TU WIEN Informatics

Computational Optimization Approaches for Distributing Service Points for Mobility Applications and Smart Charging of Electric Vehicles

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Thomas Jatschka, BSc
Matrikelnummer 00928678

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Zweitbetreuung: Dr.rer.nat. Tobias Rodemann

Diese Dissertation haben begutachtet:

Luca Di Gaspero	Kenneth Sörensen

Wien, 30. Mai 2022

Thomas Jatschka

TU WIEN Informatics

Computational Optimization Approaches for Distributing Service Points for Mobility Applications and Smart Charging of Electric Vehicles

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Thomas Jatschka, BSc
Registration Number 00928678

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Second advisor: Dr.rer.nat. Tobias Rodemann

The dissertation has been reviewed by:

Luca Di Gaspero	Kenneth Sörensen

Vienna, 30th May, 2022

Thomas Jatschka

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Thomas Jatschka, 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, 30. Mai 2022

Thomas Jatschka

Acknowledgements

I want to express my sincere gratitude to Günther Raidl for supervising me during these last four years. I am grateful for his guidance and him always taking the time to support me in my research whenever I was stuck. I also appreciate the many opportunities provided for collaborating on various interesting projects.

I would also like to thank my second supervisor, Tobias Rodemann. I greatly enjoyed the fruitful collaboration with him and Honda Research Institute Europe GmbH (HRI EU) and am thankful to Tobias and Günther for giving me this opportunity. Besides Tobias, I also want to thank Viktor Losing and Martin Heckmann from HRI for the interesting discussions in our advisory meetings.

Next, I want thank Steffen Limmer, Fabio Oberweger, and Benjamin Schaden for coauthoring with me.

I am thankful to HRI for funding my PhD position and to Honda R&D Co., Ltd. for additional financial support. At this point I also want to thank Yusuke Okamoto, Hiroaki Kataoka and Tadashi Hayashida from Honda R&D Co., Ltd. for their valuable insights.

Further, my thanks also go to my coworkers at the Algorithms and Complexity group for the great work environment and helpful discussions. I want to especially thank Nikolaus Frohner for taking the time to read this thesis and providing much appreciated feedback.

Last but not least, I want to thank my parents and my brother Johannes for supporting me throughout my whole life.

Kurzfassung

Für viele Geschäftsmodelle in der Mobilitätsbranche wird eine optimale Verteilung von Servicestellen für eine Kundengemeinschaft benötigt. Beispiele dafür sind Ladestationen oder Batterietauschstationen für elektrische Fahrzeuge, Fahrradverleihsysteme oder Reparaturstellen für Fahrzeuge. Zwei grundsätzliche Probleme sind dabei die Erhebung der notwendigen Daten, um die Kundennachfrage abzuschätzen, sowie das Identifizieren von optimalen Orten zum Platzieren von Servicestellen anhand der erhobenen Daten. Üblicherweise werden diese zwei Probleme separat gelöst. Allerdings ist das Erheben von Kundeninformation auf diese Art von Grund auf unvollständig und so gut wie immer fehleranfällig, da hier etliche Aspekte auf komplexe, nicht offensichtliche Weise zusammenspielen und deren Einfluss mit den üblichen Modelliermethoden oft nicht erfasst werden kann.

In dieser Arbeit präsentieren wir Lösungsansätze die beide Probleme, das Erheben der Daten sowie die Optimierung der Orte für Servicestellen, auf einmal lösen. Die Ansätze basieren auf einer Zusammenarbeit eines Präferenz-basierten Optimierungsalgorithmus und der Kundengemeinschaft. Anstatt die Anforderungen der Kunden auf die übliche Art und Weise im Voraus zu erheben, werden die Kunden direkt in den Optimierungsprozess eingebunden, indem diese ihre Präferenzen zu geeigneten Orten für Servicestellen kundgeben können. Dadurch kann Kundenwissen über lokale Gegebenheiten besser berücksichtigt. Vorteile dieser Methode sind eine schnellere und billigere Erfassung von Kundeninformation, die direkte Einbindung von Kunden in den Planungsprozess, eine stärkere emotionale Verbindung zwischen den Kunden und dem Produkt sowie besser akzeptierte Optimierungsergebnisse.

Ein spezielles Problem, das in so einem kooperativen Ansatz berücksichtigt werden muss ist, dass der Beitrag von einzelnen Kunden von Grund auf egoistisch ist. Daher werden spezielle Techniken benötigt, um individuelles Feedback zu verarbeiten und allgemein gültige Schlüsse daraus zu ziehen.

Ein anderer wichtiger Aspekt ist, dass Benutzer nur mit einfachen Fragen, deren Antworten aber zeitgleich stark richtungsweisend für das Zielsystem sein sollen, konfrontiert werden sollen. Ein bedeutender Nachteil von interaktiven Algorithmen ist, dass deren Performance stark von der Qualität der Kunden, die mit dem System interagieren, abhängt. Stetige Benutzerinteraktion führt schlussendlich zu einer Erschöpfung der Benutzer, wodurch ihre Fähigkeiten, zuverlässiges Feedback zu geben, stark beeinflusst werden. Aus diesem

Grund soll Benutzerinteraktion nicht nur als zeitaufwendig, sondern auch als knappe Ressource angesehen werden. Schließlich muss auch noch entschieden werden, wie das erhaltene Feeback in der Optimierung berücksichtigt wird. Die Kernoptimierung ist typischerweise bereits ein herausforderndes Problem. Daher erhöht die Berücksichtigung von Kundenfeedback in der Optimierung zusätzlich die Komplexität des Problems.

In dieser Arbeit werden drei verschiedene Probleme erforscht. Als Einführung zum Thema Verteilen von Servicestellen für Mobilitätsanwendungen, behandelt das erste Problem das Identifizieren von optimalen Orten für Servicestellen unter der Annahme, dass die benötigte Kundeninformation bereits bekannt ist. Im Speziellen betrachten wir das Problem zum Verteilen von Batterietauschstationen für elektrische Scooter, deren Batterien in ein paar einfachen Schritten ausgetauscht werden können. Entladene Batterien werden an den Tauschstationen wieder aufgeladen und, sobald sie wieder voll sind, den Kunden wieder zur Verfügung gestellt. Unser Ziel ist nicht nur das Identifizieren von optimalen Orten für diese Tauschstationen, sondern auch das Ermitteln ihrer Kapazität, um eine gewisse Menge an Kundennachfrage mit minimalen Kosten erfüllen zu können. Das problem wird als gemischt ganzzahliges Problem formuliert und eine *large neighborhood search* wird entwickelt zum Lösen von entsprechenden Instanzen, abgeleitet von realen Taxidaten von Manhattan.

Die zweite Problemformulation, die in dieser Arbeit erforscht wird, ist allgemeiner formuliert, berücksichtigt dafür aber Benutzerinteraktion zum Erheben der Kundenanforderungen verwoben mit dem Opimierungsprozess. Daher berücksichtigen wir in diesem Problem neben der eigentlichen Optimierung auch, wie Kundeninformation erfasst werden kann. Zu diesem Zweck präsentieren wir in dieser Arbeit einen kooperativen Optimierungsansatz zum Verteilen von Servicestellen für Mobilitätsanwendungen. Dieser Ansatz ist iterativ und optimiert an welchen Orten Servicestellen errichtet werden sollen, indem eine Optimierungskomponente mit Benutzerinteraktion groß angelegt kombiniert wird. Eine *machine learning* Komponente stellt dazu die Auswertungsfunktion von Lösungen während der Optimierung bereit. In jeder Iteration des Algorithmus werden Kandidatenlösugen generiert und den Kunden vorgeschlagen, die machine learning Komponente wir neu trainiert und die Optimierung wird benutzt, um eine neue Lösung, die den Benutzerbedürfnissen entspricht, zu finden. Wir stellen das Framework unseres kooperativen Ansatzes, welches in der Lage ist eine Vielzahl von Mobilitätsproblemen zu lösen, vor. Zusätzlich diskutieren wir verschiedene machine learning Modelle, um die erhaltenen Benutzerdaten zu verarbeiten, sowie verschiedene exakte und heuristische Optimierungsansätze zum Lösen des zugrunde liegenden Problems.

Schließlich betrachten wir auch ein anderes Optimierungsproblem, welches ebenso essentiell ist, um einen effektiven Mobilitätsservice zu etablieren. Im Speziellen betrachten wir die Betriebsfähigkeit des Services. Wir untersuchen das Problem des Planens von elektrischen Fahrzeugen an einer Ladestation, sodass die zeitlichen Verfügbarkeiten der Fahrzeuge sowie die maximal verfügbare Ladeleistung an der Station berücksichtigt wird. Unter Berücksichtigung von zeitlich abhängenden Energiepreisen sollen die Kosten zum Laden der Fahrzeuge minimiert werden. Ein besonderer Aspekt, den wir hierbei

untersuchen ist, dass die maximale Ladeleistung, mit der ein Fahrzeug geladen werden kann, vom aktuellen Ladezustand des Fahrzeuges abhängt. Wir schlagen zwei Methoden vor, um das Problem zu lösen. Der erste Ansatz ist ein *cutting plane* Ansatz, der die konvexe Hülle der im allgemeinen nicht konkaven Kurve des Ladezustands ausnutzt. Der zweite Ansatz basiert auf einer stückweisen Linearisierung der Kurve des Ladezustands und wird mittels *branch-and-bound* gelöst. Beide Ansätze werden in experimentellen Untersuchungen evaluiert anhand künstlich erstellter Testinstanzen, welche teilweise von realen Daten abgeleitet wurden.

Abstract

For many business models in the mobility domain an optimal distribution of service points in a customer community is needed. Examples are charging stations of electric vehicles (EVs), bicycle sharing stations, battery swapping stations, or repair stations. Two main challenges are to get the necessary data about the community and environment in order to estimate user demands, local constraints of potential locations, and other properties and to identify optimal service station locations based on these data. Traditionally, these two tasks are considered in a separated fashion. Obtaining input data for the optimization step in a classical way essentially always is inherently incomplete and error prone for larger practical scenarios since manifold aspects play roles in complex, often non-obvious ways, and not all of them can be captured with appropriate estimations of their impacts.

In this work we present approaches for solving both challenges, the data acquisition and the optimization, in a combined way by a cooperation of a preference-based optimization algorithm and customers. Instead of estimating customer demands upfront, customers are incorporated directly into the optimization process, i.e., users can interact with the optimization algorithm by expressing their preferences for where to best place service points. Potential customers further know local situations and their particular properties, including also special aspects that cannot be easily captured in a classical data acquisition approach. The expected benefits of such an approach are a faster and cheaper data acquisition, the direct integration of users into the whole planning process, possibly a stronger emotional link of the users to the product, and ultimately better and more accepted optimization results.

A particular challenge to be considered for such a cooperative approach is that the input from each individual user is inherently egoistic. Hence special techniques are necessary for aggregating, interpolating, and extrapolating individual user feedback in order to derive more globally valid aspects. Another important aspect is that users must be confronted with easy questions or tasks whose answers at the same time provide strong guidance for the target system. A major disadvantage of interactive algorithms is that their performance strongly depends on the quality of the feedback given by the interactors. Continuous user interactions will eventually result in user exhaustion, negatively influencing the reliability of the obtained feedback. Therefore, user interactions should not only be considered time consuming but users also need to be treated as a

scarce resource. Finally, one has to decide how to incorporate the obtained user feedback into the optimization. The core optimization problems are typically already challenging to solve. Hence, considering user feedback during the optimization adds an additional layer of complexity that needs to be addressed.

In this thesis three different problems are investigated. As an introduction to distributing service points for mobility applications, the first problem focuses on the challenge of identifying optimal service station locations under the assumption that demand information is already known. Specifically, we investigate the problem of setting up battery swapping stations for electric scooters. For the considered electric scooters, batteries can be swapped quickly in a few simple steps. Depleted batteries are recharged at these swapping stations and provided again to customers once fully charged. Our goal is to identify optimal battery swapping station locations as well as to determine their capacities appropriately in order to cover a specified level of assumed demand at minimum cost. The problem is formulated as a Mixed Integer Linear Program (MILP) and a Large Neighborhood Search (LNS) is developed for solving instances derived from real-world taxi data of Manhattan.

The second problem is more generic but therefore considers user interaction for obtaining demand information interleaved with the optimization. As core concept of this thesis we present a Cooperative optimization Approach (COA) for distributing service points of mobility applications. This approach is an iterative algorithm that optimizes the location of service points by combining an optimization component with user interaction on a large scale and a machine learning component that provides the objective function for the optimization. In each iteration candidate solutions are generated and suggested to the future potential users for evaluation, the machine learning component is (re-)trained on the basis of the collected feedback, and the optimization is used to find a new solution fitting the needs of the users as well as possible. We propose a framework for COA that is suitable for solving a large range of mobility applications, such as charging stations of electric vehicles or vehicle sharing systems. Additionally, we discuss different approaches for implementing the components of COA. Specifically, we propose different machine learning models for processing the obtained user feedback and different optimization techniques for solving the underlying problem w.r.t. the current user information including mixed integer programming as well as heuristic methods.

Finally we consider a different optimization problem which is also crucial for establishing an effective electric mobility service. Specifically, the operability of the service. We investigate the problem of scheduling the charging of EVs at a single charging station such that the temporal availability of each EV as well as the maximum available power at the station are considered. The total costs for charging the vehicles should be minimized w.r.t. time-dependent electricity costs. A particular aspect we investigate in this context is that the maximum power at which a vehicle can be charged depends on the current state of charge (SOC) of the vehicle. Two methods for solving the scheduling problem are proposed. The first one is a cutting plane method utilizing a convex hull of the in general nonconcave SOC-power curves. The second method is based on a piecewise

linearization of the SOC-energy curve and is effectively solved by branch-and-cut. All proposed approaches are vigorously experimentally evaluated on artificial benchmark scenarios partly derived from real world data.

Contents

CHAPTER 1

Introduction

Traveling in an urban environment has undergone many changes within the last decades due to continuous progress in the automotive industry as well as a trend towards more environmentally friendly means of transportation. Nowadays, in most cities there exists a growing number of alternative modes of transportation to traveling by classical gasoline powered vehicles. The number of people replacing their gasoline powered vehicle with an electric vehicle (EV) has strongly increased in recent years and is expected to increase even further in the future [1]. Additionally, vehicle sharing projects have become increasingly popular in many cities around the world [2, 3, 4]. A crucial requirement for establishing a new mobility service in an area is the installation of the necessary infrastructure. A major part of this process also includes the installation of *service points* at selected locations in the respective area. For example, certain types of vehicle sharing services require service points for picking up and returning vehicles. Furthermore, batteries of EVs need to be recharged or exchanged regularly at corresponding stations.

Naturally, a major challenge to overcome for a mobility service to be successful is the proper placement of such service points. Service points should be placed in such a way that the utility of the service system is maximized while taking into account limitations, such as financial budgets or inaccessible areas. We refer to such problems generally as service point distribution problems (SPDPs). Two main challenges for solving SPDPs are to first get the necessary data about the community and environment in order to estimate user demands, local constraints of potential locations, and other properties and to then identify optimal service station locations based on these data. Traditionally, in the literature the data acquisition and the optimization step are considered in a separated fashion, e.g., [5, 6, 7, 8] for setting up vehicle sharing systems or [9, 10, 11, 12] for setting up charging stations for EVs.

The first part of this thesis is dedicated to the second challenge, identifying optimal locations for service points under the assumption that user demands have already been acquired a priori. For this purpose we discuss the problem of setting up battery swapping

stations for electric scooters in an urban area. The goal is to identify optimal battery swapping station locations as well as to determine their capacities appropriately in order to cover a specified level of assumed demand at minimum costs. However, as battery swapping stations only have a limited capacity one has to consider how users behave when their preferred stations are not available. Moreover, an additional aspect that needs to be considered is that depleted batteries are recharged at these swapping stations and provided again to customers once fully charged. While there already exists work for setting up a system of battery swapping stations, e.g. [13, 14], to the best of our knowledge, there is no previous work that considers specifically the aspect of recharging and reusing returned batteries and its implications concerning station capacities when optimizing station locations and configurations. We formulate the problem as a Mixed Integer Linear Program (MILP) and present a Large Neighborhood Search (LNS) for solving it. The performance of the LNS is compared to the performance of the MILP on artificial instances as well as instances derived from real-world taxi data of Manhattan. More specifically, instances are derived from a large set recorded taxi trips between the taxi zones of Manhattan.

In the second part of this thesis, we investigate SPDPs under the assumption that user demands are not completely known a priori. Obtaining input data for the optimization in a classical way essentially always is inherently incomplete and error prone for larger practical scenarios since manifold aspects play roles in complex, often non-obvious ways, and not all of them can be captured with appropriate estimations of their impacts. For example, some customers might use multiple modes of transport for a single trip [15, 16]. Consequently, some more distant service stations might be acceptable to a customer if they are well connected by public transport used for an additional last leg.

Therefore, we propose an approach for solving both challenges, the data acquisition and the optimization, in a combined way by a cooperation of a preference-based optimization algorithm and a larger base of customers. Users are able to continuously provide feedback during the optimization, allowing them to express their preferences in a more direct way than in a classical data acquisition approach. The expected benefits of such a cooperative approach are a faster and cheaper data acquisition, the direct integration of users into the whole planning process, a stronger emotional link of the users to the product, and ultimately better and more accepted optimization results. Potential customers further know local situations and their particular properties, including also special aspects that are not foreseen to consider in a classical data acquisition approach. For example, if public transport between two locations is not reliable, e.g. due to frequent delays, users may wish for an alternative travel method. We first specify a class of Cooperative Service Point Distribution Problem (CSPDP) in which we define how users can provide feedback. Afterwards, we present a framework for a Cooperative optimization Approach (COA) for solving such problems. This framework consists of three main components, one for obtaining information from users, one for processing the obtained information, and one for generating optimized solutions based on the processed user feedback. Note that in this thesis, the focus is put on the core principles and the interaction of the different

components of COA. More work will be needed to actually apply COA in a real-world application. Further challenges concern a suitable user interface and a corresponding distributed implementation for obtaining user feedback where also psychological aspects of users need to be considered.

Depending on the concrete problem to be solved, these components might be realized in different ways, especially in regard to how the obtained user feedback is processed. Therefore, we investigate two different CSPDPs and show how these can be solved with the COA framework.

In one of the investigated CSPDPs users are considered independent of each other. For each combination of users and potential locations for service points individual machine learning models are used for processing user feedback and for making predictions about unknown user information. Initially, these machine learning models are simple linear regression models. We use an approach inspired by [17] in which a machine learning model is incrementally upgraded to higher complexity ones when the error of the model exceeds a certain threshold. Hence, as more and more user feedback is obtained during the course of COA, each model is upgraded, e.g., a feedforward neural network, to make more accurate predictions about users and appropriate service point locations. To generate optimized solutions for this problem we propose and compare two heuristic approaches, a variable neighborhood search as well as a population based iterated greedy approach. Our results show that our ensemble of machine learning models is able to learn the non-trivial user behavior of all our benchmark scenarios reliably and the optimization is able to find solutions with only small remaining optimality gaps.

The second CSPDP we investigate is modeled based on the assumption that given a sufficiently large number of customers there are users which have similar preferences about suitable locations for service points. To exploit this assumption in COA, we make use of a matrix factorization based machine learning model which is a popular approach used in recommender systems for suggesting products to users [18]. Users are asked to provide values for how suitable a service point location is to their specific needs. Based on these suitability values a matrix factorization model is used to estimate the suitability of so far unrated locations for each individual users. Moreover, for this problem COA always asks users for feedback about their most suitable service point locations. Therefore, the user information is not missing at random and tends to be biased towards more suitable service point locations. To consider this aspect, we make use of an advanced machine learning model proposed by [19] that allows us to add a bias towards so far unknown user information. Optimized solutions are obtained via heuristic and mixed integer programming techniques. Specifically, an LNS is developed which uses a MILP for repairing destroyed solutions. Further, the LNS makes use of a special graph data structure to efficiently update our rather complicated objective function. In our results we can observe that the matrix factorization based surrogate model is able to learn preferences of individual users from users with similar interests. Additionally, using the advanced matrix factorization model yields a significant improvement in the quality of the solutions. Moreover, the results show that at the cost of a slight deterioration of

usually not more than one percent in the quality of the solutions, the LNS can outperform the MILP w.r.t. to computation times by orders of magnitudes.

The COA framework is evaluated on completely artificial instances as well as instances derived form real-world data. Specifically, we derive locations for service points from bus stop shelter station data of Manhattan and user preferences are based on taxi data of Manhattan.

At this point it should be noted that setting up service points is not the only challenging aspect for successfully establishing a mobility service. Once the service infrastructure has been established, maintaining and maximizing the operability of the service points is another crucial challenge. Service points usually can handle only a limited amount of customers at a time, e.g., vehicle sharing stations can become partly unusable if they become full or empty. Moreover, charging stations for EVs usually have limitations w.r.t. how many customers can charge at once and how fast the vehicles can be charged. Hence, in the case of vehicle sharing systems, the service points need to be constantly rebalanced, ensuring that each station has the right amount of vehicles. On the other hand, charging stations require a proper scheduling strategy such that their utility can be maximized.

Therefore, the last part of this thesis is dedicated to this aspect and we consider specifically the task of finding a charging schedule for an EV fleet from the perspective of a charging station. The schedule must minimize the overall charging costs under time-dependent electricity costs while respecting each vehicle's temporal availability, its state of charge, as well as the charging station's maximum charging power. A special focus is put on the aspect that each vehicle's maximum charging power is limited by a function that depends on the vehicle's state of charge, which is particularly important for fast-charging. In related literature, e.g. [20, 21], it is typically assumed that the maximum charging power of an EV remains constant over the planning horizon. However, in practice the maximum charging power depends on the state of charge (SOC) of the EV's battery. The exact form of the charging power curve does not only depend on the type of battery and its charging controller but also on other factors like the ambient temperature or the state of health of the battery [22]. In most cases the curve is highly nonlinear. Frendo et al. [23] conclude from numerical experiments that under the constraint of a limited total charging power, up to 21% more energy can be charged if the SOC-dependent maximum charging power is considered in the planning. Considering an SOC-dependent maximum charging power for a discretized time horizon is not trivial as the maximum charging power of an EV may also change during time steps. To deal with this issue, we instead consider the maximum energy by which an EV can be charged within a time step. We propose and compare two MILP based approaches for solving the scheduling problem and evaluate these approaches in extensive numerical experiments. The first one is a cutting plane method utilizing a convex hull of the in general nonconcave SOC-power curves. The second method is based on a piecewise linearization of the SOC-energy curve and is effectively solved by branch-and-cut. The proposed approaches are evaluated on benchmark instances, which are partly based on real-world data. To deal with EVs arriving at different times as well as charging costs changing over time, a model based predictive control strategy is

usually applied in such cases. Hence, we also experimentally evaluate the performance of our approaches for such a strategy. The results show that optimally solving problems with general piecewise linear maximum power functions requires high computation times. However, problems with concave, piecewise linear maximum charging power functions can efficiently be dealt with by means of linear programming. Approximating an EV's maximum charging power with a concave function may result in practically infeasible solutions, due to vehicles potentially not reaching their specified target SOC. However, our results show that this error is negligible in practice.

1.1 Structure of the Thesis

In the next chapter we give an overview of the methodological concepts based on which the approaches in this thesis were developed.

In Chapter 3 the distribution of battery swapping stations for electric scooters is discussed. The chapter first gives an extensive overview of related problems in the literature. The problem is formally defined as a MILP, an LNS is presented for solving the problem. Afterwards, it is described how the test instances are generated and, based on these instances, the LNS is evaluated and compared to the MILP. The chapter is based on the publication

> T. Jatschka, F. F. Oberweger, T. Rodemann, and G. R. Raidl, "Distributing battery swapping stations for electric scooters in an urban area," in *Optimization and Applications, Proceedings of OPTIMA 2020 – XI International Conference Optimization and Applications* (N. Olenev, Y. Evtushenko, M. Khachay, and V. Malkova, eds.), vol. 12422 of *LNCS*, pp. 150–165, Springer, 2020.

Chapter 4 is dedicated to solving SPDPs in a cooperative way. First, related problems in the literature are discussed and it is shown how demand information for various Service Point Distribution Problem (SPDP)s have been obtained so far. The next section defines the class of CSPDPs and the way in which users can interact with the COA framework which is presented afterwards. Then, two concrete CSPDPs are defined and it is shown how these problems can be solved within the COA framework. For each of the problems it is described how the components of the COA framework are implemented and the performance of the resulting approaches is tested on artificial instances as well as real-world inspired instances. The chapter is based on the publications

- T. Jatschka, T. Rodemann, and G. R. Raidl, "A cooperative optimization approach for distributing service points in mobility applications," in *Evolutionary Computation in Combinatorial Optimization* (A. Liefooghe and L. Paquete, eds.), vol. 11452 of *LNCS*, pp. 1–16, Springer, 2019
- T. Jatschka, T. Rodemann, and G. R. Raidl, "VNS and PBIG as optimization cores in a cooperative optimization approach for distributing service points," in

Computer Aided Systems Theory – EUROCAST 2019, vol. 12013 of *LNCS*, pp. 255–262, Springer, 2020

- T. Jatschka, T. Rodemann, and G. R. Raidl, "Exploiting similar behavior of users in a cooperative optimization approach for distributing service points in mobility applications," in *The 5th International Conference on machine Learning, Optimization and Data science – LOD 2019* (G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, eds.), LNCS, pp. 738–750, Springer, 2019
- T. Jatschka, G. R. Raidl, and T. Rodemann, "A general cooperative optimization approach for distributing service points in mobility applications," *Algorithms*, vol. 14, no. 8, 2021
- T. Jatschka, T. Rodemann, and G. R. Raidl, "A large neighborhood search for a cooperative optimization approach to distribute service points in mobility applications," in *Metaheuristics and Nature Inspired Computing* (B. Dorronsoro, F. Yalaoui, E.-G. Talbi, and G. Danoy, eds.), vol. 1541 of *CCIS*, pp. 3–17, Springer, 2022.

In Chapter 5 we discuss the scheduling of EVs at charging stations. Again, an overview of related literature is given and afterwards the problem is described in more detail. Additionally, it is shown how to derive the maximum charging energy from the maximum charging power in an exact as well as an approximate way. Then, it is shown how to solve the scheduling problem for concave maximum energy function with a linear program. Afterwards, a more general approach is described for non-concave, piecewise linear energy function. Then, it is explained how the benchmark instances are generated and experimental results are presented. The chapter is based on the publications

- B. Schaden, T. Jatschka, S. Limmer, and G. R. Raidl, "Smart charging of electric vehicles considering SOC-dependent maximum charging powers," *Energies*, vol. 14, no. 22, 2021
- B. Schaden, "Scheduling the charging of electric vehicles with soc-dependent maximum charging power," Master's thesis, TU Wien, 2021. Supervised by G. R. Raidl and T. Jatschka.

Finally, Chapter 6 concludes this thesis and discusses future work.

CHAPTER 2

Methodologies

In this chapter we give a detailed description of the concepts on which the approaches presented in this thesis are based on. First, we introduce basic mathematical definitions. The second part is dedicated to mathematical programming approaches, specifically, linear programming and mixed integer linear programming. After giving formal definitions, we briefly describe the most important approaches for solving (mixed integer) linear programming models. Next, relevant (meta)heuristics are discussed, specifically, greedy construction heuristics, local search, population-based iterated greedy, variable neighborhood search, as well as large neighborhood search. Another important area for this thesis is machine learning. We give a brief introduction to machine learning in general and then present relevant machine learning approaches, such as linear regression, neural networks, and matrix factorization in more detail.

2.1 Basic Definitions

Vectors and matrices we follow the notation of [32]. We refer to an element of $\mathbb{R}^n$ as a vector of size n. To better distinguish vectors form scalar values, vectors are highlighted in bold font. Let

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{2.1}$$

be a vector of size n. Then x_i refers to the element of i^{th} element of $\mathbf{x}$.

Given dimensions $n, m \in \mathbb{N}$, a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ is an $m \cdot n$ tuple of elements, i.e.,

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \tag{2.2}$$

where w_{ij} refers to the element of row i and column j. Just as vectors, matrices are highlighted in bold font as well.

The transpose of a vector $\mathbf{x}$ and a matrix $\mathbf{W}$ is denoted by $\mathbf{x}'$ and $\mathbf{W}'$, respectively.

2.2 Mathematical Programming

The goal of mathematical programming is to find a minimum or maximum value of a real valued function subject to a set of constraints. For this thesis we only focus on mathematical programs that can be expressed with a linear optimization function as well as linear constraints. Moreover, as maximization problems can easily be transformed into minimization problems and vice versa in the following, only minimization problems are discussed. This overview is based on [33, 34, 35], and [36].

2.2.1 Linear Programming

In [33, p. 3] a Linear Program (LP) is defined as follows:

$$\min \mathbf{c}'\mathbf{x} \tag{2.3}$$
$$\text{s.t. } \mathbf{a_i}'\mathbf{x} \geq b_i \qquad \forall i \in M_1 \tag{2.4}$$
$$\mathbf{a_i}'\mathbf{x} \leq b_i \qquad \forall i \in M_2 \tag{2.5}$$
$$\mathbf{a_i}'\mathbf{x} = b_i \qquad \forall i \in M_3 \tag{2.6}$$
$$x_j \geq 0 \qquad \forall j \in N_1 \tag{2.7}$$
$$x_j \leq 0 \qquad \forall j \in N_2 \tag{2.8}$$

or in a more compact form

$$\min \mathbf{c}'\mathbf{x} \tag{2.9}$$
$$\text{s.t. } \mathbf{A}\mathbf{x} \geq \mathbf{b} \tag{2.10}$$
$$\mathbf{x} \in \mathbb{R}^n \tag{2.11}$$

The goal of an LP is to find an assignment of the *decision variables* $\mathbf{x} = (x_1, \ldots, x_n)$ such that the objective function of the problem (2.3), i.e., a dot product of the decision variables $\mathbf{x}$ and a cost vector $\mathbf{c}$, is minimized and all Constraints (2.4) - (2.8) are satisfied. In linear programming a constraint can generally be seen as a comparison between a dot product of the decision variables x with a vector $\mathrm{a_i}$ to a scalar b_i. A solution to an LP is referred to as feasible if all of the LP's constraints are satisfied w.r.t. the variable assignment of the solution and as infeasible otherwise. Additionally, if a solution is not only feasible but also minimizes the objective function, the solution is also called optimal.

The constraints of a linear program can be expressed as either equalities or inequalities. The domain of a decision variable, i.e., the set of values that can be assigned to the variable, can either be restricted or free.

As previously mentioned, a minimization problem can be transformed into a maximization problem and vice versa:

$$\min \ \mathbf{c}'\mathbf{x} = \max \ -\mathbf{c}'\mathbf{x}$$

From a geometrical point of view, the feasible region, i.e., the set of all feasible solutions, of an LP can also be described by a polyhedron:

Definition 1 ([33, p. 42]). *A polyhedron is a set that can be described in the form* $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$*, where* $\mathbf{A}$ *is an* $m \times n$ *matrix and* $\mathbf{b}$ *is a vector in* $\mathbb{R}^m$*.*

Generally, LPs are in P and can therefore be solved in polynomial time. Polynomial time algorithms include the ellipsoid method [37] and interior point methods [38]. In practice, the preferred approaches for solving LPs are usually the simplex method [39] and variations of it which have exponential worst case complexity.

The basic idea of the simplex method is to travel between the extreme points of the polyhedron of an LP along the polyhedron's edges. More specifically, consider first the following definition of an active vector:

Definition 2 ([33, p. 48]). *If a vector* $\mathbf{x}^*$ *satisfies* $\mathbf{a_i}'\mathbf{x} \leq b_i$ *for some* i *in* M_1*,* M_2*, or* M_3 *the corresponding constraint is referred to as active or binding at* $\mathbf{x}^*$*.*

Definition 3 ([33, p. 50]). *Consider a polyhedron* P *defined by equality and inequality constraints, and let* $\mathbf{x}^*$ *be an element of* $\mathbb{R}^n$*.*

(a) The vector $\mathbf{x}^*$ *is a basic solution if:*

  (i) All equality constraints are active;

  (ii) Out of the constraints that are active at $\mathbf{x}^*$*, there are* n *of them that are linearly independent.*

(b) If $\mathbf{x}^*$ *is a basic solution that satisfies all of the constraints, we say that it is a basic feasible solution.*

As there is usually only a finite number of linear inequality constraints, the number of basic feasible solutions is finite as well. Moreover, note the following relation between vertices, extreme points and basic feasible solutions:

Theorem 1 ([33, p. 50]). *Let* P *be a nonempty polyhedron* $\mathbf{x}^* \in P$*. Then the following are equivalent:*

(a) $\mathbf{x}^*$ *is a vertex;*

(b) $\mathbf{x}^*$ *is an extreme point;*

(c) $\mathbf{x}^*$ *is a basic feasible solution.*

If a polyhedron is nonempty and bounded, it has at least one extreme point. Additionally, if there exists at least one optimal solution to a linear program, then there exists an optimal solution that is an extreme point of the associated polyhedron. Therefore, we can solve LPs by exploring only the extreme points of the associated polyhedron. If an extreme point is adjacent to more than one extreme point, the algorithm chooses the most cost reducing direction (w.r.t. minimization problems).

2.2.2 Mixed Integer Linear Programming

In [33, p. 452] a Mixed Integer Linear Program (MILP) is defined as follows:

$$\begin{aligned} \min\ & \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y} && (2.12)\\ \text{s.t. } & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b} && (2.13)\\ & \mathbf{x}, \mathbf{y} \geq \mathbf{0} && (2.14)\\ & \mathbf{x} \in \mathbb{Z}^n && (2.15) \end{aligned}$$

In contrast to an LP, the domains of some of the variables are sets of integers. Such variables are referred to as integer variables. A special case of integer variables are so called 0-1 or binary variables which must always be either zero or one. Note that solving MILPs is NP-hard, even when limited to binary variables only [40].

Most state of the art approaches for solving MILPs are based on Branch-and-Bound (BB) and cutting plane methods. Branch-and-bound divides the set of feasible solutions into subproblems and computes upper and lower bounds to decide whether a subproblem should be refined or discarded. Note that for minimization problems upper and lower bounds are usually referred to as primal and dual bounds. For maximization problems upper and lower bounds are usually referred to as dual and primal bounds. A primal bound is usually derived from a feasible but not necessarily optimal solution. Dual bounds are often derived from the *linear programming relaxation* of a MILP:

Definition 4 ([33, p. 462])**.** *Given a MILP*

$$\begin{aligned} \min\ & \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y}\\ \textit{s.t. } & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b}\\ & \mathbf{x}, \mathbf{y} \geq \mathbf{0}\\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

its linear programming relaxation is defined as

$$\begin{aligned} \min\ & \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y}\\ \textit{s.t. } & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b}\\ & \mathbf{x}, \mathbf{y} \geq \mathbf{0}, \end{aligned}$$

where the requirements that $\mathbf{x}$ *is a vector of integers was relaxed.*

While every feasible MILP solution is also a feasible solution to its LP-relaxation, the other way is usually not true. Clearly, an optimal solution to the LP-relaxation is a dual bound to the optimal solution of its original MILP.

The LP-based Branch-and-Bound (LP-BB) procedure, described in Algorithm 2.1, uses the LP-relaxation for deriving dual bounds for its subproblems and is one of the most common ways for solving MILPs. At the beginning, LP-BB initializes a primal bound $\overline{z}$

Algorithm 2.1: LP-based Branch-and-Bound, [35, p. 113]

Input: Problem: $\min\{\mathbf{c}'\mathbf{x} \mid \mathbf{x} \in S\}$
1: $\mathbf{x}^* \leftarrow$ primal heuristic;
2: $\bar{z} \leftarrow \text{obj}(\mathbf{x}^*)$;
3: $L \leftarrow \{S\}$;
4: **while** $L \neq \emptyset$ **do**
5: $\quad S' \leftarrow$ select problem from L;
6: $\quad \mathbf{x}^{\text{LP}} \leftarrow$ solve LP relaxation w.r.t. S';
7: $\quad$ **if** $\mathbf{x}^{\text{LP}}$ *is infeasible* **then** prune by infeasibility;
8: $\quad \underline{z} \leftarrow \text{obj}(\mathbf{x}^{\text{LP}})$;
9: $\quad$ **if** $\underline{z} \geq \bar{z}$ **then** prune by bound;
10: $\quad$ **else if** $\mathbf{x}^{\text{LP}} \in S$ **then**
11: $\quad\quad x^* \leftarrow \mathbf{x}^{\text{LP}}$;
12: $\quad\quad \bar{z} \leftarrow \underline{z}$;
13: $\quad\quad$ prune by optimality;
14: $\quad$ **end if**
15: $\quad$ **else**
16: $\quad\quad x_j \leftarrow$ choose fractional value from $\mathbf{x}^{\text{LP}}$;
17: $\quad\quad S'_1 \leftarrow \{\mathbf{x} \in S' \mid x_j \leq \lfloor x_j^{\text{LP}} \rfloor\}$;
18: $\quad\quad S'_2 \leftarrow \{\mathbf{x} \in S' \mid x_j \geq \lceil x_j^{\text{LP}} \rceil\}$;
19: $\quad\quad L \leftarrow L \cup \{S'_1, S'_2\}$;
20: $\quad$ **end if**
21: **end while**
22: **return** $\mathbf{x}^*$;

for the problem to be solved using either some heuristic approach or just some trivial bound such as $\pm\infty$. Afterwards, for a list of subproblems L the LP relaxations are sequentially solved and based on the obtained solutions it is decided whether the current subproblem should be further divided into new subproblems or whether it should be pruned. For LP-BB a subproblem can be described as a set of solutions S' over which an optimized solution should be derived. If no solution $\mathbf{x}^{\text{LP}}$ to the LP relaxation w.r.t. S' can be obtained, the current subproblem is discarded. If the objective value $\underline{z}$ of $\mathbf{x}^{\text{LP}}$ is larger or equal to $\overline{z}$, the subproblem is discarded as well since it is not possible to achieve a solution better than the current incumbent solution $\mathbf{x}^*$ w.r.t. S'. However, in case $\mathbf{x}^{\text{LP}}$ is also a feasible solution to the original (non relaxed) formulation with $\underline{z} < \overline{z}$, a new

incumbent solution as well as primal bound is found and updated correspondingly. Just as in the previous cases, the subproblem is pruned. Finally, in the case that $\underline{z} < \overline{z}$ and $\mathbf{x}^{\mathrm{LP}}$ is not a feasible solution to the original formulation new subproblems are added to L. A subproblem S' is divided into two new subproblems by first selecting one of the integer variables x_j whose value in the LP solution w.r.t. S' is fractional and then setting fixing the value of this variable to $\lfloor x_j^{\mathrm{LP}} \rfloor$ and $\lceil x_j^{\mathrm{LP}} \rceil$, respectively. When L is empty it is guaranteed that $\mathbf{x}^*$ is an optimal solution, assuming that the problem has an optimal solution.

Note that the performance of the LP-based branch and bound algorithm can be strongly influenced by choosing good strategies for choosing the next subproblem on the list as well as choosing on which variable to branch. For both of these problems multiple strategies have been investigated. Commonly used strategy for choosing the next subproblem as described in [35] are the *depth-fist search* strategy and the *best-node first* strategy. The goal of the depth-first strategy is to find a good feasible solution by descending the enumeration tree in order to have a good lower bound for pruning. On the other hand, the best-node first strategy aims to minimize the total number of nodes evaluated in the tree by always choosing the subproblem with the best upper bound. A popular strategy for deciding on which variable to branch next is to always choose the most fractional variable, i.e., for binary variables the one whose value is closest to $\frac{1}{2}$ is chosen.

Another commonly used approach for solving MILPs are cutting plane approaches. Cutting plane approaches are based on the idea that often a small subset of the problem's constraints is sufficient for finding an optimal (and feasible) solution. Hence, these approaches initially solve a relaxed version of the MILP by discarding some of its constraints. Should the optimal solution s to this relaxation be feasible w.r.t. to the original formulation then s is also an optimal solution to the original MILP. Otherwise, there is at least one of the previously discarded constraint that is not satisfied. In this case, one or more of these violated constraints are added again to the current relaxation and the relaxation is solved anew. Finding constraints that are violated is also referred to as *separation problem*. This procedure is repeated until an optimal solution to the original MILP is obtained.

This method can also be embedded into a BB procedure yielding the branch and cut procedure. Branch and cut generates cutting planes for each of its subproblems, in order to generate stronger dual bounds. For more details regarding cutting plane approaches as well as branch and cut procedures we refer to [35].

2.3 Heuristic Methods

A major disadvantage of exact approaches, such as mathematical programming, is that in general generating optimal solutions for large instances of hard problems is a time consuming process. While finding a high quality solution already requires a lot of effort, exact approaches also have to provide some kind of guarantee that the found solution is indeed optimal, i.e., via exhaustive enumeration or by comparing dual and primal

bounds. Therefore, for most problems exacty approaches are only applicable to small size instances which are rarely relevant in practical scenarios. However, often good, non-optimal solutions might already be sufficient, especially if they can be generated quickly. Procedures for generating promising solutions without quality guarantee are referred to as *heuristics*. The advantage of heuristics often is that in comparison to exact approaches, they not only are able to generate solutions quicker but they also scale better to large size instances. In this section we will discuss a selection of heuristic approaches. First, we briefly discuss construction heuristics which often serve as basis for other heuristics that aim to improve incumbent solutions. Then iterated greedy algorithms are presented and finally local search based (meta)heuristics are introduced. The review of heuristic methods is based on [41, 42, 43, 44, 36].

2.3.1 Construction Heuristics

The idea of construction heuristics is to rather quickly generate solutions from scratch. A classical approach is to build solutions step by step, i.e., to extend a partial solution iteration wise until a feasible complete solution is obtained. There are multiple strategies for deciding on how to extend a partial solution. A frequently used strategy is to extend partial solutions in a *greedy* way by always adding the component to a partial solution that results in the best objective value w.r.t. the currently known information. This strategy is referred to as greedy construction heuristic. Algorithms that require multiple initial solutions often make use of randomized greedy heuristics in which partial solutions are extend by making stochastic decisions. Algorithm 2.2 shows a classical randomized greedy approach that is commonly used in greedy randomized adaptive search procedures [41]. A solution is element-wise created. In each iteration the element that is added to the

Algorithm 2.2: Randomized Greedy Heuristic, [41, p. 285]

```
1: s ← ∅;
2: while s is not complete do
3:     RCL ← build restricted candidate list;
4:     e ← select random element from RCL;
5:     s ← s ∪ {e};
6: end while
7: return s;
```

solution is chosen from a so called restricted candidate list (RCL). The RCL is typically constructed by selecting the elements with the cheapest induced costs when added to the current partial solution. From the RCL a random element is then chosen and added to the solution. The RCL is generated in anew in each iteration w.r.t. the current partial solution. This procedure is repeated until a complete solution is obtained.

2.3.2 Iterated Greedy

Iterated greedy is an iterative procedure consisting of three steps in each iteration. Algorithm 2.3 shows a basic pseudocode of iterated greedy. First, a current incumbent solution is partially destroyed, typically by freeing a subset of the decision variables and fixing the other to their current values. Afterwards, the destroyed solution is repaired by applying a greedy construction heuristic. Finally, it is decided whether the previous solution or the repaired one should be kept. A common strategy is to keep the solution with the better objective. However, to prevent the algorithm from converging too quickly to a non optimal value, accepting the worse solution with some probability might be a viable strategy. A specific implementation of this strategy is the Metropolis criterion [45] in which a worse solution can be accepted to some probability in dependence of its quality and a temperature value that decreases in each iteration of the algorithm. The lower the temperature value, the smaller is the likelihood of accepting a worse solution. The destroy and repair procedures are repeated until some termination criterion, e.g., a time limit or a certain number of iterations in which no improved solution was found, is reached.

Algorithm 2.3: Iterated Greedy, [41, p. 552]

```
1: s ← GenerateInitialSolution;
2: s* ← s;
3: while termination criteria not met do
4:     s' ← destroy(s);
5:     s' ← repair(s');
6:     s ← accept(s, s');
7:     if s is better than s* then s* ← s;
8: end while
9: return s*;
```

An alternative way for controlling the diversification/intensification behavior of iterated greedy is to use a Population-Based Iterated Greedy (PBIG) [46]. Instead of a single solution, a population, i.e., a set of solutions, is considered. The size of the population is decided by a parameter and is fixed throughout the iterations of the algorithm. In each iteration new solutions are derived by applying the destroy and repair procedure to each individual of the population and the accepted solutions form the new population for the next iteration.

2.3.3 Local Search

In contrast to a construction heuristic, local search does not generate solutions from scratch. Instead, the goal of local search procedures is to improve the quality of an already existing solution.

Local search forms the basis of a multitude of metaheuristics, such as variable neighborhood search and large neighborhood search. The local search procedure consists

of three components. The first component is the neighborhood function which assigns to a solution s a set of neighbors $N(s)$. Instead of explicitly defining the function, a neighborhood is usually defined by some operation which, applied to s, generates all neighbors of s. The goal of local search is to find a local optimum, i.e., a solution s whose quality is not worse than any other solution in $N(s)$. Hence, a local optimum is a solution which is optimal w.r.t. some neighborhood. A solution s can be improved by replacing it with a solution s' in $N(s)$ s.t. the quality of s' is higher than the quality of s. By repeating this procedure as long as possible one eventually reaches a local optimum.

The second local search component is the step function that decides which solution in $N(s)$ replaces the original solution s. One possibility is the so called first improvement method, which replaces s with the first found solution that has higher quality. Another way to replace s is the best improvement method, which replaces s with the solution that has the highest quality in $N(s)$. Moreover, one can also just pick a single random neighbor and replace the current solution if this leads to an improvement. Note that the choice of the most suitable step function is problem specific.

The last local search component is the termination criterion which decides when to terminate the local search. Ideally, the local search continues until a local optimum has been reached. However, sometimes this may be too time consuming and then a different termination criterion like the number of iterations or the time may be used. A local optimum of a neighborhood cannot always be found in reasonable time. Therefore, we prematurely terminate the local search if a specific criterion is met.

Algorithm 2.4 shows a basic pseudocode for the local search procedure.

Algorithm 2.4: Local Search

```
   Input: initial solution s
1: while termination criteria not met do
2:     s' ← step function(N(s));
3:     if s' is better than s then
4:         s ← s';
5:     end if
6: end while
7: return s;
```

Variable Neighborhood Descent (VND) is an extension of local search in which the local optimum over a set of neighborhoods is determined. Algorithm 2.5 shows the basic procedure of VND. Starting with some initial solution s, a new solution shall be generated w.r.t. a neighborhood $N_k(s)$. If the respective step function does not yield a better solution, the procedure is repeated and the next neighborhood $N_{k+1}(s)$ is considered. VND terminates when no neighborhood is able to yield an improved solution. However, once an improved solution is found, in the next iteration the first neighborhood

Algorithm 2.5: Variable Neighborhood Descent, [43, p. 64]

Input: initial solution s, , neighborhood structures $\{N_1, \ldots, N_{k_{\max}}\}$

```
k ← 1;
while k ≤ k_max do
    s' ← step function(N_k(s));
    if s' is better than s then
        s ← s';
        k ← 1;
    end if
    else k ← k + 1;
end while
return s;
```

is considered again. Therefore, when VND terminates, the obtained solution is a local optimum w.r.t. all considered neighborhoods.

Basic Variable Neighborhood Search (VNS) [47] extends VND by a nondeterministic component. Algorithm 2.6 shows the basic VNS pseudocode. Basic VNS is almost identical to VND. However, at the beginning of each iteration a random solution s' is randomly chosen from the current neighborhood $N_k(s)$. This process is also referred to as shaking.

Algorithm 2.6: Basic Variable Neighborhood Search, [43, p. 67]

Input: initial solution s, neighborhood structures $\{N_1, \ldots, N_{k_{\max}}\}$

```
while termination criteria not met do
    k ← 1;
    while k ≤ k_max do
        s' ← random element of N_k(s);
        s' ← LocalSearch(N_k(s'));
        if s' is better than s then
            s ← s';
            k ← 1;
        end if
        else  k ← k + 1;
    end while
end while
return s;
```

Finally, General Variable Neighborhood Search (GVNS) further extends the basic VNS by not only introducing an additional dedicated set of shaking neighborhoods but by also applying a complete VND instead of a single step function in each iteration. In each

iteration of GVNS a new solution is obtained by choosing a random element from the current shaking neighborhood N_k. Afterwards this solution is optimized with VND w.r.t. to a different set of neighborhoods $\{N'_1, \ldots, N'_{l_{\max}}\}$.

Algorithm 2.7: General Variable Neighborhood Search, [43, p. 68]

Input: initial solution s, neighborhood structures $\{N_1, \ldots, N_{k_{\max}}\}, \{N'_1, \ldots, N'_{l_{\max}}\}$

while *termination criteria not met* **do**
  $k \leftarrow 1$;
  while $k \leq k_{\max}$ **do**
    $s' \leftarrow$ random element of $N_k(s)$;
    $s' \leftarrow \text{VND}(s', \{N'_1, \ldots, N'_{l_{\max}}\})$;
    if s' *is better than* s **then**
      $s \leftarrow s'$;
      $k \leftarrow 1$;
    end if
    else $k \leftarrow k + 1$;
  end while
end while
return s;

2.3.4 Large Neighborhood Search

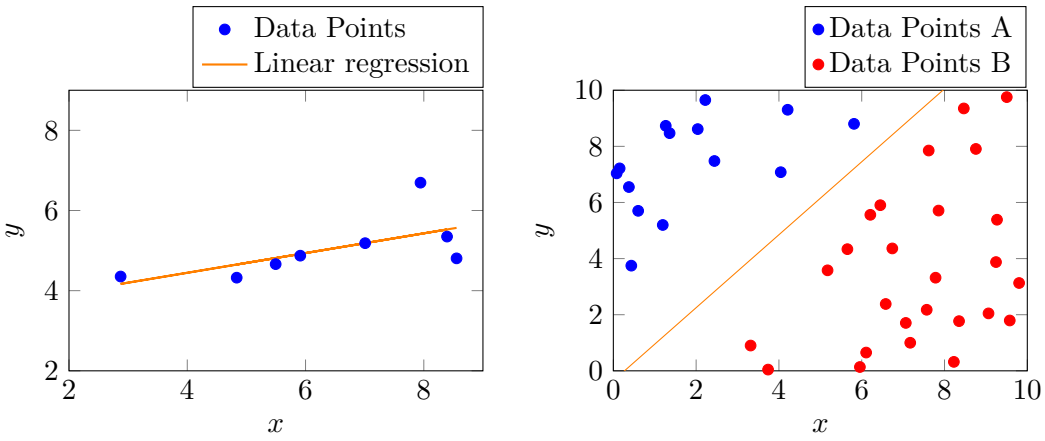
Large Neighborhood Search (LNS) [44] is a prominent metaheuristic for addressing difficult combinatorial optimization problems, which builds upon effective lower-level heuristics. A basic LNS in essence follows a classical local search framework, but usually much larger neighborhoods are considered in each iteration. The key idea is to search these neighborhoods not in a naive enumerative way but to apply some "more clever" problem-specific procedure to solve the subproblem induced by each neighborhood in order to obtain the best or a promising heuristic solution from the neighborhood. Frequently, LNS follows a destroy and recreate scheme: A current incumbent solution is partially destroyed, typically by freeing a subset of the decision variables and fixing the others to their current values, and then repaired again by finding best or at least promising values for the freed variables. Note that the basic LNS procedure is similar to the procedure of iterated greedy as described in Algorithm 2.3. However, in contrast to iterated greedy, LNS typically does not make use of construction heuristics for repairing destroyed solutions. Instead more sophisticated methods are frequently used for efficiently identifying a promising solution within a specified neighborhood for example dynamic programming or MILP-based approaches.

2.4 Machine Learning Methods

In this section we give a short overview of supervised machine learning and afterwards explain the machine learning models relevant to this thesis in more detail. The review of machine learning methods is based on [48, 32, 49, 50, 51, 52].

Machine learning has become increasingly popular in the recent years. While machine learning and its concepts have already been investigated in the 1950s [53], it is thanks to the recent increase in computing power that these concepts can today be applied in reasonable time also on huge sets of data and for more advanced applications. In [48] machine learning is defined as a computer program being able to learn from experience by performing tasks evaluated by some performance measure. More generally, the goal of machine learning is to *train* an algorithm with a set of known sample data in order to either make approximations about unknown data or to detect new patterns within the input data.

Machine learning is often divided into *supervised* and *unsupervised learning*. In supervised learning algorithms are trained from labeled data, i.e., for every input data an associated output label is provided. The most common types of supervised learning are regression and classification. For both the goal is to predict an output value for some (unknown) input. While for regression the output can be a real-valued number or vector, classification is used when the output is restricted to a set of integers representing categories. Figure 2.1 shows an example for a regression as well as a classification problem. The goal of



(a) Regression: Fitting a curve via linear regression to predict unknown data points.

(b) Classification: Fitting a function to separate two different sets of data.

Figure 2.1: Examples of supervised machine learning problems.

unsupervised learning on the other hand, is to identify patterns or similarities within data. It is often applied for clustering data into groups or for estimating the distribution of the data from which the input was sampled from. In this thesis only supervised learning algorithms are used. Note that problems involving learning from a small set of labeled

data and a large set of unlabeled data are also categorized as semi-supervised learning. For further reading on unsupervised or semi-supervised learning, we refer to [49, 54].

In supervised learning a machine learning model can be defined as a predictor

$$f_{\boldsymbol{\Theta}} : \mathbb{R}^D \to \mathbb{R} \tag{2.16}$$

mapping an D-dimensional input vector to some real-valued output. Note that the output may also be multi-dimensional. However, for simplicity we restrict ourselves to one dimensional outputs. The predictor has a set of parameters $\boldsymbol{\Theta}$. The idea of supervised learning is to find suitable parameter values $\boldsymbol{\Theta}^*$ so that $f_{\boldsymbol{\Theta}^*}$ fits the input data well, i.e.,

$$f_{\boldsymbol{\Theta}^*}(\mathbf{x}_i) \approx y_i, \quad \forall i \in \{1, \dots, n\} \tag{2.17}$$

where $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ is the input data, also referred to as ground truth.

Measuring how well $\boldsymbol{\Theta}^*$ fits the data is done by a loss function $E(\mathbf{y}, \hat{\mathbf{y}})$ comparing the ground truth labels $\mathbf{y} = \{y_i\}_{i \in \{1,\dots,n\}}$ to the predicted labels $\hat{\mathbf{y}} = \{\hat{y}_i\}_{i \in \{1,\dots,n\}}$ with

$$\hat{y}_i = f_{\boldsymbol{\Theta}^*}(\mathbf{x}_i), \quad \forall i \in \{1, \dots, n\}. \tag{2.18}$$

There are multiple ways in which E can be realized. A common loss function is the mean squared error (MSE):

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.19}$$

Naturally, the better $\boldsymbol{\Theta}^*$ fits the data, the smaller the loss reported by E. Therefore finding good parameter values is a minimization problem

$$\min_{\boldsymbol{\Theta}' \in \mathcal{T}} E((y_1, \dots, y_n), (f_{\boldsymbol{\Theta}'}(\mathbf{x}_1), \dots, f_{\boldsymbol{\Theta}'}(\mathbf{x}_n))) \tag{2.20}$$

where $\mathcal{T}$ is the domain of the parameters of f. Solving this problem is usually not trivial. Hence in many cases gradient descent based approximation approaches are employed in order to quickly compute good parameters for f.

Note however that the primary objective of supervised machine learning is not to find a model that fits the already known data but a model that also fits unknown data, i.e., values for the parameters so that the expected loss over the whole population from which the input is sampled from is minimized. While using known data to make predictions about unknown data is a reasonable approach, there is also the risk of *overfitting*, i.e., the predictor fitting too closely to the input data and not generalizing well to unknown data. To address this problem, often a parameter bias, referred to as *regularization term*, is additionally considered, i.e.,

$$\min_{\boldsymbol{\Theta}' \in \mathcal{T}} E((y_1, \dots, y_n), (f_{\boldsymbol{\Theta}'}(\mathbf{x}_1), \dots, f_{\boldsymbol{\Theta}'}(\mathbf{x}_n))) + \lambda \rho(\boldsymbol{\Theta}'). \tag{2.21}$$

Here λ determines the influence of the regularization term ρ. Introducing a regularization term to the minimization problem is an effective way for countering ill-posed parameter

configurations. However, the success of regularization depends on setting a proper value for λ that can often only be determined experimentally. Let $\mathbf{\Theta}' = \{\theta_1, \ldots, \theta_m\}$, then the two most prominent regularization methods are the L1 regularization

$$\min_{\mathbf{\Theta}' \in \mathcal{T}} E((y_1, \ldots, y_n), (f_{\mathbf{\Theta}'}(\mathbf{x}_1), \ldots, f_{\mathbf{\Theta}'}(\mathbf{x}_n))) + \lambda \sum_{i=1}^{m} |\theta_i| \tag{2.22}$$

as well as the L2 regularization

$$\min_{\mathbf{\Theta}' \in \mathcal{T}} E((y_1, \ldots, y_n), (f_{\mathbf{\Theta}'}(\mathbf{x}_1), \ldots, f_{\mathbf{\Theta}'}(\mathbf{x}_n))) + \lambda \sum_{i=1}^{m} \theta_i^2. \tag{2.23}$$

In the remainder of this section, we focus only on the machine learning models used in this thesis.

2.4.1 Linear Regression

As previously mentioned, regression deals with predicting the outcome of some observation $\mathbf{x} = (x_i)_{i \in \{1,\ldots,D\}}$ using some predictor $f_{\mathbf{\Theta}}$ with learned parameter values $\mathbf{\Theta}$. One of the simplest regression models is the linear model which combines input variables in a linear fashion, i.e.,

$$f_{\mathbf{\Theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \ldots + \theta_D x_D. \tag{2.24}$$

A *linear regression* model for D-dimensional training data, has $D+1$ trainable parameters, i.e., $\mathbf{\Theta} = \{\Theta_i\}_{i \in \{0,\ldots,D\}}$, where θ_0 is also referred to as bias. By adding a dummy input $x_0 = 1$, the model can also be written in the more compact vector form as

$$f_{\mathbf{\Theta}}(\mathbf{x}) = \mathbf{x}'\mathbf{\Theta}. \tag{2.25}$$

While one can also use gradient descent based approaches for fitting the linear model to a set of training data, a more effective approach in conjunction with the MSE as loss function is to fit the model using the *method of least squares*. Using this method $\mathbf{\Theta}$ is chosen in such way that the residual sum of squares is minimized, i.e.,

$$\min \sum_{i=1}^{n} (y_i - \mathbf{x}_i'\mathbf{\Theta})^2 \tag{2.26}$$

Let $\mathbf{y} = \{y_i\}_{i \in \{1,\ldots,n\}}$ and $\mathbf{X} = \{x_{i,j}\}_{i \in \{1,\ldots,n\}, j \in \{0,\ldots,D\}}$ where $x_{i,j}$ refers to the j^{th} variable of $\mathbf{x}_i$. Then the solution of this quadratic equation is characterized by

$$(\mathbf{y} - \mathbf{X}\mathbf{\Theta})'(\mathbf{y} - \mathbf{X}\mathbf{\Theta}) \tag{2.27}$$

By differentiating w.r.t. $\mathbf{\Theta}$ we get

$$\mathbf{X}'(\mathbf{y} - \mathbf{X}\mathbf{\Theta}) = 0. \tag{2.28}$$

Then, if $\mathbf{X}'\mathbf{X}$ is invertible, a unique solution exists, which is given by

$$\hat{\boldsymbol{\Theta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}. \tag{2.29}$$

Similar to other machine learning models, linear regression can also be regularized. Adding the L1 regularization term, the new model is referred to as *lasso regression* and as *ridge regression* when adding the L2 regularization term.

2.4.2 Feedforward Neural Networks

The perceptron is the simplest neural network and can be interpreted as a generalization of a linear regression model. The basic model of a perceptron is given by

$$f_{\boldsymbol{\Theta}}(\mathbf{x}) = \Phi(\theta_0 + \theta_1 x_1 + \ldots + \theta_D x_D) \tag{2.30}$$

where Φ is referred to as *activation function*. The activation function used for a perceptron is typically the sign function, mapping real values to {-1,+1}. Similar to before, we introduce a dummy variable $x_0 = 1$ to represent the perceptron in a more compact form, i.e.,

$$f_{\boldsymbol{\Theta}}(\mathbf{x}) = \Phi(\mathbf{x}'\boldsymbol{\Theta}). \tag{2.31}$$

Notice that if Φ is the identity function, the perceptron corresponds to a linear regression model.

The generalization of a perceptron is the multilayer perceptron consisting of three parts: an input layer, an output layer, as well as a specified number of hidden layers in between the input and output layers. The basic building block is referred to as a *neuron* which is similar to a perceptron, however, the activation function is a more general function. Each neuron feeds its output to the neurons of the next layer. As data is only fed from output to input, we refer to such neural networks as *feedforward* networks. In the standard architecture of a feedforward network all neurons of one layer are connected to all neurons of the following layer. Figure 2.2 shows an illustration of a feedforward network.

Assume that a neural network has k hidden layers, where layer i has p_i neurons. Then, the transformation of a D-dimensional input $\mathbf{x}$ can be described recursively as follows

$$\mathbf{h_1} = \Phi(\mathbf{W}_1'\mathbf{x}) \tag{2.32}$$

$$\mathbf{h_{i+1}} = \Phi(\mathbf{W}_{i+1}'\mathbf{h_i}) \qquad \forall i \in \{1, \ldots, k-1\} \tag{2.33}$$

$$\mathbf{o} = \Phi(\mathbf{W}_{k+1}'\mathbf{h_k}) \tag{2.34}$$

The learnable parameters of the input layer are given by a matrix $\mathbf{W}_1$ with size $D \times p_1$. For a hidden layer i, the matrix $\mathbf{W}_i$ is of size $p_i \times p_{i+1}$. The $k^{\text{th}} + 1$ layer is the output layer and its parameter matrix $\mathbf{W}_{k+1}$ is of size $p_k \times o$, where o is the number of outputs. Note that $\boldsymbol{\Theta}$ is the matrix formed by all $\mathbf{W}_i$ for $i \in 1, \ldots, k+1$.

The proper choice of the activation function of a network's neurons is a crucial part of a neural network's design. Figure 2.3 gives on overview of commonly used activation

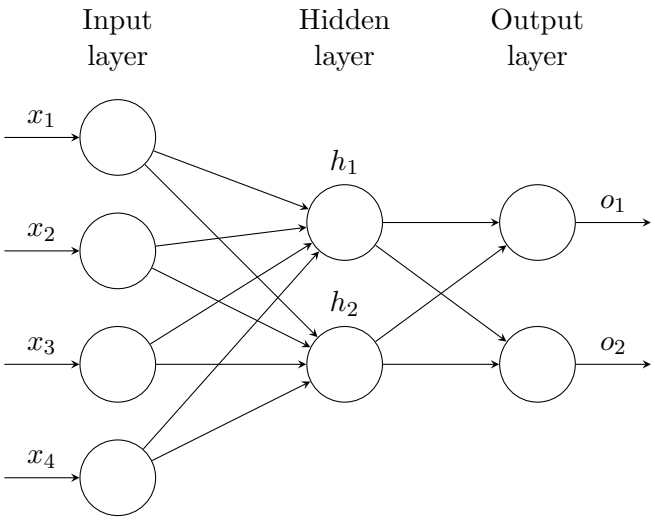
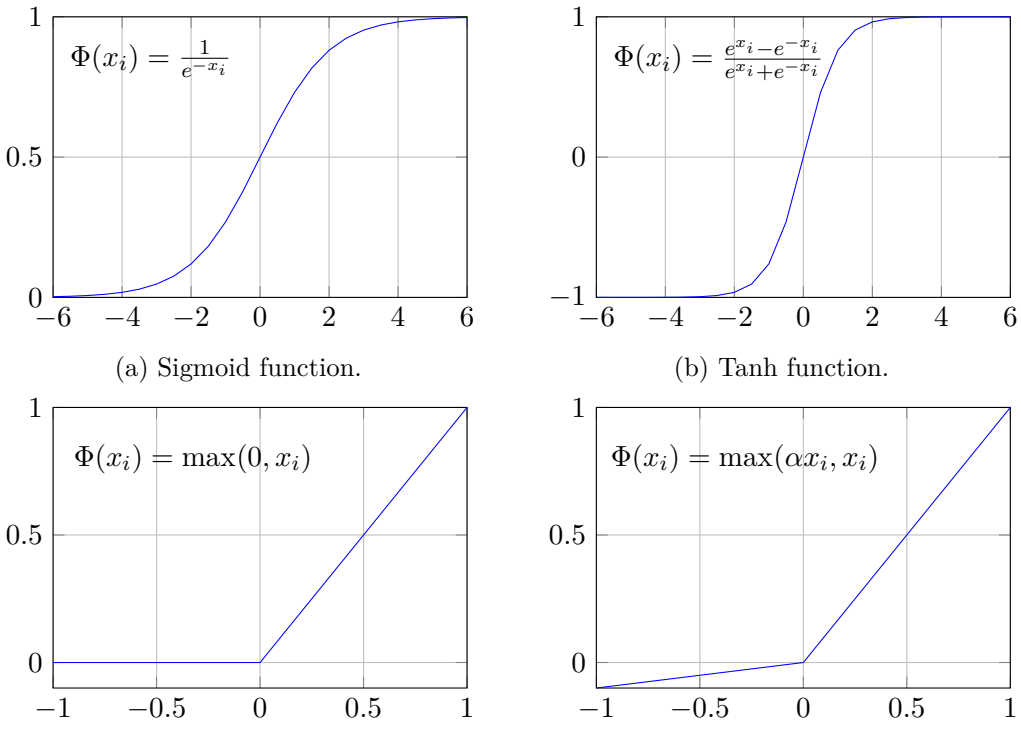


Figure 2.2: Example of a feedforward neural network.



(a) Sigmoid function.

(b) Tanh function.

(c) ReLU function.

(d) Leaky ReLU function.

Figure 2.3: Commonly used activation functions.

functions in the hidden layer. For the hidden layer a common activation function is the Rectified Linear Unit (ReLU) activation function. While the ReLU activation function works well in practice, it sometimes can get "stuck", i.e., always returning zero as output. Therefore, in such a case the leaky ReLU activation function may provide a feasible alternative.

For regression problems the output layer usually uses the identity activation function, while for classification problems the softmax function

$$\Phi_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{o} e^{x_j}} \tag{2.35}$$

is a popular choice as activation function. The softmax function is applied vector-wise, such that the sum of all outputs is one. For a more comprehensive list of activation functions we refer to [51].

Parameters $\boldsymbol{\Theta}$ can again be learned by minimizing some loss function E. Multi-layer perceptrons are usually trained via backpropagation consisting of two phases. In the forward phase training data is fed into the neural network w.r.t. the current parameter values $\boldsymbol{\Theta}$. Afterwards, the final output as well as the derivative of the loss function w.r.t. the output is calculated. In the backward phase, the gradient of the loss function w.r.t. $\boldsymbol{\Theta}$ is learned and based on this gradient $\boldsymbol{\Theta}$ is updated, i.e.,

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \alpha \nabla E(\mathbf{y}, \hat{\mathbf{y}}) \tag{2.36}$$

where α is referred to as *learning rate*.

A popular approach for training multi-layer perceptrons is the Stochastic Gradient Descent (SGD) approach. In contrast to the standard gradient descent approach, $\boldsymbol{\Theta}$ is updated in each iteration w.r.t. only a single data point $(\mathbf{x}_i, \mathbf{y}_i)$, i.e.,

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \alpha \nabla E(\mathbf{y}_i, \hat{\mathbf{y}}_i). \tag{2.37}$$

The period in which all training data are fed to the perceptron is called epoch. In each epoch the training data are fed to the perceptron in random order. The perceptron is in general trained over multiple epochs until the loss converges. Note however, that the loss might not converge if the data are not linearly separable.

Note that there also exist other training algorithms besides SGD such as Adam [55], an adaptive learning rate method which derives an individual learning rate for each parameter.

2.4.3 Matrix Factorization

Matrix factorization is a collaborative filtering technique which is frequently used in recommender systems [52]. The idea of collaborative filtering is to make recommendations for users based on the preferences of similar users, which means in our context to estimate

some user demand for a use case by the feedback already provided by other users for similar use cases. Matrix factorization is based on singular value decomposition which decomposes a matrix into two smaller matrices. Unknown values can then be estimated my multiplying the corresponding rows and columns of the decomposed matrices [52].

Given an incomplete matrix containing ratings $\mathbf{R} = (w_{i,j})_{i \in U,\, j \in P}$ for a set of users U over a set of products P, the idea behind matrix factorization is to decompose this matrix into two smaller matrices, a user/feature matrix $\boldsymbol{\xi} \in \mathbb{R}^{|U| \times \Phi}$ and a product/feature matrix $\boldsymbol{\nu} \in \mathbb{R}^{|P| \times \Phi}$, such that the product of these two matrices approximates the original matrix. The parameter Φ refers to the number of features that should be extracted and is chosen by the user. An unknown rating, i.e., a rating not contained in the original matrix $\mathbf{R}$, can then be estimated as the dot product of the corresponding feature vectors in matrix $\boldsymbol{\xi}$ and matrix $\boldsymbol{\nu}$, respectively.

Traditionally, the rating matrix $\mathbf{R}$ is factorized by solving the optimization problem

$$\min_{\boldsymbol{\xi}, \boldsymbol{\nu}} \sum_{i,j | w_{i,j} \in \mathbf{R}} E(w_{ij}, \boldsymbol{\xi}_i \boldsymbol{\nu}'_j) + \lambda \rho(\boldsymbol{\xi}, \boldsymbol{\nu}) \tag{2.38}$$

where E is again a loss function for measuring the error between the actual and the predicted ratings and ρ is a regularization term.

The two most popular techniques for decomposing a matrix with missing values are SGD [56] and Alternating Least Squares (ALS) [57]. ALS is usually only preferred over SGD for parallelization [18].

CHAPTER 3

Distributing Battery Swapping Stations for Electric Scooters

This chapter deals with identifying optimal locations for service points under the assumption that user demands are completely known in advance. Even though the demands of the users are already known, one has to also take into account the behavior of the users, once the system is operational. Demand information can only be obtained to a certain degree. However, it is difficult to estimate which service point a user will choose once the user's most preferred service points are no longer available, e.g., if their capacity is exhausted at the moment. An approach to overcome this problem is to have users assigned by some system to appropriate service points. This requires the users to specify their needs of service points in advance, e.g., via some mobile app before a trip. However, for such an approach one needs to consider that customers cannot be arbitrarily assigned to service points due to a potential unwillingness to accept large detours. We will investigate this aspect specifically for distributing battery swapping stations for electric scooters in an urban area. Due to the compactness of electric scooter batteries, depleted ones can easily be exchanged by charged ones at battery swapping stations. Exchanging such batteries can be done within a few minutes and is therefore much faster than waiting for the recharge of batteries.

We propose a Mixed Integer Linear Program (MILP) formulation for this problem that models the customer demand over time in a discretized fashion and also considers battery charging times. Moreover, we propose a Large Neighborhood Search (LNS) for addressing larger problem instances for which the MILP model cannot practically be solved anymore. The approaches are tested on artificial instances as well as instances derived from real-world taxi and bus stop shelter data of Manhattan. Part if this chapter was published as:

T. Jatschka, F. F. Oberweger, T. Rodemann, and G. R. Raidl, "Distributing battery

swapping stations for electric scooters in an urban area," in *Optimization and Applications, Proceedings of OPTIMA 2020 – XI International Conference Optimization and Applications* (N. Olenev, Y. Evtushenko, M. Khachay, and V. Malkova, eds.), vol. 12422 of *LNCS*, pp. 150–165, Springer, 2020.

Moreover note that this problem of distributing battery swapping stations was studied as part of a collaboration with Honda R&D Co., Ltd., Japan.

3.1 Introduction

Recharging the batteries of electric vehicles is usually a time-consuming process that hinders the large-scale adoption of such vehicles, especially when their range without reloading is too limited. An alternative possibility is to build electric vehicles in which the batteries can be replaced with charged ones. Batteries for electric scooters are compact enough to be replaced directly by any customer in a few simple steps. Replacement batteries are provided in exchange for the used ones at swapping stations. Returned batteries are recharged at these stations, and once fully charged, they are again provided for exchange.

We aim at investigating how to best distribute such battery swapping stations in a given urban area and how many battery slots and corresponding batteries are required at each station. Our optimization goal is to minimize the setup costs for stations in dependence of their numbers of slots and required batteries in order to cover a specified amount of user demand over multiple consecutive time periods. It is assumed that customers who want to change 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, a better utilization of the swapping stations can be achieved. However, such an automated assignment also needs to consider a certain customer dropout as not every customer is willing to travel to a predestined station if the detour is long. We assume that all scooters in our system are homogeneous and therefore require the same batteries and have the same range. Moreover, since the scooters are operating in an urban area, it is safe to assume that a scooter's range is usually larger than the length of a customer's single trip. Hence, we do not consider multiple battery swapping stops for a single trip. In fact, a scooter battery is typically exchanged after multiple trips only. We model this problem as a MILP. Smaller problem instances can be solved by directly applying a state-of-the-art MILP solver. To address the aspect of scalability to larger instances, where the MILP solver does not yield satisfactory solutions anymore, an LNS is proposed. The approaches are experimentally evaluated on artificial benchmark scenarios as well as one instance derived from real-world yellow taxi trip data and bus stop shelter station data of Manhattan.

The remainder of this chapter is structured as follows. Section 3.2 reviews relevant related work. Section 3.3 presents the problem formalization in the form of a MILP. The LNS heuristic is described in Section 3.4. Section 3.5 explains how the benchmark

scenarios are generated. Experimental results of the proposed solution methods are given in Section 3.6. Finally, Section 3.7 concludes this article and gives an outlook on future work.

3.2 Related Work

In general, our problem can be classified as a location-allocation optimization problem [58]. Specifically, our problem is closely related to the Capacitated Multiple Allocation Fixed Charge Facility Location Problem (FLP) [59] in which customers need to be assigned to facilities in order to satisfy their demand while minimizing costs for building facilities and serving customers. Moreover, the customer demand can be split arbitrarily between multiple facilities. When allocating customers to facilities from the perspective of the facility provider without considering the customers' preferences, one frequently has to expect a certain amount of customer dropout which we model with the help of a decay function as done in, e.g., [60, 61, 62]. Facility location problems with time dependent parameters are also referred to as multi-period FLPs [59]. One example for a multi-period FLP can be found in [63], where the dynamic maximal covering problem is considered.

Hodgson [64] introduced the Flow Capturing Location Model (FCLM) which is an adaptation of the Maximal Covering Location Model [65] for covering demand along paths in an underlying given graph. Customer demand is given as an *origin-destination pair* (O/D pair) and it is assumed that a shortest path is chosen to get from the origin to the destination. For each specified path there is a set of facilities that can capture the respective flow/demand. Hodgson as well as Berman et al. [61] argue that in order to cover the most demand, these facilities should be placed directly at the nodes of a path to also cover the demand of other paths going through the same node.

Moreover, our problem exhibits similarities with the Capacitated Deviation-Flow Refueling Location Model (CDFRLM) introduced in [66], which is an extension of the Flow Refueling Location Model (FRLM) introduced by Kuby and Lim [67]. The FRLM aims to locate a fixed amount of refueling stations to maximize the total flow volume refueled. In its original form, this approach for the FRLM is computationally expensive and can only be applied to small instances. More efficient formulations of the FRLM that can deal with larger instances are described in [68, 69]. Moreover, instead of placing a fixed number of stations to maximize the total flow covered, the goal in [69] is to cover all demands with as little costs as possible. Several extensions of the FRLM have been proposed in the last years, such as the capacitated FRLM [70] in which the demand a station can satisfy is limited. The Deviation Flow Refueling Location Model (DFRLM) [62] relaxes the FRLM by allowing customers to deviate from their shortest O/D pair paths in order to go to a refueling station. Moreover, it is assumed that the number of customers willing to take a deviation from the shortest path is exponentially decreasing with the length of the deviation.

In [71] a system of car charging stations shall be built gradually over time. For this purpose the model proposed in [69] is extended to a multi-period optimization model.

Existing stations cannot be relocated in later time periods.

Literature on planning gas filling station locations is surprisingly sparse. In [72] a MILP model is presented concerning the selection of filling stations to provide with unleaded fuel.

While there already exists work for setting up a system of battery swapping stations, e.g., [13], [14], to the best of our knowledge, there is no previous work that considers specifically the aspect of recharging and reusing returned batteries and its implications concerning station capacities when optimizing station locations and configurations. However, there is work dedicated to minimizing the charging costs at a battery swapping station. For example [73] deals with obtaining an optimal charging policy while ensuring a certain level of quality of service at a battery swapping station. In [74] a mixed integer non-linear programming formulation for setting up battery charging stations for electric vehicles is presented in which also the waiting time for a free charging slot at a station is considered.

3.3 Problem Definition

In this section we formalize the problem of setting up battery swapping stations for electric scooters in an urban area. The Multi-Period Battery Swapping Station Location Problem (MBSSLP), as we call it, minimizes the costs for setting up battery swapping stations to satisfy a requested expected total demand over a whole day. To be able to consider battery charging times, we consider a day in a discretized fashion as a set of equally long consecutive time intervals given as a set of the start times $\mathcal{T}$ of the intervals; w.l.o.g., we assume $\mathcal{T} = \{1, \ldots, t_{\max}\}$. We make the simplifying assumption that charging any battery always takes the same time and only completely recharged batteries are provided to customers again. Moreover, as trips in an urban environment are usually rather short, we further assume that trips start and end in the same time interval. Additionally, we make the assumption that a customer will always be able to reach a battery swapping station before his or her battery is depleted.

Let $G = (V, A, w)$ be a weighted directed graph with node set V corresponding to all relevant geographic locations, arc set $A \subseteq V \times V$, corresponding to shortest paths between locations, and arc weights $w : A \to \mathbb{R}^+$ representing the respective travel times. We assume battery swapping stations can be set up at a subset of locations $L = \{1, \ldots, n\} \subseteq V$. Moreover, each location $l \in L$ has associated a maximal number of possible battery charging slots $s_l \geq 0$, fixed setup cost c_l for setting up a station at this location, and building costs per slot $c_l^{\mathrm{s}} \geq 0$. Customer travel demands are given by origin-destination (O/D) pairs $Q \subseteq V \times V$; let $m = |Q|$. The expected number of users that need to change batteries on trip $q \in Q$ during a time interval $t \in \mathcal{T}$ is denoted as d_q^t. The minimal amount of expected total customer demand that shall be satisfied over all time intervals in $\mathcal{T}$ is denoted by $d_{\min}$. Moreover, we are given a maximum detour length $w_{\max}^{\mathrm{detour}}$ by which a feasible path including a battery swap for some $q \in Q$ may be longer than a shortest path from the origin to the destination of q. Finally, the number of time intervals required for completely recharging a battery is referred to as t^{c}.

It is assumed that customers would always take a shortest possible path p_q for an O/D pair $q = (u, v) \in Q$, except when they have to make a detour for swapping batteries. Let the set of arcs of a shortest path p_{uv} from node $u \in V$ to node $v \in V$ be $A(p_{uv}) \subseteq A$ and its length $w(p_{uv}) = \sum_{e \in A(p_{uv})} w(e)$. Moreover, we consider for an O/D pair $q = (uv) \in Q$ a shortest path that includes a certain location $l \in L$ as intermediate stop and denote it by p_q^l. The combination of a shortest path from u to l and a shortest path from l to v forms such a shortest path p_q^l, and its length is $w(p_q^l) = w(p_{ul}) + w(p_{lv})$. Let L_q be the set of locations $l \in L$ for which $w(p_q^l) \leq w(p_q) + w_{\max}^{\text{detour}}$ for $q \in Q$, i.e., the locations that may be used for battery swaps for O/D pair q.

A solution to the MBSSLP is primarily given by a pair of vectors $\mathbf{x} = (x_l)_{l \in L} \in \{0, 1\}^n$ and $\mathbf{y} = (y_l)_{l \in L}$ with $y_l \in \{0, \ldots, s_l\}$, where $x_l = 1$ indicates that a swapping station is to be established at location l and y_l is the respective number of battery slots. Moreover, let a_{ql}^t denote the part of the expected demand of O/D pair $q \in Q$ which we assign to a location $l \in L_q$ during time period $t \in \mathcal{T}$. Note that variables a_{ql}^t are continuous as the refer to the expected demand assigned to a station. Similarly to [62], we consider the loss of users in dependence of the detour length by applying a penalty coefficient $g(q, l)$ to a_{ql}^t in order to obtain the actually expected satisfied demand $\tilde{a}_{ql}^t$ of O/D pair q at location l. As suggested in [75, 62] we use the sigmoid function for this penalty coefficient, i.e., $g(q, l) = 1/(1 + \alpha e^{\beta(w(p_q^l) - w(p_q)) - \delta_q})$, where $w(p_q^l) - w(p_q)$ is the detour distance for going to the swapping station, δ_q is a reference distance, and α and β are parameters that determine the shape of the function.

Based on the variables $\mathbf{x}, \mathbf{y}, \mathbf{a}$, and $\tilde{\mathbf{a}}$ the MBSSLP can be expressed as the following MILP.

$$\min \sum_{l \in L} (c_l x_l + c_l^{\text{s}} y_l) \tag{3.1}$$

$$x_l \cdot s_l \geq y_l \qquad \forall l \in L \tag{3.2}$$

$$\tilde{a}_{ql}^t = g(q, l) \cdot a_{ql}^t \qquad \forall t \in \mathcal{T},\ q \in Q,\ l \in L_q \tag{3.3}$$

$$\sum_{l \in L_q} a_{ql}^t \leq d_q^t \qquad \forall t \in \mathcal{T},\ q \in Q \tag{3.4}$$

$$\sum_{t'=\max(1, t-t^{\text{c}})}^{t} \sum_{q \in Q | l \in L_q} \tilde{a}_{ql}^{t'} \leq y_l \qquad \forall t \in \mathcal{T},\ l \in L \tag{3.5}$$

$$\sum_{t=1}^{t_{\max}} \sum_{q \in Q} \sum_{l \in L_q} \tilde{a}_{ql}^t \geq d_{\min} \tag{3.6}$$

$$x_l \in \{0,1\} \qquad \forall l \in L \tag{3.7}$$
$$y_l \in \{0,\dots,s_l\} \qquad \forall l \in L \tag{3.8}$$
$$0 \leq a^t_{ql}, \tilde{a}^t_{ql} \leq s_l \qquad \forall t \in \mathcal{T},\ q \in Q,\ l \in L_q \tag{3.9}$$

The goal of the objective function (3.1) is to find a feasible solution that minimizes the setup costs for stations and their battery slots. Inequalities (3.2) ensure that battery slots can only be allocated to a location $l \in L$ if a station is opened there. For better readability equalities (3.3) introduce variables $\tilde{a}^t_{ql}$ by applying the penalty coefficients $g(q,l)$ to variables a^t_{ql}. Constraints (3.4) enforce that the total demand assigned from an O/D pair q to locations does not exceed d^t_q for all $t \in \mathcal{T}$. Inequalities (3.5) ensure the required capacity y_l at all locations over all time intervals. Note that by using $\tilde{a}^t_{ql}$ instead of a^t_{ql} in (3.5), we "overbook" stations to consider the expected case, similarly as in [76]. Inequalities (3.5) also model that swapped batteries can be reused after t^{c} time intervals. The minimal satisfied demand to be fulfilled over all time intervals is expressed by inequality (3.6). Finally, the domains of the variables are given in (3.7)–(3.9).

3.4 Large Neighborhood Search

In this section we propose an LNS for solving the MBSSLP making use of a MILP in the repair step. We first show how to construct an initial solution in a fast greedy way. Afterwards, the search and destroy operators of our LNS are described.

3.4.1 Greedy Construction Heuristic

The construction heuristic generates a solution station-wise. In each iteration of the algorithm a new station is opened and demand is allocated to it. In order to decide at which location to open a station next, we first calculate how much additional demand a new station at each so far unused location could satisfy w.r.t. the already opened stations. The location with the highest ratio of additionally satisfied demand to corresponding building costs is then chosen for opening the next station.

To calculate the amount of demand a station $l \in L$ can satisfy, demand is assigned from each $q \in Q \mid l \in L_q$ for all time periods $t \in T$ to l until either the station's maximum capacity is exhausted or all demand has been assigned. The iteration order of Q is hereby decided by the decay function g such that O/D pairs with lower decay value w.r.t. l are considered first.

The construction algorithm terminates when one of the following conditions is met: at least $d_{\min}$ demand is satisfied, stations are opened at all possible locations, or no more demand can be assigned to a station anymore.

3.4.2 Destroy and Repair Operators

Let $(\mathbf{x}, \mathbf{y}, \mathbf{a})$ be a solution to the MBSSLP. Moreover, let $L(\mathbf{x}) \subseteq L$ be the set of locations for which $x_l = 1$. In a first step we create an undirected graph $G^L = (V, E)$ where

$(u, v) \in E$ for $u, v \in V$ if and only if $\{u, v\} \subseteq L_q$ for at least one O/D pair $q \in Q$.

We then derive a set of locations L_{repair} that are considered for repairing via an (r, k)-*repair operator*. The operator iteratively adds k random node sets to L_{repair} where each node set is generated by choosing a random vertex $v \in V$ as well as r random neighbors of v in G^L (less if the degree of v is less than r). Afterwards, k random locations from $L(\mathbf{x})$ are added to L_{repair}. Should, during the generation of L_{repair}, a randomly selected vertex already be in L_{repair} the repair operator chooses a new random vertex if possible. From L_{repair} we derive the set $L_{\text{destroy}} = L_{\text{repair}} \cap L(\mathbf{x})$, and close all stations at these locations.

When repairing the solution, one needs to consider how much more demand needs to be satisfied in order to make the solution feasible again and how much demand from which O/D pairs is still available to be assigned to a station. Recall that variables a refer to demand of an O/D-pair assigned to a station while variables $\tilde{a}$ refer to demand that is not only assigned to a station but also satisfied, i.e., the fraction of customers that is expected to actually arrive at their assigned station. Let $D' = (d'^t_q)_{t \in T, q \in Q}$ be the demand not yet assigned to any opened location in the destroyed solution, i.e.,

$$d'^t_q = d^t_q - \sum_{l \in L(\mathbf{x}) \setminus L_{\text{destroy}}} a^t_{ql}. \tag{3.10}$$

Moreover, let d_{sat} be the amount of total demand satisfied in the partially destroyed solution, i.e.,

$$d_{\text{sat}} = \sum_{l \in L(\mathbf{x}) \setminus L_{\text{destroy}}} \sum_{t=1}^{t_{\max}} \sum_{q \in Q} \tilde{a}^t_{ql}. \tag{3.11}$$

Hence, the goal of the repair function is to assign at least $d'_{\min} = d_{\min} - d_{\text{sat}}$ demand from D' to the locations $L' = L_{\text{destroy}} \cup L_{\text{repair}}$. For this purpose, let $I(L', D', d'_{\min})$ be the residual MBSSLP instance in which L, $D = (d^t_q)_{t \in T, q \in Q}$, and $d_{\min}$ are replaced with L',D', and $d'_{\min}$. We determine a promising heuristic solution to $I(L', D', d'_{\min})$ using a relaxation of the MILP (3.1)–(3.9): Allowing the y_l variables to be continuous, i.e., replacing (3.8) by $0 \leq y_l \leq s_l$, $\forall l \in L$, while still keeping the x_l variables integral significantly speeds up the solving of the MILP. Obtained fractional values for y_l are finally rounded up to obtain a feasible solution to the original MBSSLP again, assuming one exists.

Note that the described solving of the relaxation of the MILP followed by rounding can also be used as a standalone heuristic for the original MBSSLP, which is applicable as long as the instance is not too large. We refer to this approach as $\mathbf{y}$-*Relaxed MILP Heuristic* ($\text{RMH}_{\mathbf{y}}$). Additionally, we also considered solving the full linear relaxation of the original MILP, i.e., the linear program in which all x_l as well as y_l variables are continuous, and rounding up obtained fractional x_l as well as y_l values to the next integers; we call this heuristic Linear Programming Heuristic (LPH). In Section 3.6 we compare these approaches to each other, showing that the $\text{RMH}_{\mathbf{y}}$ heuristic is a better choice for repairing solutions than the LPH heuristic.

3.5 Test Instances

As no real problem instances are available to us we created artificial test instances with characteristics that might be expected in real scenarios. Additionally, we derived one problem instance from real-world taxi trip and bus stop data of Manhattan.

In this section we give an overview of how our instances have been generated. For a detailed description see Appendix A.

3.5.1 Random Instances for the MBSSLP

The instances are simplified scenarios modeled after a typical work day where people go to work in the morning and return home in the evening. Battery swapping stations as well as origin and destination locations of customers are located within a square area. The length of the square is decided in dependence of n such that there is an average distance of 800 meters between the station locations. Following the procedure of [66, 68] we first generate a graph by sampling random points from the square and then constructing an euclidean spanning tree from the complete graph induced by the set of sampled nodes. Afterwards randomly chosen edges are added to the graph.

The set of potential battery swapping station locations is generated by choosing random nodes from the generated graph. Costs for building a station at a location are random while costs for adding a battery slot as well as the maximum number of battery slots is the same for all stations.

Origin and destination locations are again chosen from the nodes of the previously generated graph. Note however, that only a subset of the nodes is considered to speed up the computation time. Using a log-normal distribution, to each of the considered nodes popularity values are assigned, i.e., nodes with higher weights have higher incoming and outgoing traffic. Additionally, we also assume the length of a trip made by a scooter to be log-normal distributed as well with a mean length of five kilometers. We therefore use the probability density function of this distribution for assigning weights to trips, in dependence of the length of the trip. The total demand of a trip is then the product of the popularity values of the respective origin and destination and the weight of the trip. Depending on the specified number of O/D pairs m, only the m trips with the highest total demand are then kept as O/D pairs for the problem instances.

Afterwards, the total demand of an O/D pair is distributed over 24 time intervals. We assume, according to working behavior in Austria, each customer to travel twice on his corresponding path, once in the morning to get to work and once in the evening to travel back home, and we assume that customers need to swap batteries once per trip counted here as demand. Therefore, we use two normal distributions, one with mean at eight and one with mean at eighteen to determine the demand for each time interval.

The maximal deviation distance of the users, $w_{\max}^{\text{detour}}$, is set to 400 meters and the parameters of the distance decay function are set in such a way that the decay value becomes zero at approximately 400 meters. Figure 3.1 shows the decay value $g(q, l)$ in

dependence of the deviation distance $w(p_q^l) - w(p_q)$ with the chosen parameterization of the distance decay function.

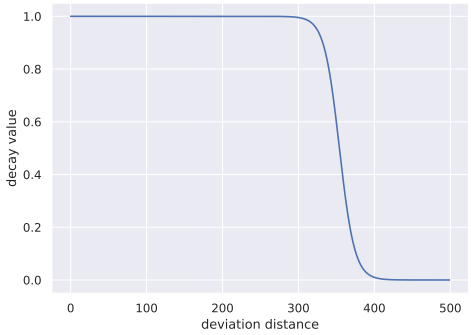


Figure 3.1: Decay $g(q, l)$ in dependence of the deviation distance $w(p_q^l) - w(p_q)$.

Eight groups of test instances for different combinations of n and m have been generated this way and each group consists of thirty instances. In Section 3.6 we evaluate the instances with $d_{\min}$ being set either to 30% or to 80% of the total swapping demand.

3.5.2 Manhattan Instance

Next to artificial benchmark instances we also derived an instance from real-world yellow taxi trip data and bus stop shelter data of Manhattan, which we call here Manhattan instance. The underlying street network of the instance corresponds to the street network graph of Manhattan provided by the Python package OSMNX[1]. Origin/Destination pairs of our instance correspond to trips between the taxi zones[2] of Manhattan. The partitioning of Manhattan into taxi zones is shown in Figure 3.3. For each taxi zone one random origin and one random destination location were chosen from the set of nodes of the network graph that are associated with the corresponding taxi zone.

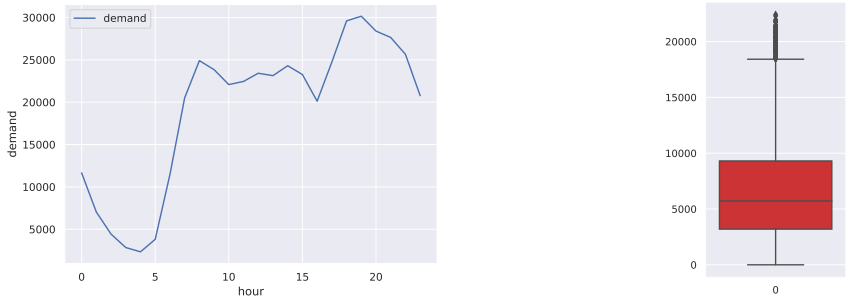
The set of O/D-pairs and their corresponding demands have been derived from the 2016 Yellow Taxi Trip Data[3]. The taxi data set was first preprocessed and all trips with invalid data as well as trips made on a weekend have been removed from the data set. Furthermore, we have also removed all trips which do not start and end in Manhattan. The preprocessed data set then consisted of 4498 unique pickup/drop-off taxi zone pairs which also constitute the instance's set of O/D pairs Q. The demand of the O/D pairs has been derived from passenger counts of trips between the respective taxi zones aggregated hourly. Figure 3.2 shows on the left how the total demand over all O/D pairs is distributed over the time intervals. Figure 3.2 shows on the right how the lengths of the O/D pairs

[1] https://github.com/gboeing/osmnx

[2] https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

[3] https://data.cityofnewyork.us/Transportation/2016-Yellow-Taxi-Trip-Data/k67s-dv2t

are distributed. Similarly to our benchmark instances, the trip lengths (given in meters) are approximately log-normal distributed with a mean between ln(5000) and ln(6000).



(a) Distribution of total daily demand.

(b) Distribution of lengths of O/D pairs in meters.

Figure 3.2: Distributions of demand and trip lengths of the O/D pairs from the real-world data based instance.

For the distance decay function and $w_{\max}^{\text{detour}}$ we use the same parameters as for the artificial benchmark instances.

The set of potential battery swapping station locations L is derived from the bus stop shelters [4] of Manhattan by selecting 500 locations randomly. Figure 3.3 shows the distribution of the stations.

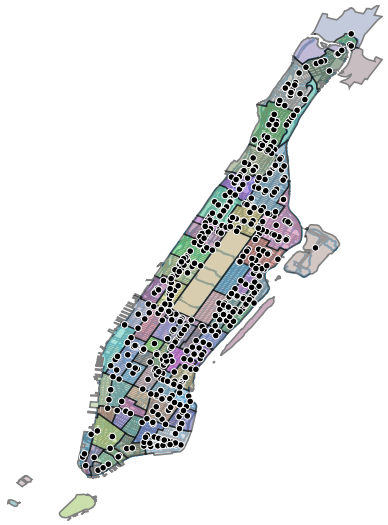


Figure 3.3: Taxi zones of Manhattan and potential locations for swapping stations.

[4] https://data.cityofnewyork.us/Transportation/Bus-Stop-Shelters/qafz-7myz

As shown in Figure 3.2 left the demand at each hour is quite high. Therefore we choose a capacity limit of 200 for each battery swapping station, The costs for building a station as well for adding a battery charging slot are chosen as for the artificial instances.

3.6 Computational Results

All algorithms were implemented in Julia[5] 1.4.2. All test runs have been executed on an Intel Xeon E5-2640 v4 2.40GHz machine in single-threaded mode with a time limit of thirty minutes. Gurobi[6] 8.1.0 was used for solving the MILPs.

First, we investigate the performance of the standalone MILP model given by Equations (3.1)–(3.9) as well as the standalone $\text{RMH}_{\mathbf{y}}$ and the LPH approach. Afterwards, the results of the LNS are discussed. Finally, in Section 3.6.3 we present the results on the instance derived from real-world data for the LNS approach as well as the MILP models. All instances are evaluated with $d_{\min}$ being set either to 30% or to 80% of the total swapping demand. Hence, let $d_{\min}[\%]$ refer to $d_{\min}$ as percentage of the total swapping demand.

3.6.1 MILP Approaches

All MILP models were solved with Gurobi 8.1.0. In case no optimal solution was found within the time limit, the solver returned the best found feasible solution if it exists.

Table 3.1 shows a summary of the performance of the exact MILP approach, $\text{RMH}_{\mathbf{y}}$ and LPH for each instance group in our benchmark set. Column “gap[%]" shows the average optimality gaps for each instance group, the median computation times are shown in column “time[s]", and column “$|L(x)|$" lists the average number of opened stations in the solutions. Note that the gaps listed for $\text{RMH}_{\mathbf{y}}$ and LPH are determined also w.r.t. the lower bounds obtained by the original MILP.

Overall, with the exact MILP solving was aborted due to the time limit for almost all instances. However, for each instance at least one feasible solution was found. Instances with up to 1000 potential battery swapping stations and 2000 O/D-pairs can be solved by the MILP almost to optimality with a gap of less than 1%. For larger instances the optimality gaps deteriorate. Compared to the results of the original MILP model, $\text{RMH}_{\mathbf{y}}$ yields in general better average optimality gaps for the three largest instance groups. The LPH approach was able to solve all instances to optimality w.r.t. the linear relaxation of the original MILP in less than 5 minutes on average. However, the derived feasible MBSSLP solutions are significantly worse than the solutions generated by $\text{RMH}_{\mathbf{y}}$ especially for $d_{\min}[\%] = 30$. For instances nearly solved to optimally, we can also observe that the number of opened stations in the solutions are as expected. $\text{RMH}_{\mathbf{y}}$ solutions require a marginally smaller number of opened stations than the MILP

[5] https://julialang.org/
[6] https://www.gurobi.com/

Table 3.1: Results of the original MILP, the $\text{RMH}_{\mathbf{y}}$ heuristic, and the LPH.

(a) MILP results for $d_{\min}[\%] = 30$.

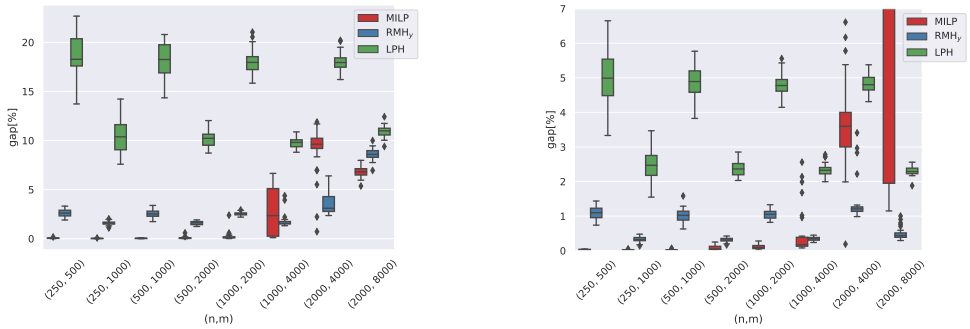
		MILP			$\text{RMH}_{\mathbf{y}}$			LPH		
n	m	gap[%]	time[s]	$\lvert L(x)\rvert$	gap[%]	time[s]	$\lvert L(x)\rvert$	gap[%]	time[s]	$\lvert L(x)\rvert$
250	500	**0.05**	1800	25	2.61	91	25	18.62	2	81
	1000	**0.02**	1800	38	1.59	125	38	10.38	4	103
500	1000	**0.03**	1800	46	2.54	287	46	18.12	5	149
	2000	**0.08**	1800	72	1.60	686	71	10.06	12	190
1000	2000	**0.24**	1800	89	2.54	1295	88	17.95	20	279
	4000	2.69	1800	192	**1.77**	1800	129	9.78	47	346
2000	4000	9.09	1800	382	**3.67**	1800	166	18.01	81	532
	8000	**6.78**	1800	531	8.60	1800	535	10.92	238	660

(b) Results for $d_{\min}[\%] = 80$.

		MILP			$\text{RMH}_{\mathbf{y}}$			LPH		
n	m	gap[%]	time[s]	$\lvert L(x)\rvert$	gap[%]	time[s]	$\lvert L(x)\rvert$	gap[%]	time[s]	$\lvert L(x)\rvert$
250	500	**0.03**	1800	47	1.09	47	47	4.98	2	86
	1000	**0.02**	1800	72	0.32	536	72	2.47	5	121
500	1000	**0.02**	1800	84	1.01	464	84	4.85	7	158
	2000	**0.08**	1800	138	0.31	1800	137	2.37	18	226
1000	2000	**0.12**	1800	160	1.04	1800	159	4.78	25	294
	4000	1.92	1800	305	**0.35**	1800	260	2.33	64	425
2000	4000	3.64	1800	488	**1.40**	1800	316	4.81	95	559
	8000	29.54	1800	1248	**0.49**	1800	515	2.31	236	815

solutions. Solutions generated from the LPH approach, on the other hand, require a much higher number of opened stations than the other approaches. Hence, LPH does not seem to be a good choice as repair procedure for the LNS.

Figure 3.4 provides a more detailed comparison of the optimality gaps of the MILP, $\text{RMH}_{\mathbf{y}}$ and LPH solutions. The figure shows boxplots of the optimality gaps for each instance group and approach and confirms our previous observations. Note that for a better comparison between the approaches Figure 3.4b is cut off and only shows optimality gaps up to 7% since solutions to the instances with $n = 1000, m = 4000$ as well as $n = 2000, m = 8000$ generated by the MILP feature optimality gaps up to 45%. For the largest instances with $n \geq 1000$ and $m \geq 4000$, $\text{RMH}_{\mathbf{y}}$ starts to produce better results than the MILP while LPH does not seem to be able to compete with $\text{RMH}_{\mathbf{y}}$ for any instance group. However, since $\text{RMH}_{\mathbf{y}}$ requires solving a large MILP as well, this approach also has its limits concerning scalability. Therefore, in the next section we investigate the LNS that uses in each iteration $\text{RMH}_{\mathbf{y}}$ to (re-)optimize only a comparably small part of a solution.



(a) Instances with $d_{\min}[\%] = 30$.

(b) Instances with $d_{\min}[\%] = 80$.

Figure 3.4: Optimality gaps of the MILP, $\text{RMH}_{\mathbf{y}}$ and LPH solutions.

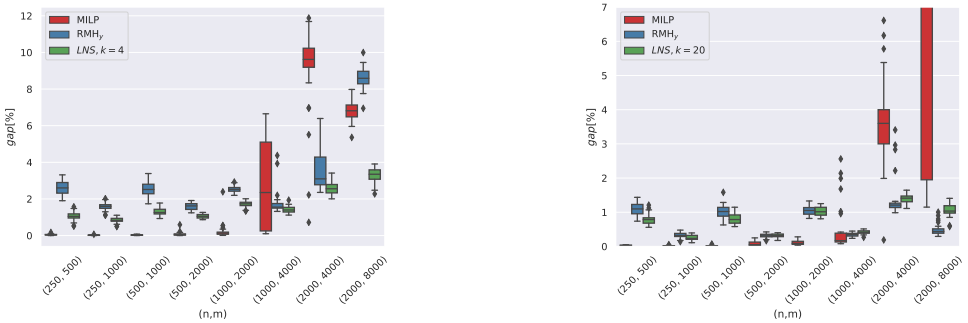
3.6.2 Large Neighborhood Search

For the size parameters of the (r, k)operator we consider here, after preliminary tests $r = 4$ and $k \in \{4, 14, 20\}$. These values are promising as the MILPs corresponding to the repair subproblems can usually be solved to a small remaining optimality gap within seconds. As the Large Neighborhood Search (LNS) is a heuristic approach, it also does not make much sense to solve the MILPs always to proven optimality; instead we terminated the MILP solver when a solution with an optimality gap of at most 0.0005% has been reached. Each LNS run was terminated after 30 minutes. The results of the LNS are shown in Table 3.2. For each considered minimum demand coverage $d_{\min}$ and each neighborhood size parameter k, the average number of iterations "iter" and the average optimality gap "gap[%]" (w.r.t. the lower bounds obtained by the original MILP).

Table 3.2: Results of the LNS.

		$d_{\min}[\%] = 30$						$d_{\min}[\%] = 80$					
		k=4		k=14		k=20		k=4		k=14		k=20	
n	m	gap[%]	iter	gap[%]	iter	gap[%]	iter	gap[%]	iter	gap[%]	iter	gap[%]	iter
250	500	**1.05**	2549	1.62	385	1.70	217	**0.57**	3222	0.75	520	0.79	270
	1000	**0.83**	1465	1.14	207	1.21	117	0.32	1843	**0.23**	263	0.25	117
500	1000	**1.31**	2094	1.64	418	1.83	207	**0.72**	2305	0.77	559	0.81	299
	2000	**1.06**	982	1.22	230	1.29	132	0.48	1115	0.33	282	**0.31**	156
1000	2000	**1.72**	1177	1.95	399	2.05	241	**1.00**	1375	1.02	482	1.03	317
	4000	**1.41**	604	1.44	203	1.45	132	0.78	606	0.46	214	**0.42**	128
2000	4000	**2.58**	698	2.64	292	2.69	211	1.59	720	1.46	331	**1.39**	251
	8000	3.28	306	3.10	128	**3.06**	93	2.00	280	1.11	128	**1.06**	87

The table shows that, naturally, the LNS can perform less iterations the larger k is. For instances with $d_{\min}[\%] = 30$ we can see that the solutions tend to deteriorate as k is increasing. However, this is not the case for instances with $d_{\min}[\%] = 80$ where we can see no such pattern. Moreover, as the instances become larger, the LNS with $k = 20$ starts to outperform the LNS with $k = 4$. Hence, for $d_{\min}[\%] = 80$ an LNS with even



(a) Instances with $d_{\min}[\%] = 30$.

(b) Instances with $d_{\min}[\%] = 80$.

Figure 3.5: Comparison of the optimality gaps of the LNS solutions to the solutions of the other approaches.

larger values for k might yield better results in theory. However, the larger k is chosen the worse the scalability of the LNS becomes as the MILP that needs to be solved in the repair procedure takes longer to solve for larger values of k.

Figure 3.5 compares the optimality gaps of solutions obtained by the LNS to the optimality gaps of the MILP and RMH$_\mathbf{y}$ solutions. Note that for a better comparison between the approaches Figure 3.5b is cut off and only shows optimality gaps up to 7%. For instances with $d_{\min}[\%] = 30$ we can see that the LNS is on average on all instance groups able to produce better solutions than RMH$_\mathbf{y}$. This particularly holds for the largest instance group, where the gap of RMH$_\mathbf{y}$ deteriorates to over 8% but the LNS' gaps are still within 4%. For instances with $d_{\min}[\%] = 80$, both, the LNS as well as RMH$_\mathbf{y}$, perform quite well with gaps usually less than 2%. The LNS solutions are here slightly worse than the RMH$_\mathbf{y}$ solutions for larger instances.

Overall, we can say that the LNS works reasonably well over all considered benchmark instances, and it is reasonable to expect it to scale much better to even larger instances than RMH$_\mathbf{y}$ or solving the original MILP directly.

3.6.3 Results on the Manhattan Instance

In this section we show how well the MILP approaches as well as the LNS were able to deal with the real-world data based Manhattan instance. While the size of n and m is similar to some of our benchmark instances, the Manhattan instance is much harder to solve than our benchmark instances due to the shape of Manhattan as well as the instance's geographic distribution of demand.

Tables 3.3 and 3.4 show respective results. Each solution approach was applied to the instance six times with different values for $d_{\min}[\%]$. For each approach the tables lists the total costs of the solutions, the corresponding optimality gaps (always w.r.t. the lower bounds obtained from the linear relaxation of the original MILP), and the computation times in seconds. The direct MILP approach was only able to find (non optimal) solutions

for the lowest levels of $d_{\min}[\%]$. $\text{RMH}_{\mathbf{y}}$ and LPH could obtain feasible solutions for all cases except with $d_{\min}[\%] = 60$. Concerning $\text{RMH}_{\mathbf{y}}$ and LPH, one can see that, as one might expect, gaps of LPH are usually significantly larger than those of $\text{RMH}_{\mathbf{y}}$, but LPH is much faster and is, in contrast to $\text{RMH}_{\mathbf{y}}$, also able to yield a feasible solution for $d_{\min}[\%] = 50$.

Table 3.4 shows the results obtained by the LNS with $r = 3$ and different values for k. Listed are total costs of the solutions, the corresponding optimality gaps (if a lower bound is known from the MILP), and the number of destroy and repair iterations. Most importantly, in contrast to the above MILP/LP approaches, the LNS could also find a feasible solution for $d_{\min}[\%] = 60$. Moreover, except for the lowest level of $d_{\min}[\%] = 10$, the LNS was able to find the best solutions. The number of performed destroy and repair iterations stays approximately the same for increasing levels of $d_{\min}[\%]$. However, as expected, the number of iterations decreases the larger the value for k.

Table 3.3: LPH, $\text{RMH}_{\mathbf{y}}$, and MILP results for the Manhattan instance.

	LPH			$\text{RMH}_{\mathbf{y}}$			MILP		
$d_{\min}[\%]$	costs	gap[%]	time[s]	costs	gap[%]	time[s]	costs	gap[%]	time[s]
10	155797	1.27	179	**153886**	0.04	1801	**153886**	0.04	1801
20	325775	2.90	140	321773	1.69	1801	320168	1.20	1801
40	692976	1.06	196	**689600**	0.57	1801	-	-	-
50	**892035**	0.77	704	-	-	-	-	-	-
60	-	-	-	-	-	-	-	-	-

Table 3.4: LNS results for the Manhattan instance.

	$k = 4$			$k = 7$			$k = 14$		
$d_{\min}[\%]$	costs	gap[%]	iter	costs	gap[%]	iter	costs	gap[%]	iter
10	153900	0.05	92	**153890**	0.05	19	154025	0.13	2
20	319769	1.07	87	319334	0.94	43	**318939**	0.82	19
40	688298	0.39	87	**687769**	0.31	42	687983	0.34	16
50	890049	0.55	83	888920	0.43	44	**887926**	0.32	24
60	1095190	-	89	**1093898**	-	43	1095097	-	15

3.7 Conclusions and Future Work

We presented the new Multi-Period Battery Swapping Station Location Problem for distributing battery swapping stations in an urban area. On our benchmark instances, directly solving the proposed MILP model is reasonable for instances with up to 1000 stations and 2000 O/D-pairs, where solutions with small gaps could be obtained. For larger instances solving the MILP model becomes quickly infeasible and heuristics need to be employed to find approximate solutions. Relaxing the y variables and rounding obtained fractional values, i.e., our $\text{RMH}_{\mathbf{y}}$, is a viable approach by which significantly larger instances can be solved reasonably well, nevertheless it also has its limits. We

therefore proposed an LNS that effectively utilizes $\text{RMH}_{\mathbf{y}}$ and provides better scalability. This can in particular be seen in the results for the real-world data based Manhattan instance.

We remark that the proposed LNS still has room for improvement. For example, different strategies for selecting the nodes to be removed or considered for addition may be investigated. Moreover, adaptive mechanisms for choosing among different destroy and re-create methods may be useful. Last but not least, there are also alternative ways to address the scalability issue, for example by approaches based on (hierarchical) clustering and iterative refinement.

In future work the MBSSLP model should also be further refined to reflect real-world aspects in a more realistic way. For example, battery swapping stations are usually not extended slot by slot but by modules which consist of multiple new battery slots. So far, we also have not yet considered a pricing model for customers or costs for maintaining the battery swapping stations and the batteries.

CHAPTER 4

Cooperative Optimization Approaches for Distributing Service Points in Mobility Applications

In the previous chapter it was assumed that the user demands have already been acquired in advance of the optimization. Deriving such demands is an intricate problem by itself, though, for which a magnitude of solutions have been proposed in literature. So far the user demand acquisition as well as the optimization step have always been considered in a separated fashion. In the approaches proposed in literature first the user demands are derived based on some given data, afterwards service point locations are optimized based on these estimated demands. In this chapter a Cooperative optimization Approach (COA) for solving SPDPs is proposed that allows users to express their preferences during the course of the optimization.

The basic concept of COA was initially presented at the *19th European Conference on Evolutionary Computation in Combinatorial Optimisation*, and published as:

> T. Jatschka, T. Rodemann, and G. R. Raidl, "A cooperative optimization approach for distributing service points in mobility applications," in *Evolutionary Computation in Combinatorial Optimization* (A. Liefooghe and L. Paquete, eds.), vol. 11452 of *LNCS*, pp. 1–16, Springer, 2019

In this original work, a Basic Variable Neighborhood Search (VNS) is used as optimization core to generate new solutions and an adaptive surrogate function [17] to process feedback.

At the *17th International Conference on Computer Aided Systems Theory* a Population-Based Iterated Greedy (PBIG) was presented as improved optimization core and compared to the VNS:

> T. Jatschka, T. Rodemann, and G. R. Raidl, "VNS and PBIG as optimization cores in a cooperative optimization approach for distributing service points," in *Computer Aided Systems Theory – EUROCAST 2019*, vol. 12013 of *LNCS*, pp. 255–262, Springer, 2020.

These first realizations of COA exhibit severe limitations in the scalability to larger numbers of potential service point locations and/or users, in particular as all users are considered independently of each other. Therefore, the COA framework was substantially revised, building upon the observation that in a larger user base there are typically users sharing the same or similar needs or preferences. This further developed framework was presented at the *5th International Conference on Machine Learning, Optimization, and Data Science*:

> T. Jatschka, T. Rodemann, and G. R. Raidl, "Exploiting similar behavior of users in a cooperative optimization approach for distributing service points in mobility applications," in *The 5th International Conference on machine Learning, Optimization and Data science – LOD 2019* (G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, eds.), LNCS, pp. 738–750, Springer, 2019.

In successive development the solution approach was extended towards applications in which the satisfaction of demand typically relies on the existence of suitable pairs or more generally tuples of service stations, such as in the case of bike or car sharing systems where a vehicle is first picked up at a rental station near the origin and returned at a station within easy reach of the destination. This approach was published as:

> T. Jatschka, G. R. Raidl, and T. Rodemann, "A general cooperative optimization approach for distributing service points in mobility applications," *Algorithms*, vol. 14, no. 8, 2021.

Finally, we further improved the scalability of the optimization component of COA by developing a Large Neighborhood Search (LNS) as optimization core. While the earlier developed VNS and PBIG considered the core problem as black box model, the LNS exploits more specific knowledge in order to efficiently generate new solutions. The LNS was presented at the *8th International Conference on Metaheuristics and Nature Inspired Computing* and published as:

.

Based on these publications we define in the following a class of problems referred to as *cooperative service point distribution problems* (CSPDPs) that we aim to solve with COA. Then we describe COA and its component in detail and afterwards present different realizations of COA for two CSPDPs. The performance of COA w.r.t. to these problems is evaluated on artificial as well as real-world inspired instances. Finally, we also provide a comparison for solving the same problem scenario with COA formulated as each of the two CSPDPs.

4.1 Introduction

A fundamental ingredient for optimizing the locations of service points in mobility applications, such as charging stations for electric vehicles or pickup and drop-off stations for car/bike sharing systems, is the distribution of existing customer demand to be potentially fulfilled in the considered geographical area. While there exists a vast amount of literature regarding setting up service points for mobility applications, such as vehicle sharing systems ([5, 6, 7, 8]) or charging stations for electric vehicles [9, 10, 11, 12], estimations of the existing demand distribution are usually obtained upfront by performing customer surveys, considering demographic data, information on the street network and public transport, and not that seldom including human intuition and political motives. However, such estimations are frequently imprecise and a system built on such assumptions might not perform as effectively as it was originally hoped for. For example in [77] GPS-based travel survey data of fossil fueled cars is used for setting up charging stations for electric vehicles. However, as pointed out by Pagany et al. [78], it cannot be assumed that the driving behavior of customers remains unchanged when switching from fossil fueled cars to electric vehicles. Furthermore, Pagany et al. [78] present a survey of 119 publications for locating charging stations for electric vehicles in which they also discuss further problems with the above mentioned demand estimation methods.

A more frequent usage of a service system by a customer will in general depend not only on the construction of a single service point on a particular location but more globally on non-trivial relationships of the customer's necessities and preferences in conjunction with larger parts of the whole service system. For example in the case of bike/car sharing systems, a well placed rental station close to the origin of a trip might be worthless if there does not also exist a suitable location near the destination for returning the vehicle. Furthermore, some customers might use multiple modes of transport for a single trip [15]. Consequently, some more distant service station for returning the vehicle might be acceptable if this place is well connected by public transport used for an additional last leg [16]. Thus, there also might be alternatives for fulfilling demand that cannot all be exactly specified by potential users. The example with an additional leg by public transport also illustrates that geographical closeness is not always the deciding factor.

To possibly improve this situation, we propose a Cooperative optimization Approach (COA) that incorporates potential customers on a large scale and more tightly into the data acquisition as well as optimization process. Instead of only acquiring demand

information from potential users upfront and evaluating candidate solutions on this basis during the optimization, we confront the potential users with certain location scenarios and ask them how these would suit their needs and how possibly these scenarios can be improved to fulfill more of their demands.

This feedback is used to incrementally gain more knowledge about how much demand may be fulfilled under which conditions. New, more promising candidate location scenarios can then be derived and again be presented to the users. The process is iterated on a large scale with many potential users and many rounds until a satisfactory solution is reached. Here, clearly many psychological aspects also come into play, which, however, shall not be our focus. For simplicity, we assume the ideal situation of perfectly reasonable and cooperative potential users who always provide a consistent and precise feedback to above basic questions.

Expected benefits of COA are a faster and cheaper data acquisition, the direct integration of users into the whole planning process, a stronger emotional link of the users to the product, and ultimately better and more accepted optimization results. Potential customers further know local situations and their particular properties, including also special aspects that are not foreseen to consider in a classical data acquisition approach.

However, for developing such a cooperative optimization approach, several issues need to be addressed. Most crucially, one needs to consider how to interact with the users. It needs to be assumed that the input from each individual user is inherently egoistic. Hence, it is necessary to employ different techniques for aggregating, interpolating, and extrapolating individual user feedback in order to derive globally valid aspects.

Another challenging aspect is that the interaction with users cannot be treated as an unlimited resource. It is a well known problem that users tend to fatigue when confronted with too many questions, substantially impacting the quality of obtained feedback [79]. Moreover, a complete direct questioning of the user would not only be extremely time consuming but users would easily be overwhelmed by the large number of possibilities, resulting in incorrect information. For example, users easily tend to only rate their preferred options as suitable and might not consider certain alternatives as also feasible although they actually might be on second thought when no other options are available. Hence, interaction with users needs to be kept to a minimum and should be done wisely to extract as much meaningful information as possible. Moreover, users must be confronted with easy questions whose answers at the same time provide strong guidance for the target system.

Finally, there need to be appropriate ways to consider the obtained customer information in the optimization. Since a heuristic approach usually generates hundreds to thousands of intermediate solutions, that all need to be evaluated, it is not an option to evaluate all these solutions via user interaction. A typical way to overcome this issue is to process the obtained user feedback via some machine learning technique to eventually derive a surrogate function. The surrogate function can then be used to estimate the quality of a solution without user interaction.

The remainder of this chapter is structured as follows: The next section gives an overview of related work. Afterwards, the in Section 4.3 we formally define the class of cooperative SPDPs. Section 4.4 introduces COA and gives a detailed description of its framework and its components. The next two sections, Section 4.5 and 4.6 then introduce two specific problems of the cooperative SPDP class and it is shown how they can be solved with COA. Finally, the section concludes with an outlook to promising future work.

4.2 Related Work

To the best of our knowledge no cooperative approach for distributing service points for mobility applications has been studied in literature so far. However, for cooperative optimization concepts a rich body of literature exists. Therefore, related work for different aspects of cooperative optimization as well as the distribution of service points is discussed in this section.

4.2.1 Distribution of Service Points in Mobility Applications

Service Point Distribution Problems (SPDPs) can be classified as a variant of the Capacitated Multiple Allocation Fixed Charge Facility Location Problem (FLP). In the FLP a set of potential facility sites and a set of demand points is given. The task is to select a subset of these sites in order to serve the demand points w.r.t. some optimization goal subject to a set of constraints. For a survey on FLPs see [80], for a more comprehensive book on location theory see [81]. More specifically, SPDPs are closely related to the uncapacitated FLPs [82] in which each facility can satisfy an arbitrary amount of demand.

SPDPs cover a large range of mobility applications. A frequently researched subclass of SPDPs are vehicle sharing systems (VSS) in which vehicles are shared among multiple users. For an overview of VSS see [5]. VSS can be realized in different operation modes. For one-way VSS a vehicle can be picked up and returned at an arbitrary station while for two-way VSS the vehicle must be picked up and returned at the same station. While one-way VSS provide a greater flexibility to the user, it is also necessary to regularly rebalance the vehicles at all stations [83]. Finally, in recent years free-floating, i.e., stationless, VSS have become popular offering the greatest flexibility but also requiring a high rebalancing effort. For an overview of various optimization problems for VSS, see [84].

Regarding the acquisition of demands for VSS, one can find multiple approaches in literature. In some work where the main focus is the optimization part, e.g., [85, 86, 6], the demands are assumed to be known in advance of the optimization, however it is not addressed how these demands can be obtained in real word scenarios.

Frade et al. [87] provide an overview of different demand estimation techniques for bike sharing systems. Early approaches identified so called generator and attractor points ([88]) which are then used to derive trip frequencies between these points. A different

approach is to derive demand from user surveys. In [89] trip frequencies are derived based on a user survey asking about preferred modes of transport, acceptable walking and waiting times, as well as acceptable associated traveling cots. Another popular method for deriving demands is geographic information system analysis ([90]). Additionally, Frade et al. [87] also suggest to estimate demand for a bike sharing system in the Portuguese city Coimbra based on the experience of similar services in other countries. Specifically, they propose a model considering the purpose of the trip, the associated distance and the slope between origin and destination. However, not all parameters in the proposed model can be derived from experience. Some parameters still need to be estimated by surveys or other benchmarking methods.

In [91] a stochastic model for planning a bike sharing system is proposed. Similar to [89], a survey is used for deriving customer demands. The survey included questions about the user's trips, preferences for a bike sharing service, maximum walking distances, as well as demographic questions related to sex, income, and age. In [92, 93] a model for a one-way car sharing system in Lisbon is proposed. User demands are estimated from mobility surveys. Only trips with a certain Euclidean distance, duration, and at a certain time interval are considered. In [94] a model for a one-way electric car sharing system is proposed for supporting operational decisions of already existing service provider in Nice, France. The demand data has been provided by the service operator. However, such an approach is only possible if a VSS exists already in the targeted area and the service provider is willing to share their data.

The problem of optimizing the distribution of service points is not only relevant for VSS but also for EVs in general, as they require stations at which their batteries can either be recharged or exchanged. For these problems different ways for determining the potential customer demands can be found. Chen et al. [95] substitute charging demand with parking demand in order to identify good locations for public charging stations. The parking demand is derived from parking information of a travel survey. In [96], a maximal covering model [65] for identifying charging stations is proposed. The demands are estimated using regression analysis based on surveys on the number of cars per household, the average travel distance of cars, the estimated range of an EV etc. In [12], the charging demand of a location is modeled as the expected duration of charging all drivers that need to charge their EV at this location. The number of drivers in a location is derived from a mobility survey as part of a case study of the city of Coimbra, Portugal. In [97] charging stations for an on-demand bus system are located using taxi probe data of Tokyo. On-demand bus systems are an SPDP in which users can request vehicles to pick them up and drop them off at designated locations. On-demand bus systems can also be combined with traditional bus services, traveling on predetermined tours. For example, in [98] a bus system with mandatory and optional stops is proposed. Whereas the mandatory stops are visited by each bus, the optional stops are only visited when requested by a customer.

Ciari et al. [99] recognize the difficulties in making demand predictions for new transport options based on estimated data and therefore propose to use an activity-based microsim-

ulation technique for the modeling of car sharing demand. The simulation is done with help of the travel demand simulator MATSim [100].

There also exist some works that take user preferences into account, e.g., in [93] a car sharing system is designed based on different assumptions on the behavior of users. The authors come to the conclusion that providing real-time information to customers can greatly improve the service level of a system if users are willing to visit a different station if their preferred one does not have a vehicle available.

In conclusion, there exists a large amount of different ways for estimating customer demands to be used by the actual optimization algorithm. There does not seem to be a consensus on which method is superior. Additionally, in many cases the necessary input data is not easy or cheap to obtain. Therefore, a cooperative approach offers an attractive alternative demand acquisition method, as it does not require any previous data, is cheap to implement and can easily be transferred to other application scenarios and locations.

4.2.2 Interactive Optimization

Interactive Optimization is a well researched area and many different approaches are proposed in the literature. Generally speaking, interactive algorithms give users the possibility to interact with the algorithm to guide the algorithm towards a satisfactory solution. Meignan et al. [101] point out several obstacles than can be handled more easily with a human in the loop: Users can consider more complex aspects which are possibly difficult to express or calculate in an optimization problem. Moreover, including users in the optimization process results in more accepted solutions than completely automated processes. For a survey on interactive optimization algorithms see [102] and [101].

Note that in literature interactive algorithms are usually designed for a single interactor only. For example, in [103] an interactive genetic algorithm for designing dresses is proposed. Instead of explicitly defining a fitness function, the fitness of a solution is decided by a user. In [104] a preference-based evolutionary algorithm for multi-objective optimization is proposed. The idea is that in each iteration a decision maker is asked to specify a reference point w.r.t. to their desired values of the objective terms. This information is then used by an evolutionary algorithm to generate solution focused in an area around the specified reference point.

However, while scarce, there also exists some literature concerning the interaction with multiple users. For example, Cheng et al. [105] propose an interactive approach for a many-objective optimization problem solved via an evolutionary algorithm that can accept multiple reference vectors as input. The reference vectors are used to partition the search space into subspaces. Note however, that these reference vectors cannot be modified by a user during the algorithm. In [101], Meignan et al. also briefly discuss crowd-solving problems, i.e., complex optimization problems that are solved via crowd-sourcing. One of the most popular among these approaches is Foldit [106] in which users have to solve puzzles in order to find special protein structures.

A crucial part of interactive algorithms is the way in which users are able to get feedback and interact with the algorithm. While in this work we do not touch upon this subject, we refer to [107] in which nine recommendations are suggested for designing interactive optimization tools.

Finally, our approach also exhibits similarities to so-called interactive machine learning approaches [108, 109]. Such methods are typically used when the number of training samples is not sufficient to properly train a machine learning model. To compensate this problem, a human is used as a guide to reduce the search space during the learning phase [110]. One way to reduce the search space is to reduce the number of features considered during the learning phase [111].

4.2.3 Surrogate Assisted Optimization

A major disadvantage of interactive algorithms is that their performance strongly depends on the quality of the feedback given by the interactors. Continuous user interactions will eventually result in user exhaustion [79], negatively influencing the reliability of the obtained feedback. Therefore, user interactions should not only be considered time consuming but users also need to be treated as a scarce resource – the interaction should be kept to a required minimum. A common way to overcome this problem is to combine interactive optimization algorithms with a surrogate-based approach [112, 113]. Surrogate models are typically used as a proxy of functions which are either unknown or extremely time consuming to compute. Classic candidates for such surrogates are machine learning models. In [114] a survey of popular surrogate functions is provided, ranging from polynomial regression [115] to more sophisticated techniques such as neural networks [116] and support vector regression [117]. For continuous optimization problems Gaussian process models, such as Kriging, are widely used as they also give information about the approximation error which can be used to improve the surrogate function, see [114].

Moreover, we also make use of matrix factorization [18] for deriving a surrogate function. Matrix factorization is a collaborative filtering technique which is frequently used in recommender systems [52]. The idea of collaborative filtering is to make recommendations for users based on the preferences of similar users. Matrix factorization is based on singular value decomposition which decomposes a matrix into two smaller matrices. Unknown values can then be estimated my multiplying the corresponding rows and columns of the decomposed matrices [52]. The two most popular techniques for decomposing a matrix with missing values are SGD [56] and Alternating Least Squares (ALS) [57]. ALS is usually only preferred over SGD for parallelization [18].

4.3 Cooperative Service Point Distribution Problems

In this section we formally define the class of problems that we aim to solve in a cooperative way. We refer to such problems as Cooperative Service Point Distribution

Problems (CSPDPs). Similar to classical SPDPs, CSPDPs are associated with a set of locations $V = \{1, \dots, n\}$ at which service points may be set up, as well as a set of potential users $U = \{1, \dots, m\}$ that are going to be served by the service points. Besides V and U, CSPDPs may have further contextual properties $\mathbf{Q}$, such as costs for setting up stations or information about users.

In contrast to SPDPs, CSPDPs have no a priori information of usage preferences of users towards specific service points. User preferences for a specific service point not only depend on the concrete location of said service point but also on which other service points are opened at the moment. For example, when considering vehicle sharing applications, a well placed service point near the destination of a user's trip might be worthless if there are no suitable service points at the origin of the trip. In the same way, if more than one suitable service point exists at the origin, one service point might be preferred over the other and hence be higher valued by the user. Additionally, for some users it might also be acceptable to use public transport for an additional last leg. Consequently, even service points not in proximity to a user's origin or destination might also be relevant to a user. Thus, for a set of available service point locations $S \subseteq V$, henceforth referred to as *location scenario*, demand information for a specific service point location $v \in S$ can only be "exactly" determined by directly asking the users. For this purpose, for CSPDPs user interaction is defined as a function

$$w \colon U \times V \times 2^V \times \mathbf{Q} \to \mathbb{R} \tag{4.1}$$

which returns demand information of a user $u \in U$ for a location v in a location scenario S w.r.t. to some other problem specific properties $\mathbf{q} \in \mathbf{Q}$. Depending on the concrete problem, demand information can be defined in multiple ways. Specifically, users may be asked to provide a potential usage frequency, ranking, or rating for a specified location. Moreover, it might also be advantageous to ask users to also consider other circumstances $\mathbf{q}$ while evaluating service point locations. For example, one might be interested in how relevant a service point location is to a user for traveling to a specific destination, e.g., the user's place of work. Figure 4.1 shows an example from the perspective of a user of how w might be evaluated.

As previously mentioned, the quality of the feedback obtained from users might substantially deteriorate when asking too much information from the users. However, for CSPDPs we do not impose any restrictions on interacting with users. Instead, it is assumed that user feedback is always accurate and we evaluate the performance of COA by considering how much user interaction is necessary for obtaining a solution.

A solution to a CSPDP is given by a subset $X \subseteq V$ and its quality is determined by some objective function $f(X)$. Note that in contrast to a solution, a location scenario is simply a set of service point locations and does not necessarily have to be a feasible solution. However, every solution is also a location scenario. Therefore, with the above considerations in mind, a CSPDP is formally defined as a quintuple $\Pi(V, U, \mathbf{Q}, w, f)$.

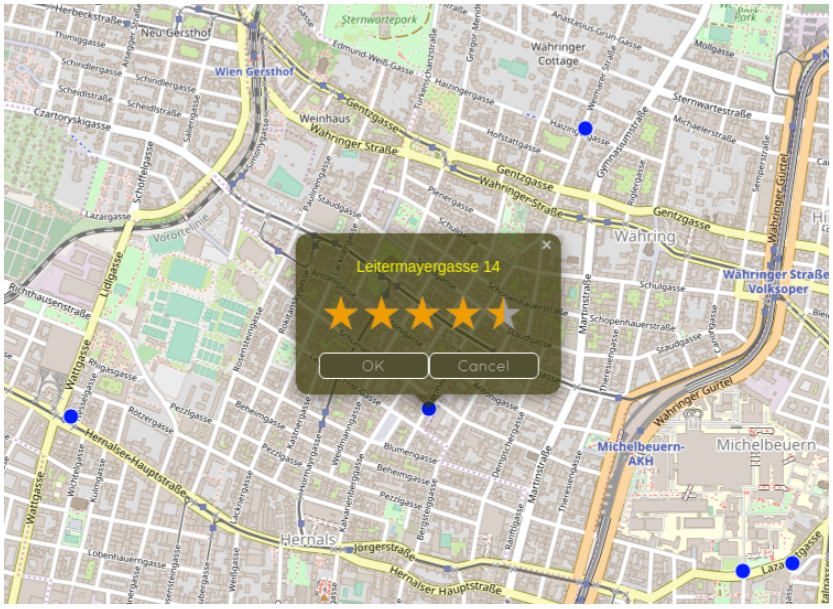


Figure 4.1: Exemplary evaluation of a location scenario by a user: A location scenario in form of a map with highlighted locations is presented and the user can then evaluate the highlighted locations in the form of ratings.

4.4 The Cooperative Optimization Framework

In this section we present the general framework of our Cooperative optimization Approach (COA) for solving CSPDPs. Figure 4.2 shows the basic methodology: In each iteration the algorithm first generates location scenarios for a subset of users to evaluate. Based on the users' feedback to these scenarios, a surrogate objective function is continuously updated over the iterations. The CSPDP instance with the current surrogate objective function is then solved, yielding a solution. In the next iteration, this solution is a basis for deriving further meaningful location scenarios to be presented to users again. The surrogate objective function thus learns to represent the users' needs better and better and the solutions obtained from the optimization will become more precise over time.

From a technical point-of-view, the COA framework consists of a Feedback Component (FC), an Evaluation Component (EC), an Optimization Component (OC), and a Solution Management Component (SMC). Figure 4.3 provides a visualization of the COA process and shows how the framework components interact with each other.

At the start of each iteration, the FC is responsible for collecting information from the user, i.e., users can interact with the framework at this stage of the algorithm. User information is collected by generating individual location scenarios $S \subseteq V$ for each user which are presented to the user in order to obtain his/her feedback. A user $u \in U$ then has to provide feedback $w(u, v, S, \mathbf{q})$ for locations v of location scenarios S presented to them w.r.t. some problem specific context $\mathbf{q}$. Additionally, if necessary, in the first iteration

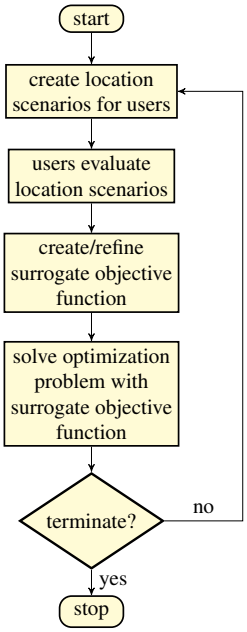


Figure 4.2: Basic methodology of the COA framework.

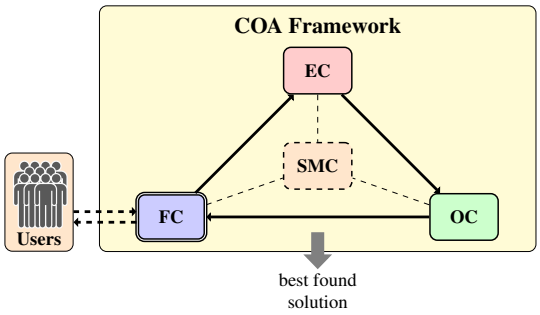


Figure 4.3: Components of COA and their interaction.

the FC also provides the possibility for users to register to provide more information about themselves. For an example of how this can be utilized, see Section 4.6.

The feedback obtained from the users is processed in the EC. As previously mentioned, user interactions should be kept to a required minimum. Hence, the EC maintains and continuously updates a *surrogate suitability function* $\tilde{w}_{\boldsymbol{\Theta}}$ approximating the evaluation function w. Using this surrogate suitability function a surrogate objective function $\tilde{f}$ is derived, making it possible to approximately evaluate solutions without user interaction. The function $\tilde{w}_{\boldsymbol{\Theta}}$ is realized by a machine learning model with parameters (weights) $\boldsymbol{\Theta}$. Based on this surrogate function, the EC also provides a surrogate problem $\Pi(V, U, \mathbf{Q}, \tilde{w}_{\boldsymbol{\Theta}}, \tilde{f})$, i.e., an approximation of the original problem $\Pi(V, U, \mathbf{Q}, w, f)$, which can be solved without user interaction. Therefore the surrogate problem can be evalu-

ated faster than the original problem, as there are no more waiting times for users to provide feedback. This fast approximate evaluation is in particular important during the optimization process at which a multitude of intermediate candidate solutions are generated. The respective learning mechanism of the surrogate objective function is also part of the EC.

A call of the OC is supposed to determine an optimal or close-to-optimal solution to the EC's current surrogate problem $\Pi(V, U, \mathbf{Q}, \tilde{w}_{\mathbf{\Theta}}, \tilde{f})$. Note that the surrogate problem $\tilde{f}$ never changes during a call of the OC, only in each major iteration of the framework after having obtained new user feedback. With the exception of the first call, the optimization procedure of the OC is warm-started with the current best solution $\tilde{X}^*$ as initial solution to possibly speed up the optimization process.

Finally, the SMC efficiently stores and manages information on all candidate solutions that are relevant for more than one of the above components and in particular also the location scenario evaluations provided by the users so far as well as the solutions χ_{OC} returned by the OC.

The whole process is repeated until some termination criterion is reached, e.g., the discrepancy of user feedback and the results of the EC is small enough or a maximum number of user interactions has been reached. In the end, COA returns a solution with the highest surrogate objective value of all of the so far generated solutions.

Note that there are multiple ways in which the components of COA can be realized. Depending on the concrete underlying CSPDP and its specific properties different approaches within the components may be necessary. Therefore, in the following we introduce two concrete CSPDPs as well as two concrete COA implementations for solving these problems.

4.5 The Independent Service Point Distribution Problem

First, we consider the Independent Service Point Distribution Problem (ISPDP). Characteristic properties of the ISPDP are that there exists no additional information about the users. Hence, as basis for the surrogate function an ensemble of machine learning models is used where each model is responsible for learning the preferences of one user each.

4.5.1 Problem Formulation

In ISPDP we are given a set of locations $V = \{1, \ldots, n\}$ at which service points may be built and a set of potential users $U = \{1 \ldots, m\}$. The fixed costs for setting up a service point at location $v \in V$ are $c_v \geq 0$ and this service point's maintenance over a defined time period is supposed to induce variable costs $z_v \geq 0$. The total construction costs must not exceed a maximum budget $B > 0$. Erected service stations may satisfy customer demand, and for each unit of satisfied customer demand a prize $q > 0$ is earned. A solution to the ISPDP is given by a set $X \subseteq V$ containing all locations where service

points are to be set up. The evaluation function is defined as $w(u, v, X, \{\})$, in short $w(u, v, X)$, which specifies the demand of user $u \in U$ fulfilled at location $v \in V$ in solution/scenario X. Note that the degree of demand fulfilled not only depends on the location v but also on other opened stations in X.

Recall that the evaluation function can only be solved “exactly" by directly asking the corresponding user. Clearly, the number of candidate solutions that are evaluated in this interactive way are a major concern. We cannot confront each user with hundreds or thousands of evaluation requests. Instead, we carefully have to select the solutions to be evaluated by each user in an individual fashion, avoiding redundancies as far as possible.

Naturally, the demand fulfilled at any location must always be non-negative and can only be positive when a service point is set up there, i.e.,

$$w(u, v, X) \geq 0, \quad v \notin X \rightarrow w(u, v, X) = 0 \qquad u \in U,\ v \in V. \tag{4.2}$$

A solution X is feasible if its total fixed costs do not exceed the maximum budget B, i.e.,

$$c(X) = \sum_{v \in X} c_v \leq B. \tag{4.3}$$

The objective is to find a feasible solution that maximizes the prizes earned for satisfied customer demands reduced by the variable costs for maintaining the service points

$$f(X) = q \cdot \sum_{u \in U} \sum_{v \in X} w(u, v, X) - \sum_{v \in X} z_v. \tag{4.4}$$

4.5.2 Cooperative Optimization Approach

This section gives a detailed description of the components of COA for solving the ISPDP, henceforth also referred to as COA[ISPDP]. Algorithm 4.1 describes the general interaction of the components. In the following we describe each component's functionality in more detail.

Solution Management Component

Next to storing all considered candidate solutions, the SMC also maintains for each candidate solution the users for which the exact fulfilled demand is already known, and for each user $u \in U$ a set of so far identified *relevant locations* V_u, which includes any location for which the user has indicated positive demand in at least one solution. Note that the complete set of relevant locations in general is unknown. However, it is the task of the FC to choose the solutions presented to the users in such a way that as many relevant locations as possible are identified.

Another important task of the SMC is to derive a user's demand for scenarios where this can be efficiently achieved through logical implications from previous scenario evaluations. For example, any scenario that is a superset of a scenario with some maximum fulfillable demand will also achieve this maximum.

Algorithm 4.1: Basic Framework

Input : an instance of the ISPDP
Output: best solution $\tilde{X}^* \subseteq V$ found

$\chi_{\text{OC}} = \{V\}$ // initial solution;
while *no termination criterion satisfied* **do**
  Feedback Component:
    for $u \in U$ **do**
      determine set of scenarios S_u to be evaluated by u from χ_{OC} and further data in the SMC;
      let user u evaluate S_u, update the SMC with evaluated solutions from S_u;
    end for
  Evaluation Component:
    train surrogate objective function $\tilde{f}(X)$ with data from the SMC;
    re-evaluate all solutions stored in the SMC with the new surrogate objective function;
  Optimization Component:
    adopt so far best solutions from the SMC as initial solutions;
    $\chi_{\text{OC}} \leftarrow$ perform optimization using the EC's surrogate objective function $\tilde{f}(X)$;
    when possible, calculate exact $f(X)$ for $X \in \chi_{\text{OC}}$;
    store the solution(s) from χ_{OC} in the SMC;
end while
return *overall best found solution* X^* *w.r.t.* $\tilde{f}$;

Feedback Component

In each major iteration of Algorithm 4.1, the FC generates for each user $u \in U$ an individual set of location scenarios to evaluate. It is assumed here that any user u evaluates each solution in a completely rational way so that the total fulfilled demand is maximal.

It appears natural that a scenario presented to a user should be similar to the best solutions identified so far by the OC or, otherwise, provide substantial information gain on locations that are potentially interesting for the user. Next to finding new relevant locations for a user, it is also necessary to gain information on the relationship between relevant locations. Note that location scenarios are different from solutions in the sense that location scenarios do not need to be feasible.

We apply the following combination of strategies for compiling a set of at most κ solutions presented to each user $u \in U$, where κ is a strategy parameter and is set to 15 in the experiments performed for this chapter:[1]

[1] All parameter values stated in the text have been tuned in comprehensive preliminary tests.

- Best solution strategy: Select the γ_1 best feasible solutions w.r.t. to f and the γ_2 best feasible solutions w.r.t. the surrogate function $\tilde{f}$ for which no exact total fulfilled demand is known yet for user u. Hereby, γ_1 and γ_2 are strategy parameters, which are both set to 2 in the experiments performed for this chapter. This strategy clearly focuses on getting exact evaluations for the currently most promising solutions.
- Relevant locations strategy: This strategy focuses purely on finding new relevant locations for a user u, which might lead to good alternative solutions. For this purpose a scenario in which the locations in $V \setminus V_u$ are selected is generated.
- Best solution mutation strategy: This strategy is a combination of the previous strategies and tries to gain information on the relationship between locations by replacing a subset of $S \cap V_u$ of a scenario S obtained from the best solution strategy for a user u with a set of locations for which it is so far unclear if they are relevant: A new scenario is constructed from a copy of an existing scenario S by removing a location $v \in S \cap V_u$ from S with a certain probability $\boldsymbol{\xi}$ for each v, where $\boldsymbol{\xi}$ is a strategy parameter which is set to 0.5 in the experiments throughout this chapter. Afterwards, we add a location v to S for n' uniformly at random chosen locations $v \in V \setminus S$ with n' being chosen uniformly at random from $\{0, \ldots, |V| - |S|\}$.

Note that the number of solutions generated with the best solution strategy as well as the relevant locations strategy is limited and yield at most five scenarios in total. Therefore, the best solution mutation strategy is applied until no more new scenarios can be generated or the threshold κ is reached.

Evaluation Component

The EC provides the means for evaluating solutions, in particular also temporary solutions generated within the OC. Within the OC the objective value of a solution is estimated by a surrogate objective function $\tilde{f}(X)$, which is defined in accordance to $f(X)$ but makes use of *estimated fulfilled demands*

$$\tilde{w}_{\boldsymbol{\Theta}}(u, v, X) = \begin{cases} 0 & \text{if } v \notin V_u \vee v \notin X \\ \max(0, g_{\boldsymbol{\Theta},u,v}(X)) & \text{else} \end{cases} \tag{4.5}$$

for each user $u \in U$ and each location $v \in V$, where $g_{\boldsymbol{\Theta},u,v}(X)$ represents a machine learning model trained by all solutions so far evaluated by user u. Note that our definition of $\tilde{w}_{\boldsymbol{\Theta}}(u, v, X)$ ensures that Conditions (4.2) are always fulfilled and gives function $g_{\boldsymbol{\Theta},u,v}(X)$ more freedom in the sense that it may return negative values, which are mapped to zero, and arbitrary values in case of $v \notin X$. Furthermore, for any location v for which user u has so far never indicated any positive fulfilled demand in any solution, i.e., for any so far not relevant location $v \in V \setminus V_u$, $g_{\boldsymbol{\Theta},u,v}(X) = 0$ is assumed and no machine learning model needs to be maintained.

Similarly to [17], we use an adaptive surrogate function in the sense that the machine learning model for each $g_{\boldsymbol{\Theta},u,v}(X)$ is initially simple and is upgraded to a higher complexity

model during the course of the algorithm when the error of the model – measured in terms of the usual MSE of $\tilde{w}(u, v, X)$ – exceeds a certain threshold τ. In this way we stay as efficient as possible from a computational perspective and substantially reduce problems with overfitting. Our initial choice for $g_{\boldsymbol{\Theta},u,v}(X)$ is the linear regression model

$$g^{\mathrm{LM}}_{u,v}(X) = \omega_{u,v} + \sum_{v' \in (V_u \cap X) \setminus \{v\}} \omega_{u,v,v'}. \tag{4.6}$$

Ridge regression with a penalization factor of one is used for determining the weights $\omega_{u,v}$ and $\omega_{u,v,v'}$. This model is sufficient for covering simple scenarios where users have independent demands that can be fulfilled at specific locations. Furthermore, it can even accurately represent the case where for a user one demand can be fulfilled at a specific primary location or, with a possibly reduced amount, at one alternative location if no service station is set up at the primary location. More complex dependencies, including in particular more than one alternative location, are, however, beyond the capability of the linear regression model.

In this case, which is detected by a remaining MSE of $\tilde{w}(u, v, X)$ larger than a threshold $\tau = 0.075$, we turn to a neural network, starting with a single layer perceptron with a leaky Rectified Linear Unit (ReLU) activation function [118]. This simple neural network realizes the function

$$g^{\mathrm{NN}}_{\boldsymbol{\Theta},u,v}(X) = \phi\left(g^{\mathrm{LM}}_{\boldsymbol{\Theta},u,v}(X)\right) \text{ with } \phi(\mathcal{S}) = \begin{cases} \mathcal{S} & \text{if } \mathcal{S} \geq 0 \\ \varepsilon \cdot \mathcal{S} & \text{else.} \end{cases} \tag{4.7}$$

The leaky ReLU activation function ϕ serves as an extension of the linear regression model in the sense that this perceptron takes actively into account that satisfied demands cannot be negative. Due to this non-linearity, it can accurately represent scenarios in which for a user a demand can be fulfilled at an arbitrary number of ordered alternative locations, where a service station at one of these locations will only fulfill a certain amount of the demand when no station is set up at any of the preceding alternative locations in the order. We use here the leaky ReLU function with parameter $\varepsilon = 0.01$ which returns small negative values in case the sum $\mathcal{S}$ is negative.

While the above perceptron is already more powerful, it is still limited when a user has more than one demand that can be fulfilled partly at the same locations, or more generally, when the different demands are related in some way, i.e., when a location can cover multiple mobility demands of a user. Again, we detect the insufficiency of the perceptron by an MSE that exceeds τ and turn in this case to a more complex feedforward neural network with one hidden layer that contains initially two hidden neurons. These neurons again make use of the leaky ReLU activation function, while the single output layer neuron corresponds to a simple summation of the inputs. Initially, we use two hidden neurons and increase this number until, after training, either the MSE does not exceed τ anymore or a maximum of $\lambda = 6$ hidden neurons is reached.

Note that the solutions used for training the models are not required to be feasible, since user evaluations do not consider the budget at all.

Optimization Component

A proper choice of optimization algorithm and corresponding configuration is vital for COA to work. The optimization algorithm should not only provide close-to-optimal solutions but should also not need too many candidate solution evaluations as they are rather time-expensive and the OC needs to be repeatedly performed. For this purpose we consider two different metaheuristics – a VNS and a PBIG – as OC for COA[ISPDP].

The VNS follows the classical scheme from [47]. An initial solution is generated via the randomized construction heuristic that considers all locations in random order and sets up a station at a location as long as the budget is not exceeded. Our local search uses an exchange & complete neighborhood following a first improvement strategy. In the first step of the neighborhood, a location in the solution is replaced by an unused location, i.e., a location at which no service point exists. As this exchange might decrease the current budget of the solution, we afterwards add further unused locations in a random order to the solution as long as the budget allows it. The k-th shaking removes k randomly selected locations from the solution and then iteratively adds unused locations in a uniform random order until no more locations can be added. Note that the VNS considers only feasible solutions.

For the general principles of the PBIG we refer to Chapter 2. An initial population of solutions is generated via the same randomized construction used for generating the initial solution of the VNS. Then, in each major iteration a new solution is derived from each solution in the current population by applying a destroy & recreate operation. The best solutions from the joint set of original and newly derived solutions are accepted as new population for the next iteration. Our destroy & recreate operation first removes a number of selected locations from the solution and then again iteratively adds unused locations in a uniform random order, such that the solution stays feasible and no more stations can be added. We reuse the exchange & complete neighborhood as well as the shakings operators of the VNS as destroy & recreate operations of the PBIG. Hence, the PBIG also considers only feasible solutions. Note that the PBIG returns its final population while the VNS only returns a single solution as result of the optimization.

4.5.3 Test Instance Generation

COA[ISPDP] is tested in a proof-of-concept manner on artificial benchmark scenarios using an idealized simulation of all user interaction.

The primary parameters for our benchmark scenarios are the number of potential locations for service stations n and the number of users m, and we consider here the combinations $n = 50, 60, \ldots, 100$ with $m = 50$ and $n = 50$ with $m = 50, 60, \ldots, 100$. The n locations correspond to points in the Euclidean plane with coordinates chosen uniformly at random from the grid $\{0, \ldots, L-1\}^2$, where $L = \lceil 10\sqrt{n} \rceil$ is the underlying width and height. The fixed costs c_v as well as the variable costs z_v for setting up a service station at each location $v \in V$ are uniformly chosen at random from $\{50, \ldots, 100\}$. The budget is assumed to be $B = \lceil 7.5 \cdot n \rceil$ so that about 10% of the stations with average costs

can be set up. We assume each of the m users $u \in U$ has ρ_u so-called *use cases*, where ρ_u is chosen randomly according to a shifted Poisson distribution with offset one and expected value three. Each of these use cases $i = 1, \ldots, \rho_u$ is associated with a particular geographical location $r_{u,i} \in \{0, \ldots, L-1\}^2$ and a respective demand $d^*_{u,i}$ that could ideally be fulfilled there. This demand can, for example, be the expected number of usages of a service point in a time period. Here, we choose each $d^*_{u,i}$ uniformly at random from {5,...,50}. In a real scenario, the locations where demand arises will clearly not be uniformly distributed over the whole considered geographic area. There will be more popular regions as well as less popular ones. We want to consider this aspect and therefore first choose n_α *attraction points* A with uniform random coordinates from $\{0, \ldots, L-1\}^2$ and then derive the location for each use case from a uniformly selected attraction point $(a_x, a_y) \in A$ by

$$r_{u,i} = (\lfloor \mathcal{N}(a_x, 20) \rfloor \bmod L, \lfloor \mathcal{N}(a_y, 20) \rfloor \bmod L), \tag{4.8}$$

where $\mathcal{N}(\cdot, \cdot)$ denotes a random value sampled from a normal distribution with the respectively given mean value and standard deviation.

For each use case $i = 1, \ldots, \rho_u$ of each user $u \in U$, demand is always only fulfilled at the closest location $v^{\text{clst}}_{u,i}(X) \in V$ w.r.t. the Euclidean distance where a service station is set up in the current candidate solution X (ties are broken according to the locations' natural order) and when a maximum distance, chosen here as 12, is not exceeded. We further assume an exponential decay of the fulfilled demand in dependence of the distance and round down to the closest integer, obtaining

$$w_i(u, v, X) = \begin{cases} \lfloor d^*_{u,i} \cdot e^{-||r_{u,i} - v^{\text{clst}}_{u,i}(x)||/10} \rfloor & \text{if } v = v^{\text{clst}}_{u,i}(X) \wedge ||r_{u,i} - v|| \leq 12 \\ 0 & \text{else,} \end{cases} \tag{4.9}$$

where $||\cdot||$ denotes the L^2 norm. These fulfilled demands for each use case i are finally summed up in order to obtain the overall fulfilled demands $w(u, v, X) = \sum_{i=1} w_i(u, v, X)$ for each user $u \in U$ and location $v \in V$ under candidate solution X. Finally, the prize earned for each unit of fulfilled demand in our objective function is assumed to be $p = 50$.

For each combination of n and m 30 independent scenarios were created, and they are available at `https://www.ac.tuwien.ac.at/research/problem-instances`. The benchmarks were also specifically designed with the ability in mind to calculate proven optimal solutions to which we will compare the solutions of our framework. Exploiting the complete knowledge of the data and specific structure in a "white-box" manner allows the problem to be expressed as MILP model, which we solved with the MILP-solver Gurobi[2].

4.5.4 Computational Results

The OC was implemented in C++, compiled with GNU G++ 5.5.0, while the remaining components of the framework were realized in Python 3.7. All test runs were executed on an Intel Xeon E5-2640 v4 with 2.40GHz machine.

[2] `http://www.gurobi.com/`

Initially, we determined the parameter configurations of standalone variants of the VNS and the PBIG, in which it is naively assumed that users can evaluate all intermediate solutions, i.e., using w directly as surrogate function instead of $\tilde{w}_{\boldsymbol{\Theta}}$. The parameters have been determined with irace [119] on a separate set of benchmark instances and have been tuned with the goal to minimize the number of iterations it takes the metaheuristics to generate near optimal solutions, i.e., solutions with an optimality gap of 0.5%. For the VNS we obtained to best use shaking neighborhoods $k \in \{1, 2\}$, for PBIG $k \in \{1, \ldots, 10\}$ with a population size of 100. The termination criteria of the metaheuristics have been chosen according to the number of iterations that were necessary on average to find the close-to-optimal solutions, which was 60 iterations without improvement for the VNS, and 300 iterations (or three generations) without improvement for PBIG. Table 4.1 shows a comparison of the standalone variants of the metaheuristics using above parameter configurations. The table shows for each instance group with n locations and m users the average optimality gap (gap[%]) and their corresponding standard deviation ($\sigma_{\%\text{-gap}}$) of the metaheuristics. Moreover, the table also shows the iteration in which the best solution has been found on average ($n_{\text{it}}^{\text{best}}$) and the median of the total computation times (t[s]).

Table 4.1: VNS vs. PBIG.

		VNS				PBIG			
n	m	gap[%]	$\sigma_{\%\text{-gap}}$	$n_{\text{it}}^{\text{best}}$	t[s]	gap[%]	$\sigma_{\%\text{-gap}}$	$n_{\text{it}}^{\text{best}}$	t[s]
50	50	**0.26**	0.48	**29**	**3**	0.28	1.22	696	8
50	60	**0.20**	0.61	**35**	**4**	0.31	1.36	656	9
50	70	**0.00**	0.02	**32**	**4**	0.10	0.42	634	8
50	80	0.31	0.74	**31**	**4**	**0.09**	0.28	631	9
50	90	**0.14**	0.42	**32**	**4**	0.37	1.01	648	11
50	100	0.37	1.03	**33**	**4**	**0.01**	0.07	706	15
60	50	0.25	0.64	**43**	**4**	**0.07**	0.34	862	9
70	50	0.34	0.62	**54**	**5**	**0.28**	0.57	1113	17
80	50	0.43	0.54	**56**	**6**	**0.24**	0.51	1389	23
90	50	0.30	0.42	**64**	**7**	**0.19**	0.60	1628	30
100	50	**0.37**	0.47	**65**	**9**	0.42	0.82	1754	39

Table 4.2: COA[VNS] vs. COA[PBIG].

		COA[VNS]				COA[PBIG]			
n	m	gap[%]	$\sigma_{\%\text{-gap}}$	n_{it}	t[s]	gap[%]	$\sigma_{\%\text{-gap}}$	n_{it}	t[s]
50	50	0.28	0.70	11	**2259**	**0.20**	0.45	**10**	2662
50	60	0.73	1.27	**9**	**2343**	**0.06**	0.18	**9**	2643
50	70	0.14	0.37	**10**	**3107**	**0.09**	0.44	**10**	3764
50	80	0.19	0.36	**10**	**3588**	**0.04**	0.15	**10**	3919
50	90	0.42	0.68	10	**3596**	**0.19**	0.71	**9**	4516
50	100	0.12	0.26	**10**	**4391**	**0.02**	0.08	**10**	4995
60	50	0.48	0.72	11	**2460**	**0.05**	0.11	**10**	2944
70	50	0.46	0.66	11	**2533**	**0.13**	0.43	**10**	3658
80	50	0.22	0.58	12	**2864**	**0.11**	0.28	**11**	4810
90	50	0.37	0.52	**11**	**2910**	**0.26**	0.65	**11**	5435
100	50	0.49	1.02	12	**3460**	**0.07**	0.15	**11**	7197

PBIG produces slightly better optimality gaps but also needs significantly more time than the VNS. Moreover, it takes PBIG much more iterations to find the best solution as opposed to the VNS.

Next, we tested COA[ISPDP] in conjunction with the VNS (denoted as COA[VNS]) and the PBIG (denoted as COA[PBIG]), respectively, as OC. Further tests have shown that COA[PBIG] yields slightly better results when using the so far best found solutions stored in the SMC as initial population. In case there are not enough solutions available, the remaining solutions are generated by the randomized construction heuristic. The other parameters remain unchanged. Independent of the implementation of the OC, COA[ISPDP] terminated after five iterations without improvement or after two hours. The comparison can be seen in Table 4.2. The table shows again the average optimality gaps (gap[%]) and their corresponding standard deviations ($\sigma_{\%\text{-gap}}$). Moreover, the table

also shows the average number of COA iterations (n_{it}) and the median of the total computation times (t[s]).

While the optimality gaps in Table 4.1 are somewhat comparable between the VNS and the PBIG, COA[PBIG] clearly outperforms COA[VNS] w.r.t. the optimality gaps. This difference can be explained by the number of solutions returned by the VNS and the PBIG. While the OC of COA[VNS] only returns one solution in every COA iteration, the OC of COA[PBIG] returns 100 solutions in every iteration. A higher number of solutions in the SMC results in more diversified solutions not only in the FC but also in the EC. Hence, the accuracy of the surrogate function increases w.r.t. larger areas of the search space, whereas for less diversified training data the accuracy of the surrogate function usually only increases in a small part of the search space. Moreover, the high number of shakings in the PBIG additionally increases the diversity of the solutions returned by the OC. Note however that COA[PBIG] does not scale so well w.r.t. computation times, especially for an increasing number of locations. The reason for this is the generous termination criterion of the PBIG in comparison to the VNS, since it takes the PBIG much more time to find a near optimal solution.

Figure 4.4 shows the computation times of the individual components of the COA framework. The total computation time is primarily split between the EC and the OC while the FC has barely any impact. Moreover, the figure also shows that the computation time of the EC mainly depends on the number of users, while the computation time of the OC primarily scales with the number of service point locations. Finally, Figure 4.4 also shows large differences in computation times between COA[VNS] and COA[PBIG].

Finally, Figure 4.5 shows the distribution of the model sizes of the surrogate function's underlying machine learning models at the final iteration of COA. A model size of zero refers to linear regression models, size one to perceptrons, and larger sizes to neural networks with the respective number of neurons in the hidden layer. The distribution shown in Figure 4.5 is typical for all instances tested. It shows that the majority of machine learning models is made up of linear regression models and perceptrons. Larger size neural networks are rarely needed. However, the figure also shows a small peak at the largest neural network with six neurons in its hidden layer. This peak is caused by unpopular service point locations resulting in training data in which most customer demands are zero. The neural networks often fail to properly learn such data, however, on the other hand, as these locations are the least popular service point locations, they usually have no large impact on the final solution.

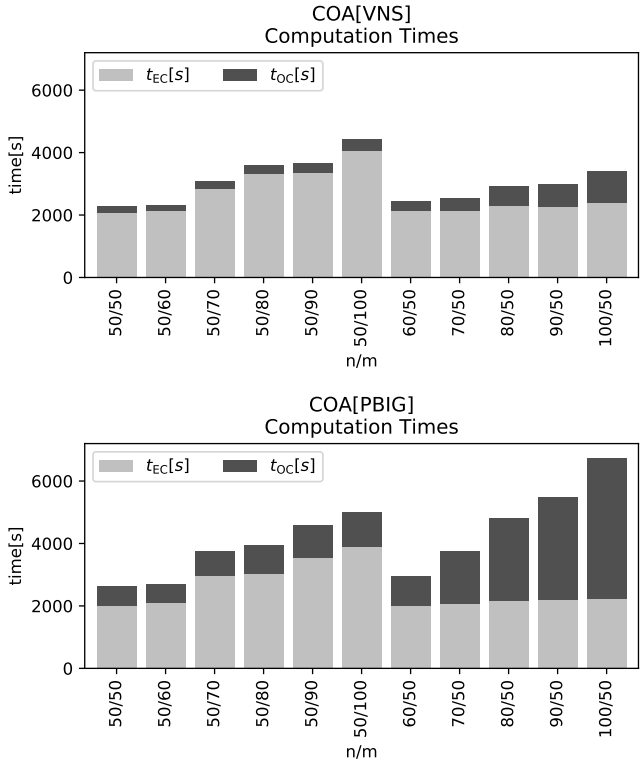


Figure 4.4: Median computation times of the COA components for each instance set

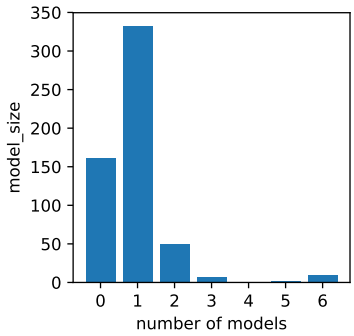


Figure 4.5: Model Distribution of an exemplary run with $n = 100$ and $m = 50$

4.6 The Generalized Service Point Distribution Problem

The Generalized Service Point Distribution Problem extends the ISPDP by considering more information about the users. COA can exploit this information by identifying similar preferences for service points between the users and deriving a surrogate function based on these preferences. Additionally, to make it easier for the user to give feedback

to service point locations we also introduce the concept of *use cases*. Use cases in our context are just labels and are not directly associated with specific geographic locations. This separation is intentionally done in order to keep flexibility: Some use cases like shopping or the visit of a fitness center may possibly be realized at different places, and as already mentioned occasionally a service station farther away from a specific target location may also be convenient if some other mode of transportation is used as additional leg. Users have to specify approximate maximal usage frequencies of the service points in the system w.r.t. their use cases in advance. Afterwards, when evaluating service points users are now asked to rate the suitability of the locations w.r.t. to a specific use case. The idea is, that the more suitable a location is, the more often a user is going to use this location when a service point is built there. Additionally, in some application scenarios, such as vehicle sharing, more than one service point is required for satisfying the demand of a user's use case. As a user may have different preferences at which locations these service points should be placed, *use cases* are further divided into so called *service point requirements*.

4.6.1 Problem Formulation

In the Generalized Service Point Distribution Problem (GSPDP) we are given a set of locations $V = \{1, \ldots, n\}$ at which service points may be set up and a set of potential users $U = \{1, \ldots, m\}$. The fixed costs for establishing a service point at location $v \in V$ are $z_v^{\text{fix}} \geq 0$, and this service point's maintenance over a defined time period is supposed to induce variable costs $z_v^{\text{var}} \geq 0$. The total setup costs of all stations must not exceed a maximum budget $B > 0$. We make the simplifying assumption that opened service stations are able to satisfy an arbitrary amount of customer demand. For each unit of satisfied customer demand a prize $q > 0$ is earned.

A solution to the GSPDP is a subset $X \subseteq V$ of all locations where service points are to be set up. A solution X is feasible if its total fixed costs do not exceed the maximum budget B, i.e.,

$$z^{\text{fix}}(X) = \sum_{v \in X} z_v^{\text{fix}} \leq B. \tag{4.10}$$

Given the set of users U, we assume that each user $u \in U$ has a certain set of *use cases* C_u, such as going to work, to a recreational facility, or shopping. Each use case $c \in C_u$ is associated with a demand $D_{u,c} > 0$ expressing how often the use case is expected to be frequented by user u within some defined time period such as a week or a month. The demand of each use case can possibly be satisfied by different service points or subsets of service points to different degrees, depending on the concrete application and the customer's preferences. We assume to have no a priori knowledge about the use cases of the users and suitable service station locations. In an initialization phase each user u just lists his or her expected use cases C_u in the form of arbitrary labels and specifies respective demands $D_{u,c},\ c \in C_u$.

Depending on the actual application and characteristics of a use case, demand may be fulfilled by a single service station, e.g., when charging batteries of an electric vehicle, or a suitable combination of multiple service stations may be needed, such as when renting a vehicle at one place and returning it somewhere else. To model this aspect formally, we associate each use case c of a user u with a set of service point requirement (SPR) $R_{u,c}$. Similar to use cases these SPRs are not directly associated with geographic locations but are an abstract entity like "place within easy reach of home to rent a vehicle" or "place close to a supermarket to return a vehicle" with which a user can express the dependency on multiple service points to fulfill the needs of one use case. Thus, the demand of such a use case can only be satisfied if a service point exists at a suitable location for each of the use case's SPRs. Note that multiple use cases of a user may also share the same SPR(s). For example a use case referring to a trip from home to work and one from home to a supermarket may share the SPRs "place within easy reach of home to rent a vehicle". We denote the set of all different SPRs over all use cases of a user u by $R_u = \bigcup_{c \in C_u} R_{u,c}$. Moreover, let $R = \bigcup_{u \in U} R_u$ be the set of all SPRs over all users. Note that in this notation, different users never share the same SPR, although SPRs of different users can be similar.

The evaluation function of the GSPDP is given by $w(u, v, \{\}, r)$, or $w(r, v)$ as the user u is already implied by his or her SPR r. The function returns a value in [0,1] and indicates the suitability of a service point at location $v \in V$ to satisfy the needs of user $u \in U$ concerning SPR $r \in R_{u,c}$ in use cases $c \in C_u$. A value of $w(r, v) = 1$ represents perfect suitability while a value of zero means that location v is unsuitable; values in between indicate partial suitability. Note that this way the evaluation of a location is independent of other existing service points.

The objective of the GSPDP is to maximize

$$f(X) = q \cdot \sum_{u \in U} \sum_{c \in C_u} D_{u,c} \cdot \min_{r \in R_{u,c}} \left(\max_{v \in X} w(r, v) \right) - \sum_{v \in X} z_v^{\text{var}}, \tag{4.11}$$

In the first term of this objective function, the obtained prize for the expected total satisfied demand is determined by considering for each user u, each use case c, and each SPR r a most suitable location $v \in V$ at which a service point is to be opened ($v \in X$). Over all SPRs of a use case, the minimum of the obtained suitability values is taken so that the full demand is only fulfilled when for each SPR an ideally suited service station is planned, and no demand is fulfilled as soon as one of the SPRs does not have an appropriate service point. The second term of the objective function represents the total maintenance costs for the service stations.

Overall, a crucial assumption we intend to exploit in our approach is that in a larger user base some users typically share use cases and/or their SPRs. Identifying and exploiting these similarities just on the basis of the specified "sampling-based" user interaction is not trivial but may reduce the required interaction per user to obtain a good final solution.

By linearizing the above objective function, the GSPDP can be formulated as a MILP with the following variables. Binary variables x_v indicate whether or not a service point is deployed at location $v \in V$, i.e., the binary vector $\mathbf{x} = (x_v)_{v \in V}$ is the incidence vector of a corresponding solution $X \subset V$. Additional variables $h_{r,v}$ are used to indicate the actually used location $v \in V$ for each SPR $r \in R$. The degree to which a use case $c \in C_u$ of a user $u \in U$ can be satisfied is expressed by continuous variables $y_{u,c} \in [0, 1]$. The GSPDP is then stated as follows.

$$\max \quad q \cdot \sum_{u \in U} \sum_{c \in C_u} D_{u,c}\, y_{u,c} - \sum_{v \in V} z_v^{\text{var}}\, x_v \tag{4.12}$$

$$\sum_{v \in V} z_v^{\text{fix}}\, x_v \leq B \tag{4.13}$$

$$\sum_{v \in V} h_{r,v} \leq 1 \qquad \forall r \in R \tag{4.14}$$

$$\sum_{v \in V} w(r, v) \cdot h_{r,v} \geq y_{u,c} \qquad \forall u \in U,\ c \in C_u,\ r \in R_{u,c} \tag{4.15}$$

$$h_{r,v} \leq x_v \qquad \forall v \in V,\ r \in R \tag{4.16}$$

$$x_v \in \{0, 1\} \qquad \forall v \in V \tag{4.17}$$

$$0 \leq y_{u,c} \leq 1 \qquad \forall u \in U,\ c \in C_u \tag{4.18}$$

$$0 \leq h_{r,v} \leq 1 \qquad \forall r \in R,\ v \in V \tag{4.19}$$

In correspondence to the definition of f, the objective value is calculated in (4.12) as the sum of the prizes earned for fulfilled demand minus the costs for opening service stations. Inequality (4.13) ensures that the budget is not exceeded. Inequalities (4.14) ensure that at most one location is selected for each SPR. As our objective is to maximize the revenue it is ensured that always a suitable service point location with the highest suitability value for each SPR is chosen. Inequalities (4.15) determine the degrees to which the use cases are satisfied, considering that the actually fulfilled demand of a use case is assumed to be proportional to the minimum suitability value of the locations selected for the SPRs of the use case. Last but not least, Inequalities (4.16) ensure that only locations at which service points are to be opened can be used for SPRs and, thus, to satisfy demand of use cases. The size of this model in terms of the number of variables as well as the number of constraints is in $\mathcal{O}(n\,|R| + |C|)$ where C refers to the set of all use cases over all users, i.e. $C = \bigcup_{u \in U} C_u$.

Theorem 2. *The GSPDP is NP-hard.*

Proof. NP-hardness of the GSPDP is proven by providing a reduction from the well known NP-hard Maximal Covering Location Problem (MCLP) [65] in the variant stated by [120]. Given are a set of possible facility locations $\mathcal{J}$, a maximum number p of facilities to be opened, and a set of demand nodes $\mathcal{D}$. Moreover, each demand node $i \in \mathcal{D}$ is associated with a demand $a_i \geq 0$ and a subset of facilities $\mathcal{F}_i \subseteq J$ of which each is able to cover the node's full demand. The goal of the MCLP is to select up to p locations for opening facilities in order to maximize the total demand covered.

Given an instance to the MCLP we construct a corresponding GSPDP instance in which the set of locations V corresponds to the set of facilities $\mathcal{J}$ and the set of users U corresponds to the set of demand nodes $\mathcal{D}$. Moreover, each user $u \in U$ only has a single use case with a single SPR and demand a_i with $u = i$. Building costs z_v^{fix} for a location $v \in V$ are set to one while the maintenance costs z_v^{var} are zero. The budget of the GSPDP instance is set to p, and the prize for a unit of covered demand q is set to one. The suitability value $w(r, v)$ is set to one for $v \in V$ and $r \in R$ if facility i can satisfy the demand of demand node j, i.e., $j \in \mathcal{F}_i$, and zero otherwise.

Let $(\mathbf{x}, \mathbf{y}, \mathbf{h})$ be a feasible solution to this derived GSPDP instance. A corresponding feasible solution to the MCLP is obtained by opening facilities at all locations $j \in \mathcal{J}$ for which $x_v = 1$. Due to the budget constraint (4.13), at most p facilities are opened in the MCLP instance, and thus, there is a bijective mapping of feasible GSPDP solutions to feasible MCLP solutions.

Since each user in the GSPDP instance only has one use case and each use case only consists of one SPR, the sets U, C, and R all contain the same elements. By our definitions, variables $y_{u,c}$ indicating the covered SPRs therefore also indicate the covered demand nodes of the MCLP instance. More generally, we also have a bijective mapping of covered SPRs in the GSPDP instance to covered demand nodes in the MCLP instance. Last but not least, due to our definitions of the suitability values $w(r, v)$, the fixed and variable costs for opened stations, and the prize per unit of fulfilled demand, the objective values of corresponding GSPDP and MCLP solutions also correspond. Since all applied transformations require polynomial time, it follows that the GSPDP is NP-hard. □

4.6.2 The Cooperative Optimization Algorithm

In this section the COA framework for solving GSPDP instances, henceforth referred to as COA[GSPDP] is described. In contrast to the COA[ISPDP], for COA[GSPDP] the FC starts with an initialization phase on its first call by asking each user $u \in U$ to specify the user's use cases C_u, associated SPRs $R_{u,c}$, as well as corresponding demands $D_{u,c}$, $c \in C_u$. Afterwards, the FC is again responsible for collecting information from the user, i.e., users can interact with the framework at this stage of the algorithm.

Figure 4.6 gives a summary of the whole COA procedure for solving GSPDP instances and of the main tasks of each of the components of COA. In the following we describe each component's functionality in more detail.

Solution Management Component

The SMC stores and manages so far considered solutions and evaluations by the users. The solutions obtained from the OC in all major iterations is stored in a set we denote by $\mathcal{X}$. For each solution $X \in \mathcal{X}$ the SMC keeps track of its current surrogate objective value $\tilde{f}(X)$. Hence, the surrogate objective values of the solutions in $\mathcal{X}$ are updated in each major COA iteration whenever the EC updates the surrogate suitability function.

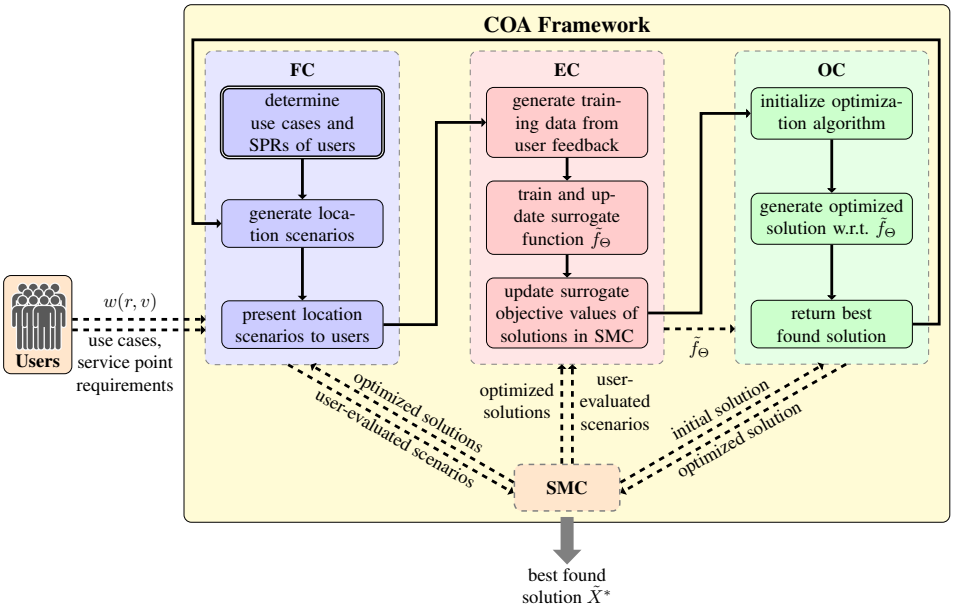


Figure 4.6: Components of COA for GSPDP instances.

The current best solution in $\mathcal{X}$, i.e., the solution with the highest surrogate objective value, is denoted by $\tilde{X}^*$.

All feedback obtained from the users via the presented location scenarios is collected and stored in the SMC in a hash map: Its set of keys, which we denote by K, is the set of pairs (r, v) with $r \in R,\ v \in V$ for which suitability values $w(r, v)$ have been obtained from the users, and the respective values are the $w(r, v)$.

Last but not least, through the FC we are also able to obtain upper bounds on suitability values $w(r, v)$, with $v \in V,\ r \in R$, as will be explained below. These upper bounds are stored in the SMC as $w^{\mathrm{UB}}_{r,v} \in [0, 1]$.

Feedback Component

Let $S \subseteq V$ be a location scenario provided to a user u in respect to one of the user's SPRs $r \in R_u$. It is then assumed that the user returns as evaluation of the scenario S either a best suited location $v_{r,S} \in S$ and the corresponding suitability value $w(r, v_{r,S}) > 0$ or the information that none of the locations of the scenario S is suitable. The latter case implies that $w(r, v) = 0$ for all $v \in S$. In case multiple locations are equally well suited, we assume that the user selects one of them at random. It is assumed here that the suitability of a location w.r.t. an SPR can be specified by the user on a five-valued scale from zero, i.e., completely unsuitable, to one, i.e., perfectly suitable; a more fine grained evaluation would not make much practical sense.

Clearly, this definition of user interaction is simplified and idealized, in particular as

we assume here that all users always give precise answers. In a real application, the uncertainty of user feedback and the possibility of misbehaving users who intentionally give misleading answers also need to be considered among other aspects. Moreover, it would be meaningful to extend the possibilities of user feedback. For example, users could be allowed to optionally rate more than one suitable locations for an SPR in one scenario or to make suggestions which locations to additionally include in a scenario.

In each iteration of COA, users get presented individual sets of location scenarios $\mathcal{S}_r$ for their service point requirements $r \in R$. These scenarios are compiled according to the following strategies.

Remember that location suitability values obtained from the users are later used in the EC for training the surrogate function $\tilde{w}_{\boldsymbol{\Theta}}$. Moreover, by enforcing that each user is required to select a best suited service point location in a presented location scenario for an SPR r, a suitability value indicated by the user for some location $v_{r,S}$ also serves as upper bound on the suitability values of all other locations in the location scenario S; thus, $w_{r,v_{r,S}} \geq w(r,v),\ \forall v \in S$. By $w^{\mathrm{UB}}_{r,v}$, the SMC maintains for each SPR $r \in R$, and each location $v \in V$ the so far best obtained upper bound on each $w(r,v)$; initially, $w^{\mathrm{UB}}_{r,v} = 1$.

Let $V_w(r) = \{v \mid w(r,v) > 0\}$ be the initially unknown set of locations that are actually relevant to a user $u \in U$ w.r.t. an SPR $r \in R_u$. A straight-forward strategy to identify this set is to iteratively present the user scenarios $S^{\mathrm{V}}_r = \{v \in V \mid (r,v) \notin K\}$, containing all locations $v \in V$ for which no entry $(r) \in K$ exists yet, i.e., locations for which no suitability values are known yet w.r.t. r. Following this strategy, it can be ensured to identify a new location of $V_w(r)$ in every iteration of COA. Note, however, that it can only be guaranteed that $V_w(r)$ is completely known once the user returns that none of the locations in the last scenario S^{V}_r are suitable for r. Consequently, $V_w(r)$ will be completely known after $|V_w(r)| + 1$ user interactions.

Hence, an upper bound I^{UB}_u on the total number of required interactions with user u for completely identifying all relevant locations for all of his/her use cases is

$$I^{\mathrm{UB}}_u = \sum_{r \in R_u} \left(|V_w(r)| + 1\right). \tag{4.20}$$

While this value is unknown in a real-world scenario, it allows us to establish a measure of quality on how well our strategy for presenting scenarios to users performs within our testing environments.

As strategies for generating scenarios we adapt strategies developed for COA[ISPDP], specifically the best solutions strategy as well as the relevant locations strategy: The first strategy generates scenarios $S^{*}_r = \{v \in \tilde{X}^{*} \mid (r,v) \notin K\}$ containing all locations from the current best solution that have not been rated yet w.r.t. r. The second strategy generates scenarios according to the approach described above, i.e., $S^{\mathrm{V}}_r = \{v \in V \mid (r,v) \notin K\}$.

Note that for users generally only a fraction of the service point locations in V is actually relevant to one of their SPRs. Hence, when presenting a user $u \in U$ two

scenarios for each of the user's SPRs every iteration the number of user interactions would quickly exceed I_u^{UB}. Therefore, in the first COA iteration a scenario S_r^{V} is generated for each $r \in R$, but in successive iterations, scenarios are only generated for subsets of R. More specifically from the second iteration onward, ς^{V} and ς^* percent of the SPRs $R_K = \{r \in R \mid \exists v \in V : (r,v) \notin K\}$ are randomly selected for generating scenarios according to S_r^{V} and S_r^*, respectively, with ς^{V} and ς^* being strategy parameters.

Evaluation Component

The exact objective function f from (4.11), which is based on the mostly unknown suitability values $w(r,v)$ with $r \in R,\ v \in V$, is approximated by the surrogate objective function $\tilde{f}$, making use of the following *surrogate suitability function*

$$\tilde{w}_{\boldsymbol{\Theta}}(r,v) = \begin{cases} w(r,v) & \text{if } (r,v) \in K \\ \max(0, \min(w_{r,v}^{\mathrm{UB}}, g_{\boldsymbol{\Theta}}(r,v))) & \text{else.} \end{cases} \tag{4.21}$$

Generally speaking, $g_{\boldsymbol{\Theta}}$ is here a learnable function with weight parameters $\boldsymbol{\Theta}$ approximating $w(r,v)$ for all unknown pairs $(r,v) \notin K$. The above definition thus ensures that $\tilde{w}_{\boldsymbol{\Theta}}$ always returns known values $w(r,v)$ and otherwise respects lower bounds zero and upper bounds $w_{r,v}^{\mathrm{UB}}$, giving function g more freedom. Upper bounds $w_{r,v}^{\mathrm{UB}}$ are initially set to one. The SMC is then responsible for deriving tighter upper bounds.

Suitability values are approximated by exploiting similarities of SPRs among users. In general we cannot expect that there exist users having the same needs in all respects, i.e., the users have the very same use cases with the same demands. However, given a sufficiently large user base it is realistic that there are users having similar SPRs and associated preferences concerning suitable locations.

A popular collaborative filtering technique for exploiting similarities among user preferences is *matrix factorization* [18], which we also apply here. Given an incomplete matrix containing ratings $\boldsymbol{\mathcal{R}} = (w_{i,j})_{i \in U,\, j \in P}$ for a set of users U over a set of products P, the idea behind matrix factorization is to decompose this matrix into two smaller matrices, a user/feature matrix $\boldsymbol{\xi}$ and a product/feature matrix $\boldsymbol{\nu}$, such that the product of these two matrices approximates the original matrix. An unknown rating, i.e., a rating not contained in the original matrix $\boldsymbol{\mathcal{R}}$, can then be estimated as the dot product of the corresponding feature vectors in matrix $\boldsymbol{\xi}$ and matrix $\boldsymbol{\nu}$, respectively.

Moreover, we also want to exploit the fact that only a small fraction of the locations in V is typically relevant for the SPR of a user and that unknown ratings are not missing at random. In our problem users are always asked to rate the most suitable location of a scenario. Therefore, known ratings tend to be biased towards more positive values while unrated locations are likely to have a low suitability for a user w.r.t. an SPR. A matrix factorization approach that takes such considerations into account has been suggested by [19]. Traditionally, the rating matrix $\boldsymbol{\mathcal{R}}$ is factorized by solving the optimization problem

$$\min_{\boldsymbol{\xi},\boldsymbol{\nu}} \sum_{i,j \mid w_{i,j} \in \boldsymbol{\mathcal{R}}} E(w_{ij}, \boldsymbol{\xi}_i \boldsymbol{\nu}_j') + \rho(\boldsymbol{\xi}, \boldsymbol{\nu}) \tag{4.22}$$

where E is a loss function for measuring the error between the actual and the predicted ratings and ρ is a regularization term. In [19] this minimization problem is expanded by adding a bias term for unknown ratings towards a certain value $\hat{w}$, i.e.,

$$\min_{\boldsymbol{\xi},\boldsymbol{\nu}} \sum_{i,j|w_{i,j}\in\mathcal{R}} E(w_{i,j}, \boldsymbol{\xi}_i\boldsymbol{\nu}'_j) + \alpha \sum_{i,j|w_{i,j}\notin\mathcal{R}} E(\hat{w}, \boldsymbol{\xi}_i\boldsymbol{\nu}'_j) + \rho(\boldsymbol{\xi}, \boldsymbol{\nu}). \tag{4.23}$$

Parameter α controls the impact of this new term in the optimization. The authors show for selected loss functions how this new optimization problem can be solved in the same time complexity as the traditional optimization problem.

In order to apply matrix factorization for approximating suitability values $w(r, v)$ in our case, we start from the sparsely filled matrix $\mathbf{W} = (w(r, v))_{(r,v)\in K}$ containing all so far known suitability values. By factorizing $\mathbf{W}$ along the r dimension and the v dimension on the basis of a feature set $F = \{1, \ldots, \phi\}$, we obtain an SPR/feature matrix $\boldsymbol{\xi} = (\boldsymbol{\xi}_{r,i})_{r\in R, i\in F}$ with $\boldsymbol{\xi}_{r,i} \in \mathbb{R}$ and a location/feature matrix $\boldsymbol{\nu} = (\boldsymbol{\nu}_{v,i})_{v\in V, i\in F}$ with $\boldsymbol{\nu}_{v,i} \in \mathbb{R}$. Feature vectors $\boldsymbol{\xi}_r$ describe the SPR r in terms of abstract features, while feature vectors $\boldsymbol{\nu}_v$ reflect the characteristics of locations v. In general, it is expected that SPRs with similar needs will have similar feature vectors in $\boldsymbol{\xi}$, and locations with similar suitability characteristics will have similar feature vectors in $\boldsymbol{\nu}$. The number of features ϕ is hereby a parameter that is chosen, e.g., in dependence of an estimation of the overall number of different service point requirements, and we assume it is considerably smaller than the overall number of SPRs as well as the number of locations n. As unknown suitability values are more likely zero than being greater than zero, we set the bias target $\hat{w} = 0$.

Having obtained matrices $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$, an unknown value of $\mathbf{W}$ is approximated by the dot product of the respective feature vectors rounded to the nearest of the five discrete suitability values we defined, i.e.,

$$g_{\boldsymbol{\Theta}}(r, v) = \lfloor 4 \cdot \boldsymbol{\xi}_r\boldsymbol{\nu}'_v + 0.5 \rfloor / 4. \tag{4.24}$$

The trainable parameters of $g_{\boldsymbol{\Theta}}$ are therefore $\boldsymbol{\Theta} = (\boldsymbol{\xi}, \boldsymbol{\nu})$.

Our loss function for the matrix factorization is

$$\min \sum_{(r,v)\in K} \left(w(r, v) - \boldsymbol{\xi}_r\boldsymbol{\nu}'_v\right)^2 + \alpha \sum_{(r,v)\notin K} (\boldsymbol{\xi}_r\boldsymbol{\nu}'_v)^2 + \lambda(\|\boldsymbol{\xi}_r\|^2 + \|\boldsymbol{\nu}_v\|^2), \tag{4.25}$$

and randomized block coordinate descent [121] is used to minimize it.

Optimization Component

Recall that the OC is performed in each major iteration of the framework and makes use of the current surrogate objective function $\tilde{f}$ provided by the EC. The surrogate objective function does not change during each individual call of the OC. The OC is thus supposed to return an optimal or close-to-optimal solution to our problem w.r.t. the current surrogate objective function.

A potential exact optimization approach is to apply a general purpose MILP solver to the MILP formulation already presented in Section 4.6.1, Equations (4.12)–(4.19), however suitability values are approximated by the surrogate suitability function $\tilde{w}_{\mathbf{\Theta}}$.

Note that for improved scalability, a metaheuristic approach might be more suitable as optimization core as it is not necessary to find an optimal solution in each iteration. Therefore, we propose an LNS as optimization core. In our LNS a solution to a GSPDP instance is destroyed in a uniform random fashion by adding k^{dest} new locations to the solution, where k^{dest} is a parameter that is varied.

To repair a solution X, we make use of a randomized greedy approach: Let $\Delta(v, X)$ denote by how much the objective value of a solution X would decrease when removing location v from X. Note that, it is discussed later how $\Delta(v, X)$ can be efficiently calculated for all $v \in X$. In each iteration we first generate a restricted candidate list of k^{rep} locations $v \in V$ for which $\Delta(v, X)$ is lowest, i.e., the candidate list contains the locations that have the lowest impact on objective value of X. Hereby, k^{rep} is another strategy parameter. Ties are broken randomly. A location is then chosen uniformly at random from this restricted candidate list and removed from X. The greedy procedure terminates once the solution is feasible, i.e., the associated budget is no longer exceeded.

To construct an initial solution in the first iteration of COA, we also make use of the repair heuristic, starting from $X = V$ and then sequentially removing locations from X for which $\Delta(v, X)$ is lowest until the solution becomes feasible, i.e. $k^{\text{rep}} = 1$ for constructing an initial solution. In subsequent iterations of COA, the LNS is warm-started with COA's current best solution $\tilde{X}^*$.

Our LNS makes use of two destroy operators with $k^{\text{dest}} = 10$ and $k^{\text{dest}} = 20$, respectively, and two repair operators with $k^{\text{rep}} = 2$ and $k^{\text{rep}} = 4$, respectively. These settings have shown to yield a robust convergence behavior across the kinds and sizes of instances in our benchmark sets. In each iteration a repair and destroy operator is chosen uniformly at random. Moreover, each LNS run terminates after 40 iterations without improvement.

A crucial aspect for developing an effective heuristic for solving the GSPDP is that computing the surrogate objective value $\tilde{f}$ of a solution in a straight-forward way from scratch is time consuming. Hence, in order to accelerate this task we maintain for a GSPDP instance a directed graph $G = (LL \cup SL \cup CL \cup \{l^{\text{obj}}\}, A_{LL} \cup A_{SL} \cup A_{CL})$ referred to as *evaluation graph*. This graph represents the objective function calculation and stores intermediate results for a current solution, allowing for an effective incremental update in case of changes in the solution. The evaluation graph consists of four layers of nodes, which are the location layer (LL), the SPR layer (SL), the use case layer (CL), and the evaluation layer containing a single node l^{obj}. The location layer contains n nodes corresponding to the locations in V, i.e., $LL = \{l^{\text{l}}_v \mid v \in V\}$. The use case layer consists of one node for each use case C_u of each user $u \in U$, i.e., $CL = \{l^{\text{c}}_c \mid c \in C_u,\ u \in U\}$, and the SPR layer contains one node for each SPR in $\in R_{u,c}$, for each use case $c \in C_u$ and user $u \in U$, i.e., $SL = \{l^{\text{r}}_{u,r} \mid r \in R_{u,c},\ c \in C_u, u \in U\}$.

There exists an arc in G from a node of the location layer l^{l}_v to a node of the SPR layer $l^{\text{r}}_{u,r}$

if $\tilde{w}_{\mathbf{\Theta}}(v,r) > 0$, i.e., $A_{LL} = \{(l^{\mathrm{l}}_v, l^{\mathrm{r}}_{u,r}) \mid l^{\mathrm{l}}_v \in LL,\ l^{\mathrm{r}}_{u,r} \in SL,\ \tilde{w}_{\mathbf{\Theta}}(v,r) > 0\}$. A node of the SPR layer is connected to a node of the use case layer if the corresponding SPR is an SPR of the corresponding use case, i.e., $A_{SL} = \{(l^{\mathrm{r}}_{u,r}, l^{\mathrm{c}}_c) \mid l^{\mathrm{r}}_{u,r} \in SL,\ l^{\mathrm{c}}_c \in CL,\ r \in R_{u,c}\}$. Finally, each node l_c of the use case layer is connected to l^{obj}, i.e., $A_{CL} = \{(l^{\mathrm{c}}_c, l^{\mathrm{obj}}) \mid l^{\mathrm{c}}_c \in CL\}$.

The location layer gets as input a binary vector $(x_v)_{v \in V}$ with $x_v = 1$ if $v \in X$ and $x_v = 0$ otherwise, w.r.t. a solution X. Figure 4.7 shows the structure of an evaluation graph. Moreover, each node in G has an activation function $\alpha()$ that decides its output value

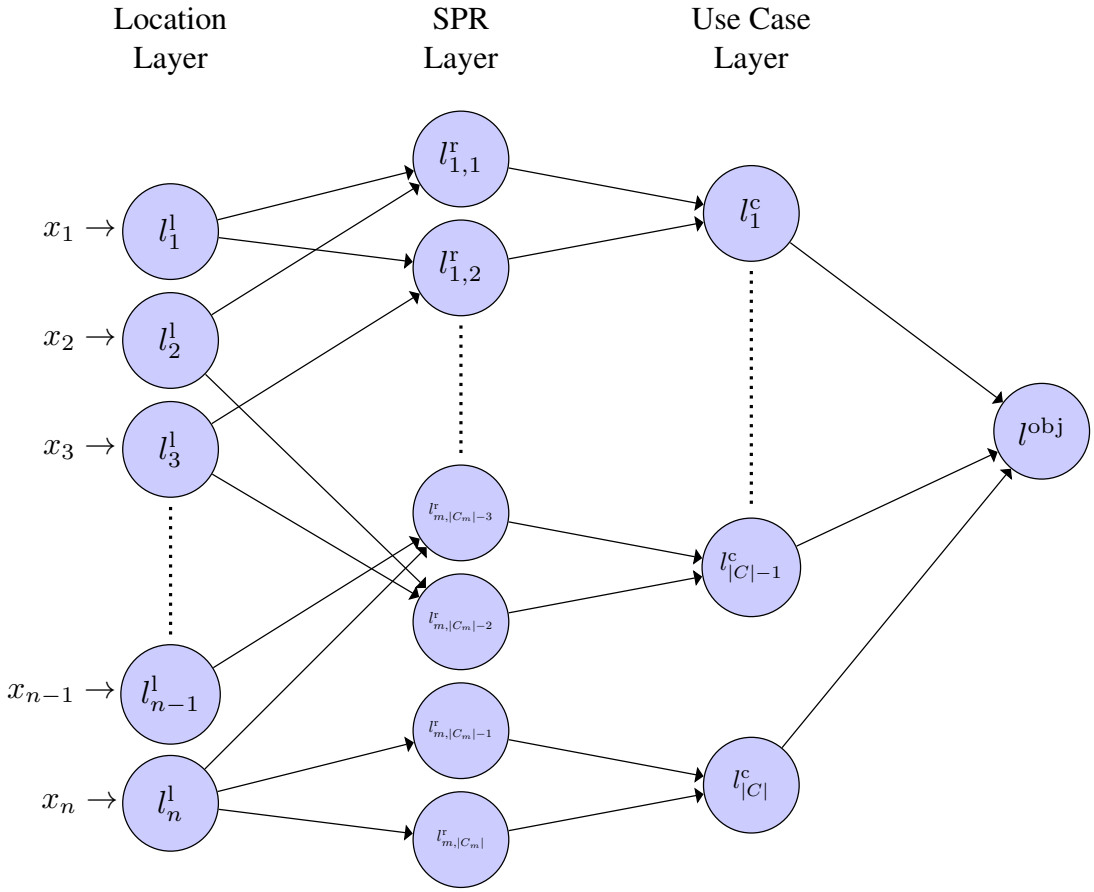


Figure 4.7: Structure of an evaluation graph where $|C|$ refers to the total number of use cases over all users.

which is propagated to its successor nodes in the next layer as their input, i.e.,

$$\alpha_{LL}(l^{\mathrm{l}}_v, X) = \begin{cases} 1 & \text{if } v \in X \\ 0 & \text{otherwise,} \end{cases} \qquad \forall l^{\mathrm{l}}_v \in LL, \tag{4.26}$$

$$\alpha_{SL}(l^{\mathrm{r}}_{u,r}, X) = \max_{(l^{\mathrm{l}}_v, l^{\mathrm{r}}_{u,r}) \in A_{LL}} (\alpha_{LL}(l^{\mathrm{l}}_v, X) \cdot \tilde{w}_{\mathbf{\Theta}}(v,r)) \qquad \forall l^{\mathrm{r}}_{u,r} \in SL, \tag{4.27}$$

$$\alpha_{CL}(l_c^{\mathrm{c}}, X) = \min_{(l_{u,r}^{\mathrm{r}}, l_c^{\mathrm{c}}) \in A_{SL}} \alpha_{SL}(l_{u,r}^{\mathrm{r}}, X) \qquad \forall l_c^{\mathrm{c}} \in C_L, \tag{4.28}$$

$$\alpha_{eval}(l^{\mathrm{obj}}, X) = \sum_{(l_c^{\mathrm{c}}, l^{\mathrm{obj}}) \in A_{CL}} \alpha_{SL}(l_c^{\mathrm{c}}, X) - \sum_{v \in X} z_v^{\mathrm{var}}. \tag{4.29}$$

The evaluation graph stores all output of the activation functions from the last evaluated solution and is therefore especially efficient for evaluating subsequent solutions that only differ in a single location $v \in V$ as not everything needs to be calculated from scratch but just the modified value v w.r.t. the current solution X needs to be propagated. Note that A_{LL} needs to be updated in each iteration of COA as the EC recalculates the surrogate suitability values $\tilde{w}_{\boldsymbol{\Theta}}$ in each iteration with newly obtained user feedback.

Additionally, the evaluation graph also makes it possible to efficiently keep track of how much each location v contributes to the objective value of a solution. For this purpose, we introduce the following new notations. Let X be a current solution and $c \in C_u$ be a use case of a user $u \in U$ that is satisfied (to some degree) in X, i.e., for each $r \in R_{u,c}$ there exists at least one location $v \in X$ such that $\tilde{w}_{\boldsymbol{\Theta}}(r, v) > 0$. Let $v^{\max}(r, X)$ refer to a location in the solution for which $\tilde{w}_{\boldsymbol{\Theta}}(r, v^{\max}(r, X)) = \max_{v \in X} \tilde{w}_{\boldsymbol{\Theta}}(r, v)$. For the sake of readability we further refer to $\tilde{w}_{\boldsymbol{\Theta}}(r, v^{\max}(r, X))$ as $\tilde{w}_{\boldsymbol{\Theta}}^{\max}(r, X)$. Additionally, let $\tilde{w}_{\boldsymbol{\Theta}}^{\mathrm{fallback}}(r, X)$ denote the second highest suitability value for an SPR r w.r.t. to the locations in X, i.e., $\tilde{w}_{\boldsymbol{\Theta}}^{\mathrm{fallback}}(r, X) = \max\{\tilde{w}_{\boldsymbol{\Theta}}(r, v) \mid \ v \in \{X \setminus \{v^{\max}(r, X)\}\} \cup \{0\}\}$. Note that $\tilde{w}_{\boldsymbol{\Theta}}^{\mathrm{fallback}}(r, X)$ is zero if $X \setminus \{v^{\max}(r, X)$ is empty. Finally, let $\tilde{w}_{\boldsymbol{\Theta}}^{\min}(u, c, X) = \min_{r \in R_{u,c}} \tilde{w}_{\boldsymbol{\Theta}}^{\max}(r, X)$.

From the definition of the surrogate objective function, it follows that the degree to which a use case c is satisfied in a solution X is only determined by the set of locations $\{v^{\max}(r, X) \mid r \in R_{u,c}\}$. Hence, let $\Delta(u, c, v, X)$ denote by how much the degree to which a use case $c \in C_u$ of a user $u \in U$ is satisfied w.r.t. a solution X would decrease when removing v from X, i.e.,

$$\Delta(u, c, r, X) = \begin{cases} q \cdot D_{u,c} \cdot (\tilde{w}_{\boldsymbol{\Theta}}^{\max}(r, X) - \tilde{w}_{\boldsymbol{\Theta}}^{\mathrm{fallback}}(r, X)) & \tilde{w}_{\boldsymbol{\Theta}}^{\mathrm{fallback}}(r, X) < \tilde{w}_{\boldsymbol{\Theta}}^{\min}(u, c, X) \\ 0 & \text{otherwise} \end{cases} \tag{4.30}$$

$$\Delta(u, c, v, X) = \max\left(\{\Delta(u, c, r, X) \mid r \in R_{u,c}, v = v^{\max}(r, X)\} \cup \{0\}\right) \tag{4.31}$$

Generally speaking, the removal of a location v from a solution X only has an impact on a use case $c \in C_u$ if it results in a change of $\tilde{w}_{\boldsymbol{\Theta}}^{\min}(u, c, X)$. Additionally, note that the GSPDP also allows cases in which one service point location can be associated to multiple SPRs of the same use case. Such a case would for example correspond to situations in which a customer returns a vehicle at the same station at which the vehicle was picked up. Therefore, the removal of a location from X may affect a use case w.r.t. more than one of its SPRs. However, only the change that affects $\tilde{w}_{\boldsymbol{\Theta}}^{\min}(u, c, X)$ the most is relevant for calculating by how much the degree to which a use case is satisfied changes.

Hence, the amount $\Delta(v, X)$ by how much the objective value of a solution would decrease

when removing location v from X is calculated as

$$\Delta(v, X) = -z_v^{\text{var}} + \sum_{u \in U} \sum_{c \in C_u} \Delta(u, c, v, X). \tag{4.32}$$

Note that the time required for determining $w^{\max}, w^{\text{fallback}}$, and $w^{\min}$ is negligible if the domain of the rating scale by which users can specify suitability values is small. Moreover, $\Delta(v, X)$ does not need to be calculated from scratch every time a location is added or removed from the solution. Let $X \circ \{v\}$ refer to the modification of a solution, by either adding or removing a location $v \subseteq V$ to/from X. Then $\Delta(v', X \circ \{v\})$ with $v' \in X$ can be determined from $\Delta(v', X)$ as follows:

$$\Delta(v', X \circ \{v\}) = \Delta(v', X) - \sum_{u \in U} \sum_{c \in C_u} \Delta(u, c, v', X) + \Delta(u, c, v', X \circ \{v\}). \tag{4.33}$$

Additionally, $\Delta(v, X)$ needs to be updated only w.r.t. use cases that are actually affected by the modification of the solution, i.e., only if $\tilde{w}_{\boldsymbol{\Theta}}^{\max}, \tilde{w}_{\boldsymbol{\Theta}}^{\text{fallback}}$, or $\tilde{w}_{\boldsymbol{\Theta}}^{\min}$ of a use case change. Finally, for each use case $c \in C_u$ at most $2 \cdot |R_{u,c}|$ locations need to updated in the worst case.

4.6.3 Test Cases

In the following it is described how test instances for evaluating COA have been generated.

Similar to COA[ISPDP], user interaction is simulated in an idealized manner in certain test cases in order to analyze the strengths and weaknesses of the framework with a focus on the algorithmic aspects. The considered test instances are of three groups. The first two groups are purely artificial test instances inspired by the location planning of stations for electric vehicle charging, denoted by EVC, and for (station-based) car sharing systems, denoted by CSS, respectively. While in group EVC each use case has one SPR, there are always two SPRs per use case in CSS. The third group of test instances is designed similarly to group CSS, also addressing the car sharing scenario, but the instances are generated from real-world taxi trip data of Manhattan; this group is therefore called MAN. Note that COA intentionally does not make use of geographic information in any of its components. Therefore, modeling preferences of users for our instances in dependence of the proximity to service point locations does not provide COA any advantage for finding an optimized solution.

All of our benchmark instances are available online at `https://www.ac.tuwien.ac.at/research/problem-instances/#spdp`.

Artificial Test Instance Groups EVC and CSS

Test instances from the groups EVC and CSS are generated with the same approach and is based on the approach used for generating instances to the ISPDP. The n possible locations for service stations are randomly distributed in the Euclidean plane with coordinates $\text{coord}(v)$, $v \in V$, chosen uniformly from the grid $\{0, \ldots, L-1\}^2$, with

$L = \lceil 10\sqrt{n} \rceil$. The fixed costs z_v^{fix} as well as the variable costs z_v^{var} for setting up a service station at each location $v \in V$ are uniformly chosen at random from $\{50, \ldots, 100\}$. The budget is assumed to be $B = \lceil 7.5 \cdot n \rceil$ so that roughly 10% of the stations with average costs can be expected to be opened.

The number of use cases for each user $u \in U$ is chosen randomly according to a shifted Poisson distribution with offset one, expected value three, and a maximum value of five, i.e., values are generated anew if they exceed five. Hence, the number of use cases never exceeds five. Each of these use cases $c \in C_u$ is associated with an individual demand $D_{u,c}$ randomly chosen from {5,...,50} and, depending on the benchmark group, with one (EVC) or two (CSS) SPRs.

Each SPR $r \in R_{u,c}$ of a use case c also is associated with a particular geographic location $q_r \in \{0, \ldots, L-1\}^2$. In order to model similarities in the users' SPRs, these locations are selected in the following correlated way. First n_α *attraction points* A with uniform random coordinates are selected from $\{0, \ldots, L-1\}^2$. Then, each use case location is derived by randomly choosing one of these attraction points $(a_x, a_y) \in A$ and adding a small individual offset to the coordinates, i.e.,

$$q_r = (\lfloor \mathcal{N}(a_x, \sigma_{\text{v}}) \rfloor, \lfloor \mathcal{N}(a_y, \sigma_{\text{v}}) \rfloor), \tag{4.34}$$

where $\mathcal{N}(\cdot, \cdot)$ denotes a random value sampled from a normal distribution with the respectively given mean value and standard deviation σ_{v}. If obtained coordinates are not in $\{0, \ldots, L-1\}^2$ a new attraction point is chosen and the deviation is re-sampled.

A service point location $v \in V$ receives a rating w.r.t. an SPR r according to a sigmoidal decay function applied to the Euclidean distance, and is also perturbed by a Gaussian noise with a standard deviation of σ_{r}:

$$w'_{r,v} = \mathcal{N}\left(\frac{1}{1 + 6e^{0.5||q_r - \text{coord}(v)|| - 6}}, \sigma_{\text{r}} \right). \tag{4.35}$$

The parameters of the sigmoid function are chosen so that $w'_{r,v}$ decreases as the distance between v and q_r increases and becomes approximately zero at a distance larger than twelve. Additionally, we discretize the rating $w'_{r,v}$ by rounding to the closest value in $\{0, 0.25, 0.5, 0.75, 1\}$, obtaining $w(r, v)$. Hence, $w(r, v) = \lfloor 4 \cdot \min(1, \max(0, w'_{r,v})) + 0.5 \rfloor / 4$.

Manhattan Test Instances

Next to the above described purely artificial benchmark instances we also derive benchmark instances from real-world yellow taxi trip data of Manhattan. As in CSS, MAN instances have two SPRs per use case. The underlying street network G of the instances corresponds to the street network graph of Manhattan provided by the Julia package LightOSM[3]. Taxi trips have been extracted from the 2016 Yellow Taxi Trip Data[4]. The

[3] https://github.com/DeloitteDigitalAPAC/LightOSM.jl

[4] https://data.cityofnewyork.us/Transportation/2016-Yellow-Taxi-Trip-Data/k67s-dv2t

taxi data set was first preprocessed by removing all trips with invalid data and trips made on a weekend. Furthermore, we have also removed all trips which do not start as well as end in Manhattan. For taxi trips within the months January to July geographic pickup and drop-off coordinates of customers are recorded in the data set. Each of these coordinates has been extracted and mapped to the geographically closest vertex in G, resulting in a list of pairs of vertices $Q \subseteq V(G) \times V(G)$. Next, as the similarity of users in our instances depends on their geographic proximity, we have reduced Q by considering only the ten taxi zones with the highest total number of pickups and drop-offs of customers, resulting in a total of approximately two million taxi trips. Geographic information of the taxi zones of Manhattan has been obtained from `https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc`. The left side of Figure 4.8 provides a visualization of the selected taxi zones.

The set of potential service point locations V has been chosen randomly from vertices of G that are located in the considered taxi zones. The fixed costs z_v^{fix} as well as the variable costs z_v^{var} for setting up a service station at each location $v \in V$ are uniformly chosen at random from $\{50, \ldots, 100\}$.

The number of use cases for each user $u \in U$ is again chosen randomly according to a shifted Poisson distribution with offset one, expected value three, and a maximum value of five. Each of these use cases $c \in C_u$ is associated with an individual demand $D_{u,c}$ randomly chosen from $\{5, \ldots, 50\}$ and the two SPRs representing the origin and destination of a trip chosen from Q uniformly at random. A rating for an SPR r is calculated for each $v \in V$ via the sigmoidal decay function

$$w'_{r,v} = \frac{1}{1 + 10e^{0.01\text{sp}(r,v)} - 6}, \tag{4.36}$$

where $\text{sp}(r, v)$ refers to the length of the shortest path between location v and the SPR r in the street network graph G. The parameters of this function have been chosen in such a way that service point locations within a distance of approximately 600 meters to r are relevant for the SPR. Finally, the discretized suitability value $w(r, v)$ is again obtained by $w(r, v) = \lfloor 4 \cdot \min(1, \max(0, w'_{r,v})) + 0.5 \rfloor / 4$. The right side of Figure 4.8 shows the distribution of SPR locations as well as potential service point locations for an example instance.

The MAN benchmark group consists of 30 instances in total with each instance having 100 potential service point locations and 2000 users. Additionally, each instance will be evaluated with different budget levels $b\,[\%] \in \{30, 50, 70\}$ such that about b percent of the stations considering average costs can be opened, i.e, the actual budget for each instance is calculated as $B = \lceil b \cdot 0.75 \cdot n \rceil$.

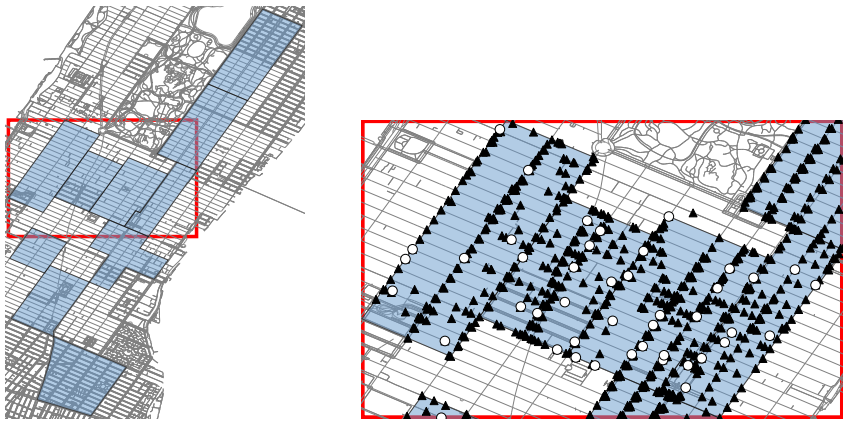


Figure 4.8: Left: The considered ten taxi zones of Manhattan with the highest number of pickups and drop-offs. Right: Exemplary distribution of SPR locations (black triangles) and potential service point locations (white points).

4.6.4 Results

The COA framework including the FC was implemented in Python 3.8. For the matrix factorization of the EC we adapted the C++ implementation of [19] provided on Github[5]. Gurobi 9.1[6] was used to solve the MILP models in the OC while the LNS was written in Julia 1.6[7]

Six sets of 30 benchmark instances were generated for EVC as well as CSS. As detailed in Table 4.3, these sets consider $n \in \{100, 200, 300\}$ potential service point locations in combination with different numbers of users and two different settings for the standard deviations of the Gaussian perturbations σ_v and σ_r. For each instance the number of attraction points n_α is ten. In the following we will denote instance sets primarily by the pair (n, m).

The parameterization of the COA[GSPDP] components have been determined through preliminary tests on an independent set of instances. For all test runs the weighting for unknown suitability values α of the matrix factorization has been set to one. Moreover, the number of features considered in the matrix factorization is set to ten for all test runs. The parameters ς^V and ς^* for controlling the number of scenarios generated according to each strategy in the FC have been set to 0.5 and 0.1, respectively.

In each COA[GSPDP] iteration a time limit of ten minutes has been set for solving the MILP. If the MILP was not solved to optimality within this time limit, the best found solution was used. 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. Note however, that all runs terminated within this time limit once all relevant locations

[5] https://github.com/rdevooght/MF-with-prior-and-updates
[6] https://www.gurobi.com/
[7] https://www.julialang.org/

Table 4.3: Main parameters of the EVC and CSS instance sets of groups EVC and CSS. Each row represents a set of 30 instances.

(n, m)	σ_v	σ_r
(100, 500)	3.0	0.03
(100, 1000)	5.0	0.15
(200, 1000)	3.0	0.03
(200, 2000)	5.0	0.15
(300, 1500)	3.0	0.03
(300, 3000)	5.0	0.15

had been discovered. Since we have full knowledge of our test instances, we are also able to calculate optimal reference solutions for each instance. Hence, by $f(x_{\text{opt}})$ we denote the objective value of a respective optimal solution x_{opt}.

To characterize the amount of user interaction performed by COA[GSPDP], we consider the total number of scenarios evaluated by a user $u \in U$ in relation to the upper bound of required interactions I_u^{UB}. Let I_u be the number of user interactions of user $u \in U$ performed within COA[GSPDP] to generate some solution. Then, $I = 100\% \cdot (\sum_{u \in U} I_u / I_u^{\text{UB}})/m$, refers to the relative average number of performed user interactions relative to I_u^{UB} over all users. Note that since scenarios are presented only to a fraction of users in every iteration, the average number of user interactions at each iteration of COA varies even for instances within the same benchmark group. Hence, in order to reasonably study results for each of our benchmark groups, we aggregate respective results to our instances at various *interaction levels* ψ by selecting for each instance the COA iteration at which I is largest but does not exceed ψ. Note that for some instances smaller levels of ψ are already exceeded in the first iteration of COA[GSPDP]. Hence, in the following we only consider interaction levels for an instance group for which results to all corresponding instances exist.

Note further that the user interaction levels can also be interpreted as the average information known about a user. This interpretation allows us to draw a direct comparison to traditional approaches for distributing service points in which information about the demands of the users is determined in advance. Each result at a certain interaction level can also be interpreted as the result of such a traditional approach with a certain level of knowledge about the users. However, to the best of our knowledge there exists no data about suitability values in other work. Additionally, for a fair comparison between COA[GSPDP] and other approaches from literature one would have to also take into account the costs required for obtaining said information about the users. Therefore, comparing COA[GSPDP] to other approaches from literature seems to be not possible without an extensive study on suitability values of users or a complex simulation of users based on various assumptions that can heavily influence the outcome of such a comparison.

First we discuss results for COA[GSPDP] with the MILP as optimization core. Afterwards, we evaluate the OC in more detail by comparing the performance of the LNS throughout COA to the performance of the MILP. Finally, we also compare the COA[GSPDP] to COA[ISPDP].

Results for the MILP as optimization core

First, we want to show how the quality of incumbent solutions develops as the number of user interactions increases during a COA[GSPDP] run. For this purpose, we calculate the optimality gap for a solution x obtained from COA[GSPDP] as

$$\text{gap} = 100\% \cdot (f(x_{\text{opt}}) - f(x))/f(x_{\text{opt}}). \tag{4.37}$$

Figure 4.9 shows the average optimality gaps of solutions to each of our benchmark sets against the interaction levels ψ. The results are grouped by σ_v and σ_r to additionally compare instances groups with similar user behavior.

Recall that the number of attraction points is the same for all EVC and CSS instances. Therefore, for instances with a higher number of users it is generally easier to find better solutions as there are more users that prefer the same locations. The plots show that in all cases solutions generally improve quickly with an increasing interaction level and close to optimal solutions can be obtained well before identifying all the users' relevant locations. Specifically, at a user interaction level of 50% the solutions generated by COA[GSPDP] feature optimality gaps of 1.45% on average. An exception of this observation are the MAN instances with $b = 30\%$. For these generated solutions do not reach an optimality gap below 1% before $\psi = 80\%$ on average. Moreover, the figure also shows that the solutions to MAN instances generally converge notably slower than the solutions to EVC and CSS instances. This behavior is likely caused by the weaker correlation of user preferences in the MAN instances and the way how the FC generates scenarios presented to the users, specifically the aspect that locations important to the individual users are tried to be identified first. Locations important to individual users might not necessarily be the best locations to add to a solution, especially if there is a lower number of users with similar preferences. Consequently, the strategies based on which scenarios are generated may still have some room for improvement for such cases. Instead of primarily identifying locations important to users, targeting locations in relation to the current best solution with a higher emphasis might be a more expedient approach here.

Note that an increased number of user interactions does not only imply a larger training set for the surrogate function but also results in better upper bounds $w^{\text{UB}}_{r,v}$ for locations $v \in V$ w.r.t. to an SPR $r \in R$. Therefore, to gain a better understanding of how much the surrogate function actually contributes to finding an optimized solution, we study what happens when the learning surrogate suitability function $\tilde{w}_{\boldsymbol{\Theta}}$ is replaced by the naive function with no learning capabilities

$$\tilde{w}_{\text{bl}}(r, v) = \begin{cases} w(r, v) & \text{if } (r, v) \in K \\ 0 & \text{else.} \end{cases} \tag{4.38}$$

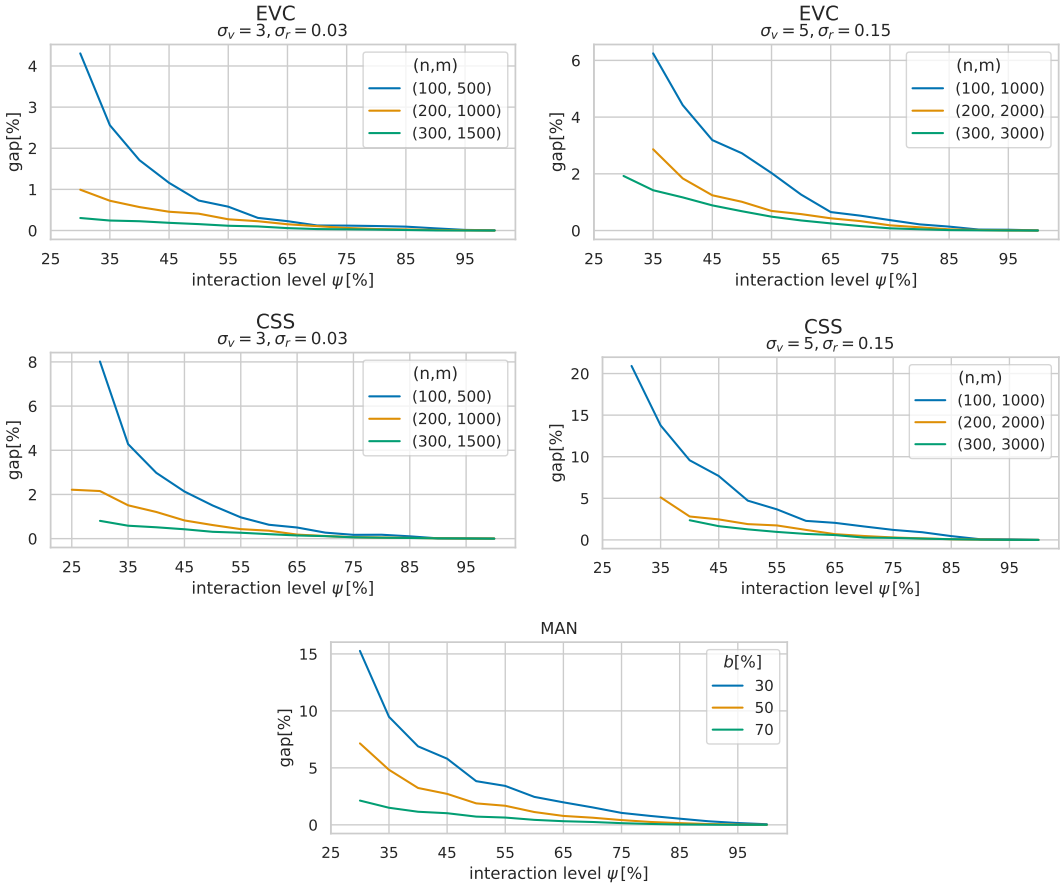


Figure 4.9: Development of average optimality gaps with an increasing number of user interactions for each benchmark instance set.

In the following we refer to our original COA implementation with the surrogate function $\tilde{w}_{\mathbf{\Theta}}$ as COA$[\tilde{w}_{\mathbf{\Theta}}]$ and denote the implementation with the naive function $\tilde{w}_{\text{bl}}$ as COA$[\tilde{w}_{\text{bl}}]$. A comparison between COA$[\tilde{w}_{\mathbf{\Theta}}]$ and COA$[\tilde{w}_{\text{bl}}]$ is shown in Table 4.4. Each table cell shows the average optimality gaps of solutions to the respective instance group at the specified interaction levels ψ. The better results among COA$[\tilde{w}_{\mathbf{\Theta}}]$ and COA$[\tilde{w}_{\text{bl}}]$ are printed bold. Additionally, as the standard deviations w.r.t. the optimality gaps are quite large, see Figure 4.10, we have also applied a one-sided Wilcoxon signed-rank test to determine for each group whether the difference in optimality gaps is significant or not. Instance groups for which the Wilcoxon test has assessed at a 95% confidence interval that either COA$[\tilde{w}_{\mathbf{\Theta}}]$ or COA$[\tilde{w}_{\text{bl}}]$ has produced better optimality gaps are marked with an asterisk.

The table shows that especially for the CSS and MAN instances COA$[\tilde{w}_{\mathbf{\Theta}}]$ generates significantly better results at almost all interaction levels ψ than COA$[\tilde{w}_{\text{bl}}]$. While for the EVC instances the average optimality gaps w.r.t COA$[\tilde{w}_{\mathbf{\Theta}}]$ are lower than the

average optimality gaps w.r.t. COA[$\tilde{w}_{\text{bl}}$], there are instance groups for which no significant difference between the optimality gaps can be determined. However, there is no instance group for which COA[$\tilde{w}_{\text{bl}}$] produced significantly better results than COA[$\tilde{w}_{\boldsymbol{\Theta}}$] over all user interaction thresholds. It can be observed that at very low levels of user interaction COA[$\tilde{w}_{\boldsymbol{\Theta}}$] and COA[$\tilde{w}_{\text{bl}}$] seem to be equally strong. However, as the amount of collected of user feedback increases, COA[$\tilde{w}_{\boldsymbol{\Theta}}$] quite quickly outperforms COA[$\tilde{w}_{\text{bl}}$].

Table 4.4: Average optimality gaps obtained by COA using the surrogate suitability functions with ($\tilde{w}_{\boldsymbol{\Theta}}$) and without learning capabilities ($\tilde{w}_{\text{bl}}$) at different interaction levels.

	EVC											
(n,m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]
30%	4.31	**4.03**	-	-	**0.99**	0.99	-	-	**0.30**	0.32	**1.93**	1.94
40%	**1.71**	2.00	**4.42**	4.95	**0.57**	0.62	**1.84**	2.06	**0.23**	0.23	**1.17***	1.31
50%	**0.73***	1.11	**2.72**	3.03	**0.41**	0.46	**1.01***	1.44	**0.16**	0.16	**0.68***	0.86
60%	**0.31***	0.69	**1.27***	1.66	**0.23***	0.29	**0.58***	0.91	0.10	**0.09**	**0.36***	0.49
70%	**0.12***	0.42	**0.53***	0.90	**0.11**	0.14	**0.33***	0.50	**0.04***	0.06	**0.16***	0.30
80%	**0.11***	0.18	**0.22***	0.53	**0.04***	0.08	**0.11***	0.21	**0.02**	0.04	**0.05***	0.10
90%	**0.05**	0.11	**0.03***	0.13	**0.01**	0.03	**0.02***	0.06	**0.00***	0.01	**0.01***	0.02

	CSS											
(n,m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]
30%	8.02	**7.60**	20.92	**19.11***	**2.15**	2.31	-	-	**0.81**	0.82	-	-
40%	**2.98***	4.28	9.57	**9.52**	**1.21***	1.63	**2.81***	3.78	**0.51**	0.52	**2.36***	2.64
50%	**1.50***	2.41	4.72	**4.41**	**0.62***	0.99	**1.90***	2.77	**0.31***	0.38	**1.27***	1.80
60%	**0.63***	1.51	**2.29***	3.76	**0.36***	0.63	**1.20***	1.83	**0.20***	0.26	**0.72***	1.15
70%	**0.27***	0.51	**1.61***	2.36	**0.12***	0.37	**0.47***	1.09	**0.11***	0.15	**0.28***	0.59
80%	**0.18***	0.43	**0.92***	1.44	**0.05***	0.12	**0.18***	0.46	**0.04***	0.07	**0.15***	0.29
90%	**0.01***	0.14	**0.08***	0.65	**0.02***	0.06	**0.05***	0.15	**0.01**	0.02	**0.03***	0.07

	MAN					
b	30%		50%		70%	
ψ	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]	COA[$\tilde{w}_{\boldsymbol{\Theta}}$]	COA[$\tilde{w}_{\text{bl}}$]
30%	15.61	**14.93**	**7.02**	7.28	2.13	**2.13**
40%	**6.25***	7.53	**3.01***	3.46	**1.12**	1.18
50%	**3.34***	4.33	**1.63***	2.14	**0.67***	0.77
60%	**2.10***	2.80	**0.93***	1.32	**0.39***	0.48
70%	**1.20***	1.86	**0.46***	0.80	**0.22***	0.28
80%	**0.52***	1.04	**0.18***	0.33	**0.07***	0.09
90%	**0.24***	0.38	**0.04***	0.10	**0.01**	0.01

Figure 4.10 gives a visual comparison between COA[$\tilde{w}_{\boldsymbol{\Theta}}$] and COA[$\tilde{w}_{\text{bl}}$] for selected instance groups and not only shows average optimality gaps but also respective standard deviations around the mean values as shaded areas. The figure confirms that the average gaps produced by COA[$\tilde{w}_{\boldsymbol{\Theta}}$] are generally lower than those of COA[$\tilde{w}_{\text{bl}}$] but also shows that the standard deviations are quite large in general for both approaches. But as COA progresses and the quality of the solutions improves, the standard deviations decrease as well.

To further investigate the learning capabilities of the surrogate function, we now look at

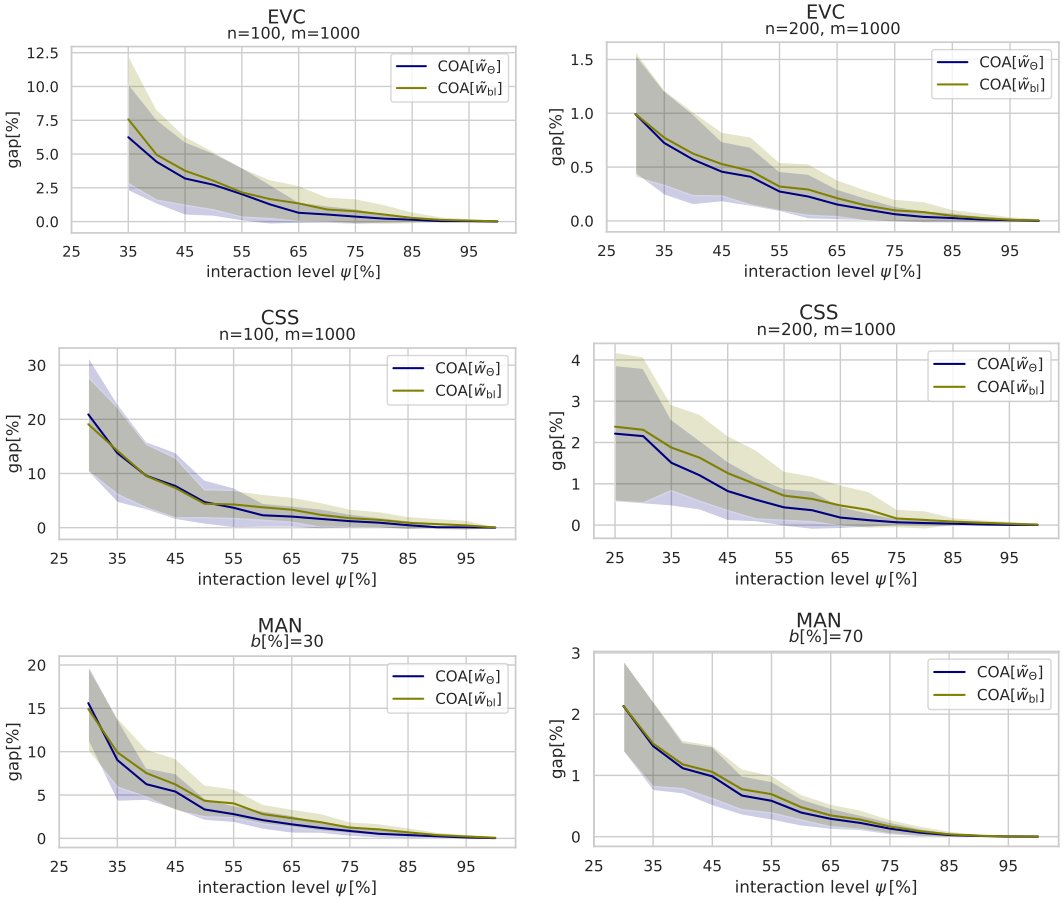


Figure 4.10: Average optimality gaps with standard deviations as shaded areas obtained by COA using the surrogate suitability functions with learning capabilities ($\tilde{w}_{\boldsymbol{\Theta}}$) and without ($\tilde{w}_{\text{bl}}$) plotted over the interaction level.

the MSE of $\tilde{w}_{\boldsymbol{\Theta}}$ in respect to the known exact values w. The left plots in Figure 4.11 show the development of this MSE calculated over all suitability values that are not known yet by COA for all instance groups. It can be seen that the MSE is generally small and approaches zero rather quickly. The reason for such small values can be found in the matrix factorization model used in the EC, which adds a bias for unknown suitability values towards zero as users typically only have a small number of locations with positive suitability values for each of their SPRs. Consequently, the MSE is distorted by the large number suitability values that are zero.

Therefore, the plots on the right side of Figure 4.11 show average mean squared errors calculated only over all *positive* suitability values that are not known yet; we denote this error by MSE+. This measure gives a clearer picture on how the surrogate function continuously improves in all cases with an increasing amount of gained knowledge. At

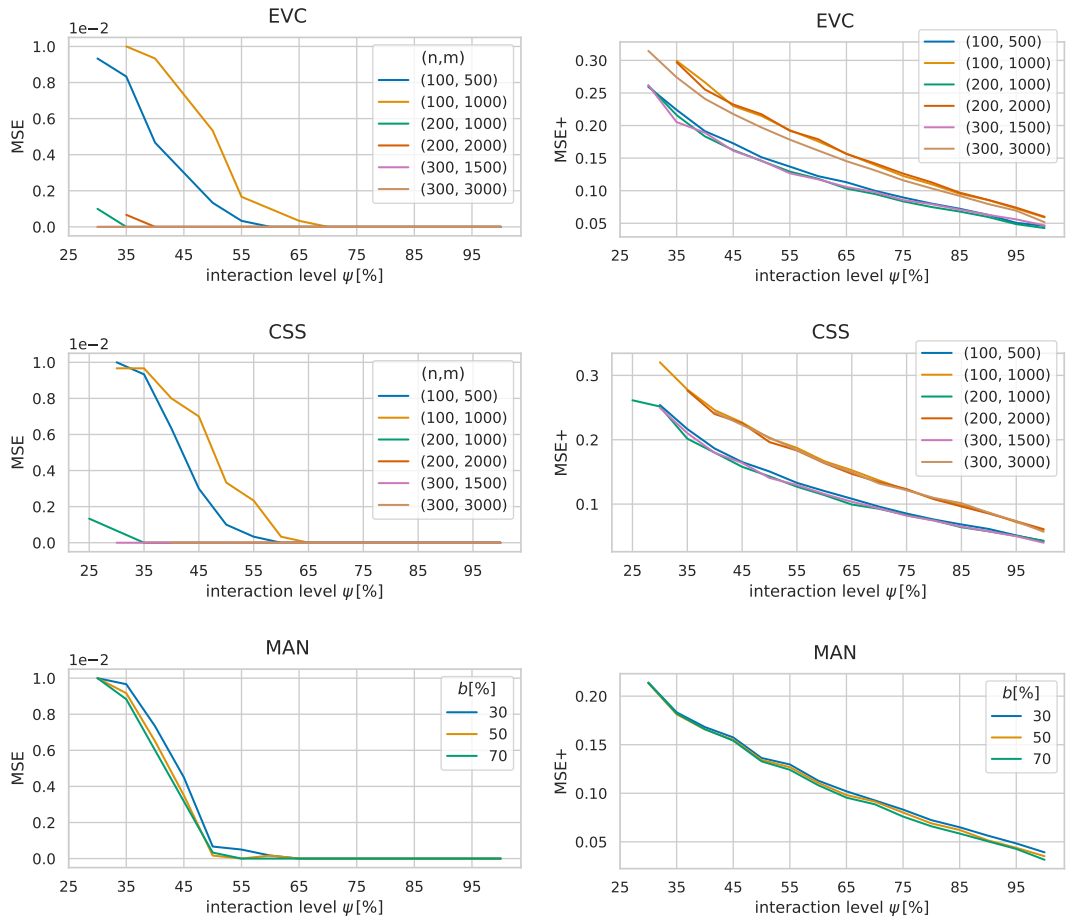


Figure 4.11: Development of average MSEs of $\tilde{w}_{\boldsymbol{\Theta}}$ over all so far unknown suitability values (left) and all so far unknown *positive* suitability values (right) for all benchmark sets.

the start of the algorithm the MSEs of the benchmark groups are between 0.2 and 0.3 on average and go towards zero almost linearly with the interaction level. Note that neither the size of an instance nor the given budget seem to have a significant impact on the size of the errors. Additionally, the figure also highlights how the instance parameters σ_v and σ_r impact the similarity of user preferences as the MSEs for instances with $\sigma_v = 5$ and $\sigma_r = 0.15$ are generally larger than the MSEs for instances with $\sigma_v = 3$ and $\sigma_r = 0.03$.

Finally, in Table 4.5 we evaluate how the predictions of the matrix factorization change due adding a bias for unknown ratings. For this purpose, let COA[$\tilde{w}_{\boldsymbol{\Theta}}$] refer to COA[GSPDP] using the matrix factorization model with $\alpha = 1$ and let COA[$\tilde{w}'_{\boldsymbol{\Theta}}$]refer to COA[GSPDP] using the matrix factorization model with $\alpha = 0$.

Each table cell shows the average optimality gaps of solutions to the respective instance

set at the specified interaction level ψ. The better results among COA[$\tilde{w}_{\mathbf{\Theta}}$] and COA[$\tilde{w}'_{\mathbf{\Theta}}$] are printed bold. A one-sided Wilcoxon signed-rank test was used to determine for each instance set whether the difference in optimality gaps is significant or not. Entries for which the Wilcoxon test has assessed at a 95% confidence level that either COA[$\tilde{w}_{\mathbf{\Theta}}$] or COA[$\tilde{w}'_{\mathbf{\Theta}}$] has produced better optimality gaps are marked with an asterisk. It can be observed that for lower levels of user interaction, solutions generated by COA[$\tilde{w}'_{\mathbf{\Theta}}$] exhibit extremely high optimality gaps. However, the higher the number of user interactions, the more COA[$\tilde{w}'_{\mathbf{\Theta}}$] can catch up with COA[$\tilde{w}_{\mathbf{\Theta}}$]. Most of the time, though, COA[$\tilde{w}_{\mathbf{\Theta}}$] still dominates COA[$\tilde{w}'_{\mathbf{\Theta}}$]. Summarizing, it can be concluded that the new matrix factorization model is a significant improvement over our previous model $\tilde{w}'_{\mathbf{\Theta}}$ resulting in COA[$\tilde{w}_{\mathbf{\Theta}}$] generating better solutions with fewer user interactions than COA[$\tilde{w}'_{\mathbf{\Theta}}$] most of the time.

Table 4.5: Average optimality gaps of solution from COA[$\tilde{w}_{\mathbf{\Theta}}$] and COA[$\tilde{w}'_{\mathbf{\Theta}}$], where the latter utilizes the former surrogate function $\tilde{w}'_{\mathbf{\Theta}}$ from [27]. COA[$\tilde{w}_{\mathbf{\Theta}}$] refers to COA[GSPDP] using the matrix factorization model with $\alpha = 1$ and COA[$\tilde{w}'_{\mathbf{\Theta}}$]refers to COA[GSPDP] using the matrix factorization model with $\alpha = 0$.

	EVC											
(n, m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]
30%	81.89	**4.31***	-	-	85.20	**0.99***	-	-	82.14	**0.30***	95.53	**1.93***
40%	16.07	**1.71***	34.82	**4.42***	3.22	**0.57**	32.79	**1.84***	9.08	**0.23**	23.21	**1.17***
50%	1.95	**0.73***	3.91	**2.72***	**0.35***	0.41	1.51	**1.01***	**0.13***	0.16	0.87	**0.68***
60%	0.62	**0.31***	2.20	**1.27***	0.24	**0.23**	0.76	**0.58***	**0.09**	0.10	0.48	**0.36***
70%	0.46	**0.12***	0.70	**0.52**	0.13	**0.11**	0.45	**0.33***	0.04	**0.04**	0.24	**0.16***
80%	0.22	**0.11***	0.31	**0.22**	0.06	**0.04***	0.17	**0.11***	**0.02**	0.02	0.11	**0.05***
90%	0.09	**0.05**	0.09	**0.03***	0.02	**0.01**	0.05	**0.02***	0.01	**0.00**	0.02	**0.01***

	CSS											
(n, m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]
30%	80.12	**8.02***	98.68	**20.92***	86.58	**2.15***	-	-	69.78	**0.81***	-	-
40%	13.80	**2.98***	24.73	**9.57***	1.38	**1.21**	13.93	**2.81***	3.84	**0.51**	6.17	**2.36***
50%	2.34	**1.50**	7.31	**4.72***	0.77	**0.62**	2.66	**1.90***	0.33	**0.31**	4.73	**1.27**
60%	1.83	**0.63***	2.53	**2.29**	0.49	**0.36***	1.56	**1.20***	0.22	**0.20**	0.83	**0.72**
70%	0.66	**0.27***	1.91	**1.61**	0.21	**0.12**	0.74	**0.47***	0.12	**0.11**	0.59	**0.28***
80%	**0.12**	0.18	0.99	**0.92**	0.10	**0.05***	0.42	**0.18***	**0.04**	0.04	0.23	**0.15***
90%	0.08	**0.01***	0.45	**0.08***	0.04	**0.02**	0.20	**0.05***	0.02	**0.01**	0.13	**0.03***

	MAN					
b	30%		50%		70%	
ψ	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]	COA[$\tilde{w}'_{\mathbf{\Theta}}$]	COA[$\tilde{w}_{\mathbf{\Theta}}$]
30%	99.78	**15.61***	99.83	**7.02***	99.83	**2.13***
40%	13.84	**6.25***	3.97	**3.01***	1.15	**1.12**
50%	8.46	**3.34***	1.93	**1.63***	**0.63**	0.67
60%	3.18	**2.10***	**0.89**	0.93	**0.35**	0.39
70%	1.86	**1.20***	0.64	**0.46***	0.26	**0.22**
80%	1.15	**0.53***	0.34	**0.18***	0.12	**0.07***
90%	0.47	**0.24***	0.14	**0.04***	0.03	**0.01***

Results for the LNS as optimization core

We compare our COA with the LNS, denoted in the following as COA[LNS], to the COA that uses the MILP (4.12)–(4.19) as optimization core and henceforth denoted as COA[MILP]. First, we provide some general information about the performance of COA[LNS]. Table 4.6 shows for each instance group at different interaction levels the average number of performed destroy and repair iterations n_{iter}, the average time in seconds required for finding the best solution $t^*[s]$, and the average total time in seconds until the LNS terminated $t[s]$. We can see that the LNS terminates within 43 to 80

Table 4.6: Results of COA[LNS].

	CSS																	
(n, m)	(100, 500)			(100, 1000)			(200, 1000)			(200, 2000)			(300, 1500)			(300, 3000)		
ψ	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$
40	50	0.21	0.87	62	0.96	2.08	60	0.21	0.61	76	1.37	2.31	59	0.35	0.66	75	1.32	1.99
50	51	0.27	1.09	67	1.18	2.82	67	0.48	1.05	68	1.03	2.42	65	0.50	0.94	71	1.32	2.34
60	46	0.22	1.17	58	1.09	3.09	58	0.41	1.17	59	1.01	2.74	66	0.61	1.19	65	1.47	2.96
70	47	0.25	1.39	53	0.79	3.09	50	0.30	1.27	58	1.18	3.07	64	0.56	1.22	64	1.45	3.27
80	45	0.18	1.51	48	0.44	2.78	45	0.16	1.16	50	0.59	2.80	56	0.47	1.33	59	1.14	2.98
90	43	0.10	1.48	44	0.25	2.64	45	0.17	1.19	49	0.60	2.73	46	0.22	1.17	44	0.43	2.51

	MAN								
b	30%			50%			70%		
ψ	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$
40	78	2.19	3.85	74	1.58	3.35	59	0.65	1.76
50	80	3.70	6.12	75	2.76	5.25	55	1.10	3.30
60	78	4.22	8.20	72	3.72	7.21	63	2.00	5.02
70	65	3.40	8.18	64	3.10	7.74	54	1.51	5.62
80	55	2.62	7.65	58	2.93	8.12	54	1.93	6.74
90	49	1.40	7.24	48	1.27	7.35	46	0.73	6.09

iterations on average and usually terminates within three seconds for the CSS instances and within eight seconds for the MAN instances. While the total number of iterations is relatively low, we later show in Table 4.7 that the solutions generated by the LNS are almost optimal w.r.t. the presented instances. The number of iterations performed tends to decrease as the number of performed user interactions increases while the total runtime increases in each iteration for the MAN instance but stays almost constant for the CSS instances. The decreasing number of iterations can be explained by the LNS being warm-started with the so far best found solution $\tilde{X}^*$. Moreover, as the number of user interactions increases, COA is able to identify more locations relevant to the SPRs of the use cases of the users, resulting in a higher number of arcs between the nodes in the service point layer and the nodes in the SPR layer of the respective evaluation graph. Therefore, the number of iterations until the LNS converges decreases while the time for performing one iteration increases.

Next, we investigate COA runs in which we apply in each iteration both, the LNS and the MILP, for solving the exact same GSPDP instances w.r.t. $\tilde{w}_{\boldsymbol{\Theta}}$ as well as the initial solution $\tilde{X}^*$. The MILP solver is able to find optimal solution in all cases, but at the expense of typically much longer running times. Note however that only the solution generated by the LNS is further used for the next iteration in COA. Table 4.7 shows the average percentage gaps between the objective values of the best solutions found by the LNS and respective optimal solutions w.r.t. $\tilde{f}$, denoted by $\text{gap}_{\tilde{f}}[\%]$, the average total running times in seconds of the LNS $t[s]$, the average times $t^{\circ}_{\text{M}}[s]$ needed by the MILP solver required for reaching a solution with at most the same objective value as the solution obtained by the LNS, as well as the average total times $t_{\text{M}}[s]$ in seconds of the MILP solver for determining a proven optimal solution. Bold values indicate best times w.r.t. t, t°_{M}, and t_{M}. First, we can see that the solutions generated by the LNS are on average only about 1% worse than an optimal solution for most instance groups. Next, the table shows that for CSS instances with a n/m ratio of 1/10, the MILP solver needs significantly more time for finding good solutions. Note that these instances have been designed in such a way that users behave less similar resulting in more complex instances. Nonetheless, the LNS significantly outperforms the MILP w.r.t. all instance groups. For all instance groups the LNS requires significantly less time on average to terminate than the MILP needs to reach a solution of the same quality as the solution obtained by the LNS. Additionally, Table 4.7 especially highlights how much more time the MILP requires for improving a solution at the same quality as the best found LNS solution to a provable optimal solution. Moreover, further tests have shown that most of the time the LNS is able to identify its best found solution while the MILP solver has still not yet solved the root relaxation in the same amount of time.

Finally, we want to compare independent COA[MILP] and COA[LNS] runs, and thus the impact of the in general slightly worse intermediate solutions of the LNS on the overall results of the two COA variants. For this purpose Table 4.8 shows for each interaction level the average optimality gaps between the best found solution during the optimization to an optimal solution w.r.t. the original objective f for COA[LNS] ($\text{gap}_{\text{L}}[\%]$) as well as COA[MILP] ($\text{gap}_{\text{M}}[\%]$). The table shows that small differences in the solution quality w.r.t. $\tilde{f}$ translate to slightly larger differences w.r.t. f. With the exception of the MAN instance group with $b[\%] = 30$, the solutions generated by COA[LNS] are usually at most 3% off from the values obtained by COA[MILP]. In most cases, the average differences are around 1% or less. Hence, in general it can be concluded that the LNS substantially outperforms the MILP in terms of computation time while still being able to generate almost optimal solutions.

Comparison between COA[ISPDP] and COA[GSPDP]

As previously mentioned, depending on the concrete CSPDP, the components of COA need to be implemented in a different way. For example, as the ISPDP does not provide any information about the use cases of the users, we are not able to apply the matrix factorization model used for deriving a surrogate function for the GSPDP. Hence, it is

Table 4.7: Average times required by the LNS, times the MILP solver needed to obtain a solution with at least the same quality as the solution of the LNS, as well as the total time required by the MILP to find a proven optimal solution. Additionally, the optimality gaps between the LNS solutions and respective optimal solutions are also shown.

	CSS											
	(100, 500)				(200, 1000)				(300, 1500)			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$
40	**0.87**	4.58	6.50	0.91	**0.61**	2.41	3.90	0.65	**0.66**	2.97	4.01	0.08
50	**1.09**	4.03	7.68	0.90	**1.05**	3.60	5.27	0.27	**0.94**	3.59	4.31	0.10
60	**1.17**	5.50	7.47	0.78	**1.17**	3.32	5.12	0.19	**1.19**	3.67	4.62	0.07
70	**1.39**	6.65	8.10	0.64	**1.27**	3.75	4.81	0.12	**1.22**	3.14	3.74	0.07
80	**1.51**	5.74	7.04	0.44	**1.16**	4.48	5.97	0.08	**1.33**	3.28	4.40	0.04
90	**1.48**	5.48	6.73	0.33	**1.19**	4.11	5.06	0.06	**1.17**	4.58	5.30	0.03

	CSS											
	(100, 1000)				(200, 2000)				(300, 3000)			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$
40	**2.08**	21.87	37.42	2.15	**2.31**	32.79	92.15	1.24	**1.99**	26.04	85.61	0.81
50	**2.82**	28.03	50.51	1.97	**2.42**	37.61	90.11	1.07	**2.34**	39.84	101.52	0.57
60	**3.09**	35.60	59.04	1.45	**2.74**	36.47	126.67	0.89	**2.96**	38.34	130.05	0.47
70	**3.09**	42.95	67.34	1.74	**3.07**	40.48	111.96	0.84	**3.27**	43.41	136.93	0.36
80	**2.78**	43.57	69.94	1.83	**2.80**	40.98	120.07	0.90	**2.98**	43.78	137.76	0.37
90	**2.64**	40.09	74.98	1.37	**2.73**	41.33	123.32	0.78	**2.51**	63.56	149.78	0.37

	MAN											
	30%				50%				70%			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}}[\%]$
40	**3.85**	67.21	326.46	2.15	**3.35**	17.24	53.54	0.87	**1.76**	6.36	10.71	0.21
50	**6.12**	80.31	328.53	1.36	**5.25**	16.76	95.43	0.59	**3.30**	10.20	15.29	0.11
60	**8.20**	131.28	368.28	1.19	**7.21**	24.36	89.15	0.43	**5.02**	14.59	21.54	0.07
70	**8.18**	140.34	375.46	1.06	**7.74**	24.86	108.59	0.35	**5.62**	13.22	21.73	0.06
80	**7.65**	160.13	414.39	1.12	**8.12**	27.70	108.01	0.34	**6.74**	18.00	24.43	0.05
90	**7.24**	154.44	411.55	1.29	**7.35**	43.43	102.70	0.27	**6.09**	13.03	17.46	0.03

not possible to solve ISPDP instances with the COA[GSPDP] framework. However, it is possible to solve GSPDP instances with the COA[ISPDP] framework. Therefore, in this section, using the CSS instance set, we compare the performance of COA[GSPDP] to COA[ISPDP]. However, as COA[ISPDP] is not as efficient as COA[GPSPD] w.r.t. computation times, we use a different benchmark set with a only 100 potential service point locations. Specifically, we consider benchmark scenarios with $n = 100$ locations and $m \in \{500, 1000, 1500\}$ users. For each combination we derive three groups of 30 independent instances with different parameters $n_\alpha \in \{10, 17, 25\}$, $\sigma_{\mathrm{v}} \in \{5, 7, 10\}$, and

Table 4.8: Quality of solutions generated by COA[LNS] and COA[MILP].

	CSS											
(n,m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]
40	3.46	**2.98**	11.92	**9.57**	1.64	**1.21**	4.34	**2.81**	0.54	**0.51**	2.81	**2.36**
50	2.06	**1.50**	7.34	**4.72**	0.81	**0.62**	2.86	**1.90**	0.43	**0.31**	1.87	**1.27**
60	1.62	**0.63**	4.31	**2.29**	0.45	**0.36**	2.21	**1.20**	0.30	**0.20**	1.31	**0.72**
70	1.20	**0.27**	4.11	**1.61**	0.22	**0.12**	1.56	**0.47**	0.19	**0.11**	0.81	**0.28**
80	0.66	**0.18**	2.72	**0.92**	0.15	**0.05**	1.30	**0.18**	0.09	**0.04**	0.58	**0.15**
90	0.43	**0.01**	1.95	**0.08**	0.08	**0.02**	0.95	**0.05**	0.06	**0.01**	0.44	**0.03**

	MAN					
b	30%		50%		70%	
ψ	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]	gap_L[%]	gap_M[%]
40	8.61	**3.46**	3.58	**3.46**	**1.32**	3.46
50	5.14	**1.88**	2.19	**1.88**	**0.77**	1.88
60	3.32	**1.14**	1.41	**1.14**	**0.46**	1.14
70	2.53	**0.63**	0.86	**0.63**	**0.25**	0.63
80	2.03	**0.26**	0.54	**0.26**	**0.12**	0.26
90	1.77	**0.10**	0.33	**0.10**	**0.05**	0.10

$\sigma_\text{r} \in \{0.03, 0.1, 0.15\}$.

In the following experiments, for COA[ISPDP] the VNS was used as optimization core. For COA[GSPDP] the number of features of the matrix factorization was set in accordance to the number of attraction points n_α of the test instances. Moreover, for the matrix factorization model, the parameter α is set to 0. Further, both COA implementations were terminated after five major iterations or when a CPU-time limit of 7200s has been reached and returned as the overall best solution $\tilde{X}^*$, i.e., the solution with the highest surrogate objective value at the end.

Table 4.9 shows the obtained results. Each line lists, for COA[GSPDP] as well as COA[ISPDP], the average number of iterations n_it, the average optimality gap "gap[%]" between the objective value of $\tilde{x}^*$ and the optimal solution, the average percentage error of the surrogate function values of the final solutions %-$\Delta\tilde{f}$, with %-$\Delta\tilde{f} = 100\% \cdot |\tilde{f}(\tilde{x}^*) - f(\tilde{X}^*)|/f(\tilde{X}^*)$, the average ratio of locations the users had to rate during the course of the algorithm per use case and their relevant locations per use case ρ, and the median computation times in seconds $t[s]$.

The results clearly show that COA[GSPDP] is able to converge to very reasonable solutions with small remaining optimality gaps of typically less than 2.3% within only five major iterations. For %-$\Delta\tilde{f}$, we can observe that the percentage errors decrease as the number of users increases. This is especially evident for the hardest instance groups C, F, and I where %-$\Delta\tilde{f}$ decreases from 8.17% to 4.42% on average. This documents that, given a sufficient amount of users, the surrogate function is able to approximate the

real objective function at the end well in the relevant parts w.r.t. the returned solution. The table also shows that not all runs have been completed with five iterations, i.e., COA[GSPDP] was aborted due to the time limit for 9 instances from the instance groups H and I. Column ρ of COA[GSPDP] also shows that in general users do not need to rate more locations than their total number of relevant locations for each of their use cases.

COA[ISPDP] is significantly outperformed by COA[GSPDP] in all aspects. COA[GSPDP] is able to generate better solutions in less time for all instance groups. In many cases COA[ISPDP] exceeded the time limit of 7200s already in the first or second iteration which explains the large difference in performance between COA[GSPDP] and COA[ISPDP]. It is not quite easy to compare ρ between COA[GSPDP] and COA[ISPDP] since COA[ISPDP] was not able to perform as many iterations as COA[GSPDP]. However, in general we can observe that users are required to evaluate significantly more locations with COA[ISPDP] than with COA[GSPDP].

Table 4.9: Average results of COA[GSPDP]and COA[ISPDP].

						COA[GSPDP]					COA[ISPDP]				
Inst.	m	n_α	σ_v	σ_r	ϕ	n_{it}	gap[%]	%-$\Delta\tilde{f}$	ρ	t[s]	n_{it}	gap[%]	%-$\Delta\tilde{f}$	ρ	t[s]
A	500	10	5	0.03	10	5.00	**0.35**	**2.28**	0.86	**751**	1.97	16.40	28.07	0.82	7172
B	500	17	7	0.10	17	5.00	**1.18**	**5.19**	0.88	**888**	2.43	18.37	21.44	1.24	7168
C	500	25	10	0.15	25	5.00	**2.23**	**8.17**	0.84	**1033**	2.07	14.61	26.54	0.89	7190
D	1000	10	5	0.03	10	5.00	**0.39**	**1.94**	0.84	**1540**	2.90	16.93	22.63	1.53	7180
E	1000	17	7	0.10	17	5.00	**1.61**	**4.73**	0.83	**2407**	2.30	13.34	21.91	1.07	7181
F	1000	25	10	0.15	25	5.00	**1.52**	**5.72**	0.86	**3383**	2.53	16.98	20.86	1.32	7191
G	1500	10	5	0.03	10	5.00	**0.26**	**1.73**	0.85	**2579**	2.83	14.78	14.81	1.50	7189
H	1500	17	7	0.10	17	4.90	**1.18**	**3.81**	0.82	**4478**	1.77	17.78	28.88	0.65	7179
I	1500	25	10	0.15	25	4.73	**1.63**	**4.42**	0.80	**5605**	1.97	18.08	26.13	0.83	7189

4.7 Conclusion and Future Work

In this chapter we proposed a COA framework for distributing service points within a geographical area in mobility applications under incomplete information. Instead of estimating user demands by combining a variety of more or less reliable sources, our method directly incorporates potential customers in the optimization process. COA offers an attractive, alternative demand acquisition method, as it does not require any previous data, is cheap to implement and can easily be transferred to other application scenarios and locations.

We have evaluated two different implementations of COA for two concrete cooperative service point distribution problems. For the first problem, the ISPDP, we refer to the corresponding COA implementation as COA[ISPDP]. We could show for COA[ISPDP] that the machine learning models in our EC are able to learn the non-trivial user behavior of all our benchmark scenarios reliably after relatively few user interactions, and the optimization is able to indeed find solutions with only small remaining optimality

gaps. The careful derivation of the candidate solutions to be presented to the users in the feedback component also plays a particularly important role. However, a major disadvantage of COA[ISPDP] is the bad scalability of the approach towards more potential locations and more users due to the large number of machine learning models that need to be solved in each iteration.

This issue was addressed in the second COA implementation for solving the GSPDP. We refer to the respective COA implementation as COA[GSPDP]. COA[GSPDP] uses a matrix factorization model as new surrogate function in the EC. More specifically, we made use of an advanced matrix factorization model which takes into account that user data is not missing at random. Moreover, we also abandoned the previous black box optimization model of the OC and use a MILP and an LNS instead. With experiments on artificial instances as well as instances derived from real-world data, we could clearly observe that the matrix factorization based surrogate model is able to learn preferences of individual users from users with similar interests. Additionally, using the advanced matrix factorization model yielded a significant improvement in the quality of the solutions. The new surrogate function as well as the new optimization core of COA[GSPDP] resulted in a major speedup and improvement in the scalability compared to COA[ISPDP]. Moreover, COA[GSPDP] also requires a significantly lower number of user interactions than COA[ISPDP].

Additionally, we also presented an LNS to be used as optimization core for COA[GSPDP]. While the LNS follows the traditional destroy and repair principle, a major challenge was to effectively guide the repair heuristic to produce promising new solutions and to efficiently calculate the surrogate objective function for modified solutions in an incremental way. Both was achieved by introducing the evaluation graph, which stores relevant intermediate results allowing efficient updates when stations are added to or removed from the current solution. In particular, the evaluation graph provides an effective way to keep track of how much impact each location in the solution has on its respective objective value. The efficient update possibility also allows to consider a larger amount of locations during the destroy procedure. The performance of the LNS within COA was tested on artificial instances as well as instances derived from real-world data and was compared to the original COA with its MILP-based optimization core. Results show that at the cost of a slight deterioration of usually not more than one percent in the quality of the solutions, the LNS can outperform the MILP w.r.t. to computation times by orders of magnitudes.

There is still potential left for future improvements. In both of our COA implementations, the strategies by which scenarios for users are generated favor the selection of unrated locations that may be important for individual users but not necessarily for a global optimal solution. In order to quickly find a good solution by the optimization it is important for our surrogate function to have higher accuracy for locations that have the potential to actually appear in a globally optimal solution. Otherwise, finding a near optimal solution requires a larger amount of user interactions as we have observed in our results on the Manhattan instances. In order to improve the scenario generation

strategies, it seems natural to enrich the FC with knowledge not only from the EC but also from the OC. As the OC finds optimized solutions via a MILP, utilizing dual solution information such as reduced costs or performing a sensitivity analysis might be a promising direction.

It would also be interesting to further improve the scalability of COA, e.g., by using hierarchical clustering and multilevel refinement strategies as applied in the context of planning a bike sharing system in [6].

Finally, we also want to emphasize that the focus was on the algorithmic and computational aspects of COA and its components. Clearly, further challenges concern a suitable user interface and a corresponding distributed implementation of at least the feedback component, in which also psychological aspects of users need to be considered. Moreover, the performed experiments are based on the assumption of perfect user feedback, which does not hold in practice. The impacts of not entirely reliable evaluation results need to be studied, and robust variants of certain components of COA devised.

CHAPTER 5

Smart Charging of Electric Vehicles Considering SOC-Dependent Maximum Charging Powers

While the main topic of this thesis is the installation of service points for mobility applications, there are also other challenges that need to be overcome for establishing a mobility service. One such challenge is to successfully run a mobility service in the sense of using one's resources as efficiently as possible while maximizing the system's utilization. For example, charging stations are not only limited by the number of vehicles they can serve at the same time but also by the maximal power they can provide to the vehicles for charging. Additionally, vehicles are only temporarily available at a station and electricity costs frequently change during a day. Therefore, a proper charging strategy is a crucial requirement for running a charging station. Hence, this chapter is dedicated to the problem of scheduling the charging of electric vehicles at a single charging station such that the temporal availability of each EV as well as the maximum available power at the station are considered. The total costs for charging the vehicles should be minimized w.r.t. time-dependent electricity costs. An additional challenge we investigate in this context is that the maximum power at which a vehicle can be charged is dependent on the current state of charge (SOC) of the vehicle. Such a consideration is particularly relevant in the case of fast charging.

Considering an SOC-dependent maximum charging power for a discretized time horizon is not trivial as the maximum charging power of an EV may change during time steps. To deal with this issue, we instead consider the maximum energy by which an EV can be charged within a time step. For this purpose, we show how to derive the maximum

charging energy in an exact as well as an approximate way.

Two methods for solving the scheduling problem are proposed. The first one is a cutting plane method utilizing a convex hull of the in general nonconcave SOC-power curves. The second method is based on a piecewise linearization of the SOC-energy curve and is effectively solved by branch-and-cut. The proposed approaches are evaluated on benchmark instances, which are partly based on real-world data. To deal with EVs arriving at different times as well as charging costs changing over time, a model based predictive control strategy is usually applied in such cases. Hence, we also experimentally evaluate the performance of our approaches in such a context. The results show that optimally solving problems with general piecewise linear maximum power functions requires high computation times. However, problems with concave, piecewise linear maximum charging power functions can efficiently be dealt with by means of linear programming. Approximating an EV's maximum charging power with a concave function may result in practically infeasible solutions, due to vehicles potentially not reaching their specified target SOC. However, our results show that this error is negligible in practice.

The approaches have been published in:

> B. Schaden, T. Jatschka, S. Limmer, and G. R. Raidl, "Smart charging of electric vehicles considering SOC-dependent maximum charging powers," *Energies*, vol. 14, no. 22, 2021

which is based on the master thesis of Benjamin Schaden:

> B. Schaden, "Scheduling the charging of electric vehicles with soc-dependent maximum charging power," Master's thesis, TU Wien, 2021. Supervised by G. R. Raidl and T. Jatschka.

5.1 Introduction

The number of EV is rapidly increasing. At the end of 2020, there were around 10 million EVs on the world's roads and the number of EV registrations increased by 41% in 2020 [122]. The uncontrolled charging of this rising number of EVs, together with an increasing share of renewable energy, imposes significant challenges for the stable operation of the power grid in terms of power quality, voltage stability, peak demand, and reliability [123]. Besides further measures, like time-of-use prices [124] or dynamic pricing schemes [125], smart charging [126, 127] is considered a promising strategy to mitigate these issues. Smart charging refers to the coordination of the charging of a number of EVs in an intelligent way. Numerous approaches for smart charging, considering different objectives and different constraints, are proposed in the literature [128, 129, 130, 131, 132, 133, 134, 135].

These approaches typically assume that the maximum charging power of an EV remains constant over the planning horizon. However, in practice the maximum charging power

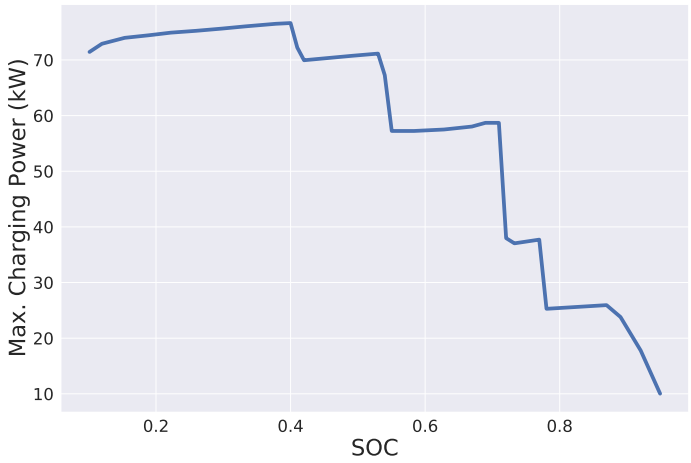


Figure 5.1: Maximum charging power of a Hyundai Kona Elektro in dependence of the EV's SOC; data obtained from Fastned [136].

depends on the SOC of the EV's battery. Typically, with an increasing SOC, the maximum charging power is regulated down by the battery controller. For slow AC charging, the decrease of the maximum power is usually only marginal and can be neglected for most applications. For modern fast DC charging, however, the effect of the decreasing maximum power can be substantial as can be seen from the exemplary SOC-power curve shown in Figure 5.1. The exact form of the curve does not only depend on the type of battery and its charging controller but also on other factors like the ambient temperature or the state of health of the battery [22]. In most cases the curve is highly nonlinear, making it difficult to consider it in MILP approaches, which are frequently used for charging planning. However, not considering the SOC-dependent maximum charging power in the charging planning is likely to result in suboptimal or even infeasible charging schedules, especially in the case of fast charging. For example, Frendo et al. [23] conclude from numerical experiments that under the constraint of a limited total charging power, up to 21% more energy can be charged if the SOC-dependent maximum charging power is considered in the planning, compared to not considering it. Frendo et al. also point out that in the literature on smart charging, the integration of nonlinear SOC-power curves is frequently mentioned as future work. However, to date the number of works, which actually address this issue, is still strongly limited.

We assume a basic use case of smart charging with the objective of minimizing the energy cost under time-varying electricity prices and with the constraint of a limited total charging power per time step. In order to allow a better integration of nonlinear SOC-power curves, we formulate the scheduling problem in terms of planning the charging energy instead of the charging power. Therefore, we consider two approaches for converting the SOC-power curves to SOC-energy curves. The first approach is an exact approach, but it can only guarantee that the *average* total charging power does not exceed the limit in a time step. The second approach is an approximate approach, which guarantees that

the total charging power never exceeds the limit, but it might lead to suboptimal costs.

We propose two methods for solving the resulting problems. The first one is an extension of a cutting plane method proposed by Korolko and Sahinoglu [20] and utilizes a convex hull of the in general nonconcave SOC-power curves. The second method makes use of a piecewise linearization of the SOC-energy curve and is accelerated by branch-and-cut. In extensive numerical experiments, we evaluate and compare the proposed approaches. The key contributions of this chapter are

- a reformulation of the scheduling problem in terms of the control of charging energy, which facilitates the integration of SOC-dependent maximum charging power,
- a proposal of two transformations of SOC-power curves into SOC-energy curves,
- and a proposal and evaluation of two mixed integer linear programming based solution methods that consider SOC-dependent maximum charging powers.

The rest of the chapter is organized as follows. The next section discusses related work. In Section 5.3 our EV charging scheduling problem is formalized. Additionally, it is shown how to derive the exact as well as an approximate maximum charging energy function from the maximum charging power function. Next, Section 5.4 presents the different problem solving approaches. Section 5.5 explains how we generated problem instances for the empirical evaluation, and respective experimental results are presented in Section 5.6. Finally, Section 5.7 concludes this chapter and outlines promising future research directions.

5.2 Related Work

Some works consider a SOC-dependent maximum charging power by integrating nonlinear physical battery models in the charging schedule optimization. Sundström and Binding [137] compare the use of a linear and a quadratic approximation of such a model in the optimization of EV schedules with the goal of minimizing charging costs. They conclude that although the linear approximation results in small violations in SOCs requested by the EV drivers, the benefit of the quadratic approximation does not justify the increase in computation time. Morstyn et al. [138] propose a nonlinear battery circuit model and integrate it in an optimization model in form of a second-order cone program. They consider the maximization of charged energy taking into account network constraints and the constraints of a limited total charging power. It is shown that problem instances with up to 500 vehicles can be solved within less than 100 seconds. In practice, the behavior of the battery (controller) can significantly differ from an idealized battery model. Thus, other works – including the present work – abstract from a specific battery model.

Different battery model-free heuristic approaches for smart charging with SOC-dependent maximum power can be found in the literature. Cao et al. [139] propose a rule-based approach for EV charging control with the objectives of energy cost reduction and load

flattening, respecting the SOC-dependent maximum charging powers of EVs. Frendo et al. [23] describe the use of a data-driven approach for the prediction of power curves of EVs. The authors propose a rule-based control, which schedules the charging of the EVs with the objective of a fair distribution of the available energy taking into account the predicted power curves.

El-Bayeh et al. [140] propose a model-free exact approach. They approximate a nonlinear power curve with a piecewise linear function. Subsequently, they draw a comparison between the charging costs resulting from charging with a constant maximum charging power and the charging costs resulting from charging with a vehicle specific SOC-dependent piecewise linear function. For solving the optimization problem, they use mixed integer nonlinear programming, which distinguishes their approach from our problem solving techniques. Han, Park, and Lee [21] consider a problem setting similar to that considered in this chapter. The authors assume that the charging station has limited grid capacity, which may be exceeded at the price of paying penalty costs. They present a MILP formulation of the problem, which integrates nonlinear power curves with help of a discretization of SOC levels. In contrast to the present work, it is assumed that EVs can only charge with maximum or zero power, which is quite restrictive and hardly the case in practice. Two network flow approaches in Schaden's Master thesis [31] extend the MILP formulation from [21] with the possibility to charge with power levels from a discrete set of values. However, we refrain from considering these approaches here as they have been found to be uncompetitive, primarily due to the much larger memory requirements even when the number of EVs is low.

A further model-free exact approach is proposed by Korolko and Sahinoglu [20]. They assume a problem setting similar to that considered in [21] but with continuous charging power values. A nonlinear problem formulation is presented and is solved as a series of linear problems with the help of a cutting plane approach. The described approach, however, requires the power curve to be concave. Our approaches partly build upon this work.

The approaches proposed in this chapter, are model-free linear exact approaches for a continuous power modulation, which are applicable to concave and nonconcave power curves. None of the previous works considers the issue that the variable maximum charging power varies within a time step of the planning horizon. To the best of our knowledge, we are the first considering this aspect in more detail.

5.3 Problem Description

The EV charging scheduling problem with SOC-dependent maximum charging power (EVS-SOC) we consider formalizes the task of scheduling the charging of a number of EVs at a single charging station such that the total charging costs are minimized. The charging schedule is preemptive, which means that the charging process of an EV may be interrupted an arbitrary number of times. It is assumed that electricity costs change over time and that they are known in advance. Discrete finite time steps $T = \{0, \dots, t_{\max}\}$

are used to model the considered time horizon. Each of these represents a time interval of constant duration Δt.

The charging is controlled by a single central entity, the so-called aggregator. The total power that can be used from the grid at any time is limited by $P^{\text{gridmax}} > 0$. Electricity costs per unit of consumed energy are given by c_t individually for each time step $t \in T$. Note that these costs may also be negative in practice.

The set of EVs to be considered is $V = \{1, \dots, n\}$, and they are all assumed to be currently connected to the charging station, i.e., immediately available for charging. Each vehicle is associated with an initial state of charge $s_{v,0} \in [0, 1]$, i.e., the SOC at the beginning of time step zero, and a minimum required state $s_v^{\text{dep}} \in [s_{v,0}, 1]$ that must be reached at the vehicle's known departure time $t_v^{\text{dep}} \in T$. Additionally, for each vehicle $v \in V$ the energy capacity $C_v > 0$ of its battery is known as well as a function $P_v^{\max} : [0, 1] \mapsto \mathbb{R}^+$ for the battery's maximum charging power given its SOC. Note that $P_v^{\max}$ must be strictly positive for any SOC less than one and is zero for SOC one. Otherwise we do not restrict this function in any way, in particular it does not necessarily have to be concave or continuous. Note that we neglect the effect of minor further factors like the battery temperature and its state of health on the maximum charging power. Furthermore, we assume a charging efficiency of 100%.

We remark that in practice, the domain of $P_v^{\max}$ is often not defined on the entire SOC interval $[0, 1]$ but just for some restricted $[s_v^{\min}, s_v^{\max}]$, $0 \leq s_v^{\min} < s_v^{\max} \leq 1$. In the following, we will regard this issue as an implementation detail and assume the domain of $P_v^{\max}$ to be $[0, 1]$.

The goal of EVS-SOC is to find a feasible charging schedule that minimizes the total charging costs while charging each vehicle v from SOC $s_{v,0}$ to (at least) SOC s_v^{dep} by time step t_v^{dep} such that the total power used from the grid at any time does not exceed $P^{\text{gridmax}} > 0$.

Since the maximum charging power function $P_v^{\max}$ depends on the SOC, it is in general not constant within a single time step of duration Δt. This may lead to the problem that a charging power value set for a time step is not allowed throughout the whole charging interval. The vehicle's charging controller will then dynamically adjust (reduce) the actually used power to never exceed the SOC-dependent maximum power. One may argue that the resulting error may be reduced by increasing the resolution of the time discretization until it becomes negligible. A larger number of time steps, however, directly affects the problem size and practical solvability. Therefore, we refrain here from decreasing Δt only because of this reason.

Instead, we turn from considering the charging power to considering the energy by which an EV may actually be charged in a time step, taking care of the above aspects. We propose alternative approaches for deducing an (approximate) maximum energy function $E_v^{\max}(s) : [0, 1] \mapsto \mathbb{R}^+$ from $P_v^{\max}$ that states the maximum energy by which EV v with SOC s can be charged within duration Δt.

In Section 5.3.1 we give an exact way for deducing $E_v^{\max}$, referred to as $E_v^{\text{max-ex}}$. However, using $E_v^{\text{max-ex}}$, we are in general only able to express that the maximum grid power is not exceeded *on average* within a time step, since we consider the time horizon in a discretized fashion. While this might be sufficient for some applications, like limiting peak load charges, it may be a too weak condition for other applications, like limiting transformer loads. Therefore, in Section 5.3.2 we also show how to deduce a lower bound $E_v^{\text{max-lb}}$ to $E_v^{\max}$ that never overestimates the real maximum power at which charging can take place.

5.3.1 Exact Maximum Energy

We determine the maximum charging energy $E_v^{\text{max-ex}}$ that is achieved when applying the dynamic charging power $P_v^{\max}$ throughout a whole time step. Considering an EV $v \in V$ with initial SOC $s_{v,t} \in [0,1]$ at some time step $t \in \{0,\dots,t_v^{\text{dep}}-1\}$, the time needed to charge the EV to some SOC $s' \in [s_{v,t},1]$ using the dynamic maximum charging power is

$$T_v^{\text{min-ex}}(s_{v,t},s') = C_v \cdot \int_{s_{v,t}}^{s'} \frac{1}{P_v^{\max}(s)}\,ds. \tag{5.1}$$

The maximum energy by which the EV can be charged during a time step of duration Δt is then

$$E_v^{\text{max-ex}}(s_{v,t}) = C_v \cdot (s'-s_{v,t}) \text{ s.t. } \begin{cases} T_v^{\text{min-ex}}(s_{v,t},s') = \Delta t & \text{for} \quad T_v^{\text{min-ex}}(s_{v,t},1) > \Delta t \\ s' = 1 & \text{else.} \end{cases} \tag{5.2}$$

Hereby we consider in the else case that charging always stops when SOC value one is reached. While calculating the integral for $\frac{1}{P_v^{\max}(s)}$ might be nontrivial from a theoretical point-of-view for some power functions, it is in practice not difficult to efficiently determine approximate values for $E_v^{\text{max-ex}}(s_{v,t})$ computationally by conventional numerical integration methods. As previously mentioned, the problem with the usage of $E_v^{\text{max-ex}}(s_{v,t})$ is primarily that it is hard to express the maximum grid power constraint since within a time step the actually used power may vary for each EV substantially, i.e., we will only be able to express that the maximum grid power is not exceeded *on average* within a time step.

5.3.2 Lower Bound for Maximum Energy

To address the aforementioned problem, we consider the largest power that can be constantly applied throughout a whole time step of duration Δt without requiring the charging controller to reduce the power. The time needed to charge the EV to some SOC $s' \in [s_{v,t},1]$ using the maximum power that can be constantly applied is

$$T_v^{\text{min-lb}}(s_{v,t},s') = \frac{C_v \cdot (s'-s_{v,t})}{\min_{s\in[s_{v,t},s']} P_v^{\max}(s)}. \tag{5.3}$$

The maximum energy by which the EV can be charged during a time step of duration Δt is then again obtained by Eq. (5.2) but in conjunction with the above $T_v^{\text{min-lb}}$ (5.3) instead of $T_v^{\text{min-ex}}$ (5.1). We refer to this variant by $E_v^{\text{max-lb}}$.

By avoiding to set for a time step a power that will have to be reduced by the charging controller at some point of time, the maximum energy $E_v^{\text{max-lb}}$ is a lower bound for the actually obtainable energy $E_v^{\text{max-ex}}$. Using $E_v^{\text{max-lb}}$ in our whole problem setting means that an obtained solution will guarantee that indeed all EVs are charged to the desired departure SOCs. As we may occasionally use a more restricted charging power than could actually be applied, the schedule might not be optimal in the original sense, and a solution's objective value will be an upper bound for the real optimum.

We want to point out the following relationships between $P_v^{\max}$ and its corresponding maximum energy functions.

- If $P_v^{\max}$ is a piecewise linear function, then $E_v^{\text{max-lb}}$ is piecewise linear as well. On the contrary, $E_v^{\text{max-ex}}$ might not be a piecewise linear function, even if $P_v^{\max}$ is piecewise linear.
- If $P_v^{\max}$ is a concave function, so are $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$.

To give the reader an impression how $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ relate to each other, Figure 5.2 shows these functions for different Δt values for a Hyundai Kona Elektro. Note that the area between $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ decreases with smaller Δt values. Hence, as we will also see in Section 5.6, the smaller Δt is chosen, the smaller will be the size of the error introduced by $E_v^{\text{max-lb}}$ in general.

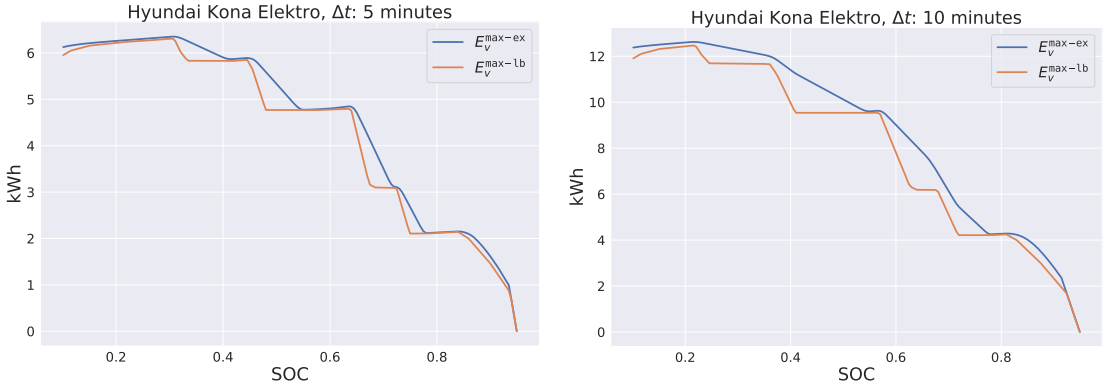


Figure 5.2: $E_v^{\max}$ functions for a Hyundai Kona Elektro for $\Delta t \in \{5, 10\}$ minutes.

In the following sections we will pursue $E_v^{\text{max-ex}}$ and $E_v^{\text{max-lb}}$ and investigate the pros and cons of each in comparison. We will use the notation $E_v^{\max}$ as a placeholder for any specific energy function from $\{E_v^{\text{max-ex}}, E_v^{\text{max-lb}}\}$.

5.3.3 Converting Energy Back to Power

In practice, the charging aggregator usually regulates the maximum charging *power* instead of the maximum charging *energy*. Consequently, when scheduling with energy values we have to convert back energy values to power values. For schedules created with $E_v^{\text{max-lb}}$, the computed energy values of a schedule can be simply divided by Δt to obtain charging power values that can be constantly applied throughout a single time step.

For schedules created with the exact $E_v^{\text{max-ex}}$, due to the possible interference of the EV's charging controller it is in general not obvious which power value $P_{v,t}$ should be provided to the charging aggregator in order to actually charge a certain amount of energy $x_{v,t}$ in a next time step t. Considering $P_v^{\max}(s)$, this value $P_{v,t}$ can be determined computationally by numerically solving the equation

$$C_v \cdot \int_{s_{v,t}}^{s_{v,t}+x_{v,t}/C_v} \frac{1}{\min(P_v^{\max}(s), P_{v,t})}\, ds = \Delta t, \tag{5.4}$$

where the left side corresponds to the time needed for charging $x_{v,t}$ when applying as power always the minimum of $P_v^{\max}(s)$ and $P_{v,t}$. Still there remains the issue that in a solution to our scheduling problem $\sum_{v\in V} P_{v,t} \leq P^{\text{gridmax}}$ is not guaranteed anymore and either P^{gridmax} may be exceeded or some $P_{v,t}$ needs to be reduced to avoid this problem. Note that Equation (5.4) is well defined for all $x_{v,t} \in [0, C_v(s' - s_{v,t})]$ where s' is determined according to Equation (5.2).

Therefore, schedules created with $E_v^{\text{max-ex}}$ mainly serve here as comparison for schedules created with $E_v^{\text{max-lb}}$ to give an idea about the size of the error introduced by time discretization.

5.3.4 Nonlinear Model

We now formally define EVS-SOC by the following nonlinear program, where variables $x_{v,t}$ represent the energy by which EV $v \in V$ is charged in time step $t = 0, \ldots, t_v^{\text{dep}} - 1$. Variables $s_{v,t}$ indicate the SOC of each EV $v \in V$ at the beginning of each time step $t = 0, \ldots, t_v^{\text{dep}}$.

$$\min \quad \sum_{v\in V} \sum_{t=0}^{t_v^{\text{dep}}-1} c_t \cdot x_{v,t} \tag{5.5}$$

$$x_{v,t} \leq E_v^{\max}(s_{v,t}) \qquad v \in V,\ t = 0, \ldots, t_v^{\text{dep}} - 1 \tag{5.6}$$

$$\sum_{v\in V \mid 0\leq t<t_v^{\text{dep}}} x_{v,t} \leq \Delta t \cdot P^{\text{gridmax}} \qquad t \in T \tag{5.7}$$

$$s_v^{\text{dep}} \leq s_{v,t_v^{\text{dep}}} \qquad v \in V \tag{5.8}$$

$$s_{v,t} = s_{v,t-1} + x_{v,t-1}/C_v \qquad v \in V,\ t = 1, \ldots, t_v^{\text{dep}} \tag{5.9}$$

$$x_{v,t} \geq 0 \qquad v \in V,\ t = 0, \ldots, t_v^{\text{dep}} - 1 \tag{5.10}$$

$$0 \leq s_{v,t} \leq 1 \qquad v \in V,\ t = 0, \ldots, t_v^{\text{dep}} \tag{5.11}$$

The objective function (5.5) minimizes the sum of the costs for the total consumed energy over all time steps. Inequalities (5.6) ensure that the energy by which each EV is charged during each time step does not exceed the SOC-dependent maximum energy. Note that this inequality is in general nonlinear. Constraints (5.7) limit the total energy consumed from the grid during each time step to $\Delta t \cdot P^{\text{gridmax}}$. The departure SOCs are enforced by Inequalities (5.8). Equalities (5.9) determine the SOC at the beginning of each time step $t = 1, \ldots, t_v^{\text{dep}}$ for each EV v. Thereunto the previous state of charge $s_{v,t-1}$ is considered together with the charging rate of the previous time slot $x_{v,t-1}$ and the total battery capacity C_v. Variable domains are defined in (5.10) and (5.11). Due to the domain of variable $x_{v,t}$, an EV may not discharge.

5.4 Problem Solving Approaches

In the following we study different ways to deal with the nonlinear maximum charging energy constraints (5.6). We first consider the simpler case that the maximum power function is concave, where we essentially can solve the problem with a LP formulation or a cutting plane approach. Afterwards, we consider a more general approach that does not make any assumptions on the concavity of the maximum power function. The approach is based on a piecewise linearization of the SOC-energy curve and is accelerated by branch-and-cut.

5.4.1 Concave Maximum Energy Functions

As already mentioned before, if $P_v^{\max}$ is concave, it follows that also $E_v^{\max} \in \{E_v^{\text{max-ex}}, E_v^{\text{max-lb}}\}$ is concave as well. For nonconcave $P_v^{\max}$, we now determine the convex hull to obtain a concave approximation of the original $P_v^{\max}$ for deriving the respective maximum energy function.

In the following, we will further assume that $E_v^{\max}$ is differentiable. We are aware that, depending on $P_v^{\max}$, this assumption might not be completely valid in practice. Actually, $E_v^{\max}$ might have breakpoints, in which the left-sided and right-sided limits of the differential do not coincide. Nevertheless, we will treat $E_v^{\max}$ as if it were differentiable at any SOC of its domain, since differing left-sided and right-sided limits will not affect the results of the following modeling approach.

Due to the assumed properties of $E_v^{\max}$, we can replace the nonlinear Inequality (5.6) from EVS-SOC with the combination of the infinite set of linear inequalities

$$x_{v,t} \leq E_v^{\max\prime}(\hat{s}) \cdot (s_{v,t} - \hat{s}) + E_v^{\max}(\hat{s}) \qquad v \in V,\ t = 0, \ldots, t_v^{\text{dep}} - 1,\ \hat{s} \in [s_{v,0}, s_v^{\text{dep}}] \quad (5.12)$$

where $E_v^{\max\prime}$ is the first derivative of $E_v^{\max}$. We call the resulting linear programming model EVS-SOC-LIN.

Note that if $P_v^{\max}$ is a piecewise linear function, then so is $E_v^{\text{max-lb}}$. The set of inequalities reduces then to a finite one where we have one inequality corresponding to each linear function segment.

In the spirit of [20], who essentially consider a similar kind of inequalities, we can solve EVS-SOC-LIN by a cutting plane approach. Thereby the relaxation of EVS-SOC-LIN without Inequalities (5.12) is first solved. Then, Inequalities (5.12) that are violated by the current LP solution are iteratively determined, added, and the LP problem is re-solved. The process is repeated until no more Inequalities (5.12) are violated.

The separation of a violated inequality for a current solution $(x^{\mathrm{LP}}, s^{\mathrm{LP}})$ to the relaxed EVS-SOC-LIN works as follows. For all $v \in V,\ t = 0, \ldots, t_v^{\mathrm{dep}} - 1$, we check if $x_{v,t}^{\mathrm{LP}} > E_v^{\max}(s_{v,t}^{\mathrm{LP}})$. In this case we add the violated Inequality (5.12) for vehicle v, time step t, and $\hat{s} = s_{v,t}^{\mathrm{LP}}$. Note that for one vehicle, multiple inequalities for different time steps can be added within a single cutting plane iteration. This separation procedure is performed for all vehicles $v \in V$ and as long as any violated inequalities are found, the augmented LP problem is then re-solved.

An alternative to the above is the following. Whenever $x_{v,t}^{\mathrm{LP}} > E_v^{\max}(s_{v,t}^{\mathrm{LP}})$ for some EV v and time step t, one can add the violated Inequality (5.12) not only for time step t but for all time steps $t' = 0, \ldots, t_v^{\mathrm{dep}} - 1$. The intention here is to possibly reduce the number of needed resolving iterations, but clearly the size of the LP formulation increases more quickly. Preliminary experiments indicated that indeed this variant performs better in practice in most cases. Therefore, we apply it in all our experiments documented in the remainder of this chapter.

We also compared this variant with the approach presented in [20], where in one iteration cuts are only added for the smallest time steps that violate Inequality (5.12). We found that our variant usually performs slightly better at least in case of our problem instances.

5.4.2 General Piecewise Linear Maximum Energy Functions

In the following model, we assume for each EV $v \in V$ that the maximum charging energy function $E_v^{\max}$ is a piecewise linear function or is approximated by such. In contrast to EVS-SOC-LIN, we do not make assumptions on the concavity of $E_v^{\max}$. We assume that we are given a finite set of SOC values $\{S_{v,k} \mid k = 1, \ldots, k_v^{\max}\}$ in increasingly sorted order, with $S_{v,1} = 0$ and $S_{v,k_v^{\max}} = 1$ and the values in between representing the breakpoints of the piecewise linear function. These values are pairwise distinct and can be unevenly distributed among the SOC interval $[0, 1]$. For each $S_{v,k}$ we know the value of the maximum charging energy $E_v^{\max}(S_{v,k})$.

We model the piecewise linear function as suggested in Chapter 10.1 of [141]. Thereunto, we use continuous variables $\alpha_{v,t,k}$ to express the SOC $s_{v,t}$ as a convex combination of $S_{v,k}$ and $\alpha_{v,t,k}$. The variables $\alpha_{v,t,k}$ are also used to represent the maximum charging energy function as a convex combination of $E_v^{\max}(S_{v,k})$ and $\alpha_{v,t,k}$.

Furthermore, we introduce additional binary variables $\beta_{v,t,k}$, which are used to ensure that at most two consecutive $\alpha_{v,t,k}$ and $\alpha_{v,t,k+1}$ variables are nonzero. By replacing Constraints (5.6) in formulation (5.5–5.11) with the following Constraints (5.13–5.21), we obtain a MILP model, which we refer to as EVS-SOC-GLIN.

$$s_{v,t} = \sum_{k=1}^{k_v^{\max}} S_{v,k} \cdot \alpha_{v,t,k} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} \tag{5.13}$$

$$x_{v,t} \leq \sum_{k=1}^{k_v^{\max}} E_v^{\max}(S_{v,k}) \cdot \alpha_{v,t,k} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} - 1 \tag{5.14}$$

$$\sum_{k=1}^{k_v^{\max}} \alpha_{v,t,k} = 1 \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} \tag{5.15}$$

$$\sum_{k=1}^{k_v^{\max}-1} \beta_{v,t,k} = 1 \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} \tag{5.16}$$

$$\alpha_{v,t,0} \leq \beta_{v,t,0} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} \tag{5.17}$$

$$\alpha_{v,t,k} \leq \beta_{v,t,k-1} + \beta_{v,t,k} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}},\ k = 2, \dots, k_v^{\max} - 1 \tag{5.18}$$

$$\alpha_{v,t,k_v^{\max}} \leq \beta_{v,t,k_v^{\max}-1} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}} \tag{5.19}$$

$$0 \leq \alpha_{v,t,k} \leq 1 \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}},\ k = 1, \dots, k_v^{\max} \tag{5.20}$$

$$\beta_{v,t,k} \in \{0, 1\} \qquad v \in V,\ t = 0, \dots, t_v^{\text{dep}},\ k = 1, \dots, k_v^{\max} - 1 \tag{5.21}$$

Equations (5.13) link the SOC values $s_{v,t}$ with the continuous weight variables $\alpha_{v,t,k}$. The charging energy $x_{v,t}$ of EV v at time slot t is limited by Inequalities (5.14) to the maximum charging energy. Constraints (5.15) set the sum of the continuous weights $\alpha_{v,t,k}$ over all discrete SOC levels $k = 1, \dots, k_v^{\max}$ to one. Equations (5.16) ensure that exactly one $\beta_{v,t,k}$ variable is active for each EV v and time slot t. The $\alpha_{v,t,k}$ variables are linked with the $\beta_{v,t,k}$ variables by Inequalities (5.17–5.19). Altogether, (5.16–5.19) are the so-called adjacency constraints, which ensure that at most two consecutive variables $\alpha_{v,t,k}$ and $\alpha_{v,t,k+1}$ are nonzero. Constraints (5.20–5.21) define the domains of $\alpha_{v,t,k}$ and $\beta_{v,t,k}$, respectively.

As we will see in Section 5.6, the previously introduced EVS-SOC-LIN formulation, which requires $E_v^{\max}$ to be concave, performs remarkably well. Therefore, we propose a branch-and-cut approach for solving EVS-SOC-GLIN, in which we initially work on the convex hull of $\{(S_{v,k}, E_v^{\max}(S_{v,k})) \mid k = 1, \dots, k_v^{\max}\} \cup \{(S_{v,1}, 0), (S_{v,k_v^{\max}}, 0)\}$. To obtain this relaxation, we consider the original EVS-SOC-GLIN formulation with all its variables and constraints except the linking constraints (5.17–5.19). Then, whenever a solution candidate is found, we check for all $v \in V,\ t = 0, \dots, t_v^{\text{dep}} - 1$ whether $x_{v,t}$ exceeds the actual $E_v^{\max}$ value at SOC $s_{v,t}$, i.e., if $x_{v,t} > E_v^{\max}(s_{v,t})$. If this is the case, a cut is added that links all nonzero $\alpha_{v,t,k}$ variables with their respective $\beta_{v,t,k}$ variables, as we did in Constraints 5.17–5.19. Such cuts are separated and added until for all $v \in V,\ t = 0, \dots, t_v^{\text{dep}} - 1$ it holds that $x_{v,t} \leq E_v^{\max}(s_{v,t})$.

Table 5.1: Used EV types with battery capacity C_v, $P_v^{\max}$ domain $[s_v^{\min}, s_v^{\max}]$ and the number of linear pieces of $P_v^{\max}$.

EV Name	C_v (kWh)	$s_v^{\min}$	$s_v^{\max}$	$\#P_v^{\max}$-lin. pieces
Energica Ego	21.5	1.1	99.9	53
MINI Cooper Electric	32.6	12.1	93.8	34
BMW i3	42.2	15.1	96.0	26
Hyundai Kona Elektro	67.5	10.1	94.9	28
Tesla Model 3 Long Range	82.0	11.1	99.0	35
Mercedes-Benz EQC	85.0	2.1	97.8	24
Jaguar I-Pace	90.0	8.0	100.0	29
Audi e-tron	95.0	3.1	99.8	44

5.5 Benchmark Instances

Due to the lack of pure real-world problem instances we randomly generate benchmark instances and use real-world data as far as possible. Specifically, battery capacities and maximum power functions are adopted from real-world data. We first consider individual EVS-SOC instances that represent snapshot scenarios at certain times with a specific number of vehicles that are assumed to have arrived at the charging station following a homogenous Poisson process. Afterwards, in Section 5.5.2, we will consider whole model based predictive control scenarios with a rolling horizon in which vehicles arrive at different times of a day.

All of the benchmark instances are available at `https://www.ac.tuwien.ac.at/research/problem-instances/`.

5.5.1 Individual EVS-SOC Instances

We distinguish between three types of problem parameters, depending on whether the parameter is set by the user, randomly generated, or based on real-world data. To the input provided by the user, we count the number n of EVs, the length Δt of a time step, and the grid's power capacity P^{gridmax}. We generate 30 instances for each combination of $n \in \{10, 20, 50, 100\}$, $\Delta t \in \{1, 5, 10\}$ minutes, and $P^{\text{gridmax}} \in \{10n, 25n, 40n\}$.

We consider eight different types of real EVs shown in Table 5.1. The EV's battery capacities were taken from the EV Database[1]. The respective maximum power functions $P_v^{\max}$ were manually extracted from plots found on the website of the Dutch EV charging station operator FASTNED[2]. More specifically, 25 up to 70 points of a plot were manually determined in dependence of notable changes of the gradient, and linear interpolation was applied in between. All these $P_v^{\max}$ functions are shown in Figure 5.3. Observe that the maximum power function's available domain of definition $[s_v^{\min}, s_v^{\max}]$ varies among

[1] `https://www.ev-database.de`
[2] `https://fastnedcharging.com`

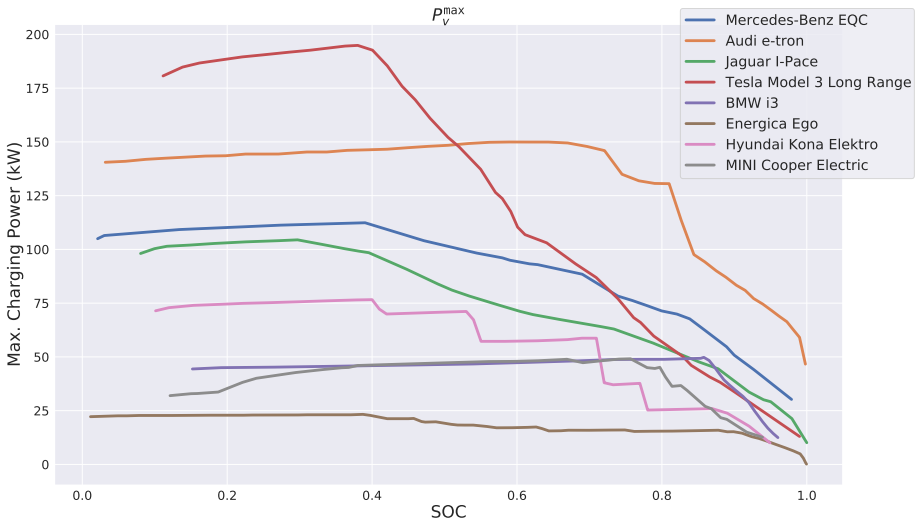


Figure 5.3: Maximum charging power functions $P_v^{\max}$ for all considered vehicle types.

the EVs. If a vehicle type supports speed charging, the respective most powerful charging curve is used.

Since the $P_v^{\max}$ data extracted from the original plots is quite fine-grained, we additionally derive simplified piecewise linear approximations with only five and ten linear pieces, respectively. For this task, we utilized the Python package `pwlf` [142] to determine approximately optimal breakpoints automatically.

A comparison between the original $P_v^{\max}$ and these simpler piecewise approximations is shown in Figure 5.4 exemplarily for the Hyundai Kona Elektro. Observe that the approximation of the original $P_v^{\max}$ function with 10 segments is already quite good for this rather challenging vehicle type. For $P_v^{\max}$ of the other vehicle types, see Appendix B.

For each EV $v \in V$ in a benchmark instance, one of the above EV types is chosen uniformly at random. Moreover, we choose an availability duration at the charging station d_v^{avail} randomly according to a normal distribution with a mean value of six hours and a standard deviation of 1.5 hours.

Next, from the interval $(-d_v^{\text{avail}}/\Delta t, 0)$ we select an arrival time t_v^{arr} uniformly at random and obtain a respective departure time $t_v^{\text{dep}} = \lceil t_v^{\text{arr}} + d_v^{\text{avail}}/\Delta t \rceil$. Considering the available domains of definition of the maximum power functions, we generally assume that each vehicle shall be charged from a SOC of 20% at arrival to a SOC of 90% at departure. In our benchmark instances, we therefore choose the initial SOC proportional to the already bygone availability time, i.e., for all $v \in V$,

$$s_{v,0} = \frac{-t_v^{\text{arr}}}{d_v^{\text{avail}}/\Delta t} \cdot 0.7 + 0.2. \tag{5.22}$$

The departure SOC s_v^{dep} is set to 90% for all EVs.

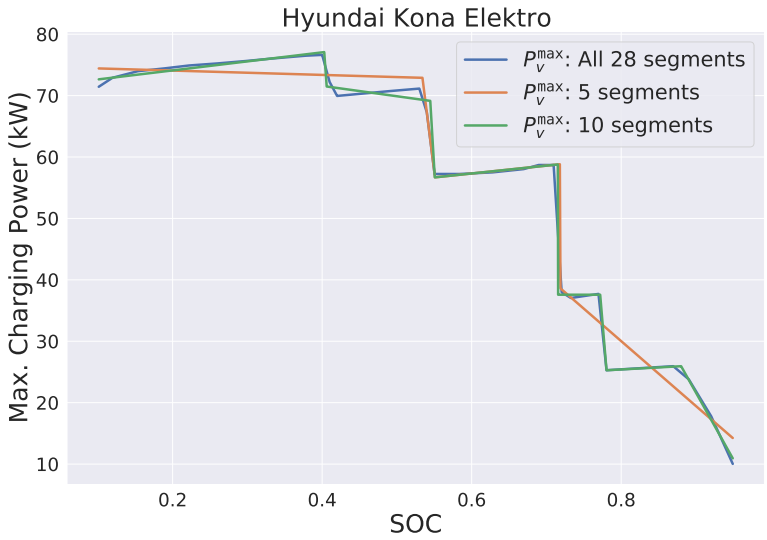


Figure 5.4: Exemplary $P_v^{\max}$ curve with different number of segments.

The end of the time horizon is obtained from the last EV's departure time, i.e., $t_{\max} = \max_{v \in V} t_v^{\text{dep}}$. Electricity costs per unit of consumed energy c_t are independently chosen for each time step $t \in T$ uniformly at random from $[1.9, 3.5)$ cent/kWh.

5.5.2 Rolling Horizon Benchmark Scenarios

In addition to the individual benchmark instances, we consider rolling horizon simulations over whole days starting at time 0:00 and ending at 24:00. To deal with such a scenario in which vehicles arrive at different times at the charging station, the schedule is (re-)optimized at time 0:00 and then every $\tau = 10$ minutes, always considering only EVs that are currently available at the charging station. The found charging schedule is then assumed to be applied for the next τ minutes until a new schedule is determined.

The time is again discretized into equally long time steps of $\Delta t \in \{5, 10\}$ minutes. Electricity costs per unit of consumed energy are chosen as explained in Section 5.5.1 and it is assumed that they are known in advance for the whole charging period. For the number of vehicles we use $n \in \{10, 20, 50, 100\}$. Again, we pick each vehicle type uniformly at random from the set of available vehicle types.

It is assumed that most vehicles arrive around two peak times at 6:00 and 14:00. For picking the arrival time t_v^{arr} for a vehicle $v \in V$, we therefore first randomly select with equal probability one of these two peak times and then sample t_v^{arr} from a normal distribution with the chosen peak time as mean value and a standard deviation of two hours. Times outside of the considered horizon of 24 hours are re-sampled.

The charging duration d_v^{avail} is chosen as described in Section 5.5.1 and t_v^{dep} is derived correspondingly. Also, s_v^{dep} and P^{gridmax} are set as before. At time 0:00 we set $s_{v,0} = 0.2$

and with each rescheduling we determine $s_{v,0}$ based on the charging schedule of the previous iteration.

Thirty independent whole-day scenarios were constructed and are considered in the experimental evaluation.

Exemplary Solutions

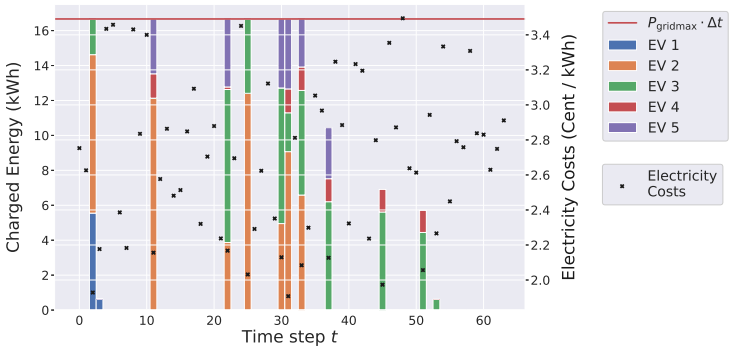
Figure 5.5 exemplarily visualizes optimal solutions for a single individual instance with $n = 5$ EVs and $\Delta t = 5$ minutes obtained from EVS-SOC-GLIN with decreasing grid power capacity $P^{\text{gridmax}} \in \{50, 125, 200\}$ kW. As maximum energy function we chose $E_v^{\text{max-lb}}$ based on $P_v^{\max}$ with five piecewise linear segments. Each sub-figure represents an optimal charging schedule of a vehicle fleet. Bars specify the energy a vehicle is charged with in each time step. The corresponding scale is located on the left y-axis. The grid's maximum energy supply $P^{\text{gridmax}} \cdot \Delta t$ is indicated as horizontal line in the plots. Crosses reveal the electricity costs for each time step and the corresponding scale is located on the right-sided y-axis.

For $P^{\text{gridmax}} = 200$kW it can be observed in Figure 5.5a that vehicles are charged usually in parallel within a single time step and cheap electricity costs can be exploited more effectively. Moreover, at some time steps the charged energy is well below the grid's power capacity. Figure 5.5b shows how the charging schedule changes when lowering P^{gridmax} to 125kW. By reducing the grid's power capacity, more time steps are required for charging the vehicles to their target SOC, resulting in higher total charging costs. Note however that in contrast to the solution shown in Figure 5.5a, the charging costs only slightly increase even though the grid's power capacity has been almost halved. When reducing P^{gridmax} even further to 50kW, as shown in Figure 5.5c, the number of time steps required for charging the vehicles drastically increases. Moreover, in contrast to Figure 5.5a at most time steps only a single vehicle is charged with usually the maximal possible energy. Finally, note that independent of the choice of P^{gridmax} the generated solutions always utilize the time steps at which charging is the cheapest. In summary, Figure 5.5 shows how the choice of P^{gridmax} affects a respective optimal charging schedule: The smaller the power capacity of the grid, the more time steps are required for charging the vehicles and therefore the higher are the total resulting charging costs.

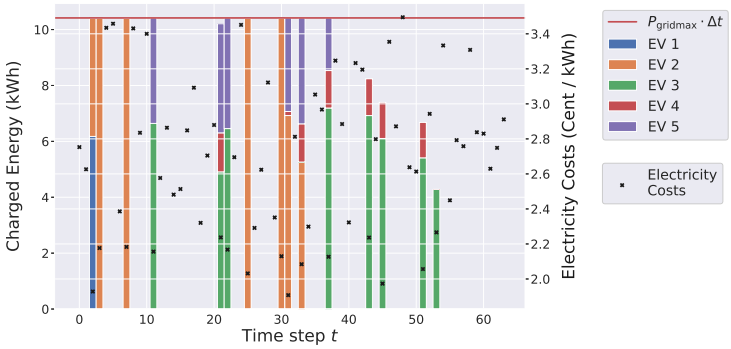
5.6 Experimental Results

All solution approaches were implemented in Julia 1.6.0[3] using the the optimization modeling package JuMP v0.21.5 and Gurobi 9.1.0[4] as LP/MILP solver. Gurobi was configured to run in single-threaded mode with a time limit of 30 minutes per instance. All remaining Gurobi parameters were kept at their default values. The experiments were conducted on an Intel Xeon E5-2640 v4 with 2.40GHz and 16GB memory limit. If

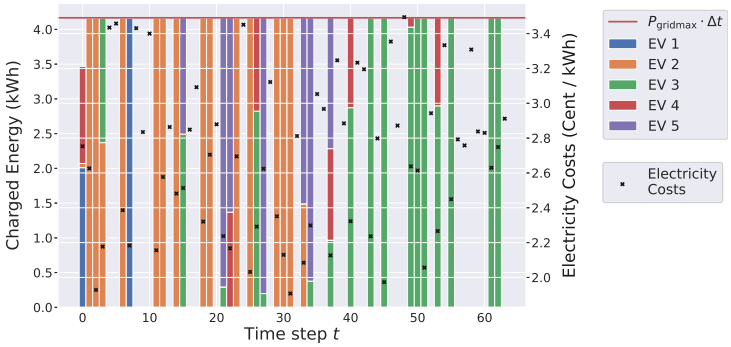
[3] https://julialang.org
[4] https://www.gurobi.com



(a) $P^{\text{gridmax}} = 200\,\text{kW}$; total charging costs: 290.42 cent.



(b) $P^{\text{gridmax}} = 125\,\text{kW}$; total charging costs: 296.91 cent.



(c) $P^{\text{gridmax}} = 50\,\text{kW}$; total charging costs: 330.10 cent.

Figure 5.5: Optimal solution for an instance with $n = 5$, $\Delta t = 5$ minutes, $P^{\text{gridmax}} \in \{50, 125, 200\}\,kW$ using EVS-SOC-GLIN.

not stated otherwise we report in the following mean or median results on the 30 problem instances per instance parameter combination $(n, \Delta t, P^{\text{gridmax}}, E_v^{\max})$.

We first show individual results for EVS-SOC-LIN and EVS-SOC-GLIN, respectively.

Afterwards, solutions generated by both approaches for the same instances w.r.t. the same configurations are compared to each other in Section 5.6.3. Finally, we present results for the rolling horizon scenarios.

5.6.1 EVS-SOC-LIN

We compare two variants of EVS-SOC-LIN. Recall that for piecewise linear $E_v^{\max}$ only a finite set of inequalities as described by (5.12) exists. Hence, next to the variant in which these constraints are dynamically separated as cuts via the cutting plane approach as described in Section 5.4.1, we also consider the variant in which all maximum charging energy constraints (5.12) are statically added to the LP upfront.

Table 5.2: EVS-SOC-LIN runtime comparison for concave maximum power functions and $P^{\text{gridmax}} = 25n$: solving the static MILP versus the cutting plane method.

n	Δt (min)	n_{seg}	Runtime (s)					n_{cuts}	
			Static			Cutting Plane		Cutting Plane	
		Mean	Median	StdDev	Median	StdDev	Mean	StdDev
$E_v^{\text{max-lb}}$								
5	1	49	0.07	0.04	1.34	0.26	10423	5209
5	5	46	0.01	0.00	1.04	0.23	574	269
5	10	43	0.01	0.00	1.03	0.24	190	85
10	1	99	0.18	0.15	1.52	0.38	15949	5580
10	5	93	0.02	0.01	1.05	0.28	1243	520
10	10	86	0.01	0.00	1.03	0.26	416	159
20	1	199	0.60	0.30	2.09	0.49	25549	6715
20	5	187	0.05	0.02	1.10	0.25	2593	747
20	10	172	0.02	0.01	1.05	0.25	862	245
50	1	495	2.78	1.02	6.72	2.07	87375	19749
50	5	464	0.16	0.06	1.28	0.31	6499	1167
50	10	427	0.06	0.02	1.10	0.23	2157	335
100	1	994	9.34	2.60	12.84	3.99	193069	27979
100	5	931	0.56	0.22	1.68	0.32	13502	1664
100	10	858	0.13	0.05	1.25	0.27	4367	475
$E_v^{\text{max-ex}}$								
5	1	901	1.19	1.02	1.31	0.38	12800	5986
5	5	901	0.23	0.11	0.90	0.25	1102	542
5	10	901	0.08	0.07	0.97	0.26	322	205
10	1	1802	4.98	3.27	1.65	0.52	25271	9541
10	5	1802	0.59	0.22	1.06	0.24	2341	950
10	10	1802	0.22	0.10	1.01	0.20	757	387
20	1	3605	14.33	8.48	3.29	0.83	60778	18725
20	5	3605	1.21	0.45	1.16	0.27	5117	1547
20	10	3605	0.68	0.20	1.07	0.21	1585	516
50	1	9041	70.69	31.89	9.11	2.66	175979	28195
50	5	9041	4.17	1.58	1.57	0.33	13737	2329
50	10	9041	1.57	0.54	1.15	0.21	3989	858
100	1	18086	280.22	100.87	25.45	9.66	390873	44162
100	5	18086	13.11	4.73	2.11	0.51	27920	3515
100	10	18086	3.80	1.35	1.32	0.34	8126	1419

The results of this comparison are reported in Table 5.2. As maximum energy function $E_v^{\text{max-lb}}$ as well as $E_v^{\text{max-ex}}$ are considered. The energy functions are derived from the convex hull of $P_v^{\max}$ as described in Section 5.4.1. Moreover, P^{gridmax} is set to $25n$ for all shown instances. The table lists for each instance group, identified by n and Δt, the average total number of piecewise linear segments n_{seg} of the $E_v^{\max}$ functions over all vehicles, a comparison of the runtimes between the cutting plane and the static approach, as well as the average total number of added cuts, denoted by n_{cuts}, for the cutting plane approach.

Note that all reported instances were solved to optimality w.r.t. both maximum energy functions. Using $E_v^{\text{max-lb}}$ as maximum energy function, the static approach as well as the cutting plane approach were both able to solve all instances within few seconds. However, the static approach is significantly faster than the cutting plane method for all considered instance groups.

Using $E_v^{\text{max-ex}}$ as maximum energy function, though, the cutting plane method shows its performance advantages with growing n. Due to how $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ are derived, the number of piecewise linear segments for $E_v^{\text{max-ex}}$ is in general much higher than for $E_v^{\text{max-lb}}$. As the number of segments increases we can observe that the cutting plane approach scales significantly better than the static approach. This improvement is particularly noticeable if we fix n and consider decreasing Δt values. Observe that, for a fixed Δt the number of cuts increases with larger n values, whereas for a fixed n the number of cuts increases with smaller Δt values. Therefore, the results indicate that the cutting plane technique shows performance benefits when a larger number of cuts has to be separated, i.e., the maximum charging power condition was not easily fulfilled. Overall, it can be said that the cutting plane variant outperforms the static model on larger instances and when n_{seg} is large. We additionally conducted the experiments for $P^{\text{gridmax}} = 10n$ and $40n$ and observed the same trends.

In Figure 5.6 we give a more detailed comparison of the runtimes between the static approach and the cutting plane approach with $E_v^{\text{max-ex}}$ as maximum energy function. The figure shows that, when fixing Δt, the static approach does not scale as well as the cutting plane approach in terms of computation time with an increasing number of vehicles. For $\Delta t \in \{5, 10\}$ the runtimes of the cutting plane approach barely increase as n grows. Only for $\Delta t = 1$ minute the runtimes of the cutting plane approach increase slightly with a growing number of vehicles. In contrast, for the static approach the computation times increase much stronger than their cutting plane counterparts. As Δt decreases the difference in performance becomes more and more obvious.

5.6.2 EVS-SOC-GLIN

Similar to before, we compare two variants of EVS-SOC-GLIN for the general nonconcave maximum charging power functions. In the first variant we directly solve the static MILP in which all linking constraints (5.17–5.19) are included from the beginning, whereas the second approach is the branch-and-cut variant (B&C) in which these linking constraints

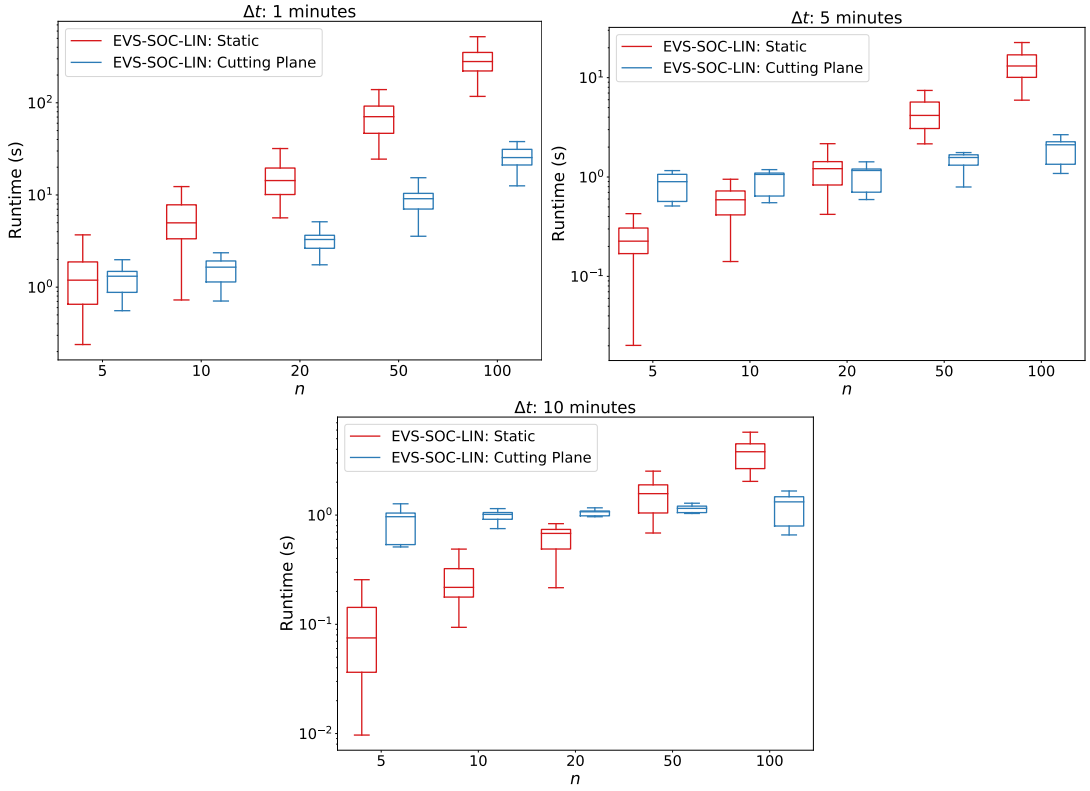


Figure 5.6: EVS-SOC-LIN runtime comparison for directly solving the LP problem versus the cutting plane approach, corresponding to results of Table 5.2.

are dynamically separated as needed, cf. Section 5.4.2. As maximum energy function we use $E_v^{\text{max-ex}}$ and $E_v^{\text{max-lb}}$, both based on the original full resolution $P_v^{\max}$ functions. For $P^{\text{gridmax}} \in \{10n, 25n, 40n\}$ we report the results in Tables 5.3, 5.4, and 5.5, respectively. Columns, n_{seg} denote the total number of piecewise linear segments functions $E_v^{\max}$ consist of, summed over all n vehicles of an instance. Columns n_{feas} indicate the numbers of instances per group to which feasible solutions have been found and columns "Runtime" list the median computation times per group. Again, n_{cuts} refers to the total number of cuts added within B&C. The last columns indicate the finally remaining optimality gaps between lower and upper bounds as reported by Gurobi. These gaps are calculated as the absolute difference between the respective upper and lower bounds divided by the upper bound. Moreover, for visual representation of the number of feasibly solved instances, the median runtimes, and the number of added cuts within B&C see Figure 5.7–5.9. Only gaps of instances with a feasible solution are considered. For parameter combinations without gaps (marked with "-"), no feasible solution has been found for any instance within the time limit. For parameter combinations where no runtime is reported, all corresponding runs terminated due to an out-of-memory error. More detailed results can be found in Appendix B where also the number of instances solved to optimality as well

as standard deviations for runtimes and the numbers of cuts are reported.

Opposed to EVS-SOC-LIN, not all instances could be solved by the EVS-SOC-GLIN variants within the time limit. Considering the results with $P^{\text{gridmax}} = 10n$, one can notice that the B&C approach shows performance benefits, as the approach was able to always find feasible solutions to as many or more instances than the static approach. It is difficult to compare the quality of the solutions obtained by each approach as the static approach sometimes found fewer feasible solutions. For groups for which both approaches could obtain feasible solutions to all instances, the quality of the generated solutions is almost identical. Moreover, except for two instance groups, the B&C approach was either as fast or faster than the static approach.

For the results with $P^{\text{gridmax}} = 25n$, the runtime performance benefit of B&C is still noticeable for small n, however it is not as strong as for $P^{\text{gridmax}} = 10n$. Moreover, for $E_v^{\text{max-lb}}$ the number of feasible solutions found by the static approach is, except for one group, never worse than for B&C. However, for $E_v^{\text{max-ex}}$ B&C still yielded significantly more feasible solutions.

A similar observation can be made for $P^{\text{gridmax}} = 40n$. For $P^{\text{gridmax}} = 40n$, the static approach has a better runtime with almost all parameter configurations.

A possible explanation for this observation seems to be that for $P^{\text{gridmax}} = 10n$ the charging energy of a vehicle v is more limited by P^{gridmax} than by $E_v^{\max}$. Initial solutions of B&C will then violate Constraints (5.14) less often, which implies spending less time for the separation of cuts. This presumption is supported by considering the number of added cuts. Fixing n and Δt, one can observe that with growing P^{gridmax} clearly more cuts are added.

When comparing $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ for any fixed P^{gridmax}, n, and Δt, $E_v^{\text{max-ex}}$ has more segments than $E_v^{\text{max-lb}}$ due to the nature of its computation. Also, for $E_v^{\text{max-lb}}$ smaller Δt values imply a higher number of $E_v^{\text{max-lb}}$ segments. For a fixed n and Δt the larger number of $E_v^{\text{max-ex}}$ segments comes with fewer feasible solutions and higher runtimes for the static approach and the B&C.

In general, regardless of P^{gridmax}, all reported median gaps for both approaches are below 0.2%. Moreover, while the B&C approach usually finds a higher number of feasible solutions, the static approach finds generally more optimal solutions, as can be seen in Appendix B.

In order to see how both solution approaches to EVS-SOC-GLIN perform on instances with fewer piecewise linear segments in $E_v^{\max}$, we conduct similar experiments using the approximations of $P_v^{\max}$ with five segments. For this we only consider $E_v^{\text{max-lb}}$, since the number of $E_v^{\text{max-ex}}$ segments does not depend on the number of $P_v^{\max}$ segments. Experimental results for $P^{\text{gridmax}} = 25n$ are given in Table 5.6. The table shows again the total number of piecewise linear segments of $E_v^{\text{max-lb}}$ (n_{seg}), the number of instances for which a feasible solutions was found within the time limit (n_{feas}), the median computation

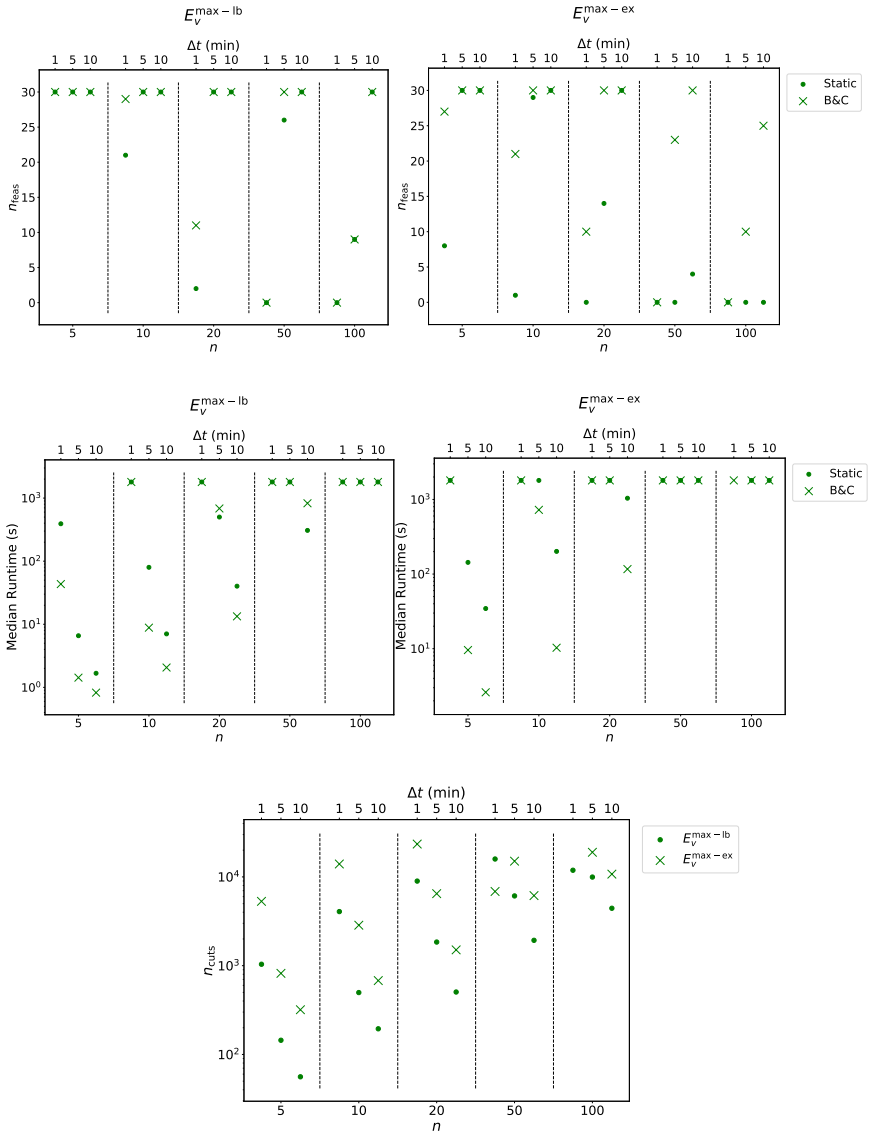


Figure 5.7: Visualization of EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ based on five-segment piecewise linear approximations of the original $P_v^{\max}$ functions, $P^{\text{gridmax}} = 10n$.

time ("Runtime"), the total number of cuts added within B&C (n_{cuts}), and optimality gaps (%-gap) of the generated solutions.

For each parameter group, B&C always finds at least as many feasible solutions as the static approach. When the static and the B&C approaches find the same number of feasible solutions, the resulting gaps are almost identical, though, the solutions of the static variant are typically slightly better than the ones of B&C. In terms of computation

Table 5.3: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 10n$.

n	Δt (min)	n_{seg}	n_{feas}		Runtime (s)		n_{cuts}	%-gap	
		Mean			Median		Median	Median	
			Static	B&C	Static	B&C	B&C	Static	B&C
$E_v^{\text{max-lb}}$									
5	1	155	30	30	391.75	43.39	1038	0.01	0.01
5	5	139	30	30	6.58	1.43	144	0.00	0.01
5	10	119	30	30	1.67	0.83	56	0.00	0.00
10	1	311	21	29	1800.00	1800.00	4068	0.03	0.03
10	5	279	30	30	79.94	8.84	498	0.01	0.01
10	10	242	30	30	7.04	2.06	194	0.00	0.01
20	1	612	2	11	1800.00	1800.00	8974	0.08	0.19
20	5	553	30	30	500.49	684.63	1846	0.01	0.01
20	10	475	30	30	40.18	13.35	505	0.01	0.01
50	1	1544	0	0	1800.00	1800.00	15910	-	-
50	5	1393	26	30	1800.00	1800.00	6106	0.05	0.05
50	10	1192	30	30	307.62	827.59	1930	0.01	0.01
100	1	3095	0	0	1800.00	1800.00	11886	-	-
100	5	2796	9	9	1800.00	1800.00	9961	0.08	0.12
100	10	2399	30	30	1800.00	1800.00	4434	0.01	0.03
$E_v^{\text{max-ex}}$									
5	1	901	8	27	1800.00	1800.00	5304	0.03	0.01
5	5	901	30	30	143.42	9.59	820	0.00	0.00
5	10	901	30	30	34.53	2.60	319	0.00	0.00
10	1	1802	1	21	1800.00	1800.00	13982	0.04	0.08
10	5	1802	29	30	1800.00	725.37	2858	0.01	0.01
10	10	1802	30	30	201.32	10.29	680	0.00	0.01
20	1	3605	0	10	1800.00	1800.00	23449	-	0.14
20	5	3605	14	30	1800.00	1800.00	6479	0.07	0.05
20	10	3605	30	30	1038.91	116.59	1507	0.01	0.01
50	1	9041	0	0	1800.00	1800.00	6856	-	-
50	5	9041	0	23	1800.00	1800.00	15048	-	0.11
50	10	9041	4	30	1800.00	1800.00	6160	0.18	0.03
100	1	18078	0	0	-	1800.00	0	-	-
100	5	18086	0	10	1800.00	1800.00	18944	-	0.08
100	10	18086	0	25	1800.00	1800.00	10750	-	0.06

times, no approach is significantly faster than the other.

Due to the smaller number of segments in the $P_v^{\max}$ functions and consequently also simpler $E_v^{\text{max-lb}}$ functions, a higher number of feasible as well as optimal solutions could generally be found, when comparing Tables 5.6 and 5.4. Moreover, the impact of fewer $P_v^{\max}$ segments is also observable when we consider the median runtimes and the number of added cuts. For almost all parameter combinations of n and Δt, fewer $P_v^{\max}$ segments lead to lower median runtimes and fewer cuts.

Table 5.4: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 25n$.

n	Δt (min)	n_{seg}	n_{feas}		Runtime (s)		n_{cuts}	%-gap	
		Mean			Median		Median	Median	
			Static	B&C	Static	B&C	B&C	Static	B&C
$E_v^{\text{max-lb}}$									
5	1	155	29	30	1800.00	1800.00	3184	0.02	0.06
5	5	139	30	30	25.52	6.68	422	0.01	0.01
5	10	119	30	30	1.27	1.62	153	0.01	0.01
10	1	312	20	23	1800.00	1800.00	7298	0.10	0.12
10	5	279	30	30	183.39	770.59	1132	0.01	0.01
10	10	242	30	30	17.87	11.88	452	0.01	0.01
20	1	612	4	3	1800.00	1800.00	11938	0.26	0.28
20	5	553	30	30	1800.00	1800.00	2702	0.01	0.05
20	10	475	30	30	60.59	201.06	967	0.01	0.01
50	1	1544	0	0	1800.00	1800.00	22034	-	-
50	5	1393	29	30	1800.00	1800.00	6997	0.08	0.11
50	10	1192	30	30	902.21	1800.00	2575	0.01	0.03
100	1	3095	0	0	1800.00	1800.00	29193	-	-
100	5	2796	14	7	1800.00	1800.00	11737	0.12	0.18
100	10	2399	30	30	1800.00	1800.00	5340	0.03	0.06
$E_v^{\text{max-ex}}$									
5	1	901	9	25	1800.00	1800.00	15258	0.21	0.20
5	5	901	30	30	448.47	761.59	2153	0.01	0.01
5	10	901	30	30	56.12	16.43	866	0.00	0.01
10	1	1802	1	18	1800.00	1800.00	23328	0.23	0.33
10	5	1802	26	30	1800.00	1800.00	5220	0.04	0.06
10	10	1802	30	30	204.26	233.60	2063	0.01	0.01
20	1	3605	0	2	1800.00	1800.00	17970	-	0.32
20	5	3605	15	29	1800.00	1800.00	10784	0.08	0.12
20	10	3605	29	30	1097.26	1800.00	4647	0.01	0.03
50	1	9041	0	0	1800.00	1800.00	23986	-	-
50	5	9041	0	17	1800.00	1800.00	23708	-	0.18
50	10	9041	16	28	1800.00	1800.00	12160	0.04	0.08
100	1	18086	0	0	1800.00	1800.00	0	-	-
100	5	18086	0	0	1800.00	1800.00	25754	-	-
100	10	18086	0	19	1800.00	1800.00	19752	-	0.09

Charging Cost Differences & Charging Errors

While the simpler approximations of the original $P_v^{\max}$ functions lead to shorter runtimes, there is clearly a tradeoff concerning the precision of the model, introduced errors, and final solution qualities. We have a closer look on these aspects in the following. Specifically, we are interested in the error made when using $E_v^{\text{max-lb}}$ instead of $E_v^{\text{max-ex}}$ and the error between the five-segment $P_v^{\max}$ approximation compared to the original $P_v^{\max}$. For this purpose, we evaluate EVS-SOC-GLIN on four different $E_v^{\max}$ functions: $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$, each based on the five-segment $P_v^{\max}$ approximation and the original $P_v^{\max}$.

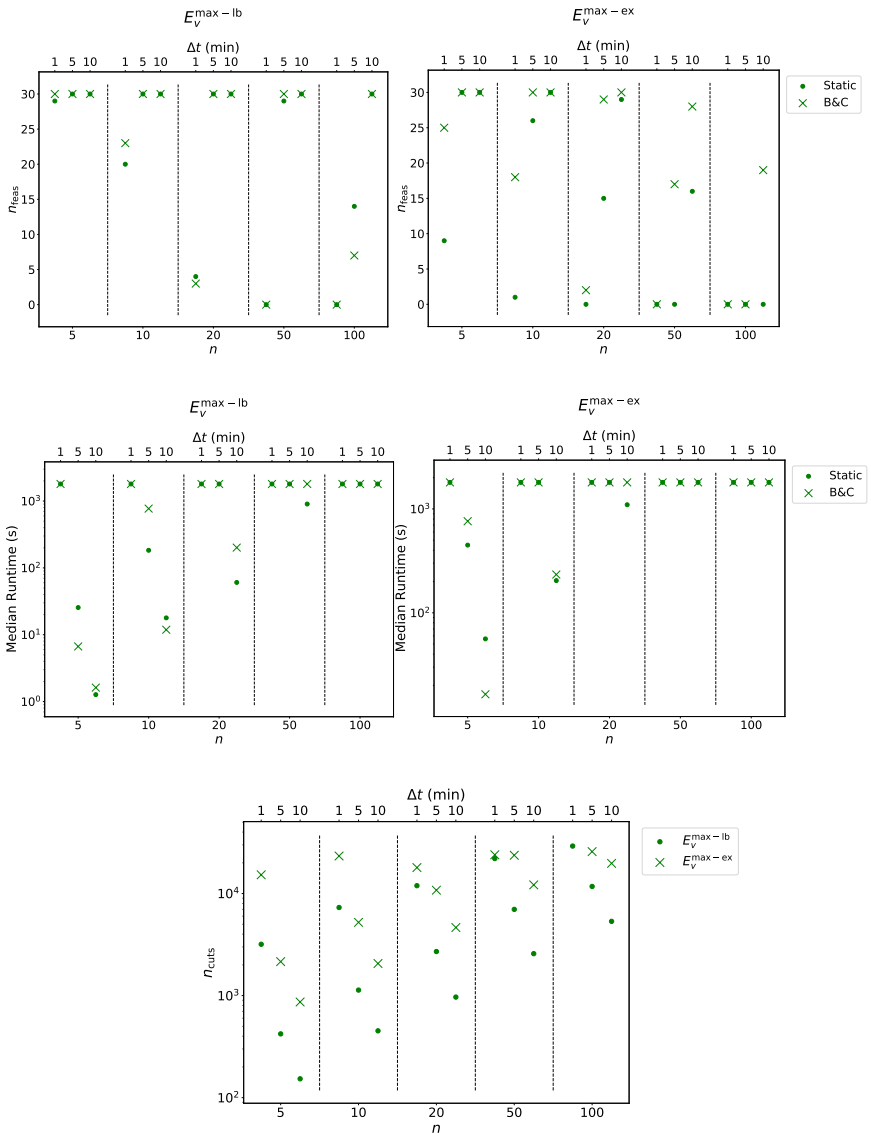


Figure 5.8: Visualization of EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ based on five-segment piecewise linear approximations of the original $P_v^{\max}$ functions, $P^{\text{gridmax}} = 25n$.

Since we want to measure the impact of the different charging curves on the charging costs, we select a high P^{gridmax} value of $40n$ as in this case the variable maximum charging power constraints have higher impact. Only results on instances solved to optimality are reported. Also, we only consider instances where an optimal solution for all four $E_v^{\max}$ functions was found. Parameter combinations where no such instances exist are omitted. The mean charging costs can be found in Table 5.7. The charging cost %-gaps are calculated by $100\% \cdot (|E_v^{\text{max-ex}} - E_v^{\text{max-lb}}|)/E_v^{\text{max-ex}}$.

Table 5.5: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 40n$.

n	Δt (min)	n_{seg}	n_{feas}		Runtime (s)		n_{cuts}	%-gap	
		Mean			Median		Median	Median	
			Static	B&C	Static	B&C	B&C	Static	B&C
$E_v^{\text{max-lb}}$									
5	1	155	29	29	1800.00	1800.00	4476	0.04	0.15
5	5	139	30	30	31.04	55.93	619	0.01	0.01
5	10	119	30	30	2.49	4.05	247	0.01	0.01
10	1	311	20	20	1800.00	1800.00	8161	0.21	0.17
10	5	279	30	30	301.14	1800.00	1410	0.01	0.03
10	10	242	30	30	27.80	36.06	456	0.01	0.01
20	1	612	2	1	1800.00	1800.00	13361	0.27	0.48
20	5	553	30	30	1800.00	1800.00	2863	0.04	0.10
20	10	475	30	30	69.51	571.16	1078	0.01	0.01
50	1	1544	0	0	1800.00	1800.00	25908	-	-
50	5	1393	28	28	1800.00	1800.00	7110	0.12	0.21
50	10	1192	30	30	1097.80	1800.00	2748	0.01	0.05
100	1	3095	0	0	1800.00	1800.00	29066	-	-
100	5	2796	7	2	1800.00	1800.00	11782	0.22	0.21
100	10	2399	29	30	1800.00	1800.00	5650	0.06	0.10
$E_v^{\text{max-ex}}$									
5	1	901	9	24	1800.00	1800.00	20190	0.23	0.44
5	5	901	30	30	582.18	1800.00	3180	0.01	0.07
5	10	901	30	30	80.12	34.07	1228	0.00	0.01
10	1	1802	1	13	1800.00	1800.00	24450	0.49	0.77
10	5	1802	26	30	1800.00	1800.00	6026	0.02	0.17
10	10	1802	30	30	245.17	1147.26	2161	0.01	0.01
20	1	3605	0	0	1800.00	1800.00	17460	-	-
20	5	3605	15	29	1800.00	1800.00	13276	0.14	0.22
20	10	3605	29	30	1437.18	1800.00	5692	0.01	0.08
50	1	9041	0	0	1800.00	1800.00	12253	-	-
50	5	9041	0	11	1800.00	1800.00	27617	-	0.21
50	10	9041	14	27	1800.00	1800.00	13538	0.10	0.12
100	1	18083	0	0	-	1800.00	0	-	-
100	5	18086	0	0	1800.00	1800.00	31692	-	-
100	10	18086	0	11	1800.00	1800.00	23081	-	0.14

Observe that for fixed Δt and varying n, the charging cost gap between $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ does not change significantly. It seems that the difference in charging costs mainly depends on Δt. Specifically, one might notice that the charging cost gaps become smaller as Δt decreases. Overall, the largest mean charging cost gap is 0.64%, the differences therefore seem to be negligible for practical purposes for the considered parameter groups. Note however that not all instances could be solved to optimality (even when increasing the time limit) and hence the number of reported instances in some instance groups varies for each instance group. Therefore, to give a better idea about the distribution

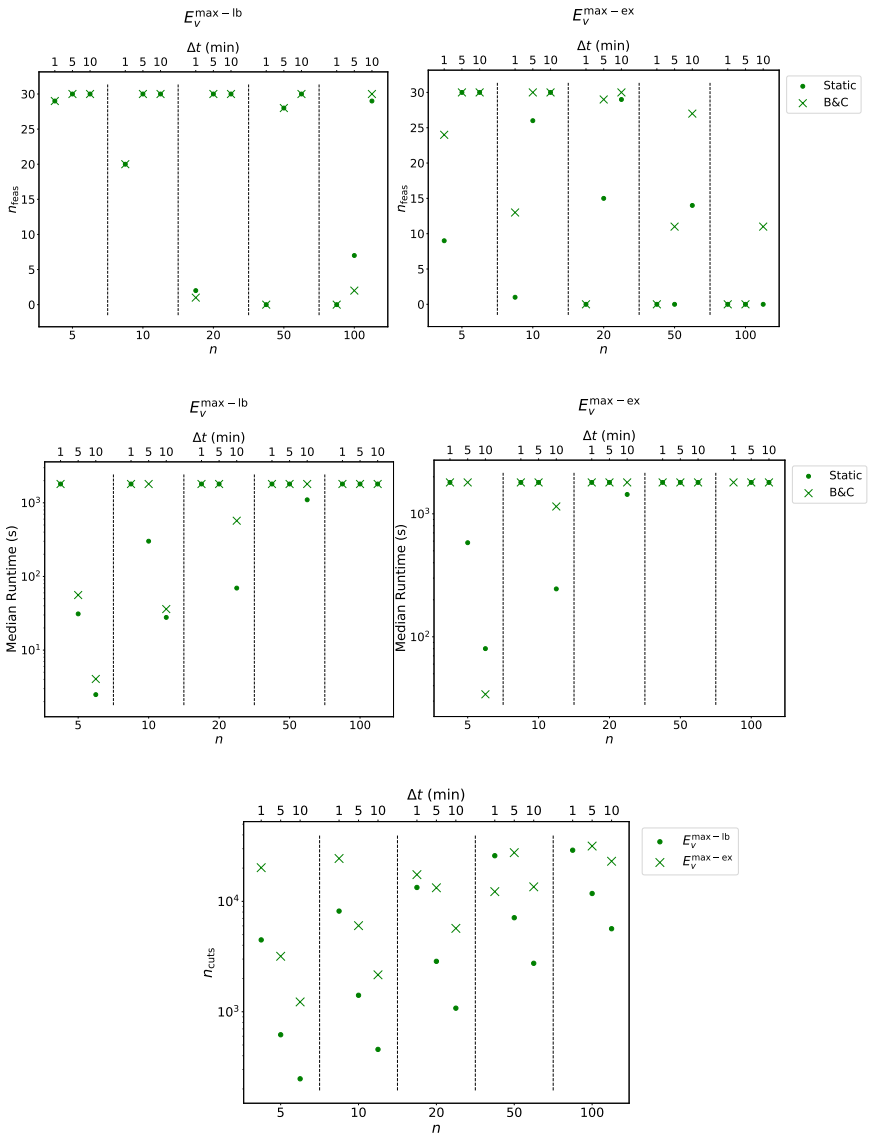


Figure 5.9: Visualization of EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ based on five-segment piecewise linear approximations of the original $P_v^{\max}$ functions, $P^{\text{gridmax}} = 40n$.

of the charging cost gaps, we additionally provide standard deviations to the charging cost gaps in Table 5.7. For groups with the same Δt we can observe that the standard deviations are quite similar.

When comparing the five-segment $P_v^{\max}$ approximation to the original $P_v^{\max}$, the difference in charging costs is marginal, even for large instances. For example consider $n = 20$, $\Delta t =$ 10 minutes and $E_v^{\text{max-ex}}$ and observe that the objective value differs on average by about

Table 5.6: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ based on five-segment piecewise linear approximations of the original $P_v^{\max}$ functions, $P^{\text{gridmax}} = 25n$.

n	Δt (min)	n_{seg}	n_{feas}		Runtime (s)		n_{cuts}	%-gap	
		Mean			Median		Median	Median	
			Static	B&C	Static	B&C	B&C	Static	B&C
$E_v^{\text{max-lb}}$									
5	1	40	30	30	60.14	19.63	387	0.01	0.01
5	5	46	30	30	2.40	1.98	88	0.01	0.01
5	10	43	30	30	0.64	1.13	42	0.00	0.01
10	1	80	30	30	509.28	1800.00	1162	0.01	0.02
10	5	92	30	30	11.01	8.34	232	0.01	0.01
10	10	87	30	30	1.49	2.68	118	0.01	0.01
20	1	160	12	30	1800.00	1800.00	2488	0.03	0.06
20	5	185	30	30	54.58	61.09	516	0.01	0.01
20	10	174	30	30	5.03	7.45	217	0.01	0.01
50	1	398	0	12	1800.00	1800.00	5598	-	0.24
50	5	459	30	30	640.74	1800.00	1556	0.01	0.02
50	10	433	30	30	37.23	36.95	624	0.01	0.01
100	1	798	0	0	1800.00	1800.00	9312	-	-
100	5	921	30	30	1800.00	1800.00	3237	0.01	0.06
100	10	871	30	30	112.16	84.83	1360	0.01	0.01

Table 5.7: Objective value comparison using EVS-SOC-GLIN and different $E_v^{\max}$ functions based on the five-segment $P_v^{\max}$ approximation and the original $P_v^{\max}$; $P^{\text{gridmax}} = 40n$.

n	Δt (min)	n_{opt}	Charging Costs			
			$E_v^{\text{max-lb}}$	$E_v^{\text{max-ex}}$	%-gap	
			Mean	Mean	Mean	StdDev
Original $P_v^{\max}$						
5	1	2	109.08	108.97	0.10	0.01
5	5	25	209.40	208.83	0.29	0.18
5	10	30	227.10	225.78	0.64	0.40
10	5	11	374.24	372.98	0.34	0.13
10	10	28	447.51	445.05	0.59	0.35
20	10	19	882.53	877.33	0.60	0.30
5-segment approx. $P_v^{\max}$						
5	1	2	109.10	108.98	0.10	0.01
5	5	25	209.38	208.82	0.29	0.17
5	10	30	227.11	225.77	0.64	0.41
10	5	11	374.14	372.92	0.33	0.13
10	10	28	447.44	445.04	0.57	0.32
20	10	19	882.39	877.26	0.60	0.30

0.07 cent only between the original $P_v^{\max}$ and the five-segment approximation. This insight seems to be particularly relevant, since it shows that approximating $P_v^{\max}$ with a lower number of linear pieces is reasonable for practice.

When realizing a charging plan in practice with a different $E_v^{\max}$ function than used for scheduling, the specified target SOCs s_v^{dep} might not be reached for some vehicles. We measure this error by generating an *optimal* charging schedule with $E_v^{\text{max-ex}}$ and simulating the actual maximum energy function with $E_v^{\text{max-lb}}$. In the simulation, the actually charged energy is set to the minimum from the corresponding planned charged energy and the actual maximum energy function. The resulting mean deviation from the target SOC in percent, the mean charging error, can be seen in Table 5.8. For a single instance, we determined the mean charging error over all vehicles, whereas for an instance group we again report the mean and the standard deviation of these mean charging errors from the individual instances.

Table 5.8: Charging error comparison when scheduling with $E_v^{\text{max-ex}}$ using EVS-SOC-GLIN and realizing the schedule with $E_v^{\text{max-lb}}$; $P^{\text{gridmax}} = 40n$.

n	Δt (min)	n_{opt}	Mean Charging Error (% SOC)			
			Original $P_v^{\max}$		5-seg. approx. $P_v^{\max}$	
			Mean	StdDev	Mean	StdDev
5	1	3	0.23	0.08	0.21	0.08
5	5	25	1.14	0.26	1.06	0.28
5	10	30	2.01	0.58	1.94	0.60
10	5	12	1.14	0.16	1.18	0.18
10	10	29	2.03	0.45	2.03	0.46
20	10	20	2.01	0.29	1.97	0.34

Similarly to before, it seems that the size of the charging error mainly depends on Δt: Fixing the number of vehicles n, the mean charging error decreases with smaller Δt, the number of vehicles does not seem to influence the mean charging error for fixed Δt.

5.6.3 Comparison of EVS-SOC-LIN and EVS-SOC-GLIN

Charging cost gaps between solutions of formulation EVS-SOC-LIN and EVS-SOC-GLIN can be found in Figure 5.10. As before, we only consider instances that were solved to optimality. For EVS-SOC-LIN we use $E_v^{\text{max-lb}}$ based on the concave $P_v^{\max}$, whereas for EVS-SOC-GLIN we use $E_v^{\text{max-lb}}$ based on $P_v^{\max}$ with five segments. The grid capacity P^{gridmax} is again set to $40n$. Charging cost gaps are calculated by dividing the difference of the EVS-SOC-GLIN objective values from the EVS-SOC-LIN objectives by the EVS-SOC-GLIN objective values. For $n \in \{50, 100\}$ and $\Delta t = 1$ minute, all mean charging cost gaps are zero, therefore the respective bars are not shown in the figure. Comparing the gaps of both formulations, one can notice that the charging costs of solutions generated by EVS-SOC-LIN are slightly too optimistic, underestimating the actual costs. In comparison to the more exact EVS-SOC-GLIN, the costs of the solutions

generated by EVS-SOC-LIN are lower by at most by 0.35%. Moreover, there are no significant differences between the charging cost gaps when varying n or Δt values. When it comes to computation times, both variants of EVS-SOC-LIN are significantly faster than any EVS-SOC-GLIN variant, as we have seen before in Table 5.2 and Table 5.4.

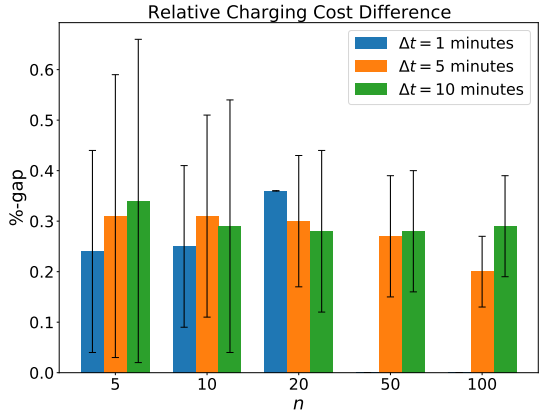


Figure 5.10: Mean charging cost gaps of EVS-SOC-LIN and EVS-SOC-GLIN with $P^{\text{gridmax}} = 40n$. Gaps are given in percent. Whiskers indicate the standard deviations. Note that for $n = 20$ and $\Delta t = 1$ only a single instance was solved to optimality and therefore the corresponding standard deviation is zero.

For the exact same setting as above, we also measure the charging error when scheduling with the convex $E_v^{\text{max-lb}}$ used in EVS-SOC-LIN and realizing the plan with the, in general, nonconvex $E_v^{\text{max-lb}}$ used in EVS-SOC-GLIN. The mean charging error is shown in Figure 5.11. It can be said that for a fixed Δt, the mean charging error does not significantly change for a varying number of vehicles n. However, for a fixed number n, the mean charging error grows with decreasing Δt. An explanation for this behavior seems to be that on instances with smaller Δt, solutions tend to be more precise in terms of the error induced by the time discretization. Therefore the difference between a convex and nonconvex $E_v^{\max}$ function could have more impact on solutions of instances with small Δt values. Overall, the mean charging cost difference does not exceed 1.5% SOC for any n and any Δt and, thus, may be negligible in practice.

5.6.4 Model Based Predictive Control Simulations

For the rolling horizon scenarios, we conduct experiments using formulations EVS-SOC-LIN and EVS-SOC-GLIN. We use $E_v^{\text{max-lb}}$ for both formulations, but for EVS-SOC-LIN the corresponding concave approximation of $P_v^{\max}$, whereas for EVS-SOC-GLIN the five-segment approximation of $P_v^{\max}$. P^{gridmax} is set to $40n$. Results of the experiments are shown in Table 5.9. Absolute charging cost differences are determined by subtracting the EVS-SOC-GLIN objective values from the EVS-SOC-LIN objective values. Relative charging costs are based on the absolute charging costs divided by the objective values of EVS-SOC-GLIN.

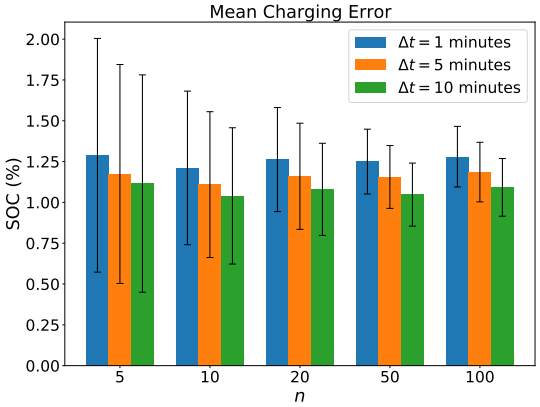


Figure 5.11: Mean charging error when scheduling with convex $E_v^{\text{max-lb}}$ and realizing the plan with nonconvex $E_v^{\text{max-lb}}$ using $P^{\text{gridmax}} = 40n$. Whiskers indicate the standard deviations.

Similarly to before, for fixed n and Δt, the charging costs of EVS-SOC-LIN and EVS-SOC-GLIN only differ marginally. The maximum gap is 0.27% for $n = 100$ and $\Delta t = 5$ minutes. As expected, the absolute charging cost difference increases with a higher number of vehicles. The gaps, however, seem to stay in the same order of magnitude for growing n.

Table 5.9: Rolling horizon charging cost difference for EVS-SOC-LIN vs. EVS-SOC-GLIN using $E_v^{\text{max-lb}}$; $P^{\text{gridmax}} = 40n$.

n	Δt (min)	Charging Cost Difference			
		Absolute (cent)		Relative (%)	
		Mean	StdDev	Mean	StdDev
5	5	0.97	0.73	0.22	0.16
5	10	0.91	0.60	0.20	0.12
10	5	1.75	0.99	0.20	0.11
10	10	1.78	0.77	0.20	0.08
20	5	3.78	1.34	0.21	0.08
20	10	3.80	1.03	0.21	0.06
50	5	9.14	2.42	0.20	0.05
50	10	9.39	2.64	0.21	0.06
100	5	24.42	2.40	0.27	0.03
100	10	19.96	4.82	0.22	0.05

5.7 Conclusions

We formally introduced the EVS-SOC problem in which we put particular focus on dealing with vehicle-specific SOC-dependent maximum charging power limitations. We

addressed the issue that the maximum charging power $P_v^{\max}$ may be regulated within a single time step in a time discretized solution approach by turning towards considering the maximum amount of energy that can be charged in a time step. To this end, we proposed an exact derivation $E_v^{\text{max-ex}}$ as well as a simpler lower bound $E_v^{\text{max-lb}}$. One should keep in mind that the gap between $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ decreases with smaller time step duration Δt. We recall that charging schedules generated with $E_v^{\text{max-lb}}$ are guaranteed to be realizable in practice, whereas schedules generated with $E_v^{\text{max-ex}}$ help us with the estimation of the charging cost differences and charging errors induced by the time discretization.

Let us recapitulate the most important experimental results. Two different MILP formulations, EVS-SOC-LIN and EVS-SOC-GLIN, were proposed, where EVS-SOC-LIN relies on the assumption that $E_v^{\max}$ is concave. When taking a closer look at EVS-SOC-LIN, both the static as well as the cutting plane variant, are quite fast. Compared to EVS-SOC-GLIN, EVS-SOC-LIN performs an order of magnitude faster in our experiments. Considering the runtime difference between the static and the cutting plane approach, a substantial performance benefit of the latter can be observed. Moreover, we have seen that the runtime of the cutting plane approach scales better with larger numbers of vehicles or decreasing Δt values. Its advantages become even more visible when the maximum charging energy of a vehicle has to be exploited, i.e., a large number of cuts has to be separated.

Concerning the static solution approach and the B&C for solving EVS-SOC-GLIN, we found that B&C performs better for instances with a small number of vehicles. For larger instances, however, the static variant is usually superior in terms of runtime. It also shows performance advantages for larger grid capacities. Results of the experiments indicate that the B&C is slower than the static variant when a large number of cuts has to be separated. Nevertheless, there are cases where B&C is faster, for example when $E_v^{\max}$ consists of many linear segments. Additionally, we realized that B&C finds more feasible solutions in the majority of the experiments, when solving to optimality is not possible anymore within the runtime limit. Overall, for both EVS-SOC-GLIN solution approaches it is also worth mentioning that fewer $P_v^{\max}$ segments usually clearly reduce the runtime.

Different approximations of the maximum charging power (e.g., piecewise linear approximation or convex hull approximation), as well as the maximum charging energy ($E_v^{\text{max-lb}}$, $E_v^{\text{max-ex}}$) have been proposed. We studied the charging cost differences and the charging errors induced by these approximations. Regarding the charging cost differences, it turned out that there are only marginal charging cost differences between schedules generated with $E_v^{\text{max-lb}}$ and schedules generated with $E_v^{\text{max-ex}}$. The number of vehicles did not show any noticeable impact on the cost differences for this comparison. Naturally, a smaller step duration Δt reduces the charging cost differences. Moreover, in case of our benchmark instances the approximation of $P_v^{\max}$ with five piecewise linear segments does not have any noticeable impact on the charging costs, despite the rather complex original functions. We also inspected the charging cost differences when generating schedules

based on the original $P_v^{\max}$ function and its concave approximation. It turned out that the charging cost differences are quite small, the mean differences did not exceed 0.35% for any shown parameter group.

As already mentioned, approximating the maximum charging energy might lead to the issue that vehicles do not reach their desired target SOCs. To measure this effect, we generated charging schedules with $E_v^{\text{max-ex}}$ and simulated the actual charging with $E_v^{\text{max-lb}}$. Experimental results have shown that the mean charging error does not exceed 2.1% SOC even for $\Delta t = 10$ minutes. For this experiments, we could also detect a correlation between the size of Δt and the charging error, more specifically the mean charging error decreases with smaller Δt. In another simulation, we considered the mean charging error when generating a charging schedule based on a concave $P_v^{\max}$ approximation and realizing it with the original $P_v^{\max}$. The mean charging error is rather small again, the mean deviation from the vehicles' target SOCs were at most 1.5%.

To see whether the concave approximation of $P_v^{\max}$ accumulates large charging cost differences in a whole day scenario, we conducted model based predictive control simulations with the original $P_v^{\max}$ and its concave approximation. The relative charging cost gaps were even smaller with a maximum value 0.27% for 100 vehicles and $\Delta t = 5$ minutes.

Overall, where we utilize one of the formulations within a model based predictive control strategy, we recommend the usage of EVS-SOC-LIN or EVS-SOC-GLIN together with a reasonably small Δt value of few minutes, in order to reduce errors introduced by time discretization. Depending on whether EVS-SOC-GLIN is performant enough for a given application setting (i.e., it finds a charging schedule within the re-optimization interval) its usage is advised to reduce the danger of significant charging cost differences and charging errors. It seems promising to approximate $P_v^{\max}$ with five to ten piecewise linear segments to improve runtime in this scenario.

In case EVS-SOC-GLIN does not find charging schedules in reasonable time, one might fall back to EVS-SOC-LIN and its cutting plane approach to rapidly generate charging schedules for a concave approximation of $P_v^{\max}$. The introduced errors are usually negligible as we have seen.

In future work it would be interesting to investigate whether the runtime of solving EVS-SOC-GLIN can be further improved. As we have seen, B&C is frequently slower than the static variant. A more detailed polyhedral study of the model may reveal additional strengthening inequalities. Concerning the computational complexity of EVS-SOC, it is an open question whether or not the problem is NP-hard if $P_v^{\max}$ is a general nonconcave function. Another aspect worth pursuing is the question whether known vehicle arrival times have a significant impact on the charging costs of a rolling horizon schedule. In the presented scenario, successively arriving vehicles are simulated, however they are not incorporated into the schedule before arrival at the charging station. One may expect that arrival times known in advance lead to better exploitation of cheap charging time slots and therefore come along with cheaper total charging costs.

A further direction of future work should be the consideration of uncertainties, e.g.,

uncertain power limits or the uncertain occupation of charging stations. Moreover, we usually do know the $P_v^{\max}$ curve in advance, but might be able to approximate it over charging time. Furthermore it would be interesting to study the effect of the rescheduling interval on charging costs and charging errors in the rolling horizon context. Last but not least, it would be interesting to consider a problem variant in which discharging of vehicles is allowed in order to enable mutual charging of EVs. This idea is referred to as vehicle-to-grid and has already been mentioned in [143], however its impact on the total charging costs has not yet been studied. One could further extend the model by allowing the charging station to supply energy to the electricity grid in exchange for monetary reward.

CHAPTER 6

Conclusion and Future Work

In the first two parts of this thesis we considered the problem of distributing service points for mobility applications. In the context of mobility applications a service point may refer to a charging station or battery swapping station for electric vehicles or a station of a vehicle sharing system where customers can pick up and return vehicles. Service points are an important part of a mobility service's infrastructure and an optimal placement of these service points is crucial for a fruitful operability. However, deciding where said service points should be placed is a challenging problem. In the literature such a problem is usually solved in two phases. In the first phase the necessary data about the community and environment in order to estimate user demands and other information, is acquired. In the second phase, based on the obtained demand information the placement of service points is optimized. Traditionally, the data acquisition and the optimization step are considered in a separated fashion.

Even when the demand information is completely known, finding optimal locations for service points still remains a difficult problem. To emphasize this aspect, we have considered in Chapter 3 the Multi-Period Battery Swapping Station Location Problem (MBSSLP), a novel problem formulation for placing battery swapping stations for electric scooters in an urban area. To the best of our knowledge the aspect of recharging and reusing returned batteries and its implications concerning station capacities when optimizing station locations and configurations has not yet been investigated in previous work. Due to various considerations, such as a dropout of customers, capacities of stations, as well as the rechargeable batteries the problem was too complex to be solved reasonably well with pure mixed integer programming approaches for instances with more than 1000 stations and 2000 origin-destination pairs. Hence, a Large Neighborhood Search (LNS) was proposed for solving larger instances. The LNS was realized as a matheuristic, as a mixed integer linear program was used for repairing solutions. Our LNS was able to find reasonable solutions for instances with up to 2000 stations and 8000 origin-destination pairs. In future work it seems promising to improve the strategies of destroying and

repairing solutions within the LNS. Specifically, the presented strategies are random to a large degree and do not consider how the remaining/allocated demand is influenced by locations previously added to the repair and destroy set, respectively. Moreover, adaptive mechanisms for choosing among different destroy and re-create methods may be useful. In order to be able to solve even larger instances, extensions utilizing, e.g., (hierarchical) clustering and multilevel refinement should be considered in future work.

In Chapter 4 a Cooperative optimization Approach (COA) was proposed for solving the demand acquisition problem as well as the station location problem in a single process. COA offers an attractive alternative demand acquisition method, as it does not require any previous data, and can easily be transferred to other application scenarios and locations and is potentially cheaper than buying data for estimating customer demands. The COA framework consists of three major components, the feedback component (FC), the evaluation component (EC), and the optimization component (OC). We have shown that the individual components can easily be realized in various ways. For example as optimization approach, one may use white-box or black-box algorithms as well as exact or heuristic approaches. We specifically have developed a particularly efficient LNS that outperforms solving a mixed integer linear programming model with a generic solver by orders of magnitude. This was achieved by to modeling the non-trivial objective function of the problem to be solved as a graph that makes it possible to efficiently update the objective value of a solution whenever it was modified.

Additionally, in order to learn users preferences, two different machine learning based surrogate models were investigated in the EC. In one model users and stations were considered independently of each other while the other model explicitly tried to exploit similar preferences of users. Ultimately, it turned out that the latter method was able learn user preferences significantly better than the former method. Moreover, by also considering that user feedback is not missing at random, we were able to improve the learning mechanism even further. However, the advantage of considering users independently of each other in the surrogate function is that the associated models cannot only be trained independently of each other but also in parallel. Recall that without the learning mechanism COA can be compared to a classical approach in the sense that it optimizes service point locations based on previously derived demand information. Our results clearly show that using our learning mechanism in COA we can achieve vastly better results than without it based on the same known demand information. Additionally, in comparison to classical approaches COA is much more flexible in how much information can be asked from individual users.

For future work there is still potential left for the further development of COA. An open problem is to investigate how the way in which user feedback is obtained affects the rate at which the surrogate function improves over the iterations of COA. Specifically, the order in which the user feedback is obtained may have a significant impact on how quickly the surrogate function can improve. Furthermore alternative strategies for generating location scenarios in the FC may also be considered which utilize knowledge not only from the EC but also from the OC, such as deriving additional information about the

importance of locations via a sensitivity analysis of a linear program.

While COA addresses many problems of the classical procedure for distributing service points, COA also comes with several challenges that need to be properly addressed when applied in a real-world scenario. The focus of this contribution was on the algorithmic and computational aspects of COA and its components. The presented problem formulations that are solved with COA were intentionally still rather general and abstract as the focus is put on the core principles and the interaction of the different components of the approach. Clearly, further challenges concern a suitable user interface and a corresponding distributed implementation of at least the feedback component, in which also psychological aspects of users need to be considered. Moreover, the performed experiments are based on the assumption of perfect user feedback, which does not hold in practice. The impacts of not entirely reliable evaluation results need to be studied, and robust variants of certain components of COA devised. For example, users might accidentally provide erroneous feedback. Hence it would be interesting to test the robustness of the EC, investigating how many and what kind of errors users can make for the surrogate function to still produce feasible predictions. Alternatively, designing methods that can detect such errors would make it possible to ignore or even correct erroneous feedback.

So far, interactive optimization algorithms are typically designed for a single person. Algorithms interacting with a large group of users are quite sparse in literature. Therefore, the algorithms developed for COA and the COA framework itself may serve as important basis for similar problems, i.e., problems where preferences or constraints of users are partially unknown or difficult to model. Such problems seem to be especially relevant in the domain of personnel scheduling.

In Chapter 5 we turned to the electric vehicle charging scheduling problem with SOC-dependent maximum charging power (EVS-SOC). We addressed the issue that the maximum charging power at which a vehicle can be charged is dependent on the current state of charge (SOC) of the vehicle. Additionally, using a discretized time horizon, the charging power of an electric vehicle (EV) may be regulated within a single time step. To deal with this issue, we instead considered the energy by which an EV can be charged within a time step. For this purpose, we showed how to derive the maximum charging energy in an exact as well as an approximate way.

We proposed two methods for solving the EVS-SOC. The first one was a cutting plane method utilizing a convex hull of the in general nonconcave glssoc-power curves. The second method was based on a piecewise linearization of the glssoc-energy curve and is effectively solved by branch-and-cut. Our results showed that optimally solving problems with general piecewise linear maximum power functions requires high computation times. However, problems with concave, piecewise linear maximum charging power functions can efficiently be dealt with by means of linear programming. Approximating an EV's maximum charging power with a concave function may result in practically infeasible solutions, due to vehicles potentially not reaching their specified target glssoc. However, our results showed that this error is negligible in practice.

In future work it would be interesting to investigate whether the runtime of solving the branch-and-cut model can be further improved. A more detailed polyhedral study of the model may reveal additional strengthening inequalities. Concerning the computational complexity of EVS-SOC, it is an open question whether or not the problem is NP-hard if a vehicle's maximum charging power is a general nonconcave function. A further direction of future work for the EVS-SOC should be the consideration of uncertainties, e.g., in the future power limits or in the future occupation of charging stations.

In conclusion, in this thesis we have investigated several practically highly relevant and challenging aspects regarding a profitable operability of a service point system. As main part of this thesis we have developed a cooperative approach for distributing service points that addresses many issues of classical approaches for optimizing service point locations. We have shown that COA has therefore the potential to be a valuable alternative to these classical approaches, offering a significantly easier and cheaper way for obtaining user demand data. Additionally, by incorporating users directly into the optimization process, COA is able to consider user preferences on a much finer grained level than the standard approaches. Finally, by allowing users to help creating the service station system, the resulting solution will ultimately be better accepted by the community.

APPENDIX A

Generation of MBSSLP Instances

Here we provide a more detailed description of how our MBSSLP benchmark instances are generated and provide all parameter values used in the generation of the instances.

A.1 Random Instances for the MBSSLP

Battery swapping stations as well as origin and destination locations of customers are located within a square of length $\lceil \xi\sqrt{n} \rceil$ with $\xi = 800$. We generate a network graph $G = (V, E)$ following a similar procedure as used in [66, 68] by first sampling $|V| = 5n$ random points from the square and then constructing an euclidean spanning tree w.r.t. V. Afterwards, n additional randomly chosen edges $(u, v) \in V \times V$ are added to E.

The set of potential battery swapping station locations L is generated by choosing n random nodes from V. Costs for building a station are chosen uniformly at random from $\{50, \ldots, 70\}$ for each station. Costs for adding a battery slot to a station are set to 40. Each battery swapping station can have at most 70 battery slots.

Origin and destination locations are chosen from a random subset $V' \subseteq V$ with $|V'| = \min(\frac{m}{2}, 5n)$. To each $v \in V'$ a random weight γ_v is assigned according to a log-normal distribution with mean $\mu = \ln(100)$ and standard deviation $\sigma = 0.5$. Moreover, we also assign weights γ_q to each OD-pair $q = (u, v) \in V' \times V'$ such that γ_q corresponds to $f_{\text{PDF}}(w(p_q), \mu, \sigma)$ with f_{PDF} being the probability density function of a lognormal distribution with mean $\mu = \ln(5000)$ and standard deviation $\sigma = 0.2$. The total demand d_q^{total} of an O/D-pair $q = (u, v)$ is then calculated as $d_q^{\text{total}} = \gamma_u \cdot \gamma_v \cdot \gamma_q$. We then set Q to be the set of m O/D-pairs q of $V' \times V'$ for which d_q^{total} is highest.

The swapping demand of each O/D-pair is distributed over 24 time periods, $\mathcal{T} = \{1, \ldots, 24\}$ and recharging a battery requires one time period, i.e., $t^c = 1$. We assume each customer to travel twice on his corresponding path, once in the morning to get to work and once in the evening to travel back home, and we assume that customers need

to swap batteries once per trip counted here as demand. The demand during each time period $t \in \mathcal{T}$ is determined by two normal distributions $\mathcal{N}_{\text{morning}}(8, 1)$ and $\mathcal{N}_{\text{evening}}(18, 2)$, respectively. From each distribution 100 samples t are generated and transformed to valid integral values by $t := (\lceil t \rceil \mod t_{\max}) + 1$. Afterwards, d_q^{total} is distributed over $\mathcal{T}$ according to the frequency in which the time periods $t \in \mathcal{T}$ appear in the generated samples.

The maximal deviation distance of the users, $w_{\max}^{\text{detour}}$, is set to $\xi/2$ and the parameters of the distance decay function are set to $\alpha = 100$, $\beta = 0.1$, and $\delta_q = w_{\max}^{\text{detour}}/10$ for all $q \in Q$.

Eight groups of test instances for different combinations of n and m have been generated as described in Section 3.5, and each group consists of thirty instances.

A.2 Manhattan Instance

Next to artificial benchmark instances we also derived an instance from real-world yellow taxi trip data and bus stop shelter data of Manhattan, which we call here Manhattan instance. The underlying street network of the instance corresponds to the street network graph of Manhattan provided by the Python package OSMNX[1]. Origin/Destination pairs of our instance correspond to trips between the taxi zones[2] of Manhattan. The partitioning of Manhattan into taxi zones is shown in Figure 3.3. For each taxi zone one random origin and one random destination location were chosen from the set of nodes of the network graph that are associated with the corresponding taxi zone.

The set of O/D-pairs and their corresponding demands have been derived from the 2016 Yellow Taxi Trip Data[3]. The taxi data set was first preprocessed and all trips with invalid data as well as trips made on a weekend have been removed from the data set. Furthermore, we have also removed all trips which do not start and end in Manhattan. From the preprocessed data set we then extracted for each trip the pickup time, the pickup zone, the drop-off zone, as well as the passenger count. Each pickup time was rounded down to the nearest hour and afterwards an average daily passenger count for each triple (pickup hour, pickup zone, drop-off zone) was calculated. In total, the final table contains 4498 unique pickup/drop-off zone pairs which also constitute the instance's set of O/D pairs Q. These passenger counts correspond to the hourly demands d_q^t of the O/D pairs $q \in Q$.

For the distance decay function and $w_{\max}^{\text{detour}}$ we use the same parameters as for the artificial benchmark instances.

[1] https://github.com/gboeing/osmnx

[2] https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

[3] https://data.cityofnewyork.us/Transportation/2016-Yellow-Taxi-Trip-Data/k67s-dv2t

The set of potential battery swapping station locations L is derived from the bus stop shelters [4] of Manhattan by selecting 500 locations randomly.

As shown in Figure 3.2 left the demand at each hour is quite high. Therefore we choose a capacity limit of 200 for each battery swapping station, The costs for building a station as well for adding a battery charging slot are chosen as for the artificial instances.

[4] https://data.cityofnewyork.us/Transportation/Bus-Stop-Shelters/qafz-7myz

APPENDIX B

EVS-SOC-GLIN - Additional Results

In Figure B.1 a comparison between the original $P_v^{\max}$ and the simpler piecewise approximations is shown for all vehicle types used in the benchmark instances.

Tables B.1, B.2, B.3, and B.4 give more detailed information to the results provided in Tables 5.3, 5.4, 5.5, and 5.6, respectively. Shown here are also the numbers of optimally solved instances in each instance groups as well as standard deviations to the runtimes and the numbers of cuts.

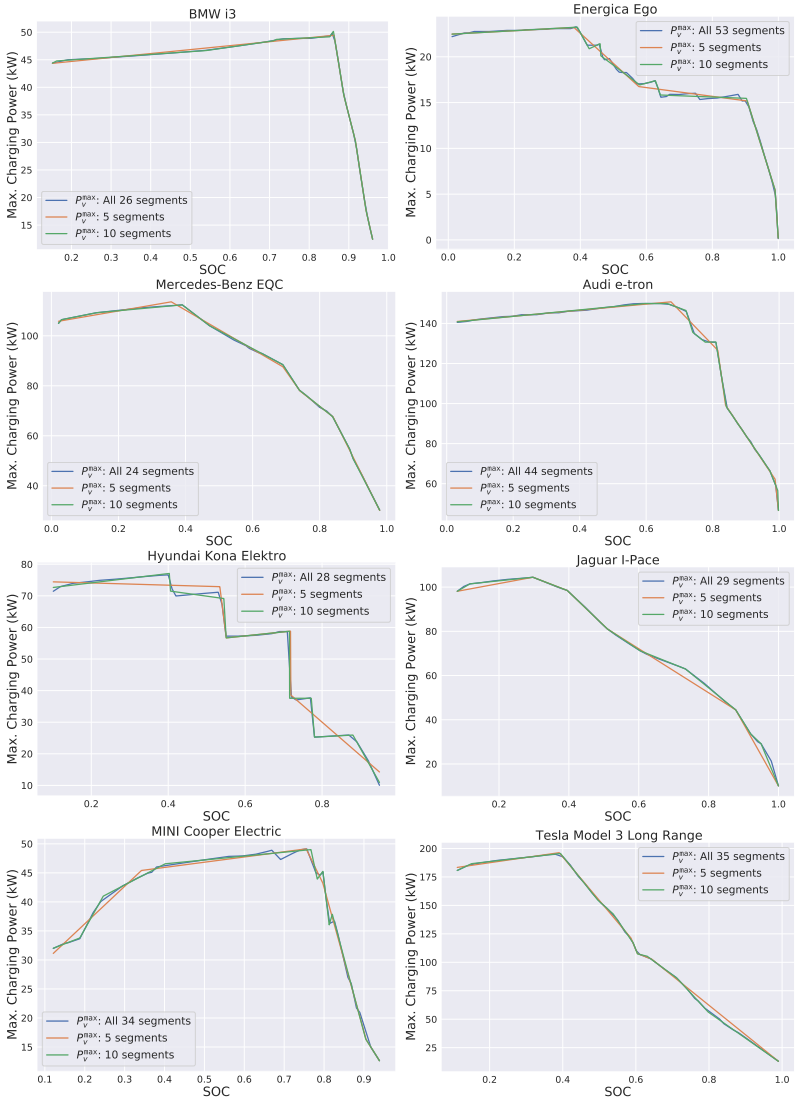


Figure B.1: Comparison of $P_v^{\max}$ curves with different numbers of segments.

Table B.1: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 10n$.

n	Δt (min)	n_{seg}	n_{opt}		n_{feas}		Runtime (s)				n_{cuts}		%-gap	
							Median		StdDev		Median	StdDev	Median	
		Mean	Static	B&C	Static	B&C	Static	B&C	Static	B&C	B&C		Static	B&C
$E_v^{\text{max-lb}}$														
5	1	155	24	24	30	30	391.75	43.39	662.53	720.28	1038	1944	0.01	0.01
5	5	139	30	30	30	30	6.58	1.43	17.30	64.96	144	307	0.00	0.01
5	10	119	30	30	30	30	1.67	0.83	6.66	1.77	56	120	0.00	0.00
10	1	311	5	12	21	29	1800.00	1800.00	389.82	764.87	4068	2726	0.03	0.03
10	5	279	29	27	30	30	79.94	8.84	353.48	544.47	498	581	0.01	0.01
10	10	242	30	30	30	30	7.04	2.06	35.58	4.31	194	167	0.00	0.01
20	1	612	0	1	2	11	1800.00	1800.00	0.00	181.68	8974	1820	0.08	0.19
20	5	553	23	19	30	30	500.49	684.63	640.95	781.48	1846	1081	0.01	0.01
20	10	475	30	29	30	30	40.18	13.35	71.71	425.71	505	274	0.01	0.01
50	1	1544	0	0	0	0	1800.00	1800.00	0.00	0.00	15910	2518	-	-
50	5	1393	2	2	26	30	1800.00	1800.00	174.76	351.60	6106	1249	0.05	0.05
50	10	1192	29	18	30	30	307.62	827.59	458.49	779.32	1930	594	0.01	0.01
100	1	3095	0	0	0	0	1800.00	1800.00	0.00	0.00	11886	3940	-	-
100	5	2796	0	0	9	9	1800.00	1800.00	0.00	0.00	9961	1319	0.08	0.12
100	10	2399	11	4	30	30	1800.00	1800.00	418.13	452.22	4434	861	0.01	0.03
$E_v^{\text{max-ex}}$														
5	1	901	3	12	8	27	1800.00	1800.00	414.46	834.31	5304	6069	0.03	0.01
5	5	901	30	26	30	30	143.42	9.59	312.39	628.62	820	1553	0.00	0.00
5	10	901	30	30	30	30	34.53	2.60	106.85	48.02	319	623	0.00	0.00
10	1	1802	0	3	1	21	1800.00	1800.00	0.00	401.91	13982	7431	0.04	0.08
10	5	1802	13	16	29	30	1800.00	725.37	568.29	872.54	2858	2600	0.01	0.01
10	10	1802	30	28	30	30	201.32	10.29	327.78	451.48	680	845	0.00	0.01
20	1	3605	0	0	0	10	1800.00	1800.00	0.00	0.00	23449	6856	-	0.14
20	5	3605	2	6	14	30	1800.00	1800.00	85.05	629.36	6479	4009	0.07	0.05
20	10	3605	22	18	30	30	1038.91	116.59	569.76	862.58	1507	1708	0.01	0.01
50	1	9041	0	0	0	0	1800.00	1800.00	0.00	0.00	6856	4528	-	-
50	5	9041	0	1	0	23	1800.00	1800.00	0.00	308.85	15048	3971	-	0.11
50	10	9041	1	7	4	30	1800.00	1800.00	123.65	585.67	6160	3202	0.18	0.03
100	1	18078	0	0	0	0	-	1800.00	-	0.00	0	5698	-	-
100	5	18086	0	0	0	10	1800.00	1800.00	0.00	0.00	18944	5630	-	0.08
100	10	18086	0	2	0	25	1800.00	1800.00	0.00	393.05	10750	3536	-	0.06

Table B.2: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 25n$.

n	Δt (min)	n_{seg}	n_{opt}		n_{feas}		Runtime (s)				n_{cuts}		%-gap	
							Median		StdDev		Median	StdDev	Median	
		Mean	Static	B&C	Static	B&C	Static	B&C	Static	B&C	B&C		Static	B&C
$E_v^{\text{max-lb}}$														
5	1	155	12	5	29	30	1800.00	1800.00	690.82	636.49	3184	3153	0.02	0.06
5	5	139	28	26	30	30	25.52	6.68	450.62	616.54	422	461	0.01	0.01
5	10	119	30	29	30	30	1.27	1.62	10.37	329.09	153	154	0.01	0.01
10	1	312	1	0	20	23	1800.00	1800.00	29.78	0.00	7298	2829	0.10	0.12
10	5	279	29	19	30	30	183.39	770.59	445.94	815.87	1132	771	0.01	0.01
10	10	242	30	29	30	30	17.87	11.88	163.75	437.28	452	223	0.01	0.01
20	1	612	0	0	4	3	1800.00	1800.00	0.00	0.00	11938	2372	0.26	0.28
20	5	553	14	6	30	30	1800.00	1800.00	608.45	637.22	2702	936	0.01	0.05
20	10	475	29	22	30	30	60.59	201.06	422.28	755.21	967	359	0.01	0.01
50	1	1544	0	0	0	0	1800.00	1800.00	0.00	0.00	22034	4220	-	-
50	5	1393	2	1	29	30	1800.00	1800.00	160.23	280.55	6997	1257	0.08	0.11
50	10	1192	20	5	30	30	902.21	1800.00	698.40	604.91	2575	653	0.01	0.03
100	1	3095	0	0	0	0	1800.00	1800.00	0.00	0.00	29193	6236	-	-
100	5	2796	0	0	14	7	1800.00	1800.00	0.00	0.00	11737	1494	0.12	0.18
100	10	2399	6	0	30	30	1800.00	1800.00	482.70	0.00	5340	1077	0.03	0.06
$E_v^{\text{max-ex}}$														
5	1	901	1	1	9	25	1800.00	1800.00	274.48	138.10	15258	9382	0.21	0.20
5	5	901	26	17	30	30	448.47	761.59	644.73	831.42	2153	2330	0.01	0.01
5	10	901	29	26	30	30	56.12	16.43	321.42	610.48	866	990	0.00	0.01
10	1	1802	0	0	1	18	1800.00	1800.00	0.00	0.00	23328	9977	0.23	0.33
10	5	1802	12	7	26	30	1800.00	1800.00	580.44	699.52	5220	3467	0.04	0.06
10	10	1802	29	22	30	30	204.26	233.60	417.05	757.12	2063	1389	0.01	0.01
20	1	3605	0	0	0	2	1800.00	1800.00	0.00	0.00	17970	9466	-	0.32
20	5	3605	1	1	15	29	1800.00	1800.00	113.34	318.65	10784	4058	0.08	0.12
20	10	3605	20	10	29	30	1097.26	1800.00	573.95	709.09	4647	2500	0.01	0.03
50	1	9041	0	0	0	0	1800.00	1800.00	0.00	0.00	23986	9245	-	-
50	5	9041	0	0	0	17	1800.00	1800.00	0.00	0.00	23708	5721	-	0.18
50	10	9041	0	3	16	28	1800.00	1800.00	0.00	439.63	12160	4186	0.04	0.08
100	1	18086	0	0	0	0	1800.00	1800.00	0.00	0.00	0	4697	-	-
100	5	18086	0	0	0	0	1800.00	1800.00	0.00	0.00	25754	8585	-	-
100	10	18086	0	0	0	19	1800.00	1800.00	0.00	0.00	19752	5121	-	0.09

Table B.3: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ and $E_v^{\text{max-ex}}$ based on the original $P_v^{\max}$ functions and $P^{\text{gridmax}} = 40n$.

n	Δt (min)	n_{seg}	n_{opt}		n_{feas}		Runtime (s)				n_{cuts}		%-gap	
							Median		StdDev		Median	StdDev	Median	
		Mean	Static	B&C	Static	B&C	Static	B&C	Static	B&C	B&C		Static	B&C
$E_v^{\text{max-lb}}$														
5	1	155	11	2	29	29	1800.00	1800.00	542.57	57.30	4476	2923	0.04	0.15
5	5	139	28	24	30	30	31.04	55.93	513.40	715.78	619	492	0.01	0.01
5	10	119	30	29	30	30	2.49	4.05	53.44	371.17	247	153	0.01	0.01
10	1	311	0	0	20	20	1800.00	1800.00	0.00	0.00	8161	3130	0.21	0.17
10	5	279	21	8	30	30	301.14	1800.00	745.75	677.68	1410	676	0.01	0.03
10	10	242	28	26	30	30	27.80	36.06	450.92	660.00	456	201	0.01	0.01
20	1	612	0	0	2	1	1800.00	1800.00	0.00	0.00	13361	2440	0.27	0.48
20	5	553	5	0	30	30	1800.00	1800.00	365.20	0.00	2863	884	0.04	0.10
20	10	475	28	19	30	30	69.51	571.16	479.77	745.04	1078	327	0.01	0.01
50	1	1544	0	0	0	0	1800.00	1800.00	0.00	0.00	25908	3569	-	-
50	5	1393	0	0	28	28	1800.00	1800.00	0.00	0.00	7110	1096	0.12	0.21
50	10	1192	18	1	30	30	1097.80	1800.00	640.13	183.90	2748	520	0.01	0.05
100	1	3095	0	0	0	0	1800.00	1800.00	0.00	0.00	29066	6072	-	-
100	5	2796	0	0	7	2	1800.00	1800.00	0.00	0.00	11782	1239	0.22	0.21
100	10	2399	1	0	29	30	1800.00	1800.00	121.93	0.00	5650	808	0.06	0.10
$E_v^{\text{max-ex}}$														
5	1	901	2	0	9	24	1800.00	1800.00	261.72	0.00	20190	9588	0.23	0.44
5	5	901	25	9	30	30	582.18	1800.00	651.87	643.80	3180	2231	0.01	0.07
5	10	901	30	23	30	30	80.12	34.07	160.32	753.56	1228	955	0.00	0.01
10	1	1802	0	0	1	13	1800.00	1800.00	0.00	0.00	24450	8643	0.49	0.77
10	5	1802	12	0	26	30	1800.00	1800.00	598.34	0.00	6026	3161	0.02	0.17
10	10	1802	29	17	30	30	245.17	1147.26	375.49	837.79	2161	1553	0.01	0.01
20	1	3605	0	0	0	0	1800.00	1800.00	0.00	0.00	17460	9716	-	-
20	5	3605	0	0	15	29	1800.00	1800.00	0.00	0.00	13276	3457	0.14	0.22
20	10	3605	19	3	29	30	1437.18	1800.00	550.74	447.72	5692	2190	0.01	0.08
50	1	9041	0	0	0	0	1800.00	1800.00	0.00	0.00	12253	7961	-	-
50	5	9041	0	0	0	11	1800.00	1800.00	0.00	0.00	27617	4805	-	0.21
50	10	9041	0	0	14	27	1800.00	1800.00	0.00	0.00	13538	2670	0.10	0.12
100	1	18083	0	0	0	0	-	1800.00	-	0.00	0	9122	-	-
100	5	18086	0	0	0	0	1800.00	1800.00	0.00	0.00	31692	13113	-	-
100	10	18086	0	0	0	11	1800.00	1800.00	0.00	0.00	23081	4035	-	0.14

Table B.4: EVS-SOC-GLIN results for solving the static model versus B&C with $E_v^{\text{max-lb}}$ based on five-segment piecewise linear approximations of the original $P_v^{\max}$ functions, $P^{\text{gridmax}} = 25n$.

n	Δt (min)	n_{seg}	n_{opt}		n_{feas}		Runtime (s)				n_{cuts}		%-gap	
							Median		StdDev		Median	StdDev	Median	
		Mean	Static	B&C	Static	B&C	Static	B&C	Static	B&C	B&C		Static	B&C
$E_v^{\text{max-lb}}$														
5	1	40	29	22	30	30	60.14	19.63	394.28	791.01	387	485	0.01	0.01
5	5	46	30	30	30	30	2.40	1.98	5.97	263.17	88	102	0.01	0.01
5	10	43	30	30	30	30	0.64	1.13	1.37	1.21	42	50	0.00	0.01
10	1	80	27	13	30	30	509.28	1800.00	582.27	830.23	1162	639	0.01	0.02
10	5	92	30	30	30	30	11.01	8.34	28.13	224.13	232	136	0.01	0.01
10	10	87	30	30	30	30	1.49	2.68	1.78	8.36	118	62	0.01	0.01
20	1	160	5	2	12	30	1800.00	1800.00	193.77	407.09	2488	722	0.03	0.06
20	5	185	30	25	30	30	54.58	61.09	199.06	659.96	516	192	0.01	0.01
20	10	174	30	30	30	30	5.03	7.45	13.02	37.35	217	79	0.01	0.01
50	1	398	0	0	0	12	1800.00	1800.00	0.00	0.00	5598	796	-	0.24
50	5	459	28	10	30	30	640.74	1800.00	516.17	754.54	1556	363	0.01	0.02
50	10	433	30	29	30	30	37.23	36.95	54.09	379.05	624	160	0.01	0.01
100	1	798	0	0	0	0	1800.00	1800.00	0.00	0.00	9312	1458	-	-
100	5	921	12	3	30	30	1800.00	1800.00	466.38	464.39	3237	568	0.01	0.06
100	10	871	30	25	30	30	112.16	84.83	156.15	652.92	1360	259	0.01	0.01

List of Figures

List of Tables

List of Algorithms

Acronyms

ALS Alternating Least Squares. 24, 48

BB Branch-and-Bound. 10, 12

CDFRLM Capacitated Deviation-Flow Refueling Location Model. 27

COA Cooperative optimization Approach. 2–5, 41–45, 49–53, 57, 60, 61, 65–68, 70, 72, 73, 76–81, 84, 85, 87–90, 126–128, 139, 141

CSPDP Cooperative Service Point Distribution Problem. 2, 3, 5, 49, 50, 52, 85

DFRLM Deviation Flow Refueling Location Model. 27

EC Evaluation Component. 50–52, 55, 60, 65, 67, 69, 72, 76, 81, 88–90

EV electric vehicle. 1, 4–6, 46, 91–106, 124, 127, 140, 141

EVS-SOC EV charging scheduling problem with SOC-dependent maximum charging power. 95

FC Feedback Component. 50, 51, 53, 54, 60, 65, 66, 76, 78, 90

FCLM Flow Capturing Location Model. 27

FLP Capacitated Multiple Allocation Fixed Charge Facility Location Problem. 27, 45

FRLM Flow Refueling Location Model. 27

GSPDP Generalized Service Point Distribution Problem. 62–66, 70, 72, 85, 86, 89, 139

GVNS General Variable Neighborhood Search. 16, 17

ISPDP Independent Service Point Distribution Problem. 52, 53, 61, 73, 85, 86, 88

LNS Large Neighborhood Search. xiv, 2–5, 17, 25, 26, 30, 35–39, 42, 70, 76, 78, 84–86, 89, 125, 126, 139, 141

LP Linear Program. 8–12, 39, 100, 101, 106, 108, 110, 140

LP-BB LP-based Branch-and-Bound. 11

LPH Linear Programming Heuristic. 31, 35–38, 139, 141

MBSSLP Multi-Period Battery Swapping Station Location Problem. 28–31, 35, 40, 125

MCLP Maximal Covering Location Problem. 64, 65

MILP Mixed Integer Linear Program. xiv, 2–5, 10–12, 17, 25, 26, 28–31, 35–39, 58, 64, 70, 76, 78, 84–86, 89, 90, 93, 95, 101, 106, 108, 109, 122, 139, 141

MSE mean squared error. 19, 20, 56, 81, 82, 139

OC Optimization Component. 50, 52, 54, 55, 57–60, 65, 69, 76, 78, 89, 90

PBIG Population-Based Iterated Greedy. 14, 42, 57, 59, 60, 141

RCL restricted candidate list. 13

ReLU Rectified Linear Unit. 23, 56

SGD Stochastic Gradient Descent. 23, 24, 48

SMC Solution Management Component. 50, 52, 53, 59, 60, 65–68

SOC state of charge. xiv, xv, 4, 5, 91–102, 104, 106, 119–121, 123, 127, 140

SPDP Service Point Distribution Problem. 1, 2, 5, 41, 45, 46, 49

SPR service point requirement. 63–76, 78, 81, 84, 139

VND Variable Neighborhood Descent. 15–17

VNS Basic Variable Neighborhood Search. 16, 41, 42, 57, 59, 60, 87

VSS vehicle sharing systems. 45, 46

Bibliography

[1] F. Sioshansi and J. Webb, "Transitioning from conventional to electric vehicles: The effect of cost and environmental drivers on peak oil demand," *Economic Analysis and Policy*, vol. 61, pp. 7–15, 2019. Special issue on: Future of transport.

[2] S. Küfeoğlu and D. Khah Kok Hong, "Emissions performance of electric vehicles: A case study from the united kingdom," *Applied Energy*, vol. 260, p. 114241, 2020.

[3] Z. Wu, M. Wang, J. Zheng, X. Sun, M. Zhao, and X. Wang, "Life cycle greenhouse gas emission reduction potential of battery electric vehicle," *Journal of Cleaner Production*, vol. 190, pp. 462–470, 2018.

[4] B. Ballinger, M. Stringer, D. R. Schmeda-Lopez, B. Kefford, B. Parkinson, C. Greig, and S. Smart, "The vulnerability of electric vehicle deployment to critical mineral supply," *Applied Energy*, vol. 255, p. 113844, 2019.

[5] D. Gavalas, C. Konstantopoulos, and G. Pantziou, "Design and management of vehicle-sharing systems: A survey of algorithmic approaches," in *Smart Cities and Homes* (M. S. Obaidat and P. Nicopolitidi, eds.), pp. 261–289, Elsevier, 2016.

[6] C. Kloimüllner and G. R. Raidl, "Hierarchical clustering and multilevel refinement for the bike-sharing station planning problem," in *International Conference on Learning and Intelligent Optimization*, pp. 150–165, Springer, 2017.

[7] Y. Xu, S.-L. Shaw, Z. Fang, and L. Yin, "Estimating potential demand of bicycle trips from mobile phone data—an anchor-point based approach," *ISPRS International Journal of Geo-Information*, vol. 5, no. 8, 2016.

[8] C. Wang, J. Bi, Q. Sai, and Z. Yuan, "Analysis and prediction of carsharing demand based on data mining methods," *Algorithms*, vol. 14, no. 6, 2021.

[9] M. Schmidt, P. Zmuda-Trzebiatowski, M. Kiciński, P. Sawicki, and K. Lasak, "Multiple-criteria-based electric vehicle charging infrastructure design problem," *Energies*, vol. 14, no. 11, 2021.

[10] A. Almaghrebi, F. Aljuheshi, M. Rafaie, K. James, and M. Alahmad, "Data-driven charging demand prediction at public charging stations using supervised machine learning regression methods," *Energies*, vol. 13, no. 16, 2020.

[11] A. Awasthi, K. Venkitusamy, S. Padmanaban, R. Selvamuthukumaran, F. Blaabjerg, and A. K. Singh, "Optimal planning of electric vehicle charging station at the distribution system using hybrid optimization algorithm," *Energy*, vol. 133, pp. 70–78, 2017.

[12] J. Cavadas, G. d. A. C. Homem, and J. Gouveia, "A MIP model for locating slow-charging stations for electric vehicles in urban areas accounting for driver tours," *Transportation Research Part E: Logistics and Transportation Review*, vol. 75, pp. 188 – 201, 2015.

[13] H.-Y. Mak, Y. Rong, and Z.-J. M. Shen, "Infrastructure Planning for Electric Vehicles with Battery Swapping," *Management Science*, vol. 59, no. 7, pp. 1557–1575, 2013.

[14] M. Zeng, Y. Pan, D. Zhang, Z. Lu, and Y. Li, "Data-driven location selection for battery swapping stations," *IEEE Access*, vol. 7, pp. 133760–133771, 2019.

[15] E. Molin, P. Mokhtarian, and M. Kroesen, "Multimodal travel groups and attitudes: A latent class cluster analysis of dutch travelers," *Transportation Research Part A: Policy and Practice*, vol. 83, pp. 14–29, 2016.

[16] A. Radzimski and M. Dzięcielski, "Exploring the relationship between bike-sharing and public transport in poznań, poland," *Transportation Research Part A: Policy and Practice*, vol. 145, pp. 189–202, 2021.

[17] L. Shi and K. Rasheed, "Asaga: an adaptive surrogate-assisted genetic algorithm," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1049–1056, ACM, 2008.

[18] R. M. Bell, Y. Koren, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 08, pp. 30–37, 2009.

[19] R. Devooght, N. Kourtellis, and A. Mantrach, "Dynamic matrix factorization with priors on unknown values," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 189–198, ACM, 2015.

[20] N. Korolko and Z. Sahinoglu, "Robust optimization of EV charging schedules in unregulated electricity markets," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 149–157, 2017.

[21] J. Han, J. Park, and K. Lee, "Optimal scheduling for electric vehicle charging under variable maximum charging power," *Energies*, vol. 10, no. 7, 2017.

[22] J. J. Mies, J. R. Helmus, and R. Van den Hoed, "Estimating the charging profile of individual charge sessions of electric vehicles in the netherlands," *World Electric Vehicle Journal*, vol. 9, no. 2, 2018.

[23] O. Frendo, J. Graf, N. Gaertner, and H. Stuckenschmidt, "Data-driven smart charging for heterogeneous electric vehicle fleets," *Energy and AI*, vol. 1, 2020.

[24] T. Jatschka, F. F. Oberweger, T. Rodemann, and G. R. Raidl, "Distributing battery swapping stations for electric scooters in an urban area," in *Optimization and Applications, Proceedings of OPTIMA 2020 – XI International Conference Optimization and Applications* (N. Olenev, Y. Evtushenko, M. Khachay, and V. Malkova, eds.), vol. 12422 of *LNCS*, pp. 150–165, Springer, 2020.

[25] T. Jatschka, T. Rodemann, and G. R. Raidl, "A cooperative optimization approach for distributing service points in mobility applications," in *Evolutionary Computation in Combinatorial Optimization* (A. Liefooghe and L. Paquete, eds.), vol. 11452 of *LNCS*, pp. 1–16, Springer, 2019.

[26] T. Jatschka, T. Rodemann, and G. R. Raidl, "VNS and PBIG as optimization cores in a cooperative optimization approach for distributing service points," in *Computer Aided Systems Theory – EUROCAST 2019*, vol. 12013 of *LNCS*, pp. 255–262, Springer, 2020.

[27] T. Jatschka, T. Rodemann, and G. R. Raidl, "Exploiting similar behavior of users in a cooperative optimization approach for distributing service points in mobility applications," in *The 5th International Conference on machine Learning, Optimization and Data science – LOD 2019* (G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, eds.), LNCS, pp. 738–750, Springer, 2019.

[28] T. Jatschka, G. R. Raidl, and T. Rodemann, "A general cooperative optimization approach for distributing service points in mobility applications," *Algorithms*, vol. 14, no. 8, 2021.

[29] T. Jatschka, T. Rodemann, and G. R. Raidl, "A large neighborhood search for a cooperative optimization approach to distribute service points in mobility applications," in *Metaheuristics and Nature Inspired Computing* (B. Dorronsoro, F. Yalaoui, E.-G. Talbi, and G. Danoy, eds.), vol. 1541 of *CCIS*, pp. 3–17, Springer, 2022.

[30] B. Schaden, T. Jatschka, S. Limmer, and G. R. Raidl, "Smart charging of electric vehicles considering SOC-dependent maximum charging powers," *Energies*, vol. 14, no. 22, 2021.

[31] B. Schaden, "Scheduling the charging of electric vehicles with soc-dependent maximum charging power," Master's thesis, TU Wien, 2021. Supervised by G. R. Raidl and T. Jatschka.

[32] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[33] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, vol. 6. Athena Scientific Belmont, MA, 1997.

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

[35] L. Wolsey, *Integer Programming.* Wiley Series in Discrete Mathematics and Optimization, Wiley, 1998.

[36] T. Jatschka, "An iterative time-bucket refinement algorithm for high resolution scheduling problems," Master's thesis, TU Wien, 2017.

[37] L. G. Khachiyan, "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 20, no. 1, pp. 53–72, 1980.

[38] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pp. 302–311, ACM, 1984.

[39] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," *New York*, 1951.

[40] C. H. Papadimitriou, "On the complexity of integer programming," *Journal of the ACM (JACM)*, vol. 28, no. 4, pp. 765–768, 1981.

[41] T. Stützle and R. Ruiz, "Iterated greedy," in *Handbook of Heuristics* (R. Martí, P. M. Pardalos, and M. G. C. Resende, eds.), pp. 547–577, Springer, 2018.

[42] W. Michiels, E. H. L. Aarts, and J. Korst, "Theory of local search," in *Handbook of Heuristics* (R. Martí, P. M. Pardalos, and M. G. C. Resende, eds.), pp. 299–339, Springer, 2018.

[43] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez, "Variable neighborhood search," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 57–97, Springer, 2019.

[44] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 99–127, Springer, 2019.

[45] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 1–35, Springer, 2019.

[46] S. Bouamama, C. Blum, and A. Boukerram, "A population-based iterated greedy algorithm for the minimum weight vertex cover problem," *Applied Soft Computing Journal*, vol. 12, no. 6, pp. 1632–1639, 2012.

[47] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[48] T. Mitchell, *Machine learning*. McGraw-Hill series in computer science, McGraw hill Burr Ridge, 1 ed., 1997.

[49] C. M. Bishop, "Pattern recognition," *Machine learning*, vol. 128, no. 9, 2006.

[50] J. H. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*. Springer, 2017.

[51] C. C. Aggarwal, *An Introduction to Neural Networks*, pp. 1–52. Springer, 2018.

[52] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems," *Foundations and Trends in Human–Computer Interaction*, vol. 4, no. 2, pp. 81–173, 2011.

[53] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[54] O. Chapelle, B. Schölkopf, and A. Zien, eds., *Semi-Supervised Learning*. The MIT Press, 2006.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[56] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[57] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 43–52, 2007.

[58] L. Cooper, "Location-allocation problems," *Operations Research*, vol. 11, no. 3, pp. 331–343, 1963.

[59] G. Laporte, S. Nickel, and F. S. da Gama, eds., *Location science*. Springer, 2015.

[60] V. Verter and S. D. Lapierre, "Location of preventive health care facilities," *Annals of Operations Research*, vol. 110, no. 1, pp. 123–132, 2002.

[61] O. Berman, R. C. Larson, and N. Fouska, "Optimal location of discretionary service facilities," *Transportation Science*, vol. 26, no. 3, pp. 201–211, 1992.

[62] J.-G. Kim and M. Kuby, "The deviation-flow refueling location model for optimizing a network of refueling stations," *International Journal of Hydrogen Energy*, vol. 37, no. 6, pp. 5406–5420, 2012.

[63] M. H. F. Zarandi, S. Davari, and S. A. H. Sisakht, "The large-scale dynamic maximal covering location problem," *Mathematical and Computer Modelling*, vol. 57, no. 3, pp. 710–719, 2013.

[64] M. J. Hodgson, "A flow-capturing location-allocation model," *Geographical Analysis*, vol. 22, no. 3, pp. 270–279, 1990.

[65] R. Church and C. ReVelle, "The maximal covering location problem," in *Papers in Regional Science*, vol. 32, pp. 101–118, Springer, 1974.

[66] M. Hosseini, S. MirHassani, and F. Hooshmand, "Deviation-flow refueling location problem with capacitated facilities: Model and algorithm," *Transportation Research Part D: Transport and Environment*, vol. 54, pp. 269–281, 2017.

[67] M. Kuby and S. Lim, "The flow-refueling location problem for alternative-fuel vehicles," *Socio-Economic Planning Sciences*, vol. 39, no. 2, pp. 125–145, 2005.

[68] I. Capar, M. Kuby, V. J. Leon, and Y.-J. Tsai, "An arc cover–path-cover formulation and strategic analysis of alternative-fuel station locations," *European Journal of Operational Research*, vol. 227, no. 1, pp. 142–151, 2013.

[69] S. A. MirHassani and R. Ebrazi, "A flexible reformulation of the refueling station location problem," *Transportation Science*, vol. 47, no. 4, pp. 617–628, 2013.

[70] C. Upchurch, M. Kuby, and S. Lim, "A model for location of capacitated alternative-fuel stations," *Geographical Analysis*, vol. 41, no. 1, pp. 85–106, 2009.

[71] S. H. Chung and C. Kwon, "Multi-period planning for electric car charging station locations: A case of korean expressways," *European Journal of Operational Research*, vol. 242, no. 2, pp. 677–687, 2015.

[72] R. Bapna, L. S. Thakur, and S. K. Nair, "Infrastructure development for conversion to environmentally friendly fuel," *European Journal of Operational Research*, vol. 142, no. 3, pp. 480–496, 2002.

[73] B. Sun, X. Tan, and D. H. K. Tsang, "Optimal charging operation of battery swapping stations with qos guarantee," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 13–18, Nov 2014.

[74] M. Ghamami, A. Zockaie, and Y. M. Nie, "A general corridor model for designing plug-in electric vehicle charging infrastructure to support intercity travel," *Transportation Research Part C: Emerging Technologies*, vol. 68, pp. 389–402, 2016.

[75] M. J. Kuby, S. B. Kelley, and J. Schoenemann, "Spatial refueling patterns of alternative-fuel and gasoline vehicle drivers in los angeles," *Transportation Research Part D: Transport and Environment*, vol. 25, pp. 84–92, 2013.

[76] P. Murali, F. Ordóñez, and M. M. Dessouky, "Facility location under demand uncertainty: Response to a large-scale bio-terror attack," *Socio-Economic Planning Sciences*, vol. 46, no. 1, pp. 78–87, 2012. Special Issue: Disaster Planning and Logistics: Part 1.

[77] J. Dong, C. Liu, and Z. Lin, "Charging infrastructure planning for promoting battery electric vehicles: An activity-based approach using multiday travel data," *Transportation Research Part C: Emerging Technologies*, vol. 38, pp. 44–55, 2014.

[78] R. Pagany, L. R. Camargo, and W. Dorner, "A review of spatial localization methodologies for the electric vehicle charging infrastructure," *International Journal of Sustainable Transportation*, vol. 13, no. 6, pp. 433–449, 2019.

[79] X. Llorà, K. Sastry, D. E. Goldberg, A. Gupta, and L. Lakshmi, "Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pp. 1363–1370, ACM, 2005.

[80] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer, 2009.

[81] G. Laporte, S. Nickel, and F. Saldanha da Gama, *Location Science*. Springer, 2015.

[82] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, "The uncapacitated facility location problem," in *Discrete Location Theory* (P. B. Mirchandani and R. L. Francis, eds.), pp. 119–171, New York, NY, USA: John Wiley and Sons, Inc, 1990.

[83] L. Di Gaspero, A. Rendl, and T. Urli, "Balancing bike sharing systems with constraint programming," *Constraints*, vol. 21, no. 2, pp. 318–348, 2016.

[84] G. Brandstätter, C. Gambella, M. Leitner, E. Malaguti, F. Masini, J. Puchinger, M. Ruthmair, and D. Vigo, "Overview of optimization problems in electric car-sharing system design and management," in *Dynamic Perspectives on Managerial Decision Making*, pp. 441–471, Springer, 2016.

[85] T. H. Yang, J. R. Lin, and Y. C. Chang, "Strategic design of public bicycle sharing systems incorporating with bicycle stocks considerations," 2010.

[86] J. R. Lin, T. H. Yang, and Y. C. Chang, "A hub location inventory model for bicycle sharing system design: Formulation and solution," *Computers & Industrial Engineering*, vol. 65, pp. 77–86, may 2013.

[87] I. Frade and A. Ribeiro, "Bicycle sharing systems demand," *Procedia - Social and Behavioral Sciences*, vol. 111, pp. 518–527, 2014.

[88] B. W. Landis, "Bicycle system performance measures," *ITE journal*, vol. 66, no. 2, pp. 18–26, 1996.

[89] L. dell'Olio, A. Ibeas, and J. Moura, "Implementing bike-sharing systems," *Municipal Engineer*, vol. 164, pp. 89–101, 06 2011.

[90] G. R. Krykewycz, C. M. Puchalsky, J. Rocks, B. Bonnette, and F. Jaskiewicz, "Defining a primary market and estimating demand for major bicycle-sharing program in philadelphia, pennsylvania," *Transportation Research Record*, vol. 2143, no. 1, pp. 117–124, 2010.

[91] C. Ciancio, G. Ambrogio, and D. Laganá, "A Stochastic Maximal Covering Formulation for a Bike Sharing System," in *Optimization and Decision Science: Methodologies and Applications* (A. Sforza and C. Sterle, eds.), pp. 257–265, Springer International Publishing, 2017.

[92] G. H. d. A. Correia and A. P. Antunes, "Optimization approach to depot location and trip selection in one-way carsharing systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, pp. 233–247, jan 2012.

[93] G. H. d. A. Correia, D. R. Jorge, and D. M. Antunes, "The Added Value of Accounting For Users' Flexibility and Information on the Potential of a Station-Based One-Way Car-Sharing System: An Application in Lisbon, Portugal," *Journal of Intelligent Transportation Systems*, vol. 18, pp. 299–308, jul 2014.

[94] B. Boyacı, K. G. Zografos, and N. Geroliminis, "An optimization framework for the development of efficient one-way car-sharing systems," *European Journal of Operational Research*, vol. 240, no. 3, pp. 718–733, 2015.

[95] T. D. Chen, K. M. Kockelman, and M. Khan, "Locating electric vehicle charging stations: Parking-based assignment method for seattle, washington," *Transportation Research Record*, vol. 2385, no. 1, pp. 28–36, 2013.

[96] I. Frade, A. Ribeiro, G. Gonçalves, and A. P. Antunes, "Optimal Location of Charging Stations for Electric Vehicles in a Neighborhood in Lisbon, Portugal," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2252, pp. 91–98, dec 2011.

[97] H. Kameda and N. Mukai, "Optimization of charging station placement by using taxi probe data for on-demand electrical bus system," in *Knowledge-Based and Intelligent Information and Engineering Systems* (A. König, A. Dengel, K. Hinkelmann, K. Kise, R. J. Howlett, and L. C. Jain, eds.), pp. 606–615, Springer, 2011.

[98] B. D. Galarza Montenegro, K. Sörensen, and P. Vansteenwegen, "A large neighborhood search algorithm to optimize a demand-responsive feeder service," *Transportation Research Part C: Emerging Technologies*, vol. 127, p. 103102, 2021.

[99] F. Ciari, N. Schuessler, and K. W. Axhausen, "Estimation of carsharing demand using an activity-based microsimulation approach: Model discussion and some results," *International Journal of Sustainable Transportation*, vol. 7, no. 1, pp. 70–84, 2013.

[100] A. Horni, K. Nagel, and K. W. Axhausen, eds., *Multi-Agent Transport Simulation MATSim*. Ubiquity Press, 2016.

[101] D. Meignan, S. Knust, J. M. Frayret, G. Pesant, and N. Gaud, "A Review and Taxonomy of Interactive Optimization Methods in Operations Research," *ACM Transactions on Interactive Intelligent Systems*, vol. 5, pp. 1–43, sep 2015.

[102] M. L. Fisher, "Interactive optimization," *Annals of Operations Research*, vol. 5, no. 3, pp. 539–556, 1985.

[103] H.-S. Kim and S.-B. Cho, "Application of interactive genetic algorithm to fashion design," *Engineering applications of artificial intelligence*, vol. 13, no. 6, pp. 635–644, 2000.

[104] L. Thiele, K. Miettinen, P. J. Korhonen, and J. Molina, "A preference-based evolutionary algorithm for multi-objective optimization," *Evolutionary computation*, vol. 17, no. 3, pp. 411–436, 2009.

[105] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 773–791, 2016.

[106] C. B. Eiben, J. B. Siegel, J. B. Bale, S. Cooper, F. Khatib, B. W. Shen, B. L. Stoddard, Z. Popovic, and D. Baker, "Increased diels-alderase activity through backbone remodeling guided by foldit players," *Nature biotechnology*, vol. 30, no. 2, pp. 190–192, 2012.

[107] J. Liu, T. Dwyer, G. Tack, S. Gratzl, and K. Marriott, "Supporting the problem-solving loop: Designing highly interactive optimisation systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1764–1774, 2021.

[108] V. Belton, J. Branke, P. Eskelinen, S. Greco, J. Molina, F. Ruiz, and R. Słowiński, "Interactive multiobjective optimization from a learning perspective," in *Multiobjective Optimization: Interactive and Evolutionary Approaches* (J. Branke, K. Deb, K. Miettinen, and R. Słowiński, eds.), pp. 405–433, Springer, 2008.

[109] A. Najar and M. Chetouani, "Reinforcement learning with human advice: A survey," *Frontiers in Robotics and AI*, vol. 8, p. 74, 2021.

[110] A. Holzinger, "Interactive machine learning for health informatics: when do we need the human-in-the-loop?," *Brain Informatics*, vol. 3, no. 2, pp. 119–131, 2016.

[111] J. A. Fails and D. R. Olsen, Jr., "Interactive machine learning," in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, (New York, NY, USA), pp. 39–45, ACM, 2003.

[112] X. Sun, D. Gong, Y. Jin, and S. Chen, "A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 685–698, 2013.

[113] X. Y. Sun, D. Gong, and S. Li, "Classification and regression-based surrogate model-assisted interactive genetic algorithm with individual's fuzzy fitness," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, (New York, NY, USA), pp. 907–914, ACM, 2009.

[114] S. Koziel, D. E. Ciaurri, and L. Leifsson, "Surrogate-based methods," in *Computational optimization, methods and algorithms*, pp. 33–59, Springer, 2011.

[115] A. I. J. Forrester and A. J. Keane, "Recent advances in surrogate-based optimization," *Progress in Aerospace Sciences*, vol. 45, no. 1-3, pp. 50–79, 2009.

[116] H. Symon, "Neural networks: a comprehensive foundation," *Prentice-Hall*, 1999.

[117] A. Forrester, S. A., A. Keane, *et al.*, *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[118] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Workshop on Deep Learning for Audio, Speech and Language Processing, ICML 2013*, 2013.

[119] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[120] R. Z. Farahani, N. Asgari, N. Heidari, M. Hosseininia, and M. Goh, "Covering problems in facility location: A review," *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 368 – 407, 2012.

[121] P. Richtárik and M. Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," *Mathematical Programming*, vol. 144, pp. 1–38, 2014.

[122] International Energy Agency, "Global EV outlook 2021," 2021.

[123] S. Deilami and S. M. Muyeen, "An insight into practical solutions for electric vehicle charging in smart grid," *Energies*, vol. 13, no. 7, 2020.

[124] M. L. Nicolson, M. J. Fell, and G. M. Huebner, "Consumer demand for time of use electricity tariffs: A systematized review of the empirical evidence," *Renewable and Sustainable Energy Reviews*, vol. 97, pp. 276–289, 2018.

[125] S. Limmer, "Dynamic pricing for electric vehicle charging—a literature review," *Energies*, vol. 12, no. 18, 2019.

[126] Q. Wang, X. Liu, J. Du, and F. Kong, "Smart charging for electric vehicles: A survey from the algorithmic perspective," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1500–1517, 2016.

[127] R. Fachrizal, M. Shepero, D. van der Meer, J. Munkhammar, and J. Widén, "Smart charging of electric vehicles considering photovoltaic power production and electricity consumption: A review," *eTransportation*, vol. 4, 2020.

[128] J. A. Lopes, F. Soares, P. Almeida, and M. Moreira da Silva, "Smart charging strategies for electric vehicles: Enhancing grid performance and maximizing the use of variable renewable energy resources," in *24th International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exhibition 2009 (EVS24)*, vol. 1, pp. 2680–2690, The European Association for Electromobility (AVERE), 2009.

[129] N. Rotering and M. Ilic, "Optimal charge control of plug-in hybrid electric vehicles in deregulated electricity markets," *IEEE Transactions on Power Systems*, vol. 26, no. 3, pp. 1021–1029, 2011.

[130] E. Sortomme, M. M. Hindi, S. D. J. MacPherson, and S. S. Venkata, "Coordinated charging of plug-in hybrid electric vehicles to minimize distribution system losses," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 198–205, 2011.

[131] R. Mehta, D. Srinivasan, and A. Trivedi, "Optimal charging scheduling of plug-in electric vehicles for maximizing penetration within a workplace car park," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3646–3653, IEEE, 2016.

[132] C. Goebel and H. A. Jacobsen, "Aggregator-controlled EV charging in pay-as-bid reserve markets with strict delivery constraints," *IEEE Transactions on Power Systems*, vol. 31, no. 6, pp. 4447–4461, 2016.

[133] E. Kontou, Y. Yin, and Y.-E. Ge, "Cost-effective and ecofriendly plug-in hybrid electric vehicle charging management," *Transportation Research Record*, vol. 2628, no. 1, pp. 87–98, 2017.

[134] I. Naharudinsyah and S. Limmer, "Optimal charging of electric vehicles with trading on the intraday electricity market," *Energies*, vol. 11, no. 6, 2018.

[135] J. Huber, K. Lohmann, M. Schmidt, and C. Weinhardt, "Carbon efficient smart charging using forecasts of marginal emission factors," *Journal of Cleaner Production*, vol. 284, 2021.

[136] Fastned, "Fastned – Supersnel laden langs de snelweg en in de stad: www.fastnedcharging.com," 2020.

[137] O. Sundström and C. Binding, "Optimization methods to plan the charging of electric vehicle fleets," in *Proceedings of the International Conference on Control, Communication and Power Engineering*, (Chennai, India), pp. 323–328, 2010.

[138] T. Morstyn, C. Crozier, M. Deakin, and M. D. McCulloch, "Conic optimization for electric vehicle station smart charging with battery voltage constraints," *IEEE Transactions on Transportation Electrification*, vol. 6, no. 2, pp. 478–487, 2020.

[139] Y. Cao, S. Tang, C. Li, P. Zhang, Y. Tan, Z. Zhang, and J. Li, "An optimized EV charging model considering TOU price and SOC curve," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 388–393, 2012.

[140] C. Z. El-Bayeh, I. Mougharbel, M. Saad, A. Chandra, D. Asber, and S. Lefebvre, "Impact of considering variable battery power profile of electric vehicles on the distribution network," in *2018 4th International Conference on Renewable Energies for Developing Countries (REDEC)*, pp. 1–8, IEEE, 2018.

[141] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear organisation*, vol. 6 of *Athena scientific optimization and computation series*. Athena Scientific, 1997.

[142] C. F. Jekel and G. Venter, *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*, 2019.

[143] T. Ishihara and S. Limmer, "Optimizing the hyperparameters of a mixed integer linear programming solver to speed up electric vehicle charging control," in *Applications of Evolutionary Computation* (P. A. Castillo, J. L. Jiménez Laredo, and F. Fernández de Vega, eds.), vol. 12104 of *LNCS*, pp. 37–53, Springer, 2020.

Thomas Jatschka

Curriculum Vitae

Personal Information

Name	Thomas Jatschka
Date/Place of Birth	November, 1989, Vienna
Email	tjatschk@ac.tuwien.ac.at

Education

2018–present **Doctoral programme in Engineering Sciences**, *TU Wien*, Vienna, Austria.
Field of Study: Computer Science
Thesis: Computational Optimization Approaches for Distributing Service Points for Mobility Applications and Smart Charging of Electic Vehicles
Advisors: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl, Dr.rer.nat. Tobias Rodemann

2014–2017 **Master of Science**, *TU Wien*, Vienna, Austria.
Field of Study: Computational Intelligence
Thesis: An Iterative Time-Bucket Refinement Algorithm for High Resolution Scheduling Problems
Advisors: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl, Dipl.-Ing. Dr.techn. Martin Riedler, Dipl.-Ing. Dr.techn. Johannes Maschler

2010–2014 **Bachelor of Science**, *TU Wien*, Vienna, Austria.
Field of Study: Software & Information Engineering

Work Experience

2018–present **Project Assistant**, *TU Wien*, Vienna, Austria.
Collaboration with Honda Research Institute Europe GmbH

Publications

2022 T. Jatschka, T. Rodemann, and G. R. Raidl, "A large neighborhood search for a cooperative optimization approach to distribute service points in mobility applications," in *Metaheuristics and Nature Inspired Computing* (B. Dorronsoro, F. Yalaoui, E.-G. Talbi, and G. Danoy, eds.), pp. 3–17, Springer, 2022

2021 B. Schaden, T. Jatschka, S. Limmer, and G. R. Raidl, "Smart charging of electric vehicles considering SOC-dependent maximum charging powers," *Energies*, vol. 14, no. 22, 2021

2020

T. Jatschka, T. Rodemann, and G. R. Raidl, "VNS and PBIG as optimization cores in a cooperative optimization approach for distributing service points," in *Computer Aided Systems Theory – EUROCAST 2019*, vol. 12013 of *LNCS*, pp. 255–262, Springer, 2020

T. Jatschka, F. F. Oberweger, T. Rodemann, and G. R. Raidl, "Distributing battery swapping stations for electric scooters in an urban area," in *Optimization and Applications, Proceedings of OPTIMA 2020 – XI International Conference Optimization and Applications* (N. Olenev, Y. Evtushenko, M. Khachay, and V. Malkova, eds.), vol. 12422 of *LNCS*, pp. 150–165, Springer, 2020

M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl, "An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem," *International Transactions in Operational Research*, Jan. 2020

2019

T. Jatschka, T. Rodemann, and G. R. Raidl, "A cooperative optimization approach for distributing service points in mobility applications," in *Evolutionary Computation in Combinatorial Optimization* (A. Liefooghe and L. Paquete, eds.), vol. 11452 of *LNCS*, pp. 1–16, Springer, 2019

T. Jatschka, T. Rodemann, and G. R. Raidl, "Exploiting similar behavior of users in a cooperative optimization approach for distributing service points in mobility applications," in *The 5th International Conference on machine Learning, Optimization and Data science – LOD 2019* (G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, eds.), LNCS, pp. 738–750, Springer, 2019

A. Hoffmann-Ostenhof and T. Jatschka, "Snarks with special spanning trees," *Graphs and Combinatorics*, vol. 35, no. 1, pp. 207–219, 2019

2016

G. R. Raidl, T. Jatschka, M. Riedler, and J. Maschler, "Time-bucket relaxation based mixed integer programming models for scheduling problems: A promising starting point for matheuristics," in *Proceedings of Matheuristics 2016: 6th International Workshop on Model-Based Metaheuristics* (T. Stützle and V. Maniezzo, eds.), (Brussels, Belgium), pp. 104–107, 2016