Technische Universität München
TUM School of Management

Algorithmic solutions for emerging challenges in last-mile logistics

Patrick Sean Klein

Vollständiger Abdruck der von der TUM School of Management der Technischen Universität München zur Erlangung eines Doktors der Wirtschafts- und Sozialwissenschaften (Dr. rer. pol.) genehmigten Dissertation.

Vorsitz : Prof. Dr. Martin Grunow

Prüfende der Dissertation:

1. Prof. Dr. Maximilian Schiffer
2. Prof. Dr. Claudia Archetti

Die Dissertation wurde am 20.03.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Management am 15.06.2024 angenommen

Contents

Acknowledgments

I want to express my heartfelt thanks to my supervisor Prof. Maximilian Schiffer, who has shaped much of my personal and professional growth, not only during my PhD but throughout my entire academic journey. If it were not for his unwavering support, insightful guidance, and the various opportunities he provided, I would not be in the position to write a thesis in the first place. I am also immensely thankful to my research group for the enriching discussions that broadened my perspectives, particularly when my research was all-consuming. Beyond the realm of academia, my group members have made this journey truly enjoyable, creating a warm and cheerful atmosphere even outside of work.

While I am grateful to everyone in the group, I owe special thanks to Paul for inspiring me with his unconditional productivity and ability to rapidly learn, to Benedikt for exemplifying how to be confident and take pride in one's work, a lesson that has been invaluable to me, to Kai for his coolness during the EURO Meets NeurIPS Competition and for the countless long nights we have shared, and to Gerhard for his valuable methodological insights and engaging conversations.

On a personal level, I want to thank my family, especially my parents, for their unconditional support that enabled me to pursue scientific studies. To all my friends, thank you for bringing joy and light into my days throughout this period.

Abstract

The logistics sector is experiencing a transformative era characterized by a paradigm shift in customer service expectations, an unprecedented demand increase, and a pressing need for sustainable operations. These changes significantly impact the last-mile distribution sector, raising new challenges that compel logistics service providers to adjust their operations. This thesis aims to provide tools and algorithms necessary to address complex optimization challenges arising in this new environment, requiring not only the *development* of appropriate algorithms but also making them *accessible* to practitioners.

This thesis includes an introduction, a literature review, three methodological chapters, and a conclusion. The introduction outlines the transformative changes currently reshaping the logistics landscape and details the specific challenges logistics service providers face, especially in the context of last-mile delivery. The literature review connects these challenges to relevant streams of literature. It provides a comprehensive overview of the current state-of-the-art, highlighting potential research gaps and positioning the contributions of this thesis within a broader methodological context. The three methodological chapters are the main contributions of this thesis.

The first methodological chapter considers a dynamic routing and dispatching problem where requests are gradually revealed over the span of the planning horizon, requiring operators to dispatch vehicles in an online fashion. The chapter develops a novel integrated approach that embeds combinatorial optimization and machine learning in a single machine learning pipeline, allowing the machine learning model to leverage the combinatorial structure in the routing problem. The algorithm was submitted to the EURO Meets NeurIPS 2022 Vehicle Routing Competition, where it won first prize, outperforming state-of-the-art Monte Carlo, reinforcement learning, and supervised learning approaches.

The second methodological chapter deals with an integrated charging and service operation scheduling problem for fleets of electric vehicles. The chapter develops a Branch-and-Price algorithm relying on a novel label-setting shortest-path algorithm, a problem-specific branching rule, a primal heuristic, and partial pricing. The individual contribution of each of these enhancements to the performance of the algorithm is shown and validated in an ablation study. The chapter further generates managerial insights on the impact of jointly scheduling charging and service operations, the impact of infrastructure capacity on operational costs, and the impact of considering time-of-use pricing.

The third methodological chapter introduces `RoutingBlocks`, an open-source software package for Vehicle Routing Problems. The package aims to make advanced routing algorithms accessible to practitioners and researchers. It provides a modular framework that separates problem-specific logic from core optimization, allowing to easily apply existing algorithms to bespoke problem settings.

The thesis concludes with a detailed summary and critical discussion of its managerial and methodological contributions. This discussion includes opportunities for future research made available through the contributions of this thesis, highlighting how to leverage the developed methodology for this purpose.

1 Introduction

Author: Patrick Sean Klein

1.1. Background

As we progress further into the digital era, industries across the globe are experiencing a myriad of transformative changes. The logistics sector, a pivotal driver of the global supply chain, is no exception. Here, a confluence of factors significantly reshapes the logistics landscape: first, technological advancements, particularly in terms of digitalization, have lowered entry barriers into the logistics sector, creating a highly competitive market where logistics service providers (LSPs) face customers demanding faster, more flexible, and more reliable deliveries (Placek, 2022). Second, population growth contributes to a substantial increase in logistic volume, such that the demand for freight transportation is projected to double by 2050 (ITF, 2023). Third, the logistics sector, being considered one of the main contributors to greenhouse gas emissions, faces increasing pressure to adopt more sustainable practices (McKinnon et al., 2015).

These developments present various challenges for LSPs. They not only need to improve their service quality, offering tighter delivery windows and drastically shortened lead times but also concurrently have to meet an expanded volume of deliveries, all while realizing sustainable operations. These challenges have a particularly profound impact on last-mile distribution, where the complexities of heightened customer expectations, increased delivery volumes, and sustainability occur on a single operational stage. Facing these challenges, the logistics sector's reliance on efficient planning becomes even more critical. Operating on typically low-profit margins, logistics companies must optimize every aspect of their operations, especially on the last mile, to stay competitive and profitable. However, existing planning methodologies fail to address the challenges arising in this new logistics landscape.

First, conventional day-ahead planning models struggle to meet the current market demands, particularly the growing expectation for shorter lead times and more flexible delivery options like same-day delivery (Placek, 2022). These flexible deliveries lead the concept of day-ahead planning, which has been the status quo in last-mile logistics for decades, ad absurdum, as orders often arrive only when vehicles have already been dispatched. To meet the demands of this evolving market, LSPs instead need to adopt dynamic planning approaches capable of responding to customer requests in real time.

Second, many LSPs are transitioning to electric commercial vehicles (ECVs) to realize sustainable transportation. ECVs, however, are limited in their driving range by battery capacity. While en-route charging is an option, LSPs often prefer recharging the fleet during off-shift hours to avoid inefficient use of drivers' time and uncertainty around station availability. This approach becomes problematic in large-scale operations, where the collective charging demand of a fleet may surpass the capacity of the charging infrastructure available at the depot. In such settings, charging operations must be strategically scheduled to ensure that all vehicles are sufficiently charged for next-day operations. Besides avoiding bottlenecks in charger capacity, this scheduling problem must also align the charging strategy with other operational objectives, such as energy cost minimization and battery life considerations.

Although research often addresses many of the planning problems faced by the logistics sector well before they emerge in practice, actual adoption of the developed solutions often occurs years or even decades after the problems become relevant. A key reason for this gap between theory and practice is a lack of accessibility: academic papers often propose innovative algorithms and sophisticated models but rarely provide the underlying code. This lack of high-quality, generalizable implementations creates a significant barrier for practitioners who aim to evaluate and apply these theoretical contributions to their specific operational context. Without accessible code, reproducing academic findings becomes daunting, necessitating considerable time and expertise to implement algorithms based only on paper descriptions. Consequently, a substantial sunk cost is associated with assessing theoretical findings' applicability to an operator's specific operational context, leading to practitioners' reluctance to test and potentially adopt recently developed methodology.

Researchers can address this issue by open-sourcing implementations alongside academic papers. This promises to lower the barriers for practitioners interested in applying and evaluating new algorithms and planning methodologies, potentially resulting in faster adoption of research findings in practice. It may also encourage more direct and effective collaboration between theoretical research and practical application. Moreover, open-source implementations are crucial for ethical and reproducible science as they enable the academic community to build upon past research, fostering future contributions and ultimately advancing the state-of-the-art.

In conclusion, the logistics sector experiences significant transformations driven by increased demand, technological advancements, and pressure to realize sustainable operations. These changes present a series of complex challenges, especially in the context of last-mile distribution:

Technological advancements have created a competitive market where customers demand fast, flexible, and reliable deliveries. These demands challenge traditional day-ahead planning methodologies unsuited to deal with the dynamic nature of short delivery lead times. Realizing these demands cost-efficiently requires a shift towards real-time, reactive planning approaches.

Increased demand and pressure to realize sustainable operations raise novel planning problems, particularly in transitioning to ECVs. Specifically, LSPs not only have to adjust their route planning to account for the range restrictions of ECV but also need to design charging schedules strategically to ensure that all vehicles are adequately charged for next-day operations. This strategic charge scheduling involves careful planning to align with time-of-use (TOU) energy tariffs, reduce battery degradation, and adhere to infrastructure capacity constraints.

1.2. Aims and Scope

Against this background, this thesis aims to provide tools and algorithms necessary to address the complex optimization challenges faced by LSPs in this new logistics landscape. Addressing these challenges requires not only to *develop* the respective planning algorithms but also to make them *accessible* to practitioners, i.e., by making them available as well-maintained open-source implementations. To achieve this, the thesis establishes a bridge between theory and practice by offering theoretical contributions, generating managerial insights, and developing a comprehensive software library. Specifically, this thesis tackles the following challenges:

Dynamic route planning and dispatching: The traditional approach of day-ahead planning, which backs many last-mile operations in practice, struggles with the dynamic environment introduced by short delivery lead times. This thesis aims to address this issue by developing adaptive solution methodology that can effectively optimize last-mile deliveries within this dynamic and uncertain context.

Charge scheduling for electric vehicles: The transition to large fleets of electric vehicles introduces complex optimization challenges not encountered in smaller-scale operations. Key issues such as efficient charger allocation, effective charge scheduling, and battery life management become critical in ensuring operational efficiency and sustainability. This thesis extends traditional planning frameworks with innovative algorithms designed to address these complexities.

Open-sourcing advanced optimization algorithms: Theoretical contributions, e.g., algorithms, are often not readily accessible in practical settings. This lack of accessibility disconnects research from practical application and threatens ethical and reproducible science. This thesis aims to enable researchers and practitioners to leverage advanced algorithmic solutions by providing practical open-source software contributions.

1.3. Contribution

The reminder of this thesis is organized as follows:

Chapter 2 reviews fundamental literature on Vehicle Routing Problems (VRPs), Dynamic Vehicle Routing Problems (DVRPs), Electric Vehicle Scheduling Problems (EVSPs), and Charge Scheduling Problems (CSPs), covering foundational works and recent advancements in these fields. It further provides an overview of recent developments in open-source software tailored for last-mile logistics and highlights pivotal open-source contributions in vehicle routing.

Chapter 3 addresses the challenge of dynamic route planning and dispatching. Specifically, it considers a dynamic routing and dispatching problem where customer requests are not known in advance but are gradually revealed over the span of the planning horizon. This setting requires operators to dispatch vehicles in waves, i.e., when new orders arrive, carefully balancing between future optimization potential of so-far unrevealed requests, robustness, and current solution quality. This dynamic dispatching entails solving two challenging problems: First, to balance the benefit of serving a request at the current time versus the cost of postponing it, and second, to assemble served requests to efficient routes. The first problem is a classical Machine Learning (ML) task, while the second problem is a classical Combinatorial Optimization (CO) task. These tasks are interrelated, e.g., the cost of postponing a request depends on the downstream routing decision. The chapter develops a novel approach that integrates these ML and CO tasks in a single Machine Learning enriched Combinatorial Optimization (ML-CO) pipeline, allowing the ML model to leverage the combinatorial structure present in the CO problem. For this purpose, it reformulates the VRP of the CO layer as a prize-collecting VRP. This transformation allows the computation of loss gradients for the CO layer, thus enabling end-to-end learning. While this novel approach already advances the state-of-the-art in dynamic route planning and dispatching problems, it also contributes to the ML-CO literature in general. Specifically, it is the first ML-CO pipeline that leverages a metaheuristic component in the CO layer, showcasing how to carefully design a metaheuristic that allows approximating gradients with sufficient accuracy to enable end-to-end learning in integrated pipelines. Extensive numerical experiments demonstrate the performance of this approach: it outperforms monte-carlo and rolling-horizon policies by 1.57% and 7.12% in terms of objective value while only taking 1.67% and 15.00% of the runtime, respectively. The approach outperformed state-of-the-art reinforcement- and supervised learning approaches, managing to win first prize in the EURO Meets NeurIPS 2022 Vehicle Routing Competition (Kool et al., 2022), despite being trained on only 20 of 250 instances available in the training set.

Chapter 4 deals with an integrated charging and service operation scheduling problem for electric vehicle fleets. It enhances the state of the art by integrating charging and service operation scheduling and accounting for realistic battery behavior, i.e., non-linear charging and battery deterioration, variable energy prices, and charging infrastructure capacity constraints. Methodologically, it develops a column-generation-based approach embedded in a Branch and Price (B&P) algorithm. The major methodological challenges tackled by this algorithm are twofold: first, it needs to account for non-linear tradeoffs between battery deterioration, charging cost, charger capacity, and service operation scheduling. Second, capacity constraints cause a large fan-out of the branch-and-bound tree, requiring the development of efficient branching and pruning strategies to maintain scalability. The chapter develops a novel label-setting shortest-path algorithm that captures non-linear tradeoffs using a function-based label representation, allowing optimal decisions even in continuous-time settings. This label-setting algorithm readily applies to problem settings beyond charge scheduling, advancing the state-of-the-art in related research areas. The branch-and-price algorithm relies on a problem-specific branching rule, a primal heuristic, and partial pricing to effectively reduce the fan-out of the branch-and-bound tree. A comprehensive numerical study shows the efficiency of these enhancements on large-scale instances and validates the benefit of the developed components in an ablation study. Beyond these theoretical findings, the chapter analyses the impact of jointly scheduling charging and service operations, generating managerial insights on i) the benefit of an integrated approach to charge and service operation scheduling, ii) the impact of infrastructure capacity on operational costs, and iii) the impact of considering TOU pricing. Specifically, integrating charging and service operation scheduling lowers the amount of charging infrastructure required by up to 57% and reduces operational costs by up to 5%. Here, both the degree of service schedule flexibility and the energy price distribution play a crucial role in realizing these benefits.

Chapter 5 introduces `RoutingBlocks`, an extensive Python package designed for implementing VRPs. Released under a permissive open-source license, it democratizes access to advanced routing algorithms, encouraging a collaborative academic environment where enhancements can be openly shared. This aspect particularly benefits researchers, enabling them to conduct thorough evaluations and comparisons between their approaches and established algorithms. The package further helps to bridge the gap between theory and practice by making state-of-the-art research in VRPs available to practitioners. The major challenge of designing such a library lies in the diversity of VRPs: especially in practice, problem settings often involve a lot of unique, case-specific constraints not addressed in pure, academic problem settings. The chapter tackles this challenge by providing a modular algorithmic framework that allows the composition of tailored solution methods based on optimized, state-of-the-art algorithmic components and data structures. By employing a unified solution and instance representation, `RoutingBlocks` separates algorithmic components from specific problem configurations. This separation allows abstracting problem-specific behavior, e.g., constraint checking, move evaluation, and cost computation, such that the same algorithm can be applied to various problem settings. The chapter outlines the architecture of `RoutingBlocks`. It explains how its design, particularly using a native `C++` implementation exposed in Python, balances modularity, user-friendliness, and high performance. The chapter includes a numerical benchmark against two monolithic state-of-the-art algorithms implemented in native code. Here, `RoutingBlocks` achieves comparable performance, maintaining a gap of 0.44% w.r.t. objective value, while being only 14.86% slower than the state-of-the-art native implementation. This is well within the bounds expected for a generic library exposed in a high-level dynamic language like Python.

Chapter 6 offers a comprehensive summary of this thesis's key managerial and methodological contributions. It highlights future research perspectives and discusses avenues for further development of the software developed as part of this thesis.

Bibliography

ITF (May 2023). *Transport Outlook 2023.* Tech. rep. DOI: `10.1787/b6cc9ad5-en`.

Kool, Wouter et al. (2022). *The EURO Meets NeurIPS 2022 Vehicle Routing Competition.*

McKinnon, Alan C. et al., eds. (2015). *Green logistics. Improving the environmental sustainability of logistics.* Third edition. Includes bibliographical references and index. Kogan Page Limited: London. 1 p. ISBN: 9781523103744.

Placek, Martin (Sept. 2022). *Same-day delivery market size in U.S. 2019-2024.* URL: `%7Bhttps://www.statista.com/statistics/1068886/us-same-day-delivery-market-size/%7D`.

2 State of the art

Author: Patrick Sean Klein

2.1. Introduction

The logistics sector is experiencing a transformative era characterized by an unprecedented increase in demand, a paradigm shift in customer service expectations, and a pressing need for sustainable operations. These changes are reshaping the logistics landscape, especially the last-mile distribution sector, raising new challenges that compel logistics service providers (LSPs) to adjust their operations.

This dissertation introduces innovative tools and algorithmic approaches to help LSPs tackle these novel operational challenges. Specifically, it i) develops methodology to strategically schedule charging operations for large fleets of electric vehicles, ii) designs algorithms to realize cost-efficient dynamic route planning, and iii) designs an innovative software package that eases the implementation of advanced vehicle routing algorithms.

Accordingly, the individual contributions of this work touch various methodological areas, ranging from data-driven Machine Learning (ML) approaches, over metaheuristics, to exact algorithms, and belong to several different research streams. This chapter aims to provide a concise review of these research streams. For this purpose, it comprises three distinct sections that align with the three main contributions of this thesis. The first section reviews literature related to dynamic route planning and dispatching, starting with a summary of conventional, i.e., static, Vehicle Routing Problems (VRPs) and gradually moving to Dynamic Vehicle Routing Problems (DVRPs). The second section reviews the literature on planning problems in the context of scheduling large fleets of electric commercial vehicles (ECVs), i.e., Charge Scheduling Problems (CSPs) and Vehicle Scheduling Problems (VSPs). The third section offers an overview of recent efforts within the operations research community to promote the development of high-quality software solutions, highlighting state-of-the-art open-source contributions in vehicle routing. The review concludes with a summary of all related fields, detailing how this thesis contributes to the existing body of research.

2.2. Vehicle Routing

The fundamental planning problem faced in last-mile distribution is designing least-cost delivery routes from a central depot to a set of geographically scattered customers. Several variants of this problem exist as real-life applications often entail operator- and scenario-specific rules and constraints. Consequently, a vast body of literature exists on this class of problems, generally referred to as VRPs. Among the most common variants are the Capacitated Vehicle Routing Problem (CVRP), where vehicles have a finite capacity that limits the maximum number of customers served on a single route, the Vehicle Routing Problem With Time Windows (VRPTW), where customers can only be served during a customer-specific time window, and Pick-up and Delivery Problems, which involve delivery and pick-up of goods at customer locations. These variants are often further extended with variant-specific enhancements, such as split deliveries allowing a single customer's demand to be met in multiple deliveries, or variant-independent features, such as incorporating demand- and other uncertainties. This review aligns with the operational problems considered in this thesis. It focuses specifically on VRPs that account for uncertainty in the planning process, as is required in the context of short delivery lead times. Interested readers can find information on classic and emerging variants of VRPs not covered in this review in various survey books and articles, see, e.g., Golden, Raghavan, and Wasil (2008), Gilbert Laporte (2009), Toth and Vigo (2014), and Vidal, Gilbert Laporte, and Matl (2020).

2.2.1. Dynamic Vehicle Routing

Real-world VRPs often bear much uncertainty. Specifically, problem data, such as customers to serve, their demand, or the travel time between them, is often not known in advance. This requires operators to revise existing or make new decisions during operation, which increases the problem's complexity significantly: operators must strike a balance between future optimization potential of so-far unrevealed data, robustness, and current solution quality.

VRPs that address this uncertainty are commonly referred to as DVRPs. More precisely, DVRPs encompass problem settings where information about the routing problem is revealed in real-time, often when the vehicles have already been dispatched. Consequently, LSPs must make decisions reactively, adapting to the new information as it becomes available. This broad definition comprises a wide range of problem settings, each differing in several key dimensions (cf. Ojeda Rios et al., 2021):

Responsiveness: DVRPs vary in the their responsiveness to new information. While some are fully reactive, responding to new information immediately, others may generate decisions on an hourly basis, as is typical in dispatching problems, or even in alignment with driver shifts, common in, e.g., e-grocery settings.

Dynamicity: Some DVRPs allow to re-route vehicles in response to new information, while others consider less dynamic settings, where decisions may not be revised at a later point in time.

Characteristics of the dynamic element: DVRPs vary in the problem characteristic that is considered uncertain. Common sources of uncertainty are customer requests, travel and service times, vehicle availability, and customer demand.

Characteristics of the routing problem: The diversity of different variants of VRPs is reflected in the DVRP. Commonly considered variants include the VRPTW, fleet size and mix problems, and the team orienteering problem.

Despite the apparent structural differences among problem settings, they all bear the same core challenge inherent in DVRPs: the need to balance the potential of future rewards against the immediate cost savings achievable through decisions based on the current problem state. Two distinct methodologies, based on ML and classic Combinatorial Optimization (CO), respectively, have emerged as prominent solutions to this fundamental challenge.

CO-based approaches often amend solution methods developed for static VRPs to suit dynamic scenarios. Here, various strategies exist that mainly differ in the extent to which they consider unrevealed data: *Myopic* approaches (see, e.g., Gendreau et al., 1999; Steever, Karwan, and Murray, 2019) focus solely on the current problem state. They typically embed (meta-)heuristics into rolling horizon frameworks, i.e., solve a static variant of the considered problem each time new information enters the system (see, e.g., Ritzinger, Puchinger, and Hartl, 2016; Ojeda Rios et al., 2021). More computationally expensive *look-ahead* approaches (e.g., Bent and Van Hentenryck, 2004; Flatberg et al., 2007) consider potential future scenarios. For this purpose, they often extend the static problem setting given by the current state with samples of potential future observations.

Conversely, ML-based approaches predict the expected reward of decisions taken at the current time based on historical observations. These approaches often model the dynamic problem as a Markov Decision Process solved with either approximate dynamic programming methods (see, e.g., Ulmer, Soeffker, and Mattfeld, 2018) or reinforcement learning (see, e.g., Nazari et al., 2018; Kool, Hoof, and Welling, 2019; Joe and Lau, 2020). Raza, Sajid, and Singh (2022) and Hildebrandt, Thomas, and Ulmer (2023) provide extensive reviews of these approaches.

The primary advantage of CO-based approaches is their efficiency in providing solutions that are close to optimal for static VRPs. However, their ability to leverage future anticipations remains limited. On the other hand, ML-based approaches excel in making accurate predictions in dynamic environments but often fall short in solution quality, as they tend to disregard the combinatorial nature of the problem. Clearly, an integrated approach that synergizes the strengths of CO and ML by utilizing the problem's combinatorial structure while leveraging the predictive power of ML promises to improve on the state-of-the-art in DVRPs. However, such an approach has yet to be developed specifically for DVRPs.

2.3. Vehicle and charge scheduling

Large-scale fleet electrification represents a paradigm shift in the logistics and transportation industries, introducing complex operational challenges that demand innovative solutions. Specifically, as fleets of ECVs become more prevalent, the need to manage their charging schedules efficiently becomes critical. The resulting planning problem joins two streams of literature, namely VSPs and CSPs.

2.3.1. Vehicle Scheduling

VSPs aim to find the optimal assignment and scheduling of tasks to vehicles. These tasks occur at different geographical locations, requiring vehicles to relocate before servicing any given task. While this problem setting, at first glance, appears similar to VRPs, the two problems consider service operations of different levels of granularity. Specifically, in VRPs, a task generally corresponds to a single customer, while in VSPs, a task usually involves servicing an entire route. These aggregated tasks typically make VSPs more scalable than their VRP counterparts (cf. Parmentier, Martinelli, and Vidal, 2023). In fact, VSPs can often be solved optimally, even in practice, typically by relying on decomposition techniques such as column generation (Klabjan, 2005). This specific technique works particularly well for VSPs, where the Resource Constrained Shortest Path Problem (RCPSPP) solved in the pricing problem is often non-elementary. Specifically, task granularity allows establishing a total order in the pricing problem's network, making cycles impossible and thus eliminating the need to enforce path elementary (Irnich and Desaulniers, 2005). As a result, VSPs usually consider longer planning horizons than their VRP counterparts and often integrate other scheduling problems, such as crew rostering (Raff, 1983; Bazargan, 2016; Bunte and Kliewer, 2009; Perumal, Lusby, and Larsen, 2022), vehicle loading (Pollaris et al., 2015), or charge scheduling (van Kooten Niekerk, van den Akker, and Hoogeveen, 2017; Parmentier, Martinelli, and Vidal, 2023). The remainder of this section focuses on VSPs that integrate charge scheduling, which are commonly referred to as Electric Vehicle Scheduling Problems (EVSPs), as this problem setting aligns most closely with the charge scheduling problem discussed in Chapter 4. Interested readers are referred to Raff (1983), Desrosiers et al. (1995), and Bunte and Kliewer (2009) for a more general discussion of VSPs.

Adler and Mirchandani (2017) are the first to address EVSPs. They consider a network with multiple depots and refueling stations, the latter allowing the replenishment of energy lost by servicing a trip or moving between network nodes. They assume instantaneous refueling, such that the penalty incurred by recharging corresponds to the time lost detouring to the station. Li (2014) consider a similar problem setting. As in Adler and Mirchandani (2017), their network comprises multiple depots and refueling stations, but they additionally penalize recharging operations with a constant waiting time. Li (2014) and Adler and Mirchandani (2017) both propose Branch and Price (B&P) algorithms, which are particularly amendable to this constant time charging model as the pricing problems reduce to the well-known weight-constrained shortest path problem with replenishment arcs (cf. Smith, Boland, and Waterer, 2012; Bolívar, Lozano, and Medaglia, 2014). While this modeling approach is accurate in battery-swapping applications, it is ill-suited to capture the general charging behavior of ECVs. Consequently, Wen et al. (2016) extend this problem setting with a more accurate battery model. Specifically, in Wen et al. (2016), charging stations replenish energy according to a fixed charging rate, which introduces a tradeoff between the time spent charging and the energy replenished. Furthermore, under this assumption, stopping charging before the battery is full may be advantageous, which increases the problem's complexity significantly. Wen et al. (2016) tackle this complexity by minimizing the overall distance traveled instead of the completion time, such that charging, besides requiring to detour to the selected charging station, does not negatively impact the objective value. van Kooten Niekerk, van den Akker, and Hoogeveen (2017) further extend the problem setting defined in Wen et al. (2016) and consider time-of-use (TOU) pricing, charging station capacity, non-linear battery behavior, and battery degradation. While modeling non-linear charging, where the charging rate depends on the battery's residual charge, significantly improves the accuracy of the underlying battery model, it comes at the price of increased complexity. As a remedy, they discretize the battery's state of charge, which allows solving the problem with off-the-shelf commercial solvers. Parmentier, Martinelli, and Vidal (2023) are the first to consider a non-linear battery model with continuous-time charging. They propose a B&P algorithm that models the pricing problem as a RCPSPP with non-linear resource constraints. Parmentier, Martinelli, and Vidal rely on the algorithm developed in Parmentier (2019) to solve this difficult optimization problem. In contrast to van Kooten Niekerk, van den Akker, and Hoogeveen (2017), they do not consider battery degradation, station capacity, or TOU pricing.

In conclusion, work on EVSPs differs mainly in the accuracy of the battery model and the extent to which charge scheduling constraints, such as infrastructure capacity, are considered. In both of these dimensions, existing work either remains limited to simple battery models, compromises accuracy for computational performance or disregards charge scheduling.

2.3.2. Charge Scheduling

CSPs focus on optimizing the charging schedule of fleets of ECVs, aiming to capitalize on potential short-term and long-term cost savings of battery degradation aware charging, TOU energy pricing, and increased charging infrastructure utilization. Unlike VSPs, CSPs typically operates under the assumption of fixed vehicle schedules. Hence, they do not alter the assignment of trips to vehicles or modify service schedules. Despite this fundamental difference, research in the field of CSPs has followed a trajectory similar to that of VSP literature, gradually improving the accuracy of the considered battery model and progressively introducing a more comprehensive range of scheduling constraints.

Early publications assume linear charging operations and focus on single vehicles (Sassi and Oulamara, 2014; Sassi and Oulamara, 2016). Hence, they do not consider fleet-level issues, such as charging station capacity. Advancements in the field have addressed these issues to a certain extent. Abdelwahed et al. (2020), for instance, focus on the role of charging infrastructure in EVSPs. In their work, charging stations differ concerning their capacity and charging speed. Methodologically, they approach their problem with mixed-integer programming, utilizing a commercial solver in their case study. They further benchmark discrete-time against discrete-event-based formulations of their Mixed Integer Program in a numerical study. Their work, however, does not consider non-linear charging dynamics, the effects of battery wear, or potential cost savings from TOU rates. To address these issues, Pelletier, Jabali, and G. Laporte (2018) extend the problem setting considered in Abdelwahed et al. (2020). Their comprehensive case study evaluates the impact of battery degradation, TOU pricing, and grid capacity constraints on charge scheduling. They propose a Mixed Integer Program solved with off-the-shelf commercial software. This may limit the scalability of their approach, such that existing literature remains limited in either level of detail or may struggle to scale with increasing fleet size. Moreover, no publication explores the benefit of integrating vehicle and charge scheduling decisions, i.e., by allowing the modification of vehicle schedules, potentially enabling more efficient charge scheduling.

2.4. Open Source Software for Vehicle Routing Problems

One of the longstanding challenges in the academic world has been translating research findings into formats more accessible to industry professionals. Open source software catalyzes this, offering a repository of readily accessible and modifiable tools and algorithms. While fields adjacent to operations research, such as computer science, already recognize the importance of open source, the concept only recently gained traction in the operations research community.

Here, some journals now accept works that contribute primarily software implementations. For instance, the INFORMS Journal on Computing has established a dedicated editorial area, titled *Software Tools*, specifically for software contributions (cf. Ralphs, 2023). This editorial area accommodates submissions that contribute novel software to the research community, where novelty may lie in advanced implementation techniques, innovative abstractions, or robust versions of existing algorithms. Even outside academic journals, the community made efforts to develop efficient tooling, especially in popular areas such as vehicle routing. Initiatives, such as the *DIMACS Implementation Challenge* (DIMACS, 2022), the *Amazon Last Mile Routing Research Challenge* (Merchán et al., 2022), and the *EURO Meets NeurIPS 2022 Vehicle Routing Competition* (Kool, Bilek, et al., 2022), have been pivotal in further supporting this development, encouraging participants to open-source their contributions. As a result, there has been steep progress on developing state-of-the-art open-source meta-heuristics and exact frameworks specifically for VRPs: Vidal (2022) open-sourced their *Hybrid Genetic Search* algorithm, Accorsi and Vigo (2021) and Accorsi and Vigo (2023) contributed *FILO*, a metaheuristic algorithm for large-scale capacitated vehicle routing problems, and Errami et al. (2023) proposed *VRPSolverEasy*, a python interface for the popular exact *VRPSolver* (cf. Pessoa et al., 2020).

While these open-source contributions represent significant strides in the field, they often remain confined to specific problem settings, particularly in the case of metaheuristic contributions. Specifically, existing open-sourced metaheuristics are implemented as monolithic algorithms and, as such, lack modularity. Thus, reusing their components in other algorithms or adapting them to other problem settings remains challenging. In fact, existing tooling lacks a modular, easy-to-use, state-of-the-art software package that eases the implementation of advanced vehicle routing algorithms for applications beyond traditionally considered variants of the VRP.

2.5. Conclusion

This thesis contributes to the research fields outlined in the preceding subsections by closing the following research gaps.

Integrated CO and ML: Methodologically, DVRP are either approached by amending CO algorithms to dynamic problem settings, i.e. by sampling future scenarios, or with ML, i.e., by predicting future rewards. These existing approaches either do not fully capitalize on the potential insights from anticipating future demand or disregard the combinatorial structure inherent in the underlying routing problem. This thesis closes this gap by developing a truly integrated approach that synergizes the strengths of CO and ML.

Integrated charge and scheduling problems: Work on VSP and CSP remains limited in at least one of two domains: it either approximates battery behavior or does not fully address either vehicle scheduling or charge scheduling constraints. This thesis closes this gap by contributing an integrated charging and service operation scheduling problem for fleets of electric vehicles that accounts for charge scheduling constraints, such as realistic battery behavior, variable energy prices, and charging infrastructure availability, while allowing to reschedule service operations.

Accessible vehicle routing algorithms: Regarding software contributions, there has been a significant shift towards open-sourcing research outputs driven by initiatives from academic journals and community challenges. However, these contributions often lack modularity, focusing on specific problem settings. This hinders their broader reuse in related research areas and limits their practical applicability in diverse real-world scenarios. This thesis closes this gap by developing a software package that provides modular algorithmic components that are easily adaptable to custom problem settings.

Bibliography

Abdelwahed, Ayman et al. (Nov. 2020). “Evaluating and Optimizing Opportunity Fast-Charging Schedules in Transit Battery Electric Bus Networks”. In: *Transportation Science* 54(6), pp. 1601–1615. ISSN: 0041-1655. (Visited on 01/01/2022).

Accorsi, Luca and Daniele Vigo (July 2021). “A Fast and Scalable Heuristic for the Solution of Large-Scale Capacitated Vehicle Routing Problems”. In: *Transportation Science* 55(4), pp. 832–856. ISSN: 0041-1655. DOI: `10.1287/trsc.2021.1059`. (Visited on 01/20/2023).

Accorsi, Luca and Daniele Vigo (June 2023). *Routing One Million Customers in a Handful of Minutes.* Tech. rep. arXiv:2306.14205 [cs] type: article. arXiv. DOI: 10.48550/arXiv.2306.14205. URL: http://arxiv.org/abs/2306.14205 (visited on 12/31/2023).

Adler, Jonathan D. and Pitu B. Mirchandani (May 2017). "The Vehicle Scheduling Problem for Fleets with Alternative-Fuel Vehicles". In: *Transportation Science* 51(2), pp. 441–456. DOI: 10.1287/trsc.2015.0615.

Bazargan, Massoud (Mar. 2016). *Airline Operations and Scheduling.* Routledge. DOI: 10.4324/9781315566474.

Bent, Russell W. and Pascal Van Hentenryck (2004). "Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers". In: *Operations Research* 52(6), pp. 977–987.

Bolívar, Manuel A., Leonardo Lozano, and Andrés L. Medaglia (Dec. 2014). "Acceleration strategies for the weight constrained shortest path problem with replenishment". en. In: *Optimization Letters* 8(8), pp. 2155–2172. ISSN: 1862-4480. DOI: 10.1007/s11590-014-0742-x. URL: https://doi.org/10.1007/s11590-014-0742-x (visited on 11/12/2023).

Bunte, Stefan and Natalia Kliewer (Nov. 2009). "An overview on vehicle scheduling models". In: *Public Transport* 1(4), pp. 299–317. DOI: 10.1007/s12469-010-0018-5.

Desrosiers, J. et al. (1995). "Time Constrained Routing and Scheduling". English. In: *Handbooks in Operations Research and Management Science* 8(C), pp. 35–139. ISSN: 0927-0507. DOI: 10.1016/S0927-0507(05)80106-9.

DIMACS (Apr. 2022). *12th DIMACS Implementation Challenge: Vehicle Routing.* URL: http://dimacs.rutgers.edu/programs/challenge/vrp/.

Errami, Najib et al. (Apr. 2023). "VRPSolverEasy: a Python library for the exact solution of a rich vehicle routing problem". URL: https://inria.hal.science/hal-04057985.

Flatberg, Truls et al. (2007). "Dynamic And Stochastic Vehicle Routing In Practice". In: *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies.* Ed. by Vasileios Zeimpekis et al. Springer US: Boston, MA, pp. 41–63.

Gendreau, Michel et al. (1999). "Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching". In: *Transportation Science* 33(4), pp. 381–390.

Golden, Bruce, S. Raghavan, and Edward Wasil, eds. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges.* Springer US. DOI: 10.1007/978-0-387-77778-8.

Hildebrandt, Florentin D., Barrett W. Thomas, and Marlin W. Ulmer (2023). "Opportunities for reinforcement learning in stochastic dynamic vehicle routing". In: *Computers & Operations Research* 150, p. 106071.

Irnich, Stefan and Guy Desaulniers (2005). “Shortest Path Problems with Resource Constraints”. en. In: ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Springer US: Boston, MA, pp. 33–65. ISBN: 9780387254869. DOI: 10.1007/0-387-25486-2_2. URL: https://doi.org/10.1007/0-387-25486-2_2 (visited on 12/18/2023).

Joe, Waldy and Hoong Chuin Lau (June 2020). “Deep Reinforcement Learning Approach to Solve Dynamic Vehicle Routing Problem with Stochastic Customers”. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30(1), pp. 394–402.

Klabjan, Diego (2005). “Large-Scale Models in the Airline Industry”. en. In: ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Springer US: Boston, MA, pp. 163–195. ISBN: 9780387254869. DOI: 10.1007/0-387-25486-2_6. URL: https://doi.org/10.1007/0-387-25486-2_6 (visited on 12/18/2023).

Kool, Wouter, Laurens Bilek, et al. (2022). *The EURO Meets NeurIPS 2022 Vehicle Routing Competition.*

Kool, Wouter, Herke van Hoof, and Max Welling (2019). “Attention, Learn to Solve Routing Problems!” In: *International Conference on Learning Representations.*

Laporte, Gilbert (Nov. 2009). “Fifty Years of Vehicle Routing”. In: *Transportation Science* 43(4), pp. 408–416. ISSN: 0041-1655. DOI: 10.1287/trsc.1090.0301. URL: https://pubsonline.informs.org/doi/abs/10.1287/trsc.1090.0301 (visited on 11/05/2023).

Li, Jing-Quan (Nov. 2014). “Transit Bus Scheduling with Limited Energy”. In: *Transportation Science* 48(4), pp. 521–539. DOI: 10.1287/trsc.2013.0468.

Merchán, Daniel et al. (Sept. 2022). “2021 Amazon Last Mile Routing Research Challenge: Data Set”. In: *Transportation Science.* ISSN: 0041-1655. DOI: 10.1287/trsc.2022.1173. URL: https://pubsonline.informs.org/doi/10.1287/trsc.2022.1173 (visited on 12/18/2023).

Nazari, MohammadReza et al. (2018). “Reinforcement Learning for Solving the Vehicle Routing Problem”. In: *Advances in Neural Information Processing Systems.* Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc.

Ojeda Rios, Brenner Humberto et al. (2021). “Recent dynamic vehicle routing problems: A survey”. In: *Computers & Industrial Engineering* 160, p. 107604.

Parmentier, Axel (Apr. 2019). “Algorithms for non-linear and stochastic resource constrained shortest path”. en. In: *Mathematical Methods of Operations Research* 89(2), pp. 281–317. ISSN: 1432-5217. DOI: 10.1007/s00186-018-0649-x. URL: https://doi.org/10.1007/s00186-018-0649-x (visited on 11/12/2023).

Parmentier, Axel, Rafael Martinelli, and Thibaut Vidal (May 2023). "Electric Vehicle Fleets: Scalable Route and Recharge Scheduling Through Column Generation". In: *Transportation Science* 57(3), pp. 631–646. DOI: `10.1287/trsc.2023.1199`.

Pelletier, S., O. Jabali, and G. Laporte (2018). "Charge scheduling for electric freight vehicles". In: *Transportation Research Part B: Methodological* 115(115), pp. 246–269.

Perumal, Shyam S.G., Richard M. Lusby, and Jesper Larsen (Sept. 2022). "Electric bus planning & scheduling: A review of related problems and methodologies". In: *European Journal of Operational Research* 301(2), pp. 395–413. DOI: `10.1016/j.ejor.2021.10.058`.

Pessoa, Artur et al. (Sept. 2020). "A generic exact solver for vehicle routing and related problems". en. In: *Mathematical Programming* 183(1), pp. 483–523. ISSN: 1436-4646. DOI: `10.1007/s10107-020-01523-z`. URL: `https://doi.org/10.1007/s10107-020-01523-z` (visited on 12/18/2023).

Pollaris, Hanne et al. (Mar. 2015). "Vehicle routing problems with loading constraints: state-of-the-art and future directions". en. In: *OR Spectrum* 37(2), pp. 297–330. ISSN: 1436-6304. DOI: `10.1007/s00291-014-0386-3`. URL: `https://doi.org/10.1007/s00291-014-0386-3` (visited on 12/18/2023).

Raff, S. (1983). "Routing and scheduling of vehicles and crews. The state of the art". English. In: *Computers and Operations Research* 10(2), pp. 63–67, 69–115, 117–147, 149–193, 195–211. ISSN: 0305-0548. DOI: `10.1016/0305-0548(83)90030-8`.

Ralphs, Ted (Dec. 2023). *INFORMS Journal on Computing - Software Tools Area Editorial Statement*. Online, last accessed: 01.12.2023. URL: `https://pubsonline.informs.org/page/ijoc/editorial-statement#Software%20Tools`.

Raza, Syed Mohib, Mohammad Sajid, and Jagendra Singh (2022). "Vehicle Routing Problem Using Reinforcement Learning: Recent Advancements". In: *Advanced Machine Intelligence and Signal Processing*. Ed. by Deepak Gupta et al. Springer Nature Singapore: Singapore, pp. 269–280.

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

Sassi, O. and A. Oulamara (Jan. 2014). "Simultaneous Electric Vehicles Scheduling and Optimal Charging in the Business Context: Case Study". English. In: *IET Conference Proceedings*. Vol. 5. 5. Institution of Engineering and Technology, 6.3–6.3(1).

Sassi, O. and A. Oulamara (June 2016). "Electric vehicle scheduling and optimal charging problem: complexity, exact and heuristic approaches". In: *International Journal of Production Research* 55(2), pp. 519–535.

Smith, Olivia J., Natashia Boland, and Hamish Waterer (May 2012). "Solving shortest path problems with a weight constraint and replenishment arcs". In: *Computers & Operations Research* 39(5), pp. 964–984. ISSN: 0305-0548. DOI: 10.1016/j.cor.2011.07.017. URL: https://www.sciencedirect.com/science/article/pii/S0305054811002103 (visited on 11/12/2023).

Steever, Zachary, Mark Karwan, and Chase Murray (July 2019). "Dynamic courier routing for a food delivery service". In: *Computers & Operations Research* 107, pp. 173–188. ISSN: 0305-0548. DOI: 10.1016/j.cor.2019.03.008.

Toth, Paolo and Daniele Vigo, eds. (2014). *Vehicle routing. Problems, methods, and applications.* Second edition. MOS-SIAM series on optimization 18. Literaturangaben. Mathematical Optimization Society: Philadelphia. 463 pp. ISBN: 9781611973587.

Ulmer, Marlin W., Ninja Soeffker, and Dirk C. Mattfeld (2018). "Value function approximation for dynamic multi-period vehicle routing". In: *European Journal of Operational Research* 269(3), pp. 883–899.

van Kooten Niekerk, M. E., J. M. van den Akker, and J. A. Hoogeveen (June 2017). "Scheduling electric vehicles". In: *Public Transport* 9(1-2), pp. 155–176. DOI: 10.1007/s12469-017-0164-0.

Vidal, Thibaut (Apr. 2022). "Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood". en. In: *Computers & Operations Research* 140, p. 105643. ISSN: 0305-0548. DOI: 10.1016/j.cor.2021.105643. (Visited on 01/20/2023).

Vidal, Thibaut, Gilbert Laporte, and Piotr Matl (Oct. 2020). "A concise guide to existing and emerging vehicle routing problem variants". In: *European Journal of Operational Research* 286(2), pp. 401–416. DOI: 10.1016/j.ejor.2019.10.010.

Wen, M. et al. (Dec. 2016). "An adaptive large neighborhood search heuristic for the Electric Vehicle Scheduling Problem". In: *Computers & Operations Research* 76, pp. 73–83. DOI: 10.1016/j.cor.2016.06.013.

3 Combinatorial Optimization enriched Machine Learning to solve the Dynamic Vehicle Routing Problem with Time Windows

This chapter is based on an article published as:

Léo Baty, Kai Jungel, Patrick S. Klein, Axel Parmentier, Maximilian Schiffer (2024). Combinatorial Optimization-Enriched Machine Learning to Solve the Dynamic Vehicle Routing Problem with Time Windows. *Transportation Science* (Articles in advance).

Léo Baty*
CERMICS, École des Ponts, Champs sur Marne, France,
leo.baty@enpc.fr

Kai Jungel*
School of Management, Technical University of Munich, Munich, Germany,
kai.jungel@tum.de

Patrick S. Klein*
School of Management, Technical University of Munich, Munich, Germany,
patrick.sean.klein@tum.de

Axel Parmentier
CERMICS, École des Ponts, Champs sur Marne, France,
axel.parmentier@enpc.fr

Maximilian Schiffer
School of Management and Munich Data Science Institute, Technical University of Munich, Munich, Germany,
schiffer@tum.de

With the rise of e-commerce and increasing customer requirements, logistics service providers face a new complexity in their daily planning, mainly due to efficiently handling same-day deliveries. Existing multi-stage stochastic optimization approaches that allow to solve the underlying dynamic vehicle routing problem are either computationally too expensive for an application in online settings, or – in the case of reinforcement learning – struggle to perform well on high-dimensional combinatorial problems. To mitigate these drawbacks, we propose a novel machine learning pipeline that incorporates a combinatorial optimization layer. We apply this general pipeline to a dynamic vehicle routing problem with dispatching waves, which was recently promoted in the EURO Meets NeurIPS Vehicle Routing Competition at NeurIPS 2022. Our methodology ranked first in this competition, outperforming all other approaches in solving the proposed dynamic vehicle routing problem. With this work, we provide a comprehensive numerical study that further highlights the efficacy and benefits of the proposed pipeline beyond the results achieved in the competition, e.g., by showcasing the robustness of the encoded policy against unseen instances and scenarios.

Key words: vehicle routing, structured learning, multi-stage stochastic optimization, combinatorial optimization, machine learning

3.1. Introduction

With the rise of e-commerce during the last decade, logistics service providers (LSPs) were exposed to increasing customer requirements, particularly with respect to (fast) delivery times. Accordingly, the concept of same-day deliveries, where LSPs guarantee to fulfill an order on the day on which

* The first three authors contributed equally to this work.

they receive it, became a key element in the B2C sector. In fact, the same-day delivery market size in the U.S. is expected to increase from \$5.87B in 2019 to \$15.6B in 2024, and grew from \$7.14B to \$8.68B solely in 2021 (Placek 2022). As e-commerce and B2C deliveries remain competitive markets with several major players, retailers and LSPs continuously aim to outbid each other, which led to continuously shortened lead times, offering delivery within as little as two hours for certain product types in selected cities (Nicolai 2016).

Realizing last-mile deliveries within such short planning horizons remains inherently challenging from an efficiency perspective as LSPs generally trade off short lead times against oversized resources, e.g., by maintaining an oversized fleet to always be able to immediately react to incoming orders. In fact, the concept of same-day deliveries leads the concept of day-ahead planning, which has been the status quo in last-mile logistics for decades, ad absurdum. Instead of solving a combinatorially complex but static planning problem to determine cost-efficient delivery routes, LSPs have to dynamically dispatch orders to vehicles and route these vehicles in an online problem setting. Taking decisions in such a setting requires anticipating the benefit of dispatching or delaying an order while still inheriting the combinatorial complexity of the corresponding static planning problem and handling uncertainty with respect to future incoming orders.

The challenges that arise in such planning problems in practice invigorate the interest in dynamic vehicle routing problem (VRP) variants from a scientific perspective. State-of-the-art methodologies to solve such problems model the underlying planning task as a multi-stage stochastic optimization problem solved either directly (Pillac et al. 2013, Soeffker, Ulmer, and Mattfeld 2022) or via reinforcement learning (Nazari et al. 2018, Hildebrandt, Thomas, and Ulmer 2023, Basso et al. 2022). However, both of these approaches bear a major drawback. Reinforcement learning based algorithms succeed in taking anticipating decisions but often struggle when being applied to high-dimensional combinatorial problems. Contrarily, stochastic optimization techniques are generally amenable to combinatorial problem settings but struggle with respect to computational efficiency in high-dimensional problems, which makes them impracticable to use in a dynamic setting (Pflug and Pichler 2014, Carpentier et al. 2015).

Against this background, we propose a new methodological approach that mitigates the aforementioned shortcomings. Specifically, we develop a machine learning (ML) pipeline with an integrated combinatorial optimization (CO)-layer that allows to efficiently solve the dynamic VRP. This pipeline mitigates the challenges of multi-stage stochastic optimization problems by design: its ML-layer allows to incorporate uncertainty by adequately parameterizing an instance of the underlying deterministic CO problem, which can then be efficiently solved within the CO-layer. We used this pipeline in the EURO Meets NeurIPS Vehicle Routing Competition at NeurIPS 2022 (Kool et al. 2022a), where it outperformed all other approaches.

3.1.1. Related work

Our work contributes to two different streams of research: from an application perspective it relates to dynamic VRPs and from a methodological perspective it relates to CO-enriched ML. We provide an overview of both related research streams in the following. For a general overview of VRPs we refer to Vidal, Laporte, and Matl (2020) and for a general overview of CO-enriched ML we refer to Bengio, Lodi, and Prouvost (2021), and Kotary et al. (2021).

Dynamic Vehicle Routing Problems: Dynamic VRPs account for the dynamic nature of real-world processes where some problem data, such as the customers to serve, their demand, or the travel time between them, is not known in advance but revealed over time. Dynamic VRPs hence have a diverse field of applications ranging from ride-hailing (Jungel et al. 2023) over grocery delivery (Fikar 2018) to emergency services (Alinaghian, Aghaie, and Sabbagh 2019). To keep this literature review concise, we focus on dynamic VRPs in the context of dynamic dispatching problems in the following and refer to Pillac et al. (2013), Ojeda Rios et al. (2021) , and Ulmer, Soeffker, and Mattfeld (2018) for comprehensive reviews of the field. Approaches to solve dynamic VRPs can be broadly categorized into CO-based approaches, which leverage the combinatorial structure of the underlying problem, and ML-based approaches, which learn prescient policies accounting for the uncertainty of future customers.

Pure CO approaches generally amend solution methods developed for static VRPs to the dynamic case. Specifically, these CO approaches embed (meta-)heuristics into rolling horizon frameworks, i.e., solve a static variant of the considered problem each time new information enters the system (see, e.g., Ritzinger, Puchinger, and Hartl 2016, Ojeda Rios et al. 2021). Here, *myopic* approaches (see, e.g., Gendreau et al. 1999, Steever, Karwan, and Murray 2019) utilize only the current problem state, while *look-ahead* approaches (e.g., Flatberg et al. 2007, Bent and Van Hentenryck 2004) take into account potential realizations of future periods, e.g., via sampling.

ML approaches learn policies which account for the uncertainty of future observations. Accordingly, they model the underlying dynamic problem as a markov decision process solved with either approximate dynamic programming methods (see, e.g., Ulmer, Soeffker, and Mattfeld 2018), i.e., policy- or value function approximation, or reinforcement learning (see, e.g., Nazari et al. 2018, Kool, van Hoof, and Welling 2019, Joe and Lau 2020). We refer to Raza, Sajid, and Singh (2022) and Hildebrandt, Thomas, and Ulmer (2023) for a review on reinforcement learning applied to dynamic VRPs.

As can be seen, various works exist that solve dynamic VRPs. Here, most approaches either utilize classical CO algorithms by sampling future scenarios or apply ML to approximate decision values which account for future expected rewards. All of these approaches contain at minimum one of the following shortcomings: classical CO-based algorithms struggle to amend to real-time requirements of dynamic problem settings, while ML-based approaches often lack solution quality as they do not take the problem's combinatorial structure into account. A truly integrated approach which combines CO and ML, thus leveraging the advantage of ML in dynamic settings without disregarding the problem's combinatorial structure, has so far not been proposed for dynamic VRPs.

Combinatorial Optimization enriched Machine Learning: Many real-world combinatorial dispatching problems are subject to uncertain future events. To find combinatorial solutions which account for these uncertain events, one can integrate CO-layers into ML-based pipelines. These pipelines leverage an ML-layer to predict uncertain events which are then used to solve the combinatorial dispatching problem in the CO-layer. Classical pipelines follow the *predict-then-optimize* paradigm and train the ML-layer in isloation (Bent and Van Hentenryck 2004, Alonso-Mora, Wallar, and Rus 2017, Wallar et al. 2018, Liu, He, and Max Shen 2021). This strict separation of ML- and CO-layers during training however may result in an ML-layer which does not account for the influence of the prediction error on the decision error of the downstream CO-problem.

More recent approaches mitigate this issue by training the ML-layer based on the decision error incurred after applying the CO-layer instead of evaluating the prediction error of the ML-layer in isolation (cf. Elmachtoub and Grigas 2022). This so called *smart predict-then-optimize* framework has been applied successfully to several optimization problems, e.g., to leverage travel time predictions in shortest path and last-mile distribution problems (cf. Elmachtoub, Liang, and McNellis 2020, Chu et al. 2021, Elmachtoub and Grigas 2022). This framework is however limited to linear cost functions in the CO-layer, and requires target solutions for the ML-layer.

For problems with complex CO-layers or when target solutions are costly to obtain, *learning by experience* techniques are often used. One example in this domain is the use of reinforcement learning techniques for (contextual) multi-stage stochastic optimization problems. Here, integrated ML-CO pipelines help to factorize combinatorial action spaces that are intractable for standard reinforcement learning techniques. Such approaches have been successfully applied to vehicle-dispatching problems (Zhou et al. 2019, Tang et al. 2019, Liang et al. 2022, Enders et al. 2023) or inventory management (Akkerman et al. 2023). Other, i.e., non reinforcement-learning-based, techniques in *learning by experience* settings have been applied to a wider range of problems, but remain limited to non-dynamic problems (Dalle et al. 2022).

A trade-off between *smart predict-then-optimize* and *learning by experience* is *learning by imitation*, which bases the learning process on target decisions of the CO-layer. This allows to train the ML-layer solely on the output of the CO-layer, mitigating the need for specially labelled data to train the

ML-layer, which would be necessary in the *smart predict-then-optimize* framework.CO-enriched ML pipelines have already been applied to combinatorial problems such as the stochastic vehicle scheduling problem (Parmentier 2021, 2022), demonstrating good performance, especially with respect to generalization on large, unseen instances. Further research in this area extended the approach to multi-stage optimization problems, such as the two-stage stochastic minimum spanning tree problem (Dalle et al. 2022), and online vehicle dispatching and re-balancing in dynamic autonomous mobility-on-demand systems (Jungel et al. 2023). These works however remain limited in several fundamental aspects: first, the downstream CO-problem of their proposed CO-enriched ML pipelines is solveable in polynominal time. Second, the ML-layer used in Jungel et al. (2023) is, as a generalized linear model, relatively simple and does not utilize potential performance gains promised of a deep learning based ML-layer.

All these CO-ML approaches learn the ML predictor in an integrated way by directly backpropagating the output gradient of the CO-layer to the ML-layer. A major challenge of these so called CO-enriched ML pipelines is therefore the piecewise constant nature of most CO problems. Specifically, gradients are zero almost everywhere, and thus uninformative in such settings, rendering straightforward backpropagation ineffective. State-of-the-art methods address this issue and introduce regularization techniques that smoothen CO-layers to enable meaningful gradient computation, allowing their usage in ML-based pipelines. Both additive (Berthet et al. 2020) and multiplicative (Dalle et al. 2022) regularization approaches have been applied successfully to CO-enriched ML pipelines in *learning by imitation* settings with Fenchel-Young losses (Blondel, Martins, and Niculae 2020).

Concluding, CO-enriched ML pipelines have been successfully utilized to solve a variety of (real-world) stochastic optimization problems, but so far either train ML-layers in isolation, or remain limited to relatively simple CO-layers, which assumes CO-layers with linear objective functions of the form $y \mapsto \theta^\top y$ where the predicted objective costs θ have the same dimension as the decision variables y.

3.1.2. Contributions

To close the research gap outlined above, we propose a novel ML-based pipeline enriched with a CO-layer, and apply it to a novel class of dynamic VRPs introduced in the EURO Meets NeurIPS Vehicle Routing Competition that is highly interesting for academia and practice. Specifically, our work contains several contributions. From a methodological perspective, we generalize the CO-layer of CO-enriched ML pipelines to a more general and potentially non-linear class of objective functions in the CO-layer. Note that in this context, we also extend the open source library `InferOpt.jl` to such non-linear settings. Moreover, we present the first CO-enriched ML pipeline that utilizes a metaheuristic component to solve the CO-layer. By so doing, we show that our general ML-CO paradigm, which formally requires optimal solutions to derive true gradients, can work well with heuristic solutions in practice. In this context, we detail how to carefully design a metaheuristic that allows to derive heuristic solutions, i.e., approximate gradients, which enable convergence in the ML-layer of our pipeline. We show how to train the ML-layer of this pipeline in a supervised learning setting, i.e., based on a training set derived from an anticipative strategy. We present a comprehensive numerical study to show the efficacy of our methodology in a benchmark against state-of-the-art approaches. Beyond this, our study validates that our learning approach generalizes well to unseen scenarios. Interestingly, our results point at the fact that, counterintuitive to common practice, imitating an anticipative strategy can work well for high-dimensional multi-stage stochastic optimization problems.

We refer to our git repository (`https://github.com/tumBAIS/euro-meets-neurips-2022`) for instructions and all material necessary to reproduce the results outlined in this paper.

3.1.3. Outline

The remainder of this paper is organized as follows. Section 3.2 provides a formal definition of our problem setting before Section 3.3 introduces our CO-enriched ML pipeline. We then detail the individual layers of our pipeline. Specifically, Section 3.4 details the algorithmic framework used in the CO-layer, while Section 3.5 details the learning methodology for the ML-layer. Section 3.6 details the design of our computational study and its results. Finally, Section 3.7 concludes the paper.

3.2. Problem setting

Our problem setting focuses on a variant of the dynamic vehicle routing problem with time windows (VRPTW) introduced in the EURO Meets NeurIPS Vehicle Routing Competition (see Kool et al. 2022a). In this dynamic VRPTW, we aim to find a cost-minimal set of routes that start and end at a central depot d and allow a fleet of vehicles to serve a set of requests $\mathcal{R}$ within a finite planning horizon $[0, T_{\max}]$. We focus on an online problem setting in which the request set $\mathcal{R}$ is initially unknown and requests continuously arrive over $[0, T_{\max}]$. While the specifics of the request set are unknown, we assume knowledge of its distribution and refer to Paragraph 3.2.6 for a critical discussion. Within this planning horizon, the fleet operator needs to serve all arriving requests and makes dispatching decisions at (equidistant) time steps $\tau \in [0, T_{\max}]$. These dispatching decisions are immutable such that vehicles cannot be rerouted to serve additional requests after being dispatched. We discretize the planning horizon into a set of n epochs $E = \{[\tau_0, \tau_1], [\tau_1, \tau_2], \ldots, [\tau_{n-1}, \tau_n]\}$ and denote the start time of an epoch e as τ_e.

In each epoch, the fleet operator solves a dispatching and vehicle routing problem for the epoch dependent request set $\mathcal{R}^e$. Each request $r \in \mathcal{R}^e$ has a certain demand q_r, and vehicles have a homogeneous vehicle capacity Q, which limits the maximum number of requests serviceable on a single route. Serving a request r takes s_r time units, and a request must be served within a request-specific time window $[\ell_r, u_r]$. Traveling from the delivery location of request i to the delivery location of another request j takes t_{ij} time units and incurs cost c_{ij}. We can straightforwardly encode an instance of an epoch's planning problem on a fully-connected digraph $\mathcal{D}^e = (\mathcal{V}^e, \mathcal{A}^e)$ with a vertex set $\mathcal{V}^e = \mathcal{R}^e \cup \{d\}$ where d is the depot, and an arc set $\mathcal{A}^e$.

3.2.1. System state: We describe a *system state* at decision time τ_e as $x^e = \mathcal{R}^e$, where e is the epoch starting at time τ_e. Here, $\mathcal{R}^e$ contains all requests revealed but not yet dispatched. In our specific setting no further information is required to describe the system state as there is no fleet limit. Note that we use redundant notation $x^e = \mathcal{R}^e$ to adhere to conventions commonly used in the domains of ML and vehicle routing, respectively. We can distinguish requests contained in $\mathcal{R}^e$ into two disjoint categories: *must-dispatch* requests need to be dispatched in e as dispatching these in a later epoch would violate the requests' time window upon delivery; *postponable* requests can but do not have to be dispatched in e.

3.2.2. Feasible decisions: Given the current state of the system x^e, the fleet operator chooses a subset of $\mathcal{R}^e$ that will be served by vehicles leaving the depot in this epoch, and computes the respective routes that allow for vehicle dispatching. Vehicles dispatched in epoch e leave the depot at time $\tau_e + \Delta\tau$ and their routes cannot be modified once they have been dispatched, i.e., vehicles dispatched in epoch e cannot serve requests revealed in epoch e', with $\tau_{e'} > \tau_e$. In this context, a feasible decision $y^e \in \mathcal{Y}(x^e)$ in state x^e corresponds to a set of routes that

(i) contains all *must-dispatch* requests,

(ii) allows each route to visit all contained requests within their respective time windows, and

(iii) the cumulative customer demand on each route does not exceed the vehicle capacity Q.

We can encode a feasible decision $y^e \in \mathcal{Y}(x^e)$ with a vector $(y^e_{i,j})_{(i,j)\in\mathcal{A}^e}$ where

$$y^e_{i,j} = \begin{cases} 1 & \text{if } (i,j) \text{ is in a route of the solution} \\ 0 & \text{otherwise.} \end{cases}$$

3.2.3. System evolution: The system transitions into the next epoch e' once the fleet operator decides on $y^e \in \mathcal{Y}(x^e)$. To describe $x^{e'}$, we derive $\mathcal{R}^{e'}$ by removing all requests contained in y^e from $\mathcal{R}^e$, and adding all requests that enter the system between τ_e and $\tau_{e'}$.

3.2.4. Policy: Let $\mathcal{X}$ denote the set of potential system states. Then, a (deterministic) policy $\pi : \mathcal{X} \to \mathcal{Y}$ is a mapping that assigns a decision $y^e \in \mathcal{Y}(x^e)$ to any system state $x^e \in \mathcal{X}$.

3.2.5. Objective: We aim to find a policy that minimizes the expected cost of serving all requests $\mathcal{R}$ over the planning horizon $[0, T_{\max}]$. Formally,

$$\min_{\pi} \mathbb{E}\left[\sum_{e \in E} c(\pi(x^e))\right], \tag{1}$$

where $c : \mathcal{Y} \to \mathbb{R}$ gives the cost of routes $y \in \mathcal{Y}$.

3.2.6. Discussion: The problem setting defined and formalized above contains various assumptions that might be questioned from a practitioner's perspective. In particular, assuming an unlimited fleet size and full knowledge of the request distribution appears to be rather unrealistic. As this paper focuses on the methodology used to win the EURO Meets NeurIPS Vehicle Routing Competition, we decided to keep these assumptions without further questioning for the sake of consistency and reproducibility. However, we like to emphasize that the methodological pipeline presented in this paper is readily applicable to problem settings with limited fleet sizes and incomplete knowledge of the underlying request distribution as long as some historical data is available.

3.3. ML pipeline with CO-layer

To explain the rationale of our CO-enriched ML pipeline, we recall that a policy π maps a system state x^e to a feasible decision y^e in $\mathcal{Y}(x^e)$. The state x^e is a set of requests $\mathcal{R}^e$, and the solution y^e is a set of feasible decisions which encode the routes covering a subset of $\mathcal{R}^e$. The set of feasible decisions $\mathcal{Y}(x^e)$ hence coincides with the set of feasible solutions of a prize-collecting VRPTW, a variant of the (static) VRPTW where it is not mandatory to serve all requests, but a prize θ^e_j is collected if request j is served. In this section, we show that any decision y^e derived from an optimal policy in a dynamic problem setting corresponds to an optimal solution of a prize-collecting VRPTW for a well chosen prize vector $\theta^e = (\theta^e_j)_{j \in \mathcal{R}^e}$. However, finding prizes θ^e_j is non-trivial, such that we resort to ML for this purpose. Specifically, we introduce a family of policies $(\pi_w)_w$ encoded by the CO-enriched ML pipeline illustrated in Figure 1: in the ML-layer, a statistical model φ_w predicts θ^e based on the given system state x^e. This yields a prize-collecting VRPTW instance (x^e, θ^e) which we solve in the CO-layer with a dedicated algorithm f. The algorithm's output y^e then corresponds to our dispatching and routing decision.

In what follows, we first formally introduce the prize-collecting VRPTW and proof that we can represent every optimal decision in epoch e as a solution of a specially constructed prize-collecting VRPTW instance. We then detail how we design our pipeline to leverage this observation for finding dispatching and routing decisions.

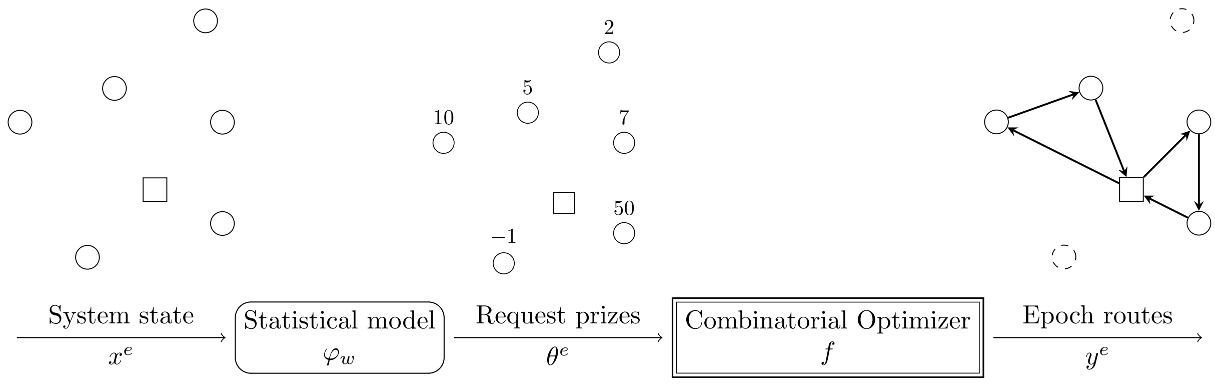


Figure 1 Our CO-enriched ML pipeline.

Prize-collecting VRPTW: Recall that $\mathcal{D}^e = (\mathcal{V}^e, \mathcal{A}^e)$ is a fully-connected digraph with vertex set $\mathcal{V}^e = \mathcal{R}^e \cup \{d\}$, comprising epoch requests $\mathcal{R}^e$ and the depot d, and that a feasible solution of the prize-collecting VRPTW $y^e \in \mathcal{Y}(x^e)$ can be encoded by the vector $(y^e_{i,j})_{(i,j)\in\mathcal{A}^e}$ where

$$y^e_{i,j} = \begin{cases} 1 & \text{if } (i,j) \text{ is in a route of the solution} \\ 0 & \text{otherwise.} \end{cases}$$

We further consider costs $(c_{i,j})_{(i,j)\in\mathcal{A}^e}$ on each arc, and prizes $(\theta^e_j)_{j\in\mathcal{R}^e}$ on each vertex. Then, we can state the objective of the prize-collecting VRPTW as follows,

$$\max_{y\in\mathcal{Y}(x^e)} \underbrace{\sum_{\substack{(i,j)\in\mathcal{A}^e \\ j\neq d}} \theta^e_j y_{i,j}}_{\text{total profit}} - \underbrace{\sum_{(i,j)\in\mathcal{A}^e} c_{i,j} y_{i,j}}_{\text{total routing cost}} . \tag{2}$$

Proposition 1. *For any x^e, there exists a $\theta \in \mathbb{R}^{|\mathcal{R}^e|}$ such that any optimal solution of* (2) *is an optimal decision with respect to* (1).

Proof Since the horizon is finite and the set of feasible decisions at each step is also finite, there exists an optimal decision $y^\star$ for x^e. Let $\bar{\mathcal{R}}^e$ be the subset of requests of $\mathcal{R}^e$ that are dispatched in $y^\star$. Then any solution y which has lower or equally low routing costs and covers $\bar{\mathcal{R}}^e$ exactly is also optimal. This follows from the Bellman equation since the routes have no impact on the evolution of the state. We can construct y by solving a prize-collecting VRPTW on $\mathcal{R}^e$ with request prizes

$$\bar{\theta}_j = \begin{cases} M & \text{if } j \in \bar{\mathcal{R}}^e \\ -M & \text{otherwise,} \end{cases} \tag{3}$$

where $M = \left(|\mathcal{R}^e| \cdot \max\limits_{(i,j)\in\mathcal{A}^e} c_{i,j}\right)$ is a large constant. The corresponding prize-collecting VRPTW solution y clearly covers $\bar{\mathcal{R}}^e$ exactly and has at most the routing cost of $y^\star$. □

CO-layer: We embed the prize-collecting VRPTW as a *layer* in our CO-enriched ML pipeline. Hence, this CO-layer must support forward and backward passes to assure compatibility with the ML-layer.

The forward pass simply solves the prize-collecting VRPTW instance defined by (x^e, θ^e) using a metaheuristic algorithm f detailed in Section 3.4. The backward pass backpropagates the gradient of the loss used in the learning algorithm through this layer. Section 3.5 introduces this loss and its gradient. To make their statement easier, we reformulate (2) as

$$f\colon \theta^e \mapsto \operatorname*{arg\,max}_{y\in\mathcal{Y}(x^e)} \theta^{e\top} g(y) + h(y) \quad \text{where} \quad g(y) = \left(\sum_{i\in\mathcal{V}^e} y_{i,j}\right)_{j\in\mathcal{R}^e} \text{ and } h(y) = \sum_{(i,j)\in\mathcal{A}^e} c_{i,j} y_{i,j}. \tag{4}$$

ML-layer: Finding the optimal prizes θ^e of Proposition 1 is non-trivial. We therefore use a statistical model φ_w to predict a vector of prizes $\theta^e = \varphi_w(x^e) \in \mathbb{R}^{|\mathcal{R}^e|}$ given the system state x^e. The only technical aspect from the ML perspective is that the dimensions of the input and the output are not fixed. Indeed, the number of requests may change from one state to another, and also differs across instances. We benchmark different choices for commonly used statistical models φ_w in our computational study (cf. Section 3.6).

Alternative ML-based methods: Alternative ML-based solution approaches presented in the EURO Meets NeurIPS Vehicle Routing Competition (e.g., van Doorn et al. 2022) generally proceed in two steps. They first apply a binary classifier, to decide on which requests to dispatch and postpone, respectively. They then construct routes covering these dispatched requests in a second step, essentially decoupling request dispatching from route construction. A major difficulty in this approach is to learn a statistical model which implicitly balances the current route costs and future route costs to find optimal dispatching decisions. We bypass this difficulty by taking dispatching and routing decisions simultaneously in our CO-layer (4), which allows us to train our statistical model based on the routing rather than the dispatching decision.

Generalization to other CO problems: The ML-CO pipeline methodology presented in this section can be used for a wider range of combinatorial problems. Indeed, our methodology can more generally be applied to any problem that can be modeled as a markov decision process. In practice, it is useful when the action space is combinatorial, which gives sense to using a CO-layer. For instance, we could handle the fixed fleet size variant of the Dynamic VRPTW by adding a constraint to the feasible set $\mathcal{Y}$ of the CO-layer. One can also use more different CO-layers in order to tackle other combinatorial and routing problems variants such as pick-up and delivery or scheduling.

3.4. Combinatorial optimization algorithm

To derive solutions of the prize-collecting VRPTW within our CO-layer, we propose a metaheuristic algorithm based on hybrid genetic search (HGS). Our algorithm extends the implementation of the HGS algorithm introduced by Kool et al. (2022b), which adapts the original HGS of Vidal (2022) to the VRPTW.

Tailoring the work of Kool et al. (2022b) to our problem setting requires various modifications to support optional requests, e.g., adaptions of the local search, initialization, and crossover procedures. We further introduce new mutation mechanisms, implement prize-collecting VRPTW specific neighborhoods, and allow to warm-start the population. In what follows, we summarize the core concepts of the HGS algorithm before detailing our prize-collecting VRPTW-specific modifications.

Hybrid Genetic Search for the Vehicle Routing Problem With Time Windows: The HGS algorithm is an evolutionary algorithm that maintains a *population* of solutions, organized in two disjoint subpopulations that contain feasible and infeasible solutions, respectively. The algorithm makes this population evolve over time, generating *offspring* solutions by combining promising *parent* solutions selected from the population in a randomized binary tournament. The algorithm then improves the generated offspring solution in a local search procedure. Note that it uses a penalty-based approach to explore infeasible regions of the solution space. The local search procedure yields a locally optimal solution, which is then added to the population. This may trigger a *survivor selection* procedure if the population size exceeds a certain threshold. This procedure eliminates a subset of solutions from the population. Survivor and parent selection are based on the *fitness* of a solution, a metric which captures the quality, i.e., objective value, of a solution and it's contribution to the population's diversity. This ensures a sufficiently diverse population, balancing diversification and intensification in the genetic algorithm. To utilize this algorithmic structure for our problem setting, i.e., the prize-collecting VRPTW, we applied the following adaptions and extensions, leading to the algorithm outlined in Figure 2.

Solution representation: We represent decisions on which requests to serve and which to ignore in our solution representation implicitly. Specifically, we use the same giant-tour representation as in Vidal (2022), but allow incomplete giant-tours. Here, requests deemed unprofitable are absent from the giant-tour and thus not considered by traditional local search operators.

Accounting for optional requests during crossover: The HGS proposed in Kool et al. (2022b) generates offspring solutions using two crossover operators: Ordered Crossover (`OX`) (Oliver, Smith, and Holland 2013) and Selective Route Exchange (`SREX`) (Nagata and Kobayashi 2010). As a first modification, we remove the `OX` operator as our benchmarking experiments indicate that it has no substantial

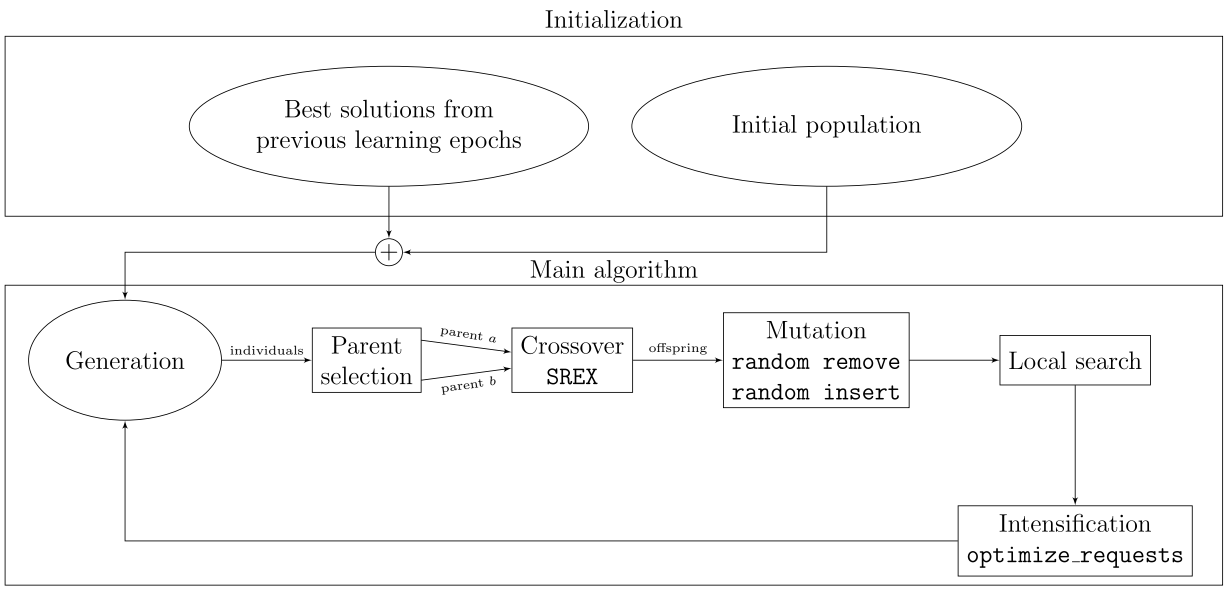


Figure 2 General structure of our metaheuristic algorithm.

Note. Round nodes indicate populations, square nodes indicate algorithmic components.

impact on the algorithm's performance for our problem variant. Our second modification amends the `SREX` operator to the prize-collecting VRPTW. Specifically, we preserve the set of requests served by the first parent in any generated offspring by re-inserting requests that are currently not served as in regular `SREX`, i.e., one at a time.

Additional diversification and intensification mechanisms: We further introduce two new mutation operators that diversify and intensify solutions based on the set of served requests. The first operator (`random remove/insert`), based on the result of a coin toss, either removes $\nu \cdot |R^\sigma|$ of the requests currently served, or inserts $\eta \cdot |C \setminus R^\sigma|$ of the currently unserved requests. This mutation occurs with a probability of ρ right after offspring generation.

The second operator (`optimize_request_set`) optimizes the set of requests served by a given solution. Specifically, this operator first removes any requests from the solution that cause a detour whose cost is higher than the request's profit, and re-inserts any profitable requests that are not part of the current solution. The operator perturbs insertion and removal costs with a random factor drawn uniformly from the interval $[\zeta^-, \zeta^+]$. In contrast to the first operator, we delay evaluating the second operator until the LS converges to avoid removing an excessive amount of requests due to poor solution quality. We run this operator with probability μ. To intensify our search in regions around promising solutions, we further run this operator (without perturbation) on any solution that improves on the current best solution.

Type	Name	Description
Traditional (Small)	`relocate`	removes a single request from it's route and re-inserts it at a different position in the solution
	`relocate pair`	removes a pair of consecutive requests from their route and re-inserts them at a different position in the solution
	`relocate reversed pair`	removes a pair of consecutive requests from their route and re-inserts them in reverse order at a different position in the solution
	`swap`	exchanges the position of two requests in the solution
	`swap pair`	exchanges the positions of two pairs of consecutive requests in the solution
	`swap pair with single`	exchanges the positions of a pair of consecutive requests with a single request in the solution
	`2-opt`	reverses a route segment
	`2-opt*`	splits two routes into two segments each, swapping a segment from the first with a segment from the second route
Traditional (Large)	`relocate*`	removes a request from it's route and inserts it at the best possible position in any spatially overlapping route
	`swap*`	removes requests a and b from routes r_a and r_b with spatial overlap, inserting a and b at the best position in r_b and r_a, respectively
PC-VRPTW	`serve request`	inserts a currently unserved request into the solution
	`remove request`	removes a request from the solution

Table 1 Local search operators

Local Search: Table 1 shows the local search operators used in our algorithm. We consider traditional and prize-collecting VRPTW specific operators. Specifically, traditional operators refer to those defined for the VRPTW (see Kool et al. 2022b, Vidal 2022). These traditionally explore neighborhoods by exchanging arcs within a solution, i.e., by changing the position of one or several requests in the current solution. Hence, they preserve the set of requests that receive service in a given solution. Prize-collecting VRPTW specific operators on the other hand work with the request set exclusively, that is, they remove or insert a request from a given solution. We evaluate these after evaluating traditional VRPTW operators to avoid removing profitable requests prematurely, i.e., due to bad routing decisions. As in Vidal (2022), we further distinguish between small and large neighborhood operators: small neighborhoods consider only those moves that involve requests which are geographically close and compatible w.r.t. their time windows. Large neighborhoods on the other hand remain unrestricted.

Initialization: We apply a pre-processing technique to account for extreme weights encountered during the learning procedure. Specifically, we maintain a set of certainly profitable and certainly unprofitable requests based on the maximum and minimum detour required, respectively. We impose constraints on our operators to avoid removing and including these profitable and unprofitable requests, respectively. We determine certainly unprofitable requests based on the following observation: if $\min_{j\in\mathcal{R}^e\cup\{d\}} c_{jr} + \min_{j\in\mathcal{R}^e\cup\{d\}} c_{rj} - \theta_r \geq \max_{i,j\in(\mathcal{R}^e\cup\{d\})^2} c_{i,j}$ holds for some request $r \in \mathcal{R}^e$, then for any route σ that includes r, a cheaper route σ' exists. To see this, let $u, v \in \mathcal{R}^e \cup \{d\}$ such that $(u,r),(r,v) \in \sigma$. Then $c_{u,r} + c_{r,v} - \theta_r \geq \max_{i,j\in(\mathcal{R}^e\cup\{d\})^2} c_{i,j} \geq c_{u,v}$, such that removing r from σ

yields a cheaper route. Analogously, we force the inclusion of a request $r \in \mathcal{R}^e$ if $c_{d,r} + c_{r,d} < \theta_r$ holds. This procedure further allows to account for must-dispatch customers during the evaluation phase by setting θ_r accordingly.

The magnitude of changes in customer prices decreases as the learning procedure converges. We exploit this behaviour in our algorithm and seed the initial population, generated as in Kool et al. (2022b), with solutions of previous learning epochs. Specifically, we generate promising solutions in a new construction heuristic which first applies our `optimize_customers` operator to the solution to account for changed request prizes, and then optimizes the routing decisions using the local search procedure. Note that this is only possible while training the pipeline. When evaluating the pipeline, we use starting solutions generated as in Kool et al. (2022b) exclusively.

Discussion: Focusing on our algorithmic design decisions, we note that we have tailored our algorithm to the unique challenges of our learning methodology and training environment. Specifically, successfully computing approximate gradients that ensure convergence during training requires solutions with low variance. Here, the algorithm needs to generalize well to the different prize distributions observed during training. Beyond this, it must be capable of providing such solutions within tight time constraints to allow training within a reasonable amount of time.

We have tackled these challenges with a design that focuses on guiding the population towards promising regions of the search space through aggressive diversification (e.g., randomized insertion and deletion of requests) and intensification (`optimize_request_set`) operators, explicit request insertion and deletion neighborhoods, and warmstarting. The resulting algorithm behaves more greedily than implicit approaches (e.g. Vidal et al. 2016), but converges reliably within the tight time constraints imposed during training. We further cope with extreme prizes observed during training in a preprocessing step.

3.5. Learning approach

The objective of our learning problem is to find parameters w such that the statistical model φ_w predicts a prize vector θ that leads to "good" decisions in the CO-layer. To reach this objective, we train our CO-enriched ML pipeline to imitate a good policy. To do so, we follow a supervised learning setting and therefore build a dataset $\mathcal{D} = \{(x^1, \bar{y}^1), \ldots, (x^n, \bar{y}^n)\}$ of state instances x^i with the decisions $\bar{y}^i$ taken by the imitated policy. In this supervised learning setting, we define a loss $\mathcal{L}(\theta, \bar{y})$ which quantifies the error when we predict $f(\theta)$ instead of $\bar{y}$. Then, we formulate the learning problem as finding the parameter $\hat{w}_{\mathcal{D}}$ that minimizes the empirical risk, i.e., the average loss on the training dataset,

$$\hat{w}_{\mathcal{D}} = \arg\min_{w} \sum_{i=1}^{n} \mathcal{L}(\varphi_w(x^i), \bar{y}^i). \tag{5}$$

The rest of this section introduces the imitated policy, the training set, the loss, and our algorithm to solve learning problem (5).

3.5.1. Anticipative decisions in the training set

The dynamic VRPTW is difficult because future requests are unknown when taking decisions. If we know the future, i.e., if we knew all the requests at the beginning of the horizon, we would just have to solve the corresponding static VRPTW at the beginning of the horizon, and then take the corresponding dispatching decisions at each epoch. This procedure would give us the optimal anticipative policy. However, such an anticipative policy can of course not be used in practice because it relies on unavailable information. Nonetheless, we can rebuild the decisions taken by the anticipative policy a posteriori, when all information has been revealed, i.e., on historical data. We therefore can learn to imitate an anticipative policy, which we gained from historical data, in our learning problem (5).

Practically, we rebuild the decision of the anticipative policy as follows. We know all the requests of an historical VRPTW instance. Then, we associate a *release time* to each request i, which is equal to the time τ_i at which the request would be revealed in a dynamic setting, so the earliest point in time at which request i can be served. We then seek routes to serve all these requests at minimum cost while respecting the release times and the time windows. We use the adapted HGS of Kool et al. (2022b) to solve this variant of the static VRPTW. This yields a set of routes P from which we reconstruct the decisions taken in each epoch as follows. For route $p \in P$, let $\tau_p = \max_{r \in p} \tau_r$ be the latest release time of any request route p serves. The anticipative policy dispatches route p in the first epoch e where $\tau_e \geq \tau_p$, i.e., the first epoch where all served requests have been revealed.

Instead, one could also derive an anticipative decision by dispatching a route at the latest possible epoch, such that the route is still feasible with respect to the requests' time windows. This may increase the degree of freedom for decisions in subsequent periods. However, in the environment studied, there is no leeway in the time windows to do so, such that early dispatching and late dispatching lead to almost identical decisions. While we accordingly excluded such a strategy from this study, it may prove worthwhile to consider a delayed anticipative decision when working in a different problem setting.

3.5.2. Loss function

We recall that the objective of our learning problem is to find parameter values w of a statistical model φ_w such that for any state x, the CO-layer predicts a good decision $y = f(\theta)$ for the prize-collecting VRPTW instance $(x, \theta = \varphi_w(x))$. More precisely, for each state-decision pair $(x, \bar{y})$ in the data set, we want the target decision $\bar{y}$ to be as close as possible to the optimal solution of the prize-collecting VRPTW (2). Hence, it is natural to take the non-optimality of $\bar{y}$ as a solution of $f(\theta)$ as loss function:

$$\mathcal{L}(\theta,\bar{y}) = \max_{y\in\mathcal{Y}(x)} \{\theta^\top g(y) + h(y)\} - (\theta^\top g(\bar{y}) + h(\bar{y})), \tag{6}$$

with g and h defined in (4). Note that what follows in this subsection is valid for any CO-layer of the form $f : \theta \mapsto \arg\max_y \theta^\top g(y) + h(y)$, for any g and h.

We clearly have $\mathcal{L}(\theta,\bar{y}) \geq 0$ in general, and $\mathcal{L}(\theta,\bar{y}) = 0$ only if y is an optimal solution of (4). Unfortunately, the polyhedron $\mathcal{P}(\bar{y}) = \{\theta \in \mathbb{R}^{|x|} \,|\, \theta^\top g(\bar{y}) + h(\bar{y}) \geq \theta^\top g(y) + h(y), \forall y \in \mathcal{Y}(x)\}$ is generally highly degenerate. For instance, if $h = 0$, then $\theta = 0$ belongs to $\mathcal{P}(y)$ for all y in $\mathcal{Y}$. This implies that $\theta = 0$ would be an optimum of our learning problem, which is a problem because such a θ allows our CO-layer to return any solution in $\mathcal{Y}(x)$. This degeneracy can be removed by considering the perturbed loss

$$\mathcal{L}_\varepsilon(\theta,\bar{y}) = \mathbb{E}\Big[\max_{y\in\mathcal{Y}(x)} \{(\theta+\varepsilon Z)^\top g(y) + h(y)\}\Big] - (\theta^\top g(\bar{y}) + h(\bar{y})). \tag{7}$$

where $\varepsilon > 0$ and $Z \sim \mathcal{N}(0, \mathbf{I}_{|x|})$ is a Gaussian perturbation. The perturbed loss $\mathcal{L}_\varepsilon$ has been considered in the literature only in the case where $g(y) = y$ and $h(y) = 0$ (cf. Berthet et al. 2020). Due to non-zero h, the geometry of the loss changes. Notably, while the size of ε does not matter when $h = 0$ (cf. Parmentier and T'Kindt 2021), it does matter when h is non-zero: the larger ε, the smaller the impact of h on the prediction. Proposition 2 summarizes the geometry of these losses in our more general context.

Proposition 2. *Let $x \in \mathcal{X}$, $\bar{y} \in \mathcal{Y}(x)$. Let $\mathcal{C}(\bar{y}) = \{\theta \in \mathbb{R}^{|x|} : \theta^\top g(\bar{y}) \geq \theta^\top g(y), \forall y \in \mathcal{Y}(x)\}$ be the* normal cone *associated to $g(\bar{y})$.*

1. *$\theta \mapsto \mathcal{L}(\theta,\bar{y})$ is piecewise linear and convex, with subgradient*

$$\underbrace{g\Big(\arg\max_{y\in\mathcal{Y}(x)} \theta^\top g(y) + h(y)\Big)}_{=g(f(\theta))} - g(\bar{y}) \in \partial_\theta \mathcal{L}(\theta,\bar{y}).$$

2. *$\theta \mapsto \mathcal{L}_\varepsilon(\theta,\bar{y})$ is $\mathcal{C}^\infty$ and convex with gradient*

$$\nabla_\theta \mathcal{L}_\varepsilon(\theta,\bar{y}) = \mathbb{E}\left[g\left(\arg\max_{y\in\mathcal{Y}(x)} (\theta+\varepsilon Z)^\top g(y) + h(y)\right)\right] - g(\bar{y}) = \mathbb{E}[g(f(\theta+\varepsilon Z))] - g(\bar{y}).$$

3. *$\mathcal{L}_\varepsilon(\theta,\bar{y}) \geq \mathcal{L}(\theta,\bar{y})$.*
4. *$\mathcal{C}(\bar{y})$ is the recession cone of $\mathcal{P}(\bar{y})$.*
5. *Let $\theta \in \mathbb{R}^{|x|}$. If η is in $\mathcal{C}(\bar{y})\backslash\{0\}$, then $\lambda \mapsto \mathcal{L}(\theta+\lambda\eta,\bar{y})$ is non increasing. If in addition $\mathcal{C}(\bar{y}) \neq \mathbb{R}^{|x|}$, then $\lambda \mapsto \mathcal{L}_\varepsilon(\theta+\lambda\eta,\bar{y})$ is decreasing.*
6. *Let $\theta \in \mathbb{R}^{|x|}$. If η is in the interior $\mathring{\mathcal{C}}(\bar{y})$ of $\mathcal{C}(\bar{y})$, then $\lim_{\lambda\to\infty} \mathcal{L}(\theta+\lambda\eta,\bar{y}) = \lim_{\lambda\to\infty} \mathcal{L}_\varepsilon(\theta+\lambda\eta,\bar{y}) = 0$.*

Proof 1. $\theta \mapsto \mathcal{L}(\theta, \bar{y})$ is linear and convex as it is the maximum of mappings that are linear in θ. Let $(\theta, \tilde{\theta}) \in \mathbb{R}^{|x|} \times \mathbb{R}^{|x|}$. Let y^* be in $\arg\max_{y \in \mathcal{Y}(x)} \theta^\top g(y) + h(y)$ and $\tilde{y}$ in $\arg\max_{y \in \mathcal{Y}(x)} \tilde{\theta}^\top g(y) + h(y)$. By definition of $\tilde{y}$, we have $\tilde{\theta}^\top g(\tilde{y}) + h(\tilde{y}) \geq \tilde{\theta}^\top g(y^*) + h(y^*)$, which gives $\mathcal{L}(\tilde{\theta}, \bar{y}) \geq \mathcal{L}(\theta^*, \bar{y}) + (\tilde{\theta} - \theta^*)^\top \big(g(y^*) - g(\bar{y})\big)$ and the subgradient in $\partial_\theta \mathcal{L}(\theta, \bar{y})$.

2. Since $\mathcal{L}_\varepsilon(\theta, y) = \mathbb{E}[\mathcal{L}(\theta + \varepsilon Z, y)]$, we obtain that $\theta \mapsto \mathcal{L}_\varepsilon(\theta, y)$ is C^∞ as a convolution product of $\theta \mapsto \mathcal{L}(\theta, \bar{y})$ with a Gaussian density (which is C^∞), and is convex as an expectation of a convex function. The gradient follows from the subgradient of $\mathcal{L}(\theta, y)$.

3. Let $\theta \in \mathbb{R}^{|x|}$. For all $\tilde{y}$ in $\mathcal{Y}(x)$, we have

$$\begin{aligned}
\text{for all } z \in \mathbb{R}^{|x|}, \quad & \max_y (\theta + \varepsilon z)^\top g(y) + h(y) \geq (\theta + \varepsilon Z)^\top g(\tilde{y}) + h(\tilde{y}) \\
\text{hence,} \quad & \mathbb{E}[\max_y (\theta + \varepsilon Z)^\top g(y) + h(y)] \geq \theta^\top g(\tilde{y}) + h(\tilde{y}) \quad (\text{since } \mathbb{E}[Z] = 0) \\
\text{and,} \quad & \mathbb{E}[\max_y (\theta + \varepsilon Z)^\top g(y) + h(y)] - (\theta^\top g(\bar{y}) + h(\bar{y})) \geq \theta^\top g(\tilde{y}) + h(\tilde{y}) - (\theta^\top g(\bar{y}) + h(\bar{y})).
\end{aligned}$$

Finally: $\mathbb{E}[\max_y (\theta + \varepsilon Z)^\top g(y) + h(y)] - (\theta^\top g(\bar{y}) + h(\bar{y})) \geq \max_y \{\theta^\top g(y) + h(y)\} - (\theta^\top g(\bar{y}) + h(\bar{y}))$, i.e. $\boxed{\mathcal{L}_\varepsilon(\theta, \bar{y}) \geq \mathcal{L}(\theta, \bar{y})}$.

4. Let $\theta \in \mathcal{P}(\bar{y})$. We have $\forall y$, $(\theta + \lambda\eta)^\top g(\bar{y}) + h(\bar{y}) \geq (\theta + \lambda\eta)^\top g(y) + h(y) \Leftrightarrow \forall y$, $\theta^\top g(\bar{y}) + h(\bar{y}) + \lambda\eta^\top \big(g(\bar{y}) - g(y)\big) \geq \theta^\top g(y) + h(y)$. If η is in $\mathcal{C}(\bar{y})$, then $\eta^\top \big(g(\bar{y}) - g(y)\big) \geq 0$, and $\theta + \lambda\eta$ is in $\mathcal{P}(\bar{y})$ for any $\lambda > 0$. If η is not in $\mathcal{C}(\bar{y})$, then $\eta^\top \big(g(\bar{y}) - g(y)\big) < 0$ and there exists a $\lambda > 0$ such that $\theta + \lambda\eta$ is not in $\mathcal{P}(\bar{y})$.

5. Let $\eta \in \mathcal{C}(\bar{y})$. Let us denote by $\hat{y} = \arg\max_y (\theta + \lambda\eta)^\top g(y) + h(y)$. We have

$$\text{for all } \lambda, \quad \mathcal{L}(\theta + \lambda\eta, \bar{y}) = (\theta + \lambda\eta)^\top g(\hat{y}) + h(\hat{y}) - [(\theta + \lambda\eta)^\top g(\bar{y}) + h(\bar{y})]$$

By taking the derivative with respect to λ, we obtain: $\eta^\top (g(\hat{y}) - g(\bar{y}))$ which is negative by definition of η. This gives us the non increasing-property of $\lambda \mapsto \mathcal{L}(\theta + \lambda\eta, \bar{y})$.

Similarly, by denoting $\hat{y}_\varepsilon(Z) = \arg\max_y (\theta + \lambda\eta + \varepsilon Z)^\top g(y) + h(y)$ for all Z, we obtain the derivative of $\lambda \mapsto \mathcal{L}_\varepsilon(\theta + \lambda\eta)$: $\eta^\top (\mathbb{E}[g(\hat{y}_\varepsilon(Z))] - g(\bar{y}))$. If $\mathcal{C}(\bar{y}) \neq \mathbb{R}^{|x|}$, then $\mathbb{P}(\theta + \lambda\eta + \varepsilon Z \notin \mathcal{C}(\bar{y})) > 0$. Hence, $\mathbb{E}[g(\hat{y}_\varepsilon(Z)] \neq g(\bar{y})$, $\eta^\top (\mathbb{E}[g(\hat{y}_\varepsilon(Z)] - g(\bar{y})) < 0$, and therefore $\lambda \mapsto \mathcal{L}_\varepsilon(\theta + \lambda\eta)$ is decreasing.

6. Let $\eta \in \mathring{\mathcal{C}}(\bar{y})$. By definition of $\mathring{\mathcal{C}}(\bar{y})$, we have $\eta^\top (\bar{y} - y) > 0$ for all $y \neq \bar{y}$. Hence, there exists $M > 0$ such that for all $\lambda \geq M$ and $y \neq \bar{y}$, we have $(\theta + \lambda\eta)^\top \bar{y} + \geq (\theta + \lambda\eta)^\top y$. That is, for all $\lambda \geq M$ we have $\theta + \lambda\eta \in \mathcal{P}(\bar{y})$, i.e. $\mathcal{L}(\theta + \lambda\eta, \bar{y}) = 0$. Hence $\boxed{\lim_{\lambda \to \infty} \mathcal{L}(\theta + \lambda\eta, \bar{y}) = 0}$.

If $\mathcal{C}(\bar{y}) \neq \mathbb{R}^{|x|}$, from point 5 and the previous limit, we get that $\lambda \mapsto \mathcal{L}(\theta + \varepsilon z + \lambda\eta, \bar{y})$ monotonically decreases to zero for any z. The monotone convergence theorem therefore gives $\boxed{\lim_{\lambda \to \infty} \mathcal{L}_\varepsilon(\theta + \lambda\eta, \bar{y}) = 0}$. If $\mathcal{C}(\bar{y}) = \mathbb{R}^{|x|}$, the result is immediate. □

The perturbed loss $\mathcal{L}_\varepsilon$ has properties that make it very suitable for a learning problem. Indeed, it is convex and smooth, and while the expectation in its gradient is intractable, sampling Z gives a stochastic gradient. The learning problem (5) with loss $\mathcal{L}_\varepsilon$ can therefore be solved using stochastic gradient descent. Furthermore, $\theta \mapsto \mathcal{L}_\varepsilon(\theta, \bar{y})$ tends to 0 only if θ is far in the interior of $\mathcal{C}(\bar{y})$, which means that there is no ambiguity on the fact that $\bar{y}$ is an optimal solution of (4) for θ, and removes the degeneracy issue of $\theta \mapsto \mathcal{L}(\theta, \bar{y})$. This situation is illustrated in two dimensions in Figure 3. Figure 3a represents the polytope corresponding to the convex envelope $\mathrm{Conv}(g(\mathcal{Y})) = \mathrm{Conv}(\{g(y), \forall y \in \mathcal{Y}\})$. The algorithm f necessarily outputs a vertex of this polytope (red square) as the objective function of the CO-layer is linear in $g(y)$. The dark blue hexagon is the output $\mathbb{E}[g(f(\theta + \varepsilon Z))]$ of the perturbed CO-layer, which can be seen as the expectation over a distribution (blue circles) on the vertices of the polytope. On Figure 3b, with $h = 0$ and no perturbation, we can see that the loss $\mathcal{L}(\theta, \bar{y})$ is 0 for any θ in the normal cone of $\bar{y}$ (with $g(\bar{y})$ being the red square), and that the origin belongs to the normal cone of any vertex y. On Figure 3c, the perturbation has been added and we can see that $\mathcal{L}(\theta, \bar{y})$ goes to zero only when we are safely inside the cone, which means that the output of the combinatorial optimization layer is non-ambiguous. Finally, on Figure 3d and 3e, we can see that a non-zero h changes the geometry for small θ values but not for large θ values.

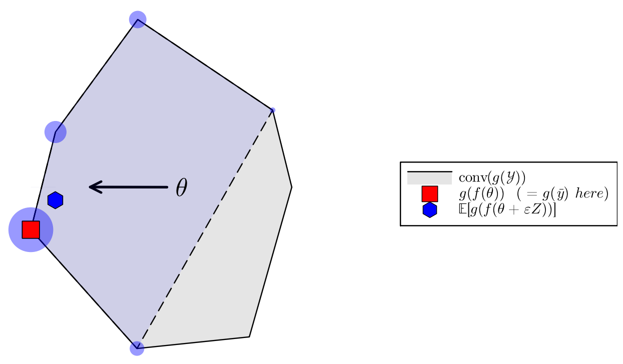
Non-optimal CO-layer: Note that our CO-layer f is a (meta)-heuristic algorithm and thus does not guarantee an optimal solution, which is not in line with the assumptions of the theory reviewed in this section. However, we observe that this is not a problem in practice as f outputs solutions close enough to optimal ones.

Generalization to other CO problems: As mentioned at the beginning of this subsection, this learning approach works for any CO-layer of the form (4) for any g, h, and feasible set $\mathcal{Y}$. Additionally, at the end of Section 3.3 we explain that we can build a pipeline for any problem modeled as markov decision process, the only requirement in practice being able to compute $f(\theta)$. However, for being able to learn this pipeline, we need an additional assumption: the dynamic of the system needs to be of the form $x_{t+1} = f(x_t, y_t, \psi_t)$, with ψ_t an exogeneous noise independent of decision y_t. This is needed in order to derive the anticipative policy to imitate.

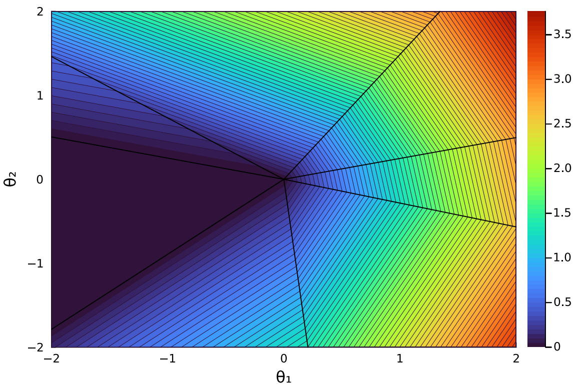
Perturbed loss and Fenchel-Young loss: The perturbed loss $\mathcal{L}_\varepsilon$ defined in (7) is a generalized version the Fenchel-Young loss defined in Berthet et al. (2020). It has a missing term $C(\hat{y})$, a constant depending on the target solution only, and therefore not affecting the gradient with respect to θ.

3.6. Computational study

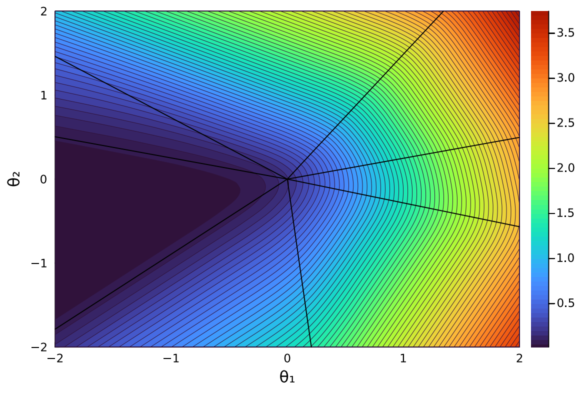
The aim of our computational study is twofold. First, we validate the performance of our CO-enriched ML pipeline in a benchmark against several state-of-the-art approaches. Second, we conduct extensive numerical experiments to assess the impact of different training settings on the performance of our CO-enriched ML pipeline. Specifically, we investigate the impact of i) the feature set, ii) the size of


θ
$\mathrm{conv}(g(\mathcal{Y}))$
$g(f(\theta))$ $(= g(\bar{y})$ *here*$)$
$\mathbb{E}[g(f(\theta+\varepsilon Z))]$

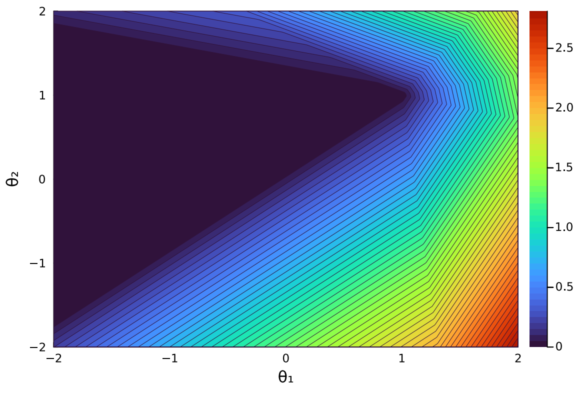
(a) Polytope $\mathrm{Conv}(g(\mathcal{Y}))$



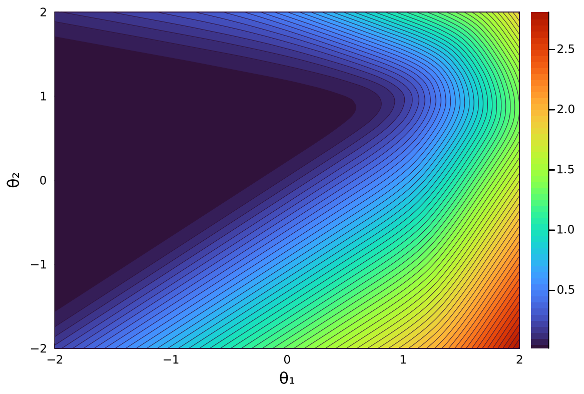
(b) $\theta \mapsto \mathcal{L}(\theta, \bar{y})$ for $h = 0$



(c) $\theta \mapsto \mathcal{L}_\varepsilon(\theta, \bar{y})$ for $h = 0$



(d) $\theta \mapsto \mathcal{L}(\theta, \bar{y})$ for $h \neq 0$



(e) $\theta \mapsto \mathcal{L}_\varepsilon(\theta, \bar{y})$ for $h \neq 0$

Figure 3 **Example with two-dimensional θ and its respective polytope 3a. Contour plots of the loss $\mathcal{L}(\cdot, \bar{y})$ and its perturbed version $\mathcal{L}_\varepsilon(\cdot, \bar{y})$ for two different values of h. Thick lines on 3b and 3c represent the normal fan associated to the polytope. We can see the perturbed regularization "pushing" the loss inside the normal cone $\mathcal{C}(\bar{y})$.**

instances in the training set, iii) the training set size, iv) the imitated target strategy, and v) the type of statistical model used.

For this purpose, we first detail the design of our computational study in Section 3.6.1, and the different benchmarks in Section 3.6.2. We then present the results of our benchmark study in Section 3.6.4, and discuss the results of our experiments on different training settings in Section 3.6.5.

3.6.1. Design of experiments

We design our computational study in line with the problem setting presented in the EURO Meets NeurIPS Vehicle Routing Competition (Kool et al. 2022a). We consider a set of VRPTW instances derived from real-world data of a US-based grocery delivery service. We refer to these instances as *static* instances. A static instance contains a set of requests, each with a specific location, a service time, a demand, and a time window. Hence, each instance implicitly defines a distribution of request locations, service times, demands, and time windows. We generate an instance of the *dynamic* problem from a static instance as follows: we first discretize the planning horizon of the static instance, which is between 5 and 9 hours, into one hour epochs. Then, for each epoch e, we sample m requests randomly, constructed by drawing from the static instance's location, demand, service time, and time window distributions independently. We refer to m as the *sample size* in the remainder of this section. We discard requests that are infeasible with respect to the starting time of the current epoch and the chosen time window. The remaining requests form the set of requests revealed in epoch e. This sampling approach has two implications: First, the expected number of requests arriving in each epoch decreases with advancing epochs as the probability of sampling a feasible time window decreases. Second, time windows that end closer to the end of the planning horizon are more likely to appear, as these time windows have a higher probability to be feasible. In line with the challenge's problem setting, we minimize travel time only, such that the number of vehicles, service times, and waiting times are not part of the objective. Furthermore, we do not limit the number of vehicles and assume knowledge over the static instance's request distribution. Finally, we note that we can generate several dynamic instances from one static instance by varying the seed used to sample from the request distribution. In what follows, we refer to this seed as the *instance seed*.

While we already discussed shortcomings of the problem setting in Section 3.2, a few comments on the problem setting, especially with respect to the epoch length and unlimited vehicle fleet are in order: When interpreting our results, one should be aware of the fact that the epoch length of one hour limits the responsiveness of the system to new requests. However, our methodology is not dependent on the epoch length. We can vary the epoch length between a dynamic epoch length, which defines the time between two requests entering the system, or a rather long static epoch length, as considered in this experimental design, as long as we can state the underlying problem as a markov decision process. A dynamic short epoch length leads to an immediate response to new requests when entering the system. The unlimited vehicle fleet size is also no methodological limitation in practice, as we can introduce a limited fleet size by adapting the constraints in the CO-layer. In parallel works, we provide evidence that our methodology is not only in theory applicable to such settings but also provides a competitive performance. For details, we refer the interested reader to Jungel et al. (2023) where we study a mobility-on-demand problem setting with a limited fleet size in which we use a point process to model requests entering the system, leading to a short response time of one minute. In this setting, our methodological paradigm improves upon existing benchmarks and allows to provide a new state-of-the-art.

3.6.2. Benchmark policies

We evaluate the performance of our CO-enriched ML pipeline against a set of benchmark policies. These policies follow a two-stage approach by first deciding on the set of requests to dispatch and then form covering routes in a second step by solving a VRPTW on the dispatched requests using the HGS algorithm (cf. Kool et al. 2022b, Vidal 2022). For each benchmark policy, we allow a time limit for finding decisions of 90 seconds per epoch, if not stated differently. In what follows, we briefly introduce the benchmark policies considered in this computational study.

Greedy policy: The greedy policy dispatches all requests as soon as they enter the system.

Lazy policy: The lazy policy delays dispatching a request as long as possible, i.e., to the epoch where it becomes must-dispatch.

Random policy: The random policy dispatches postponable requests with a probability of 50%. It further dispatches all must-dispatch requests.

Rolling-horizon policy: The rolling-horizon policy samples a scenario for the remaining epochs, and applies the HGS in a similar way as the anticipative strategy assuming that the sampled scenario represents the true scenario. This yields a set of routes based on which it decides which requests to dispatch. Specifically, it dispatches a request of the current epoch if and only if that request shares a route with a must dispatch request of the current epoch. We assign a time limit of 600 seconds to ensure convergence of the HGS since this policy entails solving a VRPTW on a complete scenario. Note that this extends the 90 second time limit allowed to the other benchmark policies by 510 seconds.

Monte-carlo policy: The monte-carlo policy samples nine scenarios for the remaining epochs, solved individually as in the rolling-horizon policy. It dispatches a request based on a majority decision, i.e., if and only if the rolling-horizon policy dispatches the request in at least 50% of the sampled scenarios. We raise the time-limit accordingly, i.e., allow a total of 5400 seconds.

ML-CO policy: The ML-CO policy is encoded in the CO-enriched ML pipeline introduced in Section 3.3. Unless specified otherwise, we train our ML-CO policy on a set of 15 training instances using a sample size of 50 requests per epoch, derived using the anticipative strategy as detailed in Section 3.5.1. Remark that we train on some instances and generalizes on others to stick to the challenge practice. Note that a firm that would use such a policy in practice may have one single instance corresponding to a city. It would use a training set with some scenarios and generalize on a test set with different scenarios. Such a setting enables to overfit the instance and get performance on it. We limit the runtime of the HGS used to derive the anticipative solutions to 3600 seconds. Our ML-layer uses a feedforward neural network with four hidden layers, each comprising ten neurons. For the ML-CO policy, we set the time limit for finding decisions to 90 seconds per epoch. Table 2 indicates that this time limit does not impact the performance of our ML-CO policy significantly.

	Decision time limit during evaluation [seconds]					
	30	60	90	120	180	240
Relative distance to anticipative baseline	5.66%	5.18%	5.15%	5.01%	4.89%	4.87%

Table 2 Performance of our ML-CO policy for different time limits that we allow for the PC-HGS during evaluation.

We note that the *greedy*, *lazy*, and *random* policies are the baseline heuristics used in the EURO Meets NeurIPS Vehicle Routing Competition and refer the interested reader to Kool et al. (2022a) for more details.

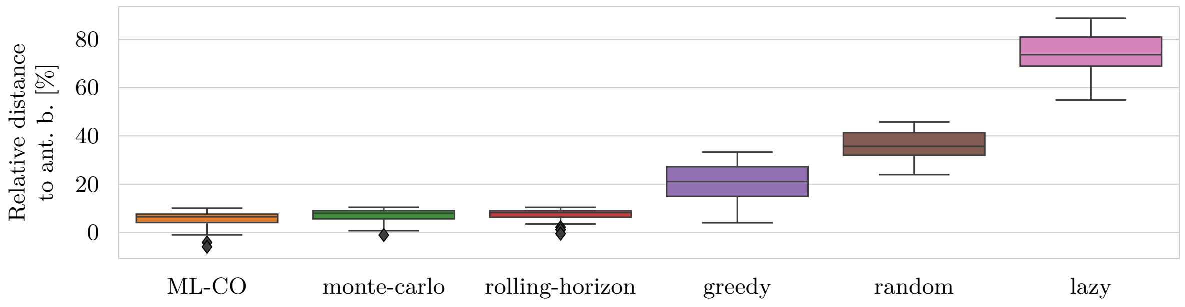
We additionally include the anticipative strategy used during training (cf. Section 3.5.1) as a baseline. To derive the anticipative baseline, we solve the offline VRPTW using the HGS algorithm proposed in Kool et al. (2022b) with a time limit of 3600 seconds. Note that the HGS algorithm is a heuristic which implies that it is possible to find a solution which outperforms the anticipative baseline.

3.6.3. Study Setup

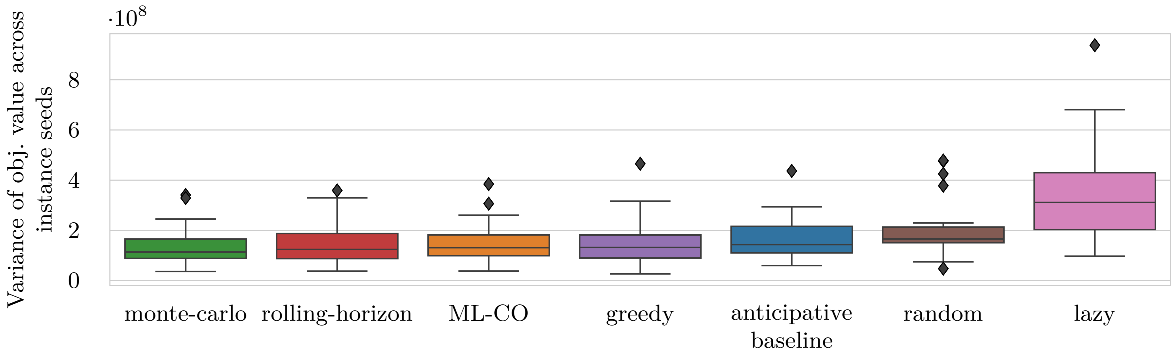
We evaluate the performance of the benchmark policies on a total of 500 dynamic instances generated from 25 different request distributions, i.e., static instances (cf. Section 3.6.1). Our results base on aggregated objective values for each considered static instance, i.e., the average over the 20 different dynamic instances generated from each request distribution. We generate dynamic instances as detailed in Section 3.6.1 using a sample size of $m = 100$ requests per epoch. We note that evaluation instances do not correspond to those used to train individual models and particularly point out that we, unless noted otherwise, generate training instances for the ML-CO policy using a lower sample size ($m = 50$). All experiments limit the runtime of the ML-CO policy to 90 seconds per epoch.

3.6.4. Performance Analysis

Figure 4a compares the performances of the introduced benchmarks. The figure shows the gap of the policy's objective value relative to the anticipative baseline (ant. b.). In general, we can see that benchmark policies which consider information about the uncertain appearance of future requests, i.e., rolling-horizon, monte-carlo, ML-CO, and the anticipative baseline, outperform benchmarks which make dispatching decision based on the current epoch only, i.e., lazy, random, and greedy. Specifically, these perform on average 74.05%, 36.29%, and 20.99% worse than the anticipative baseline. The *rolling-horizon* policy performs 7.12% worse than the anticipative baseline. We see that considering the future impact of our dispatching decision based on only a single scenario already improves the performance in comparison to the greedy policy significantly. This is surprising as the scenario drawn is often only a poor representation of the requests actually observed in later epochs due to the size of the sample space. However, the distance between the drawn scenario and the correct scenario might be outweighed by the benefit of combining future requests with actual requests in low-cost routes. The *monte-carlo* policy outperforms the rolling-horizon policy and only has a gap of 6.97% to the anticipative baseline. This shows that a better approximation of future uncertainties, achieved through a higher number of sampled scenarios, leads to better dispatching decisions, thus improving



(a) Performance relative to anticipative baseline solution.



(b) Variance of objective value with respect to instance seeds.

Figure 4 Performance of benchmark policies.

performance. Yet, the improvement with respect to the rolling-horizon benchmark is rather small in comparison to the improvement from the greedy policy to the rolling-horizon policy. Our *ML-CO* policy outperforms all other online policies and is only 5.15% worse than the anticipative baseline. This is surprising as the general consensus in literature on multi-stage stochastic problems indicates the contrary, i.e., that imitating an anticipative strategy does not generalize well. In our extended analysis we point to some signals in our results, which indicate that one may achieve improved performance by imitating a better policy.

Result 1. Our ML-CO policy performs best across all online benchmark policies and outperforms the monte-carlo policy by 1.57%. This indicates that learning a policy by imitating an anticipative strategy yields good performances in this problem setting.

Figure 4b shows the variance of the objective value over several instance seeds for the considered benchmark policies. Comparing Figures 4a and 4b shows a clear trend: policies with low variance outperform policies with high variance. This trend results from finding more robust solutions that generalize well over uncertain future observations. Note, that, although performing worse, the monte-carlo policy has less variance than our ML-CO policy. This relates to the sampling approach the monte-carlo policy bases on, which yields a robust solution that performs well over all sampled scenarios and therefore focuses too much on variance minimization. Our ML-CO policy on the other hand balances the trade-off between robustness and solution quality.

Result 2. The monte-carlo policy and our ML-CO policy generalize well over uncertain future observations. The learning component in the ML-CO policy balances the trade-off between robustness and solution quality, leading to a superior performance.

3.6.5. Extended analysis

This analysis assesses the impact of different training settings on the performance of our ML-CO policy. Specifically, we investigate the impact of i) the feature set, ii) the size of each training instance, iii) the training set size, iv) the imitated target strategy, and v) the type of statistical model used. We evaluate each model's performance as detailed in Section 3.6.2 and report the relative gap to the anticipative baseline.

	Feature sets		
	complete (baseline)	model-aware	model-free
Relative distance to ant. b.:	5.15%	13.83%	6.78%

Table 3 Performance of ML-CO policy using different feature sets.

Different feature sets: Table 3 compares the performance of our ML-CO policy on three feature sets (i.e., *complete*, *model-aware*, and *model-free*). Table 4 details the features each set comprises.

model-free		model-aware	
x coordinate of location	x_r	*Quantiles from distribution of travel time to all locations:*	
y coordinate of location	y_r	1% quantile	$Pr[X < x] \leq 0.01, X \sim t_{r,:}$
demand	q_r	5% quantile	$Pr[X < x] \leq 0.05, X \sim t_{r,:}$
service time	s_r	10% quantile	$Pr[X < x] \leq 0.1, X \sim t_{r,:}$
time window start	l_r	50% quantile	$Pr[X < x] \leq 0.5, X \sim t_{r,:}$
time window end	u_r	*Quantiles from distribution of slack time to all time windows:*	
time from depot to request	$t_{d,r}$	0% quantile	$Pr[X < x] \leq 0, X \sim u_: - (l_r + s_r + t_{r,:})$
relative time depot to request	$t_{d,r}/(u_r - s_r)$	1% quantile	$Pr[X < x] \leq 0.01, X \sim u_: - (l_r + s_r + t_{r,:})$
time window start / rem. time	$l_r/(T_{max} - \tau_e)$	5% quantile	$Pr[X < x] \leq 0.05, X \sim u_: - (l_r + s_r + t_{r,:})$
time window end / rem. time	$u_r/(T_{max} - \tau_e)$	10% quantile	$Pr[X < x] \leq 0.1, X \sim u_: - (l_r + s_r + t_{r,:})$
is must dispatch	$\mathbb{1}_{\tau_e + \Delta\tau + t_{d,r} > u_r}$	50% quantile	$Pr[X < x] \leq 0.5, X \sim u_: - (l_r + s_r + t_{r,:})$

Table 4 Different feature sets.

Specifically, the *model-free* feature set contains features computable from the current state x^e, while the *model-aware* feature set only contains features which include distributional information from the static instance. We further include the *complete* feature set which combines the features from the *model-aware* and *model-free* feature sets. The *complete* feature set was used in the model submitted to the challenge, i.e., our baseline. Our results show that the performance of our ML-CO policy does not rely solely on *model-aware* information not available in real-world scenarios. Specifically, considering *model-aware* features derived from the static instance decreases the gap of our ML-CO policy to the anticipative baseline by only 1.63 percentage points on average.

Result 3. Our ML-CO policy performs well without considering *model-aware* features derived from the static instance.

	Sample size of training instances				
	10	25	50	75	100
Relative distance to ant. b.:	8.71%	6.31%	5.15%	7.41%	13.08%

Table 5 Performance of ML-CO policy when trained on different sized training instances.

Different sample size of training instances: Table 5 shows the performance of the ML-CO policy relative to the performance of the anticipative baseline when training the ML-CO policy on training instances derived from different sample sizes. Note that we evaluate the trained models on instances generated with a sample size of 100 regardless of the sample size used during training. Our results, indicate that there exists a trade-off between different sample sizes. Specifically, our pipeline performs best when training on instances with a sample size of 50, reaching an average gap of 5.15% to the anticipative baseline. Increasing or decreasing the sample size reduces the performance. It is not surprising to see the performance decrease when the sample size is small: The policy overfits the training set in that case and generalizes poorly. The decrease of performance on large sample sizes is more unexpected. It might be due to the fact that we overfit the portion of the state space visited by the anticipative policy on the training instances, which may be different from those visited by our non-anticipative policy on the test instances.

Result 4. It is crucial to balance learning accuracy and model generalization when training the ML-CO policy.

	Num. of training instances							
	1	2	5	10	15	20	25	30
Relative distance to ant. b.:	10.19%	7.77%	7.26%	5.89%	5.15%	5.23%	4.50%	4.94%

Table 6 Performance of ML-CO policy when training on different numbers of training instances.

Different numbers of training instances: Table 6 shows the performance of the ML-CO policy relative to the performance of the anticipative baseline when training the ML-CO policy on different training set sizes. Recall that the policy is fitted on a set of training instances and tested on a test of different instances. And that for each instance, we solve 15 anticipative scenarios. The size of the traning set therefore increases with the number of training instances. Our results convey that the performance our ML-CO policy improves with the training set size. This is unsurprising, because first, a larger training set enables to reduce single decisions overfitting (independently on the number instances), and second a more diverse training set enables to improve cross-instances generalization. The marginal improvement decreases with the training set size. Specifically, performance saturates at a training set size of 10 instances, which shows that our ML-CO policy learns to generalize well even from small training sets. This indicates that our approach requires only a few instances to extract most of the structural information contained in the problem setting, which is a stark contrast to findings from classic supervised learning, which generally requires significantly larger training sets.

Result 5. The ML-CO policy only needs few training instances to learn a general policy.

	Imitated anticipative strategies							
	best seed	3600 sec	900 sec	300 sec	240 sec	180 sec	120 sec	60 sec
Relative distance to ant. b.:	6.18%	5.15%	4.79%	7.61%	6.01%	5.25%	5.33%	5.81%
Average objective value [×10000]:	20.49	20.56	20.70	20.78	20.80	20.87	20.97	21.21

Table 7 Performance of ML-CO policy for different target strategies.

Varying target strategies: To investigate the impact of the quality of the solutions imitated by our ML-CO policy, we vary the time limit allocated to the HGS used to derive the underlying anticipative solutions. Here the intuition is as follows: solutions derived with a low time limit should be of lower quality than solutions derived with a high time limit. As the HGS algorithm is subject to random decisions, we further include a training set derived from the best solutions found across 10 runs with a time limit of 3600 seconds each. Table 7 shows the performance of the ML-CO policy relative to the performance of the anticipative baseline when training the ML-CO policy on different target solutions with different solution qualities. Surprisingly, there is no clear trend between the solution quality of the respective training set and the performance of the ML-CO policy. This leads to the assumption that an improved quality of the anticipative target solution does not increase the performance of the trained ML-CO policy and therefore the anticipative strategy might not be the best policy to imitate.

Result 6. The anticipative strategy might not be the best policy to imitate and there might exist a target solution which yields better performances.

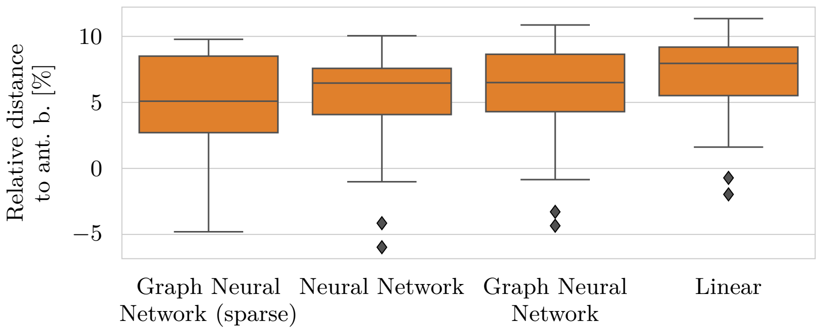


Figure 5 **Performance of ML-CO policy using different statistical models.**

Different statistical models: Figure 5 compares the performance of our ML-CO policy when using different statistical models φ_w. All statistical models rely on a feature mapping ϕ that maps a state x^e and a request i of this state to a feature vector $\phi(i, x^e)$ in $\mathbb{R}^{|\phi|}$. Our *linear model* $\varphi_w(x^e) = (w^\top \phi(i,x))_i$ manages to handle variable size inputs and outputs by applying the same linear model $\phi \mapsto w^\top \phi$ independently to each dimension i. Similarly, our *neural network* $\varphi_w(x^e) = \big(g_w(\phi(i,x))\big)_i$ applies an auxiliary neural network g_w independently for each dimension i. Finally, we propose two *graph neural networks*. Both graph neural networks consider requests as nodes and the connection between requests as edges. We include a regular and a sparsified graph neural network, the latter contains only those edges which are feasible with respect to request time windows and travel times. Both graph neural networks receive an input vector $\phi(i,x)$ for each node i, such that $\varphi_w(x^e) = \big(h_w(\phi(i,x)_{i \in I})\big)$.

The linear model performs worst with an average gap of 6.85% to the anticipative baseline while the sparsified graph neural network performs best with a 5.04% gap to the anticipative baseline. Comparing the performance of the linear model to the neural networks' performance indicates the importance of a feature generator. In fact, using a simple neural network already lowers the gap to the anticipative baseline to 5.15%. As expected, using a graph neural network that calculates structural features further improves the performance of the ML-CO policy. However, the improvement is rather small in comparison to the performance of the neural network. This suggests that most of the structural information is already included in the structured learning approach.

Result 7. Feature-generating statistical models yield the best performing ML-CO policies.

3.7. Conclusion

We presented a novel CO-enriched ML pipeline for a dynamic VRP that was introduced in the EURO Meets NeurIPS Vehicle Routing Competition. Specifically, our work contains several methodological contributions and an extensive computational study. From a methodological perspective, we extend ML-based pipelines to objective functions where the dimension of the predicted objective costs does not match the dimension of the decision variables. These objective functions amend to other problem settings such that we have made them available in the open source library `InferOpt.jl`. Moreover, we presented the first pipeline that utilizes a metaheuristic component to solve the CO-layer and showed how to carefully design a metaheuristic that finds heuristic solutions which allow to compute approximate gradients. We showed how to train the ML-layer of this pipeline in a supervised learning fashion, i.e., based on a training set derived from an anticipative strategy. We presented a comprehensive numerical study and show that our policy encoded via the CO-enriched ML pipeline outperforms greedy policies by 13.18% and even monte-carlo policies, which were granted a longer runtime, by 1.57% in terms of travel time on average. Interestingly, our results point at the fact that counterintuitive to common practice, imitating anticipative strategies can work well for high-dimensional multi-stage stochastic optimization problems, even if the anticipative strategy might not be the best strategy to imitate.

In this work, we limited the evaluation of the presented CO-enriched ML pipeline to the problem setting provided in the EURO Meets NeurIPS Vehicle Routing Competition. This problem setting was tailored to the competition and therefore had some shortcomings in comparison with general real-world practices, e.g., an unlimited vehicle fleet and long epoch durations of one hour. To which extent the performance of our proposed CO-enriched ML pipeline and CO-enriched ML pipelines in general transfers to other problem settings hence remains an open question. While some early works point into this direction (see, e.g., Jungel et al. 2023), answering this question requires future research on the added value of these pipelines in the context of more general dynamic problem settings, e.g., in the areas of mobility-on-demand, freight and food delivery, and inventory management. Furthermore, although the use of CO-enriched ML to encode policies seems very promising, stochastic optimization experience and artifacts in our results indicate that imitating the anticipative strategy might not be the best learning approach. Improving the learning algorithm may therefore be a fruitful research direction. Contrarily, if imitating a "bad" anticipative policy does work on other applications, it would be interesting to understand this phenomenon from a theoretical lens. Finally, we believe that exploring applications with shorter, possibly fully dynamic epoch durations and problem settings that allow dynamic adaption of routes after initial dispatch could reveal areas where CO-enriched ML pipelines provide added value over existing approaches, and thus lead to a more profound understanding on when CO-enriched ML pipelines in general perform well.

Acknowledgments

This work has partially been supported by the German Research Foundation (DFG) under grant no. 449261765.

We thank Wouter Kool and the organizational committee of the EURO Meets NeurIPS 2022 Vehicle Routing Competition for providing the problem setting studied in this paper.

References

Akkerman F, Luy J, van Heeswijk W, Schiffer M, 2023 *Handling large discrete action spaces via dynamic neighborhood construction* ArXiv:2305.19891.

Alinaghian M, Aghaie M, Sabbagh MS, 2019 *A mathematical model for location of temporary relief centers and dynamic routing of aerial rescue vehicles. Computers & Industrial Engineering* 131:227–241.

Alonso-Mora J, Wallar A, Rus D, 2017 *Predictive routing for autonomous mobility-on-demand systems with ride-sharing. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3583–3590, URL `http://dx.doi.org/10.1109/IROS.2017.8206203`.

Basso R, Kulcsár B, Sanchez-Diaz I, Qu X, 2022 *Dynamic stochastic electric vehicle routing with safe reinforcement learning. Transportation Research Part E: Logistics and Transportation Review* 157:102496.

Bengio Y, Lodi A, Prouvost A, 2021 *Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research* 290(2):405–421.

Bent RW, Van Hentenryck P, 2004 *Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Operations Research* 52(6):977–987.

Berthet Q, Blondel M, Teboul O, Cuturi M, Vert JP, Bach F, 2020 *Learning with differentiable pertubed optimizers. Advances in neural information processing systems* 33:9508–9519.

Blondel M, Martins AF, Niculae V, 2020 *Learning with fenchel-young losses. The Journal of Machine Learning Research* 21(1):1314–1382.

Carpentier P, Chancelier JP, Cohen G, De Lara M, 2015 *Stochastic multi-stage optimization. Probability Theory and Stochastic Modelling* 75.

Chu H, Zhang W, Bai P, Chen Y, 2021 *Data-driven optimization for last-mile delivery. Complex & Intelligent Systems* 1–14.

Dalle G, Baty L, Bouvier L, Parmentier A, 2022 *Learning with Combinatorial Optimization Layers: a Probabilistic Approach.* ArXiv:2207.13513.

Elmachtoub AN, Grigas P, 2022 *Smart "predict, then optimize". Management Science* 68(1):9–26.

Elmachtoub AN, Liang JCN, McNellis R, 2020 *Decision trees for decision-making under the predict-then-optimize framework. International Conference on Machine Learning*, 2858–2867 (PMLR).

Enders T, Harrison J, Pavone M, Schiffer M, 2023 *Hybrid Multi-agent Deep Reinforcement Learning for Autonomous Mobility on Demand Systems* .

Fikar C, 2018 *A decision support system to investigate food losses in e-grocery deliveries. Computers & Industrial Engineering* 117:282–290.

Flatberg T, Hasle G, Kloster O, Nilssen EJ, Riise A, 2007 *Dynamic And Stochastic Vehicle Routing In Practice*, 41–63 (Boston, MA: Springer US).

Gendreau M, Guertin F, Potvin JY, Taillard E, 1999 *Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science* 33(4):381–390.

Hildebrandt FD, Thomas BW, Ulmer MW, 2023 *Opportunities for reinforcement learning in stochastic dynamic vehicle routing. Computers & Operations Research* 150:106071.

Joe W, Lau HC, 2020 *Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. Proceedings of the International Conference on Automated Planning and Scheduling* 30(1):394–402.

Jungel K, Parmentier A, Schiffer M, Vidal T, 2023 *Learning-based online optimization for autonomous mobility-on-demand fleet control.* ArXiv:2302.03963.

Kool W, Bliek L, Numeroso D, Reijnen R, Afshar RR, Zhang Y, Catshoek T, Tierney K, Uchoa E, Vidal T, Gromicho J, 2022a *The EURO meets NeurIPS 2022 vehicle routing competition.*

Kool W, Juninck JO, Roos E, Cornelissen K, Agterberg P, van Hoorn J, Visser T, 2022b *Hybrid Genetic Search for the Vehicle Routing Problem with Time Windows: a High-Performance Implementation.*

Kool W, van Hoof H, Welling M, 2019 *Attention, learn to solve routing problems! International Conference on Learning Representations.*

Kotary J, Fioretto F, Van Hentenryck P, Wilder B, 2021 *End-to-End Constrained Optimization Learning: A Survey.* ArXiv:2103.16378.

Liang E, Wen K, Lam WHK, Sumalee A, Zhong R, 2022 *An integrated reinforcement learning and centralized programming approach for online taxi dispatching. IEEE Transactions on Neural Networks and Learning Systems* 33(9):4742–4756, URL `http://dx.doi.org/10.1109/TNNLS.2021.3060187`.

Liu S, He L, Max Shen ZJ, 2021 *On-time last-mile delivery: Order assignment with travel-time predictors. Management Science* 67(7):4095–4119, URL `http://dx.doi.org/10.1287/mnsc.2020.3741`.

Nagata Y, Kobayashi S, 2010 *A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. Parallel Problem Solving from Nature, PPSN XI*, 536–545 (Springer Berlin Heidelberg).

Nazari M, Oroojlooy A, Snyder L, Takac M, 2018 *Reinforcement learning for solving the vehicle routing problem.* Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds., *Advances in Neural Information Processing Systems*, volume 31 (Curran Associates, Inc.).

Nicolai B, 2016 *Amazon liefert pakete innerhalb von zwei stunden.* URL `https://www.welt.de/wirtschaft/article153690106/Amazon-liefert-Pakete-innerhalb-von-zwei-Stunden.html`.

Ojeda Rios BH, Xavier EC, Miyazawa FK, Amorim P, Curcio E, Santos MJ, 2021 *Recent dynamic vehicle routing problems: A survey. Computers & Industrial Engineering* 160:107604.

Oliver I, Smith D, Holland JR, 2013 *A study of permutation crossover operators on the traveling salesman problem. Genetic Algorithms and their Applications*, 224–230 (Psychology Press).

Parmentier A, 2021 *Learning structured approximations of operations research problems.* ArXiv:2107.04323.

Parmentier A, 2022 *Learning to approximate industrial problems by operations research classic problems. Operations Research* 70(1):606–623.

Parmentier A, T'Kindt V, 2021 *Learning to solve the single machine scheduling problem with release times and sum of completion times.* ArXiv:2101.01082.

Pflug GC, Pichler A, 2014 *Multistage stochastic optimization*, volume 1104 (Springer).

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

Placek M, 2022 *Same-day delivery market size in u.s. 2019-2024.* URL `https://www.statista.com/statistics/1068886/us-same-day-delivery-market-size/`.

Raza SM, Sajid M, Singh J, 2022 *Vehicle routing problem using reinforcement learning: Recent advancements.* Gupta D, Sambyo K, Prasad M, Agarwal S, eds., *Advanced Machine Intelligence and Signal Processing*, 269–280 (Singapore: Springer Nature Singapore).

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

Soeffker N, Ulmer MW, Mattfeld DC, 2022 *Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. European Journal of Operational Research* 298(3):801–820.

Steever Z, Karwan M, Murray C, 2019 *Dynamic courier routing for a food delivery service. Computers & Operations Research* 107:173–188.

Tang X, Qin Z, Zhang F, Wang Z, Xu Z, Ma Y, Zhu H, Ye J, 2019 *A deep value-network based approach for multi-driver order dispatching. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* URL `https://api.semanticscholar.org/CorpusID:196194675`.

Ulmer MW, Soeffker N, Mattfeld DC, 2018 *Value function approximation for dynamic multi-period vehicle routing. European Journal of Operational Research* 269(3):883–899.

van Doorn J, Lan L, Pentinga L, Wouda N, 2022 *Solving a static and dynamic vrp with time windows using hybrid genetic search and simulation. EURO Meets NeurIPS Vehicle Routing Competition* URL `https://github.com/ortec/euro-neurips-vrp-2022-quickstart/blob/main/papers/OptiML.pdf`.

Vidal T, 2022 *Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. Computers & Operations Research* 140:105643.

Vidal T, Laporte G, Matl P, 2020 *A concise guide to existing and emerging vehicle routing problem variants. European Journal of Operational Research* 286(2):401–416.

Vidal T, Maculan N, Ochi LS, Vaz Penna PH, 2016 *Large neighborhoods with implicit customer selection for vehicle routing problems with profits. Transportation Science* 50(2):720–734.

Wallar A, Van Der Zee M, Alonso-Mora J, Rus D, 2018 *Vehicle rebalancing for mobility-on-demand systems with ride-sharing. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4539–4546, URL `http://dx.doi.org/10.1109/IROS.2018.8593743`.

Zhou M, Jin J, Zhang W, Qin Z, Jiao Y, Wang C, Wu G, Yu Y, Ye J, 2019 *Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2645–2653, CIKM '19 (New York, NY, USA: Association for Computing Machinery), ISBN 9781450369763, URL `http://dx.doi.org/10.1145/3357384.3357799`.

4 Electric vehicle charge scheduling with flexible service operations

This chapter is based on an article published as:

Klein P. S., Schiffer M. (2023). Electric vehicle charge scheduling with flexible service operations. *Transportation Science* 57(6):1605-1626.

Patrick Sean Klein
TUM School of Management, Technical University of Munich, 80333 Munich, Germany, patrick.sean.klein@tum.de

Maximilian Schiffer
TUM School of Management & Munich Data Science Institute, Technical University of Munich, 80333 Munich, Germany, schiffer@tum.de

Operators who deploy large fleets of electric vehicles often face a challenging charge scheduling problem. Specifically, time-ineffective recharging operations limit the profitability of charging during service operations such that operators recharge vehicles off-duty at a central depot. Here, high investment cost and grid capacity limit available charging infrastructure such that operators need to schedule charging operations to keep the fleet operational. In this context, flexible service operations, i.e. allowing to delay or expedite vehicle departures, can potentially increase charger utilization. Beyond this, jointly scheduling charging and service operations promises operational cost savings through better utilization of time-of-use energy tariffs and carefully crafted charging schedules designed to minimize battery wear. Against this background, we study the resulting joint charging and service operations scheduling problem accounting for battery degradation, non-linear charging, and time-of-use energy tariffs. We propose an exact Branch & Price algorithm, leveraging a custom branching rule and a primal heuristic to remain efficient during the Branch & Bound phase. Moreover, we develop an exact labeling algorithm for our pricing problem, constituting a resource-constrained shortest path problem that considers variable energy prices and non-linear charging operations. We benchmark our algorithm in a comprehensive numerical study and show that it can solve problem instances of realistic size with computational times below one hour, thus enabling its application in practice. Additionally, we analyze the benefit of jointly scheduling charging and service operations. We find that our integrated approach lowers the amount of charging infrastructure required by up to 57% besides enabling operational cost savings of up to 5%.

Key words: charge scheduling; branch and price; flexible service

4.1. Introduction

Increasing societal and political environmental awareness resulting from climate change and local and global emission problems call for a paradigm change towards sustainable transportation systems. Herein, electric commercial vehicles (ECVs) are seen as a promising alternative to internal combustion engine vehicles (ICEVs), allowing up to 20% reduction in life-cycle greenhouse gas emissions when considering the current European energy mix (cf. EEA 2018). Moreover, ECVs may provide an economic advantage due to lower operational costs (Taefi 2016, Schiffer et al. 2021). Accordingly, major players in the freight and passenger transportation sectors started to electrify their fleets. Seminal examples of this development include the Deutsche Post DHL Group (DPDHL), UPS, FedEx, General Electric, Hertz, and Amazon in the freight transportation sector, as well as Uber, Lyft, and Addison Lee in the passenger transportation sector (cf. DPDHL 2017, Clark 2019, Juan et al. 2016, Rodriguez, Hildermeier, and Jahn 2020, Lyft 2021, Griffin 2021).

A central challenge in all of these applications is the efficient scheduling of charging operations, which are often conducted during off-service periods using private charging infrastructure installed at a central depot to avoid inefficient use of drivers' time. Here, grid constraints and high investment costs limit the availability of dedicated charging infrastructure, such that there are generally fewer (fast) chargers than vehicles. Accordingly, operators must synchronize the fleet's charging operations to avoid charger capacity bottlenecks. Moreover, time-of-use (TOU) energy tariffs, which charge different prices depending on the time of consumption, further complicate this scheduling problem: with on-peak prices up to three times as high as off-peak prices (OpenEI 2022), it becomes economically worthwhile to consider energy prices when planning charging operations. Generally, charger capacity and TOU pricing favor schedules where a vehicle's state of charge (SoC) peaks at certain times, e.g., when a fast charger becomes available or energy is cheap (cf. Pelletier, Jabali, and Laporte 2018). However, these schedules impose considerable stress on an ECV's battery, such that the long-term effects of battery degradation may mitigate short-term energy cost savings (see Appendix C). Operators who want to utilize this trade-off between charger utilization, off-peak energy prices, and battery degradation must consider an accurate (non-linear) charging model as simple (e.g., linear) approximations may over- or underestimate charging rates, which potentially distorts the cost savings attainable through the trade-off mentioned above (cf. Montoya et al. 2017, Pelletier, Jabali, and Laporte 2018).

In practice, operators often determine service schedules in an upstream planning problem, e.g., by solving a respective vehicle routing problem (VRP) or vehicle scheduling problem (VSP), and schedule charging operations for the resulting fixed service schedule and vehicle assignment subsequently. This hierarchical decomposition often stems from applications with complex rostering constraints, when operators value consistent service (Stavropoulou 2022), or when compatibility dependencies between service operations and vehicles or drivers exist (Batsyn et al. 2021). Examples of such applications are abundant: in city logistics, narrow or particularly congested roads may limit vehicle length, height, or weight. In law enforcement and military applications, access clearance may constrain the driver pool. Maintenance problems may place requirements on vehicle equipment or crew skill, while continuity of care may be a hard constraint in health care applications. These predetermined service schedules often have some (unavoidable) slack due to, e.g., driver service regulations and restrictive time windows (cf. Kok, Hans, and Schutten 2011), such that individual service operations are *flexible*, i.e., can be shifted in time to a limited extent without violating upstream scheduling constraints. Operators may benefit from this flexibility and delay or expedite a service operation to allow charging at a slower charger, e.g., to make a faster charger available to another vehicle of the fleet, to charge during cheap off-peak periods, or to balance charging operations across the planning horizon to avoid charging patterns with high impact on battery health.

Concluding, operators who deploy (large) fleets of ECVs face an inherently complex planning problem comprising decisions on charging and service operation schedules, which may significantly impact the viability and practicability of an ECV fleet. Here, they need to account for *i)* capacity restrictions of available charging infrastructure, *ii)* battery degradation effects, *iii)* TOU energy tariffs, *iv)* non-linear battery behavior, and *v)* flexible service operations. We study the resulting planning problem in the remainder of this paper. In the following, we first review related work in Section 4.1.1 before we state our aims and scope in Section 4.1.2 and outline the paper's structure in Section 4.1.3.

4.1.1. State-of-the-Art

We concisely review the state-of-the-art of related research areas, namely electric vehicle routing problems (EVRPs), vehicle scheduling problems (VSPs), and charge scheduling problems (CSPs). For in-depth reviews of these research fields, we refer to Schiffer et al. (2019) and Olsen (2020).

Most publications in the context of EVRPs focus on routing decisions and simplify charging-related issues such as battery degradation, variable energy prices, and non-linear battery behavior. Traditionally, most publications in the context of EVRPs have simplified charging behavior and focused on routing decisions. Specifically, charging operations were either modeled as a fixed time penalty (Conrad and Figliozzi 2011, Erdoğan and Miller-Hooks 2012) or were considered to be linear with respect to time and residual battery capacity (Schneider, Stenger, and Goeke 2014, Desaulniers et al. 2016, Schiffer and Walther 2018). More recent works extend on these simplified models and assume non-linear charging (Montoya et al. 2017, Lee 2020, Liang, Dabia, and Luo 2021, Lam, Desaulniers, and Stuckey 2022). Overall, the current trend in literature on EVRP increasingly considers charging-related concerns. This includes issues such as battery degradation (Guo et al. 2022, Zang, Wang, and Qi 2022), variable energy prices (Lin, Ghaddar, and Nathwani 2021), and charger capacity constraints (Froger et al. 2022, Bruglieri, Mancini, and Pisacane 2019, Lam, Desaulniers, and Stuckey 2022). However, these works tend to isolate individual charging-related challenges and thus do not address potential trade-offs.

VSPs, which focus on assigning a set of (fixed) trips to a fleet of vehicles, have been limited similarly. Here, most publications focused on conventional vehicles and did not consider charging operations. Electric vehicle scheduling problems (EVSPs) assumed either instantaneous (Adler and Mirchandani

	[1]	[2]	[3]	[4]	[5]	[6]	Our work
Service scheduling							✓
Vehicle assignment	✓	✓	✓		✓		
Continuous charging	✓	✓		✓	✓	✓	✓
Non-linear charging				✓			✓
Battery degradation			✓	✓			✓
Energy price	✓	✓	✓	✓			✓
Heterogeneous chargers		✓	✓	✓	✓	✓	✓
Station capacity			✓	✓		✓	✓
Exact	✓	✓		✓	✓	✓	✓
Scalable	✓		✓		✓		✓

Indices [1] to [6] signify publications as follows: [1] Sassi and Oulamara (2016), [2] Sassi and Oulamara (2014), [3] van Kooten Niekerk, van den Akker, and Hoogeveen (2017), [4] Pelletier, Jabali, and Laporte (2018), [5] Parmentier, Martinelli, and Vidal (2021), [6] Abdelwahed et al. (2020).

Table 1: Related publications.

2016, Yao et al. 2020), linear (Wen et al. 2016, Alvo, Angulo, and Klapp 2021, Parmentier, Martinelli, and Vidal 2021), or non-linear charging (van Kooten Niekerk, van den Akker, and Hoogeveen 2017).

CSPs differ from EVSPs by assuming a fixed assignment of trips to vehicles, which reduces the problem's complexity to scheduling charging operations in-between trips. In the realm of CSPs, early publications have assumed linear charging operations and did not consider station capacity constraints or heterogeneous chargers (Sassi and Oulamara 2014, 2016). More recent work on charge-scheduling problems alleviated some of these shortcomings. Specifically, Abdelwahed et al. (2020) consider station capacity and heterogeneous chargers but do not account for non-linear battery charging and degradation. They derive and compare discrete-time and discrete-event mixed-integer formulations using a commercial solver. Pelletier, Jabali, and Laporte (2018) contribute a mixed integer program (MIP) that models realistic battery behavior, accounting for non-linear battery degradation and charging. They conduct an extensive case study using a commercial solver to assess the influence of battery aging, energy price, and grid restrictions in several city logistics scenarios.

Concluding, Table 1 categorizes the most-related publications in the realm of EVSPs and CSPs. As can be seen, related publications do not consider the charging process in sufficient detail, particularly with respect to non-linear charging and, with the exception of van Kooten Niekerk, van den Akker, and Hoogeveen (2017), charger capacity constraints. The work of Pelletier, Jabali, and Laporte (2018) and Abdelwahed et al. (2020) does not consider service scheduling and relies on a standard MIP, solved with commercial solvers, such that it remains limited in its computational scalability. To the best of our knowledge, a comprehensive, integrated approach for joint charging and service operation scheduling of ECVs has not been studied so far.

4.1.2. Contribution

This paper proposes a joint charging and service operation scheduling problem that accounts for a realistic battery behavior model with *i)* limited charger capacity, *ii)* non-linear charging operations, *iii)* battery degradation, and *iv)* variable energy prices. Here, our contribution is twofold:

from a methodological perspective, we develop an efficient branch and price (B&P) algorithm that significantly outperforms commercial solvers and allows to solve problem sizes encountered in practice. This algorithm relies on a problem-specific branching rule adopted from Foster and Ryan (1976), a primal heuristic, and partial pricing to remain scaleable. Our pricing problem constitutes a so-far unconsidered extension to the fixed-route vehicle charging problem (FRVCP) (cf. Baum et al. 2019, Froger et al. 2019, Kullman et al. 2021). Specifically, we consider time-constrained charging operations. We develop a label-setting algorithm that utilizes a continuous label representation and relies on a set-based dominance rule to solve this pricing subproblem efficiently. A comprehensive numerical study shows the efficiency of our approach and asserts its scalability to large problem sizes.

From a managerial perspective, we analyze the impact of jointly scheduling charging and service operations on the amount of charging infrastructure required and the benefit of accounting for variable energy prices and battery degradation. Specifically, our computational study shows that integrated planning of charging and service operations improves the utilization of variable energy prices, lowers the cost incurred from battery degradation, and allows charging infrastructure savings. Specifically, our integrated approach lowers the amount of charging infrastructure required by up to 57% and reduces operational costs by up to 5%. We further reveal that both the degree of service schedule flexibility and the energy price distribution significantly impact these savings.

4.1.3. Outline

The remainder of this paper is as follows. Section 4.2 provides a formal definition of our problem setting and derives a set-covering-based integer program (IP) before Section 4.3 develops a column generation procedure. Section 4.4 embeds this column generation procedure into a branch and bound (B&B) algorithm, leveraging a primal heuristic and problem-specific branching- and node-selection rules. Section 4.5 details the design of our computational and managerial studies, the results of which we discuss in Section 4.6. Finally, Section 4.7 concludes this paper with a summary and an outlook on future research.

4.2. Problem definition

We consider a set of vehicles $k \in \mathcal{K}$, each required to service a set of operations $\vartheta \in \Theta_k$. These vehicles operate from a central depot, where each operation starts and ends. Servicing an operation

$\vartheta \in \Theta, \Theta = \cup_{k \in \mathcal{K}} \Theta_k$, consumes a certain amount of energy $\Delta\texttt{SoC}_\vartheta$, takes a certain amount of time $\Delta\tau_\vartheta$, and is restricted to an operation-specific time window, such that vehicle k must depart between a_ϑ and d_ϑ to serve ϑ. Two comments on this setting are in order. First, we assume a fixed assignment of vehicles to service operations, determined in an upstream planning problem. This assumption is realistic in applications where compatibility constraints between service operations and vehicles or drivers exist, or where operators value consistent service (cf. Batsyn et al. 2021, Stavropoulou 2022). We nevertheless outline how our algorithm can be extended to a problem setting that relaxes this assumption in Section 4.4.5. Second, we specify that we do not assume any ordering of service operations such that any two operations $\vartheta_i, \vartheta_j \in \Theta_k$ assigned to the same vehicle $k \in \mathcal{K}$ may be served in arbitrary order if their departure time windows allow. We note that our solution methodology can be extended straightforwardly to account for such precedence constraints.

We schedule operations on a finite time horizon, discretized with a time step width of ξ minutes, given as an ordered set $\mathcal{P}$. This discretization is conservative with respect to operation time windows as it shifts departure and arrival times to the beginning and end of the respective periods. We denote the i^{th} period as $\mathcal{P}_i$ and assign to each period $p \in \mathcal{P}$ an energy price e_p.

Vehicles can be recharged using a set of (heterogeneous) charging stations $f \in \mathcal{F}$ available at the depot, each capable of simultaneously charging up to C_f vehicles. To avoid charging more energy than necessary, we allow partial charging operations independent of the time discretization, such that charging may be started or interrupted at any time. However, our discretization remains conservative concerning charger capacity, such that vehicles charging in some period $p \in \mathcal{P}$, occupy the respective charger for the entire period. This assumption is common in CSP literature (see, e.g., Pelletier, Jabali, and Laporte 2018, Abdelwahed et al. 2020) and necessary to efficiently model station capacity constraints while accounting for variable energy prices. Specifically, alternative, i.e., time-continuous, formulations are advantageous in problem settings where only few conflicts occur or where it is relatively easy to resolve capacity conflicts, e.g., by cutting involved routes from the solution space. This is the case in some vehicle routing problems (Froger et al. 2022, Lam, Desaulniers, and Stuckey 2022), but does not apply to the CSP studied in this paper. We direct readers to Boland et al. (2019) and Boland et al. (2020) for a more comprehensive exploration of the trade-off between continuous and discrete time formulations.

Finally, in contrast to Pelletier, Jabali, and Laporte (2018) and Sassi and Oulamara (2014), we do not limit the number of uninterrupted charging operations but note that including such constraints is straightforward in our solution methodology.

We model non-linear charging behavior with charger-specific piecewise linear charging functions Φ_f : Time $\mapsto \texttt{SoC}$, which capture a vehicle's SoC evolution over time when charging with an initially empty battery. We assume concave charging functions Φ_f in line with Montoya et al. (2017) and Pelletier et al. (2017). We note that $\Phi_f^{-1}(\beta)$ is well defined on $[\texttt{SoC}_{\min}, \texttt{SoC}_{\max}]$. We let $\Phi_f^{-1}(\beta) = 0$ for $\beta < \Phi_f(\texttt{SoC}_{\min})$ and $\Phi_f(\beta) = \tau_{\max}$ for $\beta > \Phi_f(\texttt{SoC}_{\max})$, such that we can use bivariate $\Phi_f(\beta, \tau) := \Phi_f(\Phi_f^{-1}(\beta) + \tau)$ to denote the SoC after charging for time τ with an initial SoC of β.

We quantify the charging cost attributed to battery deterioration in a *(cumulative) wear density function*, denoted by $\Upsilon : \texttt{SoC} \mapsto \texttt{cost}$. This function is piecewise linear and convex on the battery's operational range, $[\texttt{SoC}_{\min}, \texttt{SoC}_{\max}]$, and maps SoC q to the total cost of charging an initially empty battery up to q. We define a bivariate $\Upsilon(q, q_t) := \Upsilon(q_t) - \Upsilon(q)$, which describes the battery degradation related cost of charging from an initial SoC q to a target SoC q_t. We refer to Appendix C for a formal definition of our charging functions Φ_f and WDF Υ. Finally, we denote the set of breakpoints of some piecewise linear function g by $\mathcal{B}(g)$ and use $\angle_g(x)$ to refer to its right derivative, i.e., the slope at x.

With this setting and notation, we state the objective of our optimization problem: Given a fleet of vehicles $k \in \mathcal{K}$, each assigned operations $\vartheta \in \Theta_k$ with energy consumption $\Delta\texttt{SoC}_\vartheta$, duration $\Delta\tau_\vartheta$, and departure time window $[a_\vartheta, d_\vartheta]$, we aim to find a cost-minimal, feasible *fleet schedule*. We consider a fleet schedule feasible if it respects charger capacity constraints in each period and each *vehicle schedule* satisfies the following constraints: i) the vehicle's SoC remains within its operational limits $[\texttt{SoC}_{\min}, \texttt{SoC}_{\max}]$ at all times, ii) the vehicle provides service to all assigned operations, and iii) the vehicle meets each operation's departure time window. Formally, we represent a vehicle schedule ω as a two-tuple with components

$\boldsymbol{A}^\omega \in \mathbb{B}^{|\mathcal{P}| \times (\mathcal{F} \cup \Theta_k)}$: a binary matrix indicating scheduled operations. Here, $\boldsymbol{A}^\omega_{i,\eta}$, $\eta \in \mathcal{F} \cup \Theta_k$ indicates that the vehicle uses period $\mathcal{P}_i$ to charge at charger $f \in \mathcal{F}$ if $\boldsymbol{A}^\omega_{i,f} = 1$, services operation $\vartheta \in \Theta_k$ if $\boldsymbol{A}^\omega_{i,\vartheta} = 1$, or remains idle if $\boldsymbol{A}^\omega_{i,\eta} = 0 \ \forall \eta \in \mathcal{F} \cup \Theta_k$. We note that feasible schedules satisfy $\sum_{\forall \eta \in \mathcal{F} \cup \Theta_k} \boldsymbol{A}^\omega_{i,\eta} \leq 1 \, \forall i \in [1, |\mathcal{P}|]$.

$\boldsymbol{B}^\omega \in \mathbb{R}^{|\mathcal{P}|}$: a vector that holds the amount of charge replenished in each period $p \in \mathcal{P}$. Here, negative values $\boldsymbol{B}^\omega_i < 0$ indicate consumption. We allocate the consumption of operations that span multiple periods to the departure period.

With this notation, we state the cost of a vehicle schedule ω as the sum of energy and battery degradation related costs incurred in each period:

$$c(\omega) := \sum_{i=1}^{|\mathcal{P}|} \max(0, e_{\mathcal{P}_i} \cdot \boldsymbol{B}^\omega_i + \Upsilon(\sum_{j=1}^{i-1} \boldsymbol{B}^\omega_j, \sum_{j=1}^{i-1} \boldsymbol{B}^\omega_j + \boldsymbol{B}^\omega_i)). \tag{1}$$

Here, energy costs result from the amount of charge replenished in each period multiplied by the respective period's energy price. Degradation costs result from the WDF. Note that periods $\mathcal{P}_i$ that consume energy, i.e., where $\boldsymbol{B}^\omega_i < 0$, do not incur cost as Υ is increasing. We model this optimization problem as a set-covering problem over the set of vehicle schedules. For this purpose, we refer to the set of feasible schedules for vehicle $k \in \mathcal{K}$ as $\mathcal{A}_k$ and let $\mathcal{A} = \cup_{k\in\mathcal{K}}\mathcal{A}_k$ denote the set of all feasible vehicle schedules. Using binary variables x^k_ω, which indicate the inclusion of a schedule in the final solution ($x^k_\omega = 1$), we propose the following IP.

$$\min \sum_{k\in\mathcal{K}} \sum_{\omega\in\mathcal{A}_k} x^k_\omega c(\omega) \tag{2a}$$

$$\sum_{k\in\mathcal{K}} \sum_{\omega\in\mathcal{A}_k} x^k_\omega \cdot \boldsymbol{A}^\omega_{p,f} \leq C_f \qquad f \in \mathcal{F}, p \in \mathcal{P} \tag{2b}$$

$$\sum_{\omega\in\mathcal{A}_k} x^k_\omega \geq 1 \qquad k \in \mathcal{K} \tag{2c}$$

$$x^k_\omega \in \{0,1\} \qquad \omega \in \mathcal{A}_k \tag{2d}$$

Objective (2a) minimizes overall scheduling costs. *Linking Constraints* (2b) enforce charger capacity limitations, while *Convexity Constraints* (2c) ensure that each vehicle is assigned a schedule. Finally, Constraints (2d) state our decision variables' domain.

4.3. Column generation

The proposed set covering formulation comprises a variable for each feasible schedule, such that solving IP 2 using standard integer programming techniques remains intractable even for small instances. As a remedy, we approach IP 2 with column generation (CG). The fundamental idea of CG is to consider only a small subset of variables (*columns*) in a restricted master problem (RMP), which we obtain by relaxing IP 2 and substituting each $\mathcal{A}_k$ with some subset $\tilde{\mathcal{A}}_k \subseteq \mathcal{A}_k$. The CG procedure then iteratively extends $\tilde{\mathcal{A}}_k$ with schedules that improve the RMP's objective value until no such schedules can be identified. For this purpose, CG solves the current RMP to obtain dual prices $\pi^{(2b)}_{p,f}$ and $\pi^{(2c)}_k$ of Constraints (2b) and (2c), respectively. With these, we state the reduced cost of a schedule $\omega \in \mathcal{A}_k$ as:

$$rc(\omega) := c(\omega) - \pi^{(2c)}_k - \sum_{p\in\mathcal{P}} \sum_{f\in\mathcal{F}} \boldsymbol{A}^\omega_{p,f} \cdot \pi^{(2b)}_{p,f}. \tag{3}$$

Then, improving columns correspond to schedules $\omega \in \mathcal{A}$ with negative reduced costs, which we generate in a so-called *pricing problem* by solving $|\mathcal{K}|$ shortest path problems with resource constraints (SPPRCs) on vehicle specific time-expanded networks G^k.

In the following, we first detail the construction of these time-expanded networks in Section 4.3.1 and give an overview of our labeling algorithm in Section 4.3.2. We then detail each central algorithmic component, namely label representation (4.3.3), label dominance (4.3.4), label propagation (4.3.5), and non-dominated charging decisions (4.3.6) in separate sections (4.3.3-4.3.6). Section 4.3.7 provides a detailed example that applies our algorithm to a simplified pricing network. Finally, we outline speedup techniques used to establish the computational efficiency of our labeling algorithm in Section 4.3.8.

4.3.1. Pricing networks

For each vehicle $k \in \mathcal{K}$, we model the pricing problem as a SPPRC defined on a time-expanded network $G^k = (\mathcal{V}^k, \mathcal{E}^k)$. Here, vertices $v \in \mathcal{V}^k$ represent the vehicle's location in time and space. More precisely, for each period $p \in \mathcal{P}$, G^k comprises a set of *station vertices* and a single *garage vertex.* Station vertices correspond to the chargers available in $p \in \mathcal{P}$, while garage vertices represent locations where vehicles can idle without occupying a charger. Additionally, we add (dummy) *source* and *sink* vertices s^- and s^+, which serve as network entry and exit points to *virtual* periods $\mathcal{P}_0$ and $\mathcal{P}_{n+1}$, respectively. These dummy periods correspond intuitively to the first period before and after the planning horizon. We use functions $p(v)$ and $f(v)$ to denote the period and charger associated with some vertex $v \in \mathcal{V}^k$.

Arcs $(i, j) \in \mathcal{E}^k$ correspond to actions that a vehicle performs in period $p(i)$ and allow a vehicle to move in time and space. Traversing an arc incurs a certain (fixed) cost $c_{i,j}$ and consumes $q_{i,j}$ units of energy. Our network comprises three types of arcs:

Charging arcs $(i, j) \in \mathcal{E}^k_{\mathcal{F}}$ represent charging at $f(i)$ and incur fixed costs according to the station's dual multiplier ($c_{i,j} := \pi^{(2b)}_{p(i),f(i)}$) without consuming energy ($q_{i,j} := 0$). Charging with arrival SoC q replenishes Δq SoC at a price of $c(q, \Delta q) := e_{p(i)} \cdot \Delta q + \Upsilon(q, q + \Delta q)$, such that traversing a charging arc incurs a total cost of $c_{i,j} + c(q, \Delta q)$. Here, Δq is variable and bounded implicitly by the period length ξ, such that $0 < \Delta q \leq \Phi_{f(i)}(q, \xi)$. Charging arcs connect station vertices to all vertices of the following period.

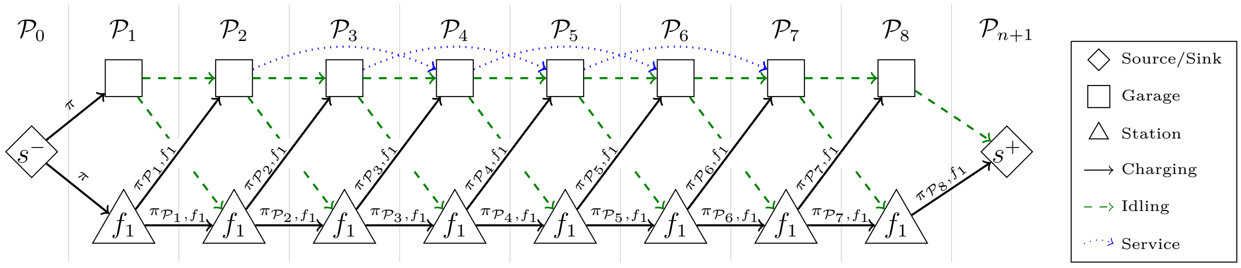
Idle arcs $(i, j) \in \mathcal{E}^k_I$ model a vehicle idling at the depot. Idling is possible at zero cost ($c_{i,j} := 0$) and does not consume energy ($q_{i,j} := 0$). Analogous to charging arcs, idling arcs connect *garage vertices* to all vertices of the following period.

Service arcs $(i, j) \in \mathcal{E}^k_{\Theta_k}$ model a vehicle's departure in period $p(i)$ to service an operation $\vartheta \in \Theta_k$. This consumes energy according to the operation's consumption ($q_{i,j} := \Delta\texttt{SoC}_\vartheta$) but incurs no additional costs ($c_{i,j} := 0$). For each $\vartheta \in \Theta_k$, service arcs connect all garage vertices that lie within the departure time window of ϑ to garage vertices in the respective arrival period. We denote the operation serviced by arc (i, j) with $\vartheta_{i,j} := \{\vartheta\}$, and let $\vartheta_{i,j} := \emptyset$ for non-service arcs.

Finally, we add dummy arcs (i, j) connecting the source vertex with all vertices of the first period at a cost according to the coverage dual ($c_{i,j} := \pi^{(2c)}_k$). Figure 1 shows an example of a time-expanded network with a single charger (f_1) and a single service operation.

4.3.2. Labeling algorithm

By construction, each source-sink path ρ in the pricing network G^k corresponds to a vehicle schedule ω. Moreover, the cost of ρ in G^k matches the reduced cost of schedule ω. Consequently, by defining resource constraints that establish the feasibility of the corresponding schedule ω (cf. Section 4.2),



Note. This example time-expanded network models the scheduling decisions of some vehicle $k \in \mathcal{K}$ that is assigned a single service operation with a departure time window of $[\mathcal{P}_2, \mathcal{P}_5]$ and duration $\Delta\tau = 2$. Solid triangles and black squares represent station and garage vertices, respectively. Diamonds illustrate source and sink vertices. Solid black, dashed green, and dotted blue arcs represent charging, idle, and service arcs. We denote $\pi^{(2b)}_{p,f}$ and $\pi^{(2c)}_{k}$ as $\pi_{p,f}$ and π, respectively.

Figure 1: An example time-expanded network.

we can express the pricing subproblem of vehicle $k \in \mathcal{K}$ as a SPPRC on G^k. Specifically, we treat SoC and serviced operations as constrained resources, i.e., restrict the SoC at non-source vertices to $[\mathtt{SoC}_{\min}, \mathtt{SoC}_{\max}]$ (*energy feasibility*), and require that all operations have been serviced when reaching the sink (*service feasibility*).

We obtain a solution to these SPPRCs using a problem-specific label-setting algorithm. The design of this algorithm remains challenging as continuous (non-linear) charging raises a trade-off between cost and SoC at charging stations. Specifically, charging any $0 < \Delta q \leq \Phi_f(q, \xi)$ with arrival SoC q results in a potentially optimal pair of cost and SoC, such that visits to charging stations may generate an unbounded number of labels. While this is so far unconsidered in previous work on EVSP and CSP, we note that recent work on vehicle routing (Baum et al. 2019, Froger et al. 2019) faced a similar challenge: here, charging operations raise a trade-off between arrival time and SoC. To resolve this issue, these works proposed a function-based label representation, which allows capturing all time-SoC trade-offs at the last visited station in a single label. This effectively delays the charging decision at the respective station until a finite subset of potentially optimal charging decisions can be identified. Although related, the methodology developed in these works is insufficient for our problem setting. The reason for this relates to variable energy prices in combination with implicit bounds on the SoC rechargeable at station vertices imposed by the time discretization. This violates a central assumption of the methodology developed in Baum et al. (2019) and Froger et al. (2019), such that their algorithms fail to find an optimal solution if energy prices vary across periods. We give proof to this issue in Appendix A and note that these additional challenges are not unique to charge scheduling but may occur in other problem settings. For example, in time-dependent routing, travel times vary over time analogously to how energy prices vary in our CSP and can hence raise a similar issue. Accordingly, we contribute to the state of the art by presenting a new label-setting algorithm that accounts for such bounded charging operations. Moreover, we present speed-up and preprocessing techniques specifically tailored to the challenges of this new label-setting algorithm.

The following outlines our algorithm and introduces core algorithmic components before we detail each component separately in Sections 4.3.3-4.3.8. We refer to Appendix D for an in-depth technical description of our algorithm.

Our algorithm iteratively explores pricing networks G^k, starting with an empty path at the source vertex s^-. We extract the, w.r.t. cost, currently cheapest path in each iteration and extend it to all neighboring vertices, thus creating new paths. The algorithm terminates when extracting a path at the sink or when no *unsettled* paths remain. We capture path resources (operations serviced, SoC, and cost) in labels $\ell \in \mathcal{L}_v$ to ensure that paths $\rho \coloneqq (s^-, \ldots, v)$ are feasible. As in Baum et al. (2019), and Dabia et al. (2013), we use a function-based label representation for this purpose. Specifically, our labels encode the parameters of a function that maps the SoC on arrival at the last node of the corresponding path to the total path cost. In contrast to Baum et al. (2019), our labels are not limited to capturing only the last visited charging station but instead capture charging decisions at several previously visited charging stations instead (see Section 4.3.3).

When extending paths along arc (i, j), we thus need to create new labels at j to capture the extended path's state. We derive these in so-called *propagation functions*, which generate a set of new labels at vertex j according to arc resources and charging decisions based on the label of the current path $\rho \coloneqq (s^-, \ldots, i)$ and the newly extended arc (i, j) (Section 4.3.5). Here, our algorithm considers already generated labels at vertex j to avoid exploring *dominated* paths. Specifically, it discards labels that do not improve on the set of labels already developed at j using a set-based dominance criterion (Section 4.3.4).

4.3.3. Label representation

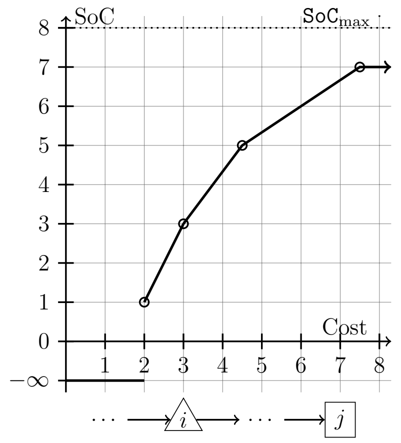
Our labels $\ell \in \mathcal{L}_j$ store path cost and SoC in *cost profiles* $\psi_\ell(c)$, which map the total cost c of the labeled path to the resulting (arrival) SoC at vertex j (cf. Figure 2). Formally, we represent labels as tuples with components:

$\psi_\ell(c) : \mathbb{R} \mapsto [\mathit{SoC}_{\min}, \mathit{SoC}_{\max}] \cup \{-\infty\}$ denoting the vehicle's SoC at a total cost of c, and

$O_\ell \in \mathbb{P}(\Theta_k)$ denoting the operations serviced so far.

We call a label energy feasible if there exists some $c \in \mathbb{R}$ such that $\psi_\ell(c) \neq -\infty$ and denote the first and last breakpoints of ψ_ℓ with $(c_{\min}(\psi_\ell), q_{\min}(\psi_\ell))$ and $(c_{\max}(\psi_\ell), q_{\max}(\psi_\ell))$, respectively. We further note that $\psi_\ell(c)$ is not well defined on $\mathbb{R}$ and hence define $\psi_\ell^{-1}(q)$, preserving piecewise-linearity, as follows:

$$\psi_\ell^{-1}(q) \coloneqq \begin{cases} -\infty & \text{if } q \leq q_{\min}(\psi_\ell) \\ c_{\max}(\psi_\ell) & \text{if } q \geq q_{\max}(\psi_\ell) \\ \arg\min\limits_{c \in \mathbb{R}} \{c \mid \psi_\ell(c) = q\} & \text{otherwise.} \end{cases} \tag{4}$$



Note. In this example, the lowest cost of the corresponding path is $c = 2$, at which we reach v with a SoC of 1. Spending less than $c = 2$ is infeasible, which we indicate with a SoC of $-\infty$. We reach a SoC of 5 at a cost of 4.5, i.e., when spending $\Delta c := 4.5 - 2 = 2.5$ to charge at i. The maximum reachable SoC is 7, which corresponds to spending $\Delta c := 7.5 - 2 = 5.5$ at j. Any $c > 7.5$ does not increase the SoC further. This upper bound on the SoC can be caused by either the battery capacity or the period length, which limits the time spent charging at the tracked station. The example illustrates the latter case.

Figure 2: Example of a cost profile tracking the cost-SoC trade-off of path $\rho = (s^-, \ldots, i, \ldots, j)$.

Note that this extended label representation remains efficient as ψ_ℓ is piecewise linear and can thus be represented as a finite sequence of *breakpoints*. Lastly, we define the label corresponding to the empty path used to initialize the algorithm at the source as $\ell_{s^-} := (\psi_{\ell_{s^-}}(c), \emptyset)$, with $\psi_{\ell_{s^-}}(c)$ as follows:

$$\psi_{\ell_{s^-}}(c) := \begin{cases} 0 & \text{if } c \geq 0 \\ -\infty & \text{otherwise.} \end{cases} \tag{5}$$

Intuitively, the source represents a (dummy) station with a charging rate 0.

4.3.4. Dominance

We use two dominance rules to discard sub-optimal labels early and thus reduce the number of explored paths. The first rule extends standard pairwise Pareto-dominance of scalar-valued labels to our function-based representation. The second rule is unique to function-based label representations and defines a *set-based* dominance relationship to capture cases where the union of a set of labels dominates a single label. We note that a similar logic has been applied successfully to time-dependent VRPs and VRPs with weight-related costs (cf. Dabia et al. 2013, Luo et al. 2017).

Definition 1. *Pairwise dominance.*

Let there be labels $\ell_a, \ell_b \in \mathcal{L}_v$ for some $v \in \mathcal{V}^k$. Then label $\ell_a \in \mathcal{L}$ dominates label ℓ_b, denoted $\ell_a \succeq \ell_b$, if and only if

$$\psi_{\ell_a}(c) \geq \psi_{\ell_b}(c) \, \forall c \in \mathbb{R} \text{ and } O_{\ell_a} \supseteq O_{\ell_b}.$$

Definition 1 states that a label ℓ_a dominates another label ℓ_b if ℓ_a achieves i) a higher or equally high SoC at any given cost c and ii) schedules a superset of operations. Accordingly, we do not define a dominance relationship between labels of paths ending at different vertices. Note that as ψ_ℓ are piecewise linear, it suffices to check $\psi_{\ell_a}(c) \geq \psi_{\ell_b}(c)$ at breakpoints $c \in \mathcal{B}(\psi_{\ell_a}) \cup \mathcal{B}(\psi_{\ell_b})$ such that the dominance check's complexity remains linear. Unfortunately pairwise dominance remains relatively weak in cases where the images of ψ_{ℓ_a} and ψ_{ℓ_b} are not equal. To mitigate this issue, we extend pairwise dominance to sets of labels as follows.

Definition 2. *Set-based dominance.*
A set of labels $\mathcal{L}' \subseteq \mathcal{L}_v$ at some vertex v dominates label $\ell \in \mathcal{L}_v \setminus \mathcal{L}'$, denoted $\mathcal{L}' \succeq \ell$, if and only if

$$\max_{\ell' \in \mathcal{L}'}\{\psi_{\ell'}(c)\} \geq \psi_\ell(c)\ \forall c \in \mathbb{R} \text{ and } \bigcap_{\ell' \in \mathcal{L}'} O_{\ell'} \supseteq O_\ell.$$

Essentially, set-based dominance occurs if, for any given cost, at least one label in $\mathcal{L}'$ achieves a higher SoC than label ℓ, and each $\ell' \in \mathcal{L}'$ schedules a superset of operations.

4.3.5. Label propagation

We distinguish propagating a label $\ell \in \mathcal{L}_i$ along a service or idling arc from propagating along a charging arc as the latter constitutes a station visit and thus allows to replenish additional energy. We detail our propagation methodology for both arc types separately in Sections 4.3.1 and 4.3.2.

4.3.1. Service and idling arcs Propagating along service and idling arcs, denoted as $\ell' := \ell \underset{(i,j)}{\leftarrow} /$, generates a new label at the target vertex j. This requires updating ψ_ℓ according to arc consumption $q_{i,j}$ and cost $c_{i,j}$, which we obtain from shifting ψ_ℓ on the cost and SoC axes, respectively: $\psi_{\ell'}(c) := \psi_\ell(c - c_{i,j}) - q_{i,j}$. Additional modifications are necessary when some charging decisions become infeasible after propagating ℓ along (i,j). Specifically, we need to maintain $\psi_{\ell'}(c) \in [\mathtt{SoC}_{\min}, \mathtt{SoC}_{\max}] \cup \{-\infty\}$ for all $c \in \mathbb{R}$ such that we derive $\psi_{\ell'}$ as follows:

$$\psi_{\ell'}(c) := \begin{cases} -\infty, & \text{if } \psi_\ell(c - c_{i,j}) - q_{i,j} < \mathtt{SoC}_{\min} \\ \mathtt{SoC}_{\max} & \text{if } \psi_\ell(c - c_{i,j}) - q_{i,j} \geq \mathtt{SoC}_{\max} \\ \psi_\ell(c - c_{i,j}) - q_{i,j} & \text{otherwise.} \end{cases} \tag{6}$$

Recall that $\vartheta_{i,j} = \{\vartheta\}$ if (i,j) serves operation ϑ and $\vartheta_{i,j} = \emptyset$ otherwise, such that $\ell' := (\psi_{\ell'}, O_\ell \cup \vartheta_{i,j})$ yields the propagated label.

4.3.2. Charging arcs Charging arc (i,j) represents a charging opportunity at station $f(i)$ where it is possible to charge for some duration $0 < \tau \leq \xi$ at a cost resulting from the energy price $e_{p(i)}$, battery degradation Υ, and fixed cost $c_{i,j}$. For the sake of conciseness, we will interchangeably refer to a charging decision at i using the charging time τ and the total cost according to the corresponding cost profile.

Propagating label $\ell \in \mathcal{L}_i$ along (i,j) realizes this charging opportunity such that the newly created label ℓ' should capture the arrival SoC at vertex j. Recall that ℓ tracks charging decisions at the last visited station v such that we are unaware of the actual arrival SoC at i. The resulting label would thus be ill-defined: the arrival SoC q at vertex j with cost c depends on how we distribute our charging budget between stations v and i.

Resolving this ambiguity requires a decision at either station. Specifically, we either need to fix a decision at v and thus the arrival SoC at i, or commit to a decision at i and continue to track decisions at v. Intuitively, the first decision *replaces* the currently tracked station v with the new charging opportunity at i such that we will refer to this operation as *station replacement*, while the latter decision entails charging *intermediately* at i and will be referred to as such. Our propagation function covers both cases, such that charging arcs generate several labels. In what follows, we describe our methodology for both cases separately.

Case *i)*: *station replacements*. We denote this operation with $\ell' = \ell \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c'$. Here, $c' \in [c_{\min}(\psi_\ell), c_{\max}(\psi_\ell)]$ corresponds to a decision on the sunk cost up to vertex i and thus the arrival SoC $q' = \psi_\ell(c')$ at i. Recall that the generated label ℓ' should give the arrival SoC at j depending on the charging budget at i, i.e., $c - c'$. We compute the according cost profile $\psi_{\ell'}$ based on the charging cost at vertex i, given as:

$$\Delta c_{p(i),f(i)}\langle q'\rangle(\Delta q) \coloneqq e_{p(i)} \cdot \Delta q + \Upsilon(q', q' + \Delta q). \tag{7}$$

Here, q' and Δq correspond to the arrival SoC and the SoC recharged at i, respectively. We note that $\Delta c_{p(i),f(i)}\langle q'\rangle(\Delta q)$ is well defined for $0 \le q' \le \texttt{SoC}_{\max}$ and $0 \le \Delta q \le \texttt{SoC}_{\max} - q'$, such that we can formally state $\psi_{\ell'}$ using the inverse of Equation (7):

$$\psi_{\ell'}(c) \coloneqq \begin{cases} -\infty & \text{if } c < c' + c_{i,j} + \Delta c_{p(i),f(i)}\langle q'\rangle(0) \\ \Phi_{f(i)}(q', \xi) & \text{if } c \ge c' + c_{i,j} + \Delta c_{p(i),f(i)}\langle q'\rangle(\Phi_{f(i)}(q',\xi) - q') \\ q' + \Delta c^{-1}_{p(i),f(i)}\langle q'\rangle(c - c_{i,j} - c') & \text{otherwise.} \end{cases} \tag{8}$$

Here, we utilize precomputed *station cost profiles* to compute Equation (8) efficiently. These give the arrival SoC at vertex j when spending c on charging at i with an empty battery:

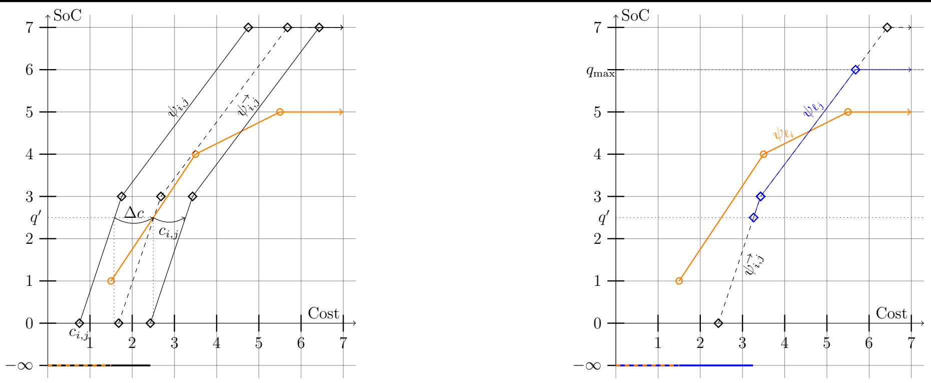
$$\psi_{i,j}(c) \coloneqq \begin{cases} -\infty & \text{if } c < c_{i,j} \\ \texttt{SoC}_{\max} & \text{if } c \ge c_{i,j} + \Delta c_{p(i),f(i)}\langle \texttt{SoC}_{\min}\rangle(\texttt{SoC}_{\max} - \texttt{SoC}_{\min}) \\ \Delta c^{-1}_{p(i),f(i)}\langle \texttt{SoC}_{\min}\rangle(c - c_{i,j}) & \text{otherwise.} \end{cases} \tag{9}$$

Station cost profiles allow to compute the replacement profile $\psi_{\ell'}$ in two steps: First, we shift $\psi_{i,j}$ by $c' + c_{i,j} - \psi^{-1}_{i,j}(q')$ on the cost axis to obtain $\psi^{\rightarrow}_{i,j}$. Second, we establish upper and lower bounds on the SoC and cut-off any unreachable SoC levels. Finally, we get

$$\psi_{\ell'}(c) \coloneqq \begin{cases} -\infty, & \text{if } c < (\psi^{\rightarrow}_{i,j})^{-1}(q') \\ \texttt{SoC}_{\max} & \text{if } c \ge (\psi^{\rightarrow}_{i,j})^{-1}(\Phi_f(q', \xi)) \\ \psi^{\rightarrow}_{i,j}(c) & \text{otherwise.} \end{cases} \tag{10}$$

See Figure 3 for an illustration of this procedure.

Case *ii)*: *intermediate charging*. Deciding to commit to charging for a fixed amount of time $0 < \tau \le \xi$ at vertex i while continuing to track decisions at previous station v, denoted $\ell' \coloneqq \ell \underset{(i,j)}{\leftarrow} \tau$, transforms the



(a) Shifting $\psi_{i,j}$ according to q' and fixed cost.

(b) Cutting off any unreachable SoC.

Note. The illustrated station replacement operation charges up to a cost of $c' = 2.5$, i.e., spends $c' - c_{\min}(\psi_\ell) = 1.0$ on charging at the previous station. The orange (circles) and blue (diamonds) cost profiles correspond to the cost profile at the origin and target vertices (i, j), respectively. The precomputed (shifted) station cost profile $\psi_{i,j}$ ($\psi_{i,j}^{\rightarrow}$) is illustrated in black (diamonds). q' denotes the arrival SoC at station i. $\Delta c = c' - \psi_{i,j}^{-1}(q')$ offsets the station cost profile by the path's fixed cost. $q_{\max} = \Phi_f(q', \xi)$ corresponds to the maximum reachable SoC at station i.

Figure 3: Geometric interpretation of station replacement.

cost profile non-linearly: decisions on charging at v influence the arrival SoC at i and thus (possibly) the total energy recharged within timespan τ. This, in turn, impacts the cost incurred from energy price and battery degradation. Formally, we can state the cost of charging at i such that we arrive with SoC q at j using Equation (11):

$$\psi_{\ell'}^{-1}(q) := \psi_\ell^{-1}\big(\overbrace{\Phi_{f(i)}(\Phi_{f(i)}^{-1}(q) - \tau)}^{\text{arrival SoC at } i}\big) + \underbrace{\overbrace{c_{i,j} + \psi_{i,j}^{-1}(q)}^{\text{sunk cost after } i} - \overbrace{\psi_{i,j}^{-1}(\Phi_{f(i)}(\Phi_{f(i)}^{-1}(q) - \tau))}^{\text{sunk cost before } i}}_{\text{total cost of charging at } i}. \tag{11}$$

We note that the breakpoints of $\psi_{\ell'}$ correspond to the breakpoints of $\psi_{\ell'}^{-1}$ with reversed domain and co-domain such that we can derive an explicit representation of $\psi_{\ell'}$ by evaluating $\psi_{\ell'}^{-1}(q)$ at each breakpoint q of $\psi_{\ell'}^{-1}$, computing segment slopes accordingly. We compute these as stated in Equation 12, denoting the slope of some function f at x by $\angle_f(x)$:

$$\begin{aligned} \angle_{\psi_{\ell'}^{-1}}(q) := {} & \overbrace{\angle_{\psi_{i,j}^{-1}}(q) + c_{i,j}}^{\text{cost of charging at } i} + \underbrace{\angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q) - \tau) \cdot \angle_{\Phi_{f(i)}^{-1}}(q)}_{\text{flow of charge from } i \text{ to } v \text{ (Term 12.1)}} \\ & \cdot \big[\underbrace{\angle_{\psi_\ell^{-1}}(\Phi_{f(i)}(\Phi_{f(i)}^{-1}(q) - \tau))}_{\text{additional charging cost at } v} - \underbrace{\angle_{\psi_{i,j}^{-1}}(\Phi_{f(i)}(\Phi_{f(i)}^{-1}(q) - \tau))}_{\text{cost saving by charging less at } i}\big]. \end{aligned} \tag{12}$$

Every value q at which Equation (12) changes is a breakpoint of $\psi_{\ell'}^{-1}$. Hence, we can compute the breakpoints of $\psi_{\ell'}^{-1}$ based on the union of the breakpoints of functions ψ_ℓ and $\psi_{i,j}$. We construct $\psi_{\ell'}$

from all breakpoints that lie within the SoC interval

$$[\Phi_{f(i)}(q_{\min}(\psi_\ell),\tau), \min\{\Phi_{f(i)}(q_{\max}(\psi_\ell),\tau), \sum_{\vartheta\in\Theta_k\setminus O_\ell} \Delta\mathtt{SoC}_\vartheta, \mathtt{SoC}_{\max}\}]$$

and set $q_{\min}(\psi_\ell), c_{\min}(\psi_\ell), q_{\max}(\psi_\ell)$, and $c_{\max}(\psi_\ell)$ accordingly. Note that the derived cost profile $\psi_{\ell'}$ is not necessarily concave or even increasing. However, as we argue in the following, decreasing segments of such cost profiles are always dominated such that they can be discarded.

Proposition 1. *The slope of a cost profile $\psi_{\ell'}$ obtained from $\ell' := \ell \underset{(i,j)}{\leftarrow} \tau$ can only be negative at $c \in [c_{\min}(\psi_{\ell'}), c_{\max}(\psi_{\ell'})]$ if for $q' := \Phi_{f(i)}(\Phi^{-1}_{f(i)}(\psi^{-1}_{\ell'}(c)) - \tau)$, $\angle_{\psi^{-1}_{\ell}}(q') < \angle_{\psi^{-1}_{i,j}}(q')$ holds.*

Proof: see Appendix B. □

Intuitively, Proposition 4 concerns cases where charging at v is more expensive than charging at i, such that it pays off to *shift* energy recharged at vertex i to the previously tracked vertex v, i.e., use the charging opportunity captured by ℓ.

Proposition 2. *For any c on a decreasing segment $[c', c' + \epsilon]$ with $\epsilon > 0$ of some cost profile $\psi_{\ell'}$ obtained from $\ell' := \ell \underset{(i,j)}{\leftarrow} \tau$, there exists some $\ell'' \in \mathcal{L}_j \setminus \{\ell'\}$ such that $\psi_{\ell''}(c) \geq \psi_{\ell'}(c)$.*

Proof: see Appendix B. □

With Proposition 5 in mind, we ensure that cost profiles are non-decreasing by replacing non-concave cost profiles with their upper concave envelope. We argue that this preserves the correctness of our algorithm. Specifically, Proposition 5 implies that for any c, there exists some label $\ell'' \in \mathcal{L}_j$ that reaches a higher SoC at the same cost c. Hence, modifying ℓ' accordingly does not lead to the domination of any optimal labels.

4.3.6. Non-dominated charging decisions

Propagating a label ℓ along a charging arc (i,j) creates new labels according to our propagation functions $\ell' := \ell \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c'$ (station replacement) and $\ell' := \ell \underset{(i,j)}{\leftarrow} \tau$ (intermediate charge). Station replacements require a decision on the cost c', i.e., the charging budget at the tracked station, and thus the respective arrival SoC q' at i. When charging intermediately, we need to decide on the time τ spent charging at i. In both cases, time-continuous charging operations allow any decision that respects time and SoC bounds, such that the set of choices for c' and τ is unbounded in the general case. In what follows, we argue that in our problem setting only a finite set of values for c' and τ generates non-dominated labels, such that it suffices to consider these during propagation. Specifically, we show that labels $\mathcal{L}' := \{\ell_j, \ell_i \underset{(i,j)}{\leftarrow} \xi\} \cup \{\ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c \mid \forall c \in \mathcal{B}(\psi_{\ell_i})\}$, with $\ell_j := \ell \underset{(i,j)}{\leftarrow} /$, at j capture all possibly optimal charging decisions.

Theorem 1. *For any charging arc (i,j) and label $\ell_i \in \mathcal{L}_i$, the set of labels*

$$\mathcal{L}' := \{\ell_j, \ell_i \underset{(i,j)}{\leftarrow} \xi\} \cup \{\ell_i \overset{f(i)}{\underset{(i,j)}{\leftarrow}} c \mid \forall c \in \mathcal{B}(\psi_{\ell_i})\},$$

with $\ell_j := \ell_i \underset{(i,j)}{\leftarrow} /$ dominates any label

$$\ell' \in \{\ell_i \overset{f(i)}{\underset{(i,j)}{\leftarrow}} c \mid c \in [c_{\min}(\psi_{\ell_i}), c_{\max}(\psi_{\ell_i})]\} \cup \{\ell_i \underset{(i,j)}{\leftarrow} \tau \mid 0 \le \tau \le \xi\} \setminus \mathcal{L}'.$$

To prove Theorem 2, we assume the contrary, i.e., that there exists some ℓ' not dominated by $\mathcal{L}'$. We then show that for each $c \in \mathbb{R}$ there exists some $\ell \in \mathcal{L}'$ with $\psi_\ell(c) \ge \psi_{\ell'}(c)$, hence contradicting the assumption. Our proof bases on the trade-off between charging at the station captured by ℓ versus charging at i: for any Δc spent to charge using the charging opportunity captured by ℓ, it will either be cheaper to continue charging at the tracked charger, or to utilize the new charging opportunity for this purpose. In the first case, the label spending $\Delta c + \epsilon$ for some $\epsilon > 0$ will dominate ℓ, while the label spending $\Delta c - \epsilon$ will prevail in the second case. We can then show our claim using the piecewise-linearity of our cost profiles. We refer to Appendix B for a full proof.

4.3.7. Example

The following example illustrates the behavior of our labeling algorithm on the example network illustrated in Figure 4. Figure 5 depicts the labels created at each step of the algorithm. We provide the data basis for Figures 4 and 5 in Appendix G.

Figure 5a: The algorithm initially extracts the empty path labeled $\ell_{s^-} \in \mathcal{L}_{s^-}$ at the source s^-. It then extends this path along the adjacent source arc (s^-, v_f) at a cost of $c_{s^-,v_f} = 2$. Accordingly, we create a new label $\ell_{v_f} := \ell_{s^-} \underset{(s^-,v_f)}{\leftarrow} /$ with a minimum SoC of one at a total cost of two at vertex v_f. Geometrically, we obtain the new cost profile ψ_{ℓ_f} from *shifting* $\psi_{\ell_{s^-}}$ on the SoC axis.

Figure 5b: Our algorithm then continues and extracts ℓ_{v_f} at v_f in the next iteration, propagating it along charging arc (v_f, v_2). Here, we can charge up to a maximum SoC of $\Phi_{v_f}(q=0, \xi=4) = 3$ at a cost according to energy price $e_{v_f} = 2.5$, battery degradation Υ, and fixed cost $c_{(v_f,v_2)} = 0$. The algorithm creates a new label $\ell_{v_2} := \ell_f \overset{f}{\underset{(v_f,v_2)}{\leftarrow}} 2$ that captures this trade-off as detailed in Sections 4.3.5 and 4.3.6. Note that it suffices to create a single label in this special case, as v_f is the first charging opportunity considered such that the arrival SoC at v_f is unique. We obtain the updated cost profile $\psi_{\ell_{v_2}}$ from shifting $\psi_{(v_f,v_2)}$ on the cost axis such that it intersects $\psi_{\ell_{v_f}}$ at $q_{\min}(\psi_{\ell_{v_f}}) = 0$, and subsequently cutting off any SoC outside $[0, \Phi_{f(i)}(0,\xi)]$.

Figure 5c: The next iteration extracts ℓ_{v_2} at vertex v_2 and propagates it along service arc (v_2, v_3) such that $\ell_{v_3} := \ell_{v_2} \underset{(v_2,v_3)}{\leftarrow} /$. Providing service to the respective operation consumes 1.5 units of energy,

which renders some charging decisions captured in $\psi_{\ell_{v_2}}$ infeasible: spending less than $c = 6.5$ does not replenish sufficient energy to provide service. Accordingly, we obtain the new cost profile $\psi_{\ell_{v_3}}$ from shifting $\psi_{\ell_{v_2}}$ by $q = 1.5$ on the SoC axis and subsequently cutting of any value below $\texttt{SoC}_{\min} = 0$. Essentially, we (implicitly) commit to charging at least 1.5 units of energy at v_f to maintain feasibility.

Figure 5d: The figure shows two iterations of our labeling algorithm. First, the algorithm propagates label ℓ_{v_3} along idle arc (v_3, v_g). This neither consumes SoC nor incurs cost, such that $\psi_{\ell_{v_g}} = \psi_{\ell_{v_3}}$ with $\ell_{v_g} := \ell_{v_3} \underset{(v_3,v_g)}{\leftarrow} /$. The next iteration propagates label ℓ_{v_g} along charging arc (v_g, v_5), which again provides a charging opportunity. Here, in contrast to the previous charging opportunity (v_f, v_2), the arrival SoC q at v_g is not unique. Our algorithm creates new labels according to Section 4.3.6 at v_5. Specifically, we create labels $\ell^1_{v_5} := \ell_{v_g} \overset{g}{\underset{(v_g,v_5)}{\leftarrow}} 6.5$, $\ell^2_{v_5} := \ell_{v_g} \underset{(v_g,v_5)}{\leftarrow} \xi\ (4)$, $\ell^3_{v_5} := \ell_{v_g} \overset{g}{\underset{(v_g,v_5)}{\leftarrow}} 8$, and $\ell^4_{v_5} := \ell_{v_g} \overset{g}{\underset{(v_g,v_5)}{\leftarrow}} 11.7$.

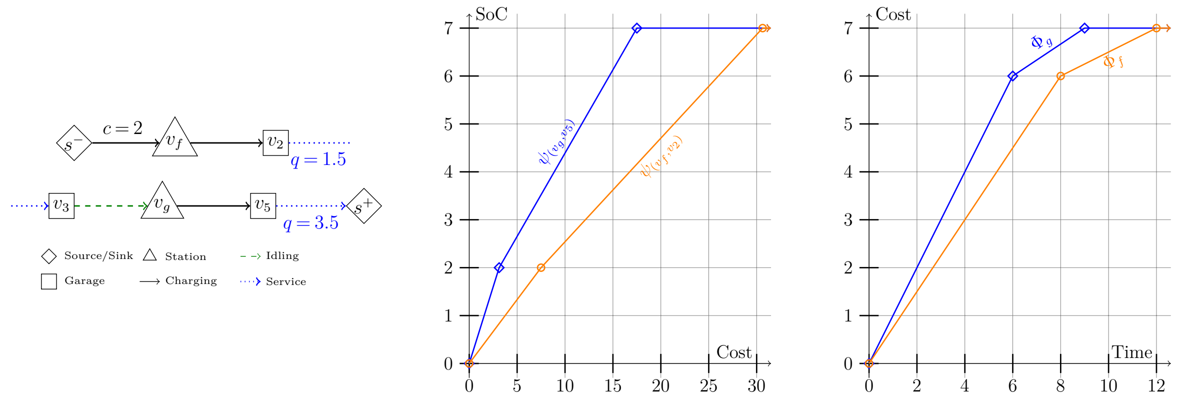
We observe two effects: first, some of the created labels are dominated according to Definitions 1 & 2. Second, the cost profiles span segments of different lengths on the cost and SoC axes. Here, non-linear charging functions Φ and the charging limit imposed by period length ξ cause mismatches on the SoC domain, while non-linear battery degradation is responsible for mismatching cost domains.

Figure 5e: The algorithm proceeds to extract the labels in $\mathcal{L}_{v_5}$ in order of least cost. Hence, it first propagates label $\ell^1_{v_5}$ along service arc (v_5, s^+), which creates a new label $\ell^1_{s^+} := \ell^1_{v_5} \underset{(v_5,s^+)}{\leftarrow} /$ at s^+ accordingly. The algorithm then extracts label $\ell^1_{s^+}$ at the sink as it has lower costs than label $\ell^2_{v_5}$. This terminates the algorithm and yields a path that reaches the sink. The cost profile of $\ell^2_{v_5}$ then gives the total cost of this shortest path as $c_{\min}(\psi_{\ell^2_{v_5}}) = 11.875$ with final SoC $q_{\min}(\psi_{\ell^2_{v_5}}) = 0$.

Figure 5f: Assuming a consumption of $q_{v_5,s^+} := 4.25$ for service arc (v_5, s^+), we observe a slightly different behavior. In this case, it does not suffice to charge as little as possible at v_f such that label $\ell^1_{s^+}$ is infeasible. Instead, label $\ell^2_{s^+}$ now gives the optimal path. This constitutes an example where considering only station replacements does not yield an optimal solution.

4.3.8. Speedup techniques

A straightforward application of our label-setting algorithm suffers from several phenomena which lead to the generation of many superfluous labels: first, it detects infeasible charge and service scheduling decisions only when extending a service arc or reaching the sink. Second, our algorithm is biased towards schedules that charge late and little, which results in the generation of many infeasible paths before reaching the sink and further yields schedules with low diversity. Third, redundancy in the network's structure creates many (cost-)equivalent labels.



(a) The example network. **(b)** Station cost profiles. **(c)** Charging functions.

Note. The example network corresponds to a simplified time-expanded network as introduced in Section 4.3.1. We assume a period length of $\xi = 4$, $\texttt{SoC}_{\min} = 0$, and $\texttt{SoC}_{\max} = 7$. Orange profiles with circle markers indicate station f, blue profiles with diamond markers station g.

Figure 4: Example network (4a), station cost profiles (4b), and charging functions (4c).

We mitigate the first issue by establishing lower bounds on the SoC and propagating service operation time windows to station and garage vertices such that we can discard infeasible paths early. We further derive a *potential function* for each $v \in \mathcal{V}^k$ which estimates the remaining cost required to reach the sink. This potential function then offsets costs of partial paths accordingly to eliminate the bias towards labels with low SoC. Finally, we reduce symmetry issues caused by network redundancy through additional checks in our feasibility condition.

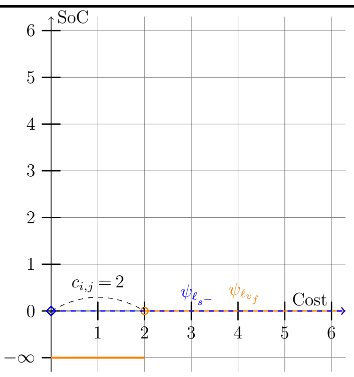
SoC bounds: We seek to establish a lower bound on the SoC at vertex $v \in \mathcal{V}^k$ for each label $\ell \in \mathcal{L}_v$, such that we can discard ℓ if $q_{\max}(\psi_\ell)$ falls below this bound. We express this bound using an auxiliary function $\texttt{ub}(p, p', \Theta')$, which gives an upper bound on the maximum SoC rechargeable in the interval $[\mathcal{P}_i, \mathcal{P}_j]$, $1 \leq i < j \leq |\mathcal{P}|$, when servicing operations $\Theta' \subseteq \Theta_k$:

$$\texttt{ub}(\mathcal{P}_i, \mathcal{P}_j, \Theta') := \underset{f \in F}{max} \angle_{\Phi_f} \cdot \Big(j - i - \sum_{\vartheta \in \Theta'} (\Delta\tau_\vartheta + 1)\Big) \cdot \xi - \sum_{\vartheta \in \Theta'} \Delta\texttt{SoC}_\vartheta.$$

In other words, $\texttt{ub}(\mathcal{P}_i, \mathcal{P}_j, \Theta')$ corresponds to the SoC reached when charging with the maximum charging rate $\underset{f \in F}{max} \angle_{\Phi_f}$ in all non-service periods, accounting for service-related consumption. With this in mind, the following set of conditions needs to hold for a path labeled with label ℓ to be feasibly extensible to the sink:

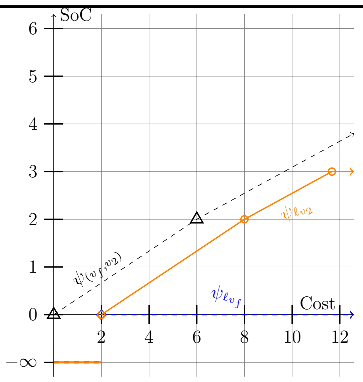
$$\forall \vartheta \in \Theta_k \setminus O_\ell : \overbrace{\texttt{ub}(p(v), d_\vartheta, \underbrace{\{\vartheta' \in \Theta_k \setminus O_\ell \mid \vartheta' \gg \vartheta\}}_{\text{not yet serviced operations that precede } \vartheta})}^{\text{maximum SoC rechargeable before } d_\vartheta} + q_{\max}(\psi_\ell) \geq \Delta\texttt{SoC}_\vartheta.$$

Here, $\gg$ is a precedence relation between operations $\vartheta_a, \vartheta_b \in \Theta_k$, i.e., it indicates whether ϑ_a has to be served before ϑ_b, formally, $\vartheta_a \gg \vartheta_b \Leftrightarrow a_{\vartheta_b} + \Delta\tau_{\vartheta_b} > d_{\vartheta_a}$. Essentially, for each $\vartheta \in \Theta_k$ not yet serviced,



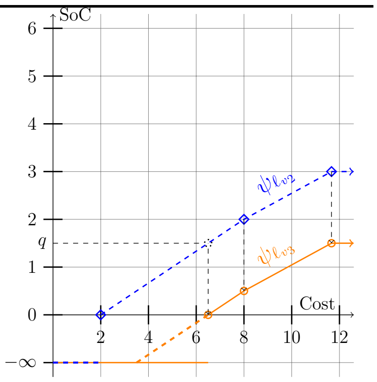
(a) Propagating along source arc (s^-, v_f).

Note. Source arc (s^-, v_f) has a fixed cost of 2, such that we shift $\psi_{\ell_{s^-}}$ (blue, dashed, diamonds) by 2 on the cost axis to obtain $\psi_{\ell_{v_f}}$ (orange, solid, circles).



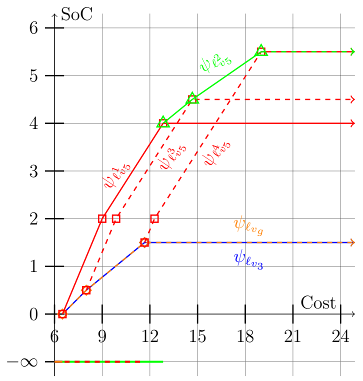
(b) Propagating along charging arc (v_f, v_2).

Note. Propagating profile $\psi_{\ell_{v_{v_f}}}$ (blue, dashed, diamonds) captures the charging trade-off at v_f in $\psi_{\ell_{v_2}}$ (orange, solid, circles). The dashed black profile with triangle markers corresponds to the station cost profile of arc (v_f, v_2).



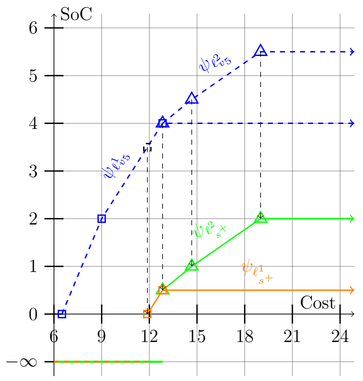
(c) Propagating along service arc (v_2, v_3).

Note. We shift profile $\psi_{\ell_{v_2}}$ (blue, dashed, diamonds) according to the consumption of service arc (v_2, v_3), specifically by $q = 1.5$, on the SoC axis. The dashed part of profile $\psi_{\ell_{v_3}}$ (orange, solid, circles) is cut off.



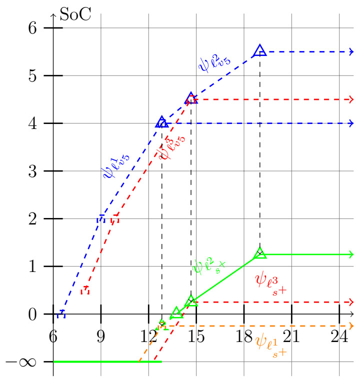
(d) Propagating along idle and charging arcs (v_3, v_g), (v_g, v_5).

Note. Labels $\ell^1_{v_5}, \ell^3_{v_5}$, and $\ell^5_{v_5}$ (red, squares) settle on charging 1.5, 2, and 3 units of energy at v_f, such that the arrival SoC at v_g equals 0, 0.5, and 1.5 respectively. Label $\ell^2_{v_5}$ (green, triangles) commits to charge for $\tau = \xi$ at v_g. Together, the solid red and green profiles dominate the dashed red profiles as they provide higher SoC at a lower cost. The profile of label $\ell^3_{v_5}$ would not be dominated by the profile of label $\ell^1_{v_5}$ or $\ell^2_{v_5}$ alone.



(e) Propagating along service arc (v_5, s^+) with $q_{v_5,s^+} = 3.5$.

Note. The optimal path reaches the sink at a cost of $c_{\min}(\psi_\ell) = 11.875$, charging 1.5 and 3.5 units of energy at v_f and v_g, respectively. Note that the algorithm terminates after propagating $\ell^1_{v_5}$ (blue, dashed, squares) as label $\ell^1_{v_{s^+}}$ (orange, squares) is extracted before $\ell^2_{v_5}$ (blue, dashed, triangles). Label $\ell^2_{v_{s^+}}$ (green, triangles) is thus never created.



(f) Propagating along service arc (v_5, s^+) with $q_{v_5,s^+} = 4.25$.

Note. In this case, propagating label $\ell^1_{v_5}$ (blue, dashed, squares) yields an infeasible label $\ell^1_{s^+}$ (orange, dashed, squares). Propagating label $\ell^2_{v_5}$ (blue, dashed, triangles) yields optimal label $\ell^2_{v_5}$ (green, solid, triangles), which reaches the sink at a cost of $c_{\min}(\psi_\ell) = 14.1875$, charging 1.75 and 4 at v_g and v_f respectively. Here, none of the labels obtained from station replacements at v_g yields the optimal solution.

Figure 5: Running our labeling algorithm on the example network illustrated in Figure 4.

we check whether the residual SoC and the maximum SoC reachable before having to leave for ϑ jointly suffice to service operation ϑ.

Potential functions: We key our labels according to a potential function $h(\ell, v)$ that, for a label $\ell \in \mathcal{L}_v$ at vertex v, computes lower bound on the cost required to feasibly reach the sink:

$$\begin{aligned} h(\ell, v) := c_{\min}(\psi_\ell) + \max(0, (\overbrace{\sum_{\vartheta \in \Theta_k \setminus O_\ell} \Delta\texttt{SoC}_\vartheta}^{\text{remaining consumption}}) - q_{\min}(\psi_\ell)) \\ \cdot \left(\underbrace{\min_{q' \in [\texttt{SoC}_{\min}, \texttt{SoC}_{\max}]} \angle_{\Upsilon}(q')}_{\text{minimum battery degradation cost}} + \overbrace{\min_{p' \in \mathcal{P}, p' \geq p(v)} e_{p'}}^{\text{minimum energy price}} \right) + c(v). \end{aligned} \tag{13}$$

Equation (13) comprises three cost components. First, the cost required to feasibly reach v. Second, a lower bound on the charging-related costs required to reach the sink. Third, the network-related cost $c(v)$ of the shortest path to the sink vertex.

Extended feasibility check: We perform the following additional checks when testing label feasibility to detect energy infeasible or service-incomplete schedules early, reduce label redundancy, and avoid superfluous iterations:

1. We terminate the search when we extract a label with positive cost at any vertex. This remains correct as our algorithm extracts labels in order of lowest cost, such that path cost monotonically increases due to non-positive station capacity duals. Hence, the corresponding vehicle schedule cannot have negative reduced cost.
2. We propagate service arcs only if the associated operation has not been covered yet.
3. We discard labels ℓ extracted at vertex i if $p(i) > d_\vartheta$ for any $\vartheta \notin O_\ell, \vartheta \in \Theta_k$. We relax this strict inequality for non-garage vertices, i.e., $p(i) \geq d_\vartheta$.
4. We do not propagate charging arcs if a label's minimum reachable SoC suffices to cover all operations, such that additional charging is superfluous.

4.4. Branch-and-Price

We embed our column generation procedure into a B&B algorithm to address cases where the RMP's final solution is fractional. The resulting B&P algorithm relies on a problem-specific branching rule (Section 4.4.1), partial pricing (Section 4.4.2), a primal heuristic (Section 4.4.3), and uses a two-stage vertex selection strategy (Section 4.4.4).

4.4.1. Branching rule

Our branching rule bases on the observation of Foster and Ryan (1976) adopted to our problem setting: a basic feasible solution of the RMP is only fractional if charger capacity constraints are binding. In other words, when two or more schedules compete for a charger that is already at capacity.

Proposition 3. *Let σ be a fractional basic feasible solution to the RMP. Then it holds that $\exists (p,f) \in \mathcal{P} \times \mathcal{F}$ such that $\sum_{x_\omega^k \in \sigma} x_\omega^k \boldsymbol{A}_{p,f}^\omega = C_f$, and there exist at least two $x_\omega^k \in \sigma$ with $0 < x_\omega^k < 1$.*

Proof: see Appendix B.

We refer to pairs $(p,f) \in \mathcal{P} \times \mathcal{F}$ where this occurs as *conflicts* and resolve these by creating a new branch for each vehicle $k \in \mathcal{K}$ that participates in the conflict, formally where $\sum_{\omega \in \tilde{\mathcal{A}}_k} x_\omega^k \boldsymbol{A}_{p,f}^\omega > 0$. Each of these branches cuts off solutions where vehicle k uses charger f in period p. We enforce these constraints in our subproblems to avoid additional dual variables in our master problem. For this purpose, we purge all violating columns from $\tilde{\mathcal{A}}_k$ and remove the station vertex representing the charging opportunity from G^k.

Our algorithm relies on a hierarchial selection strategy to resolve cases with multiple conflicts. Specifically, we prioritize conflicts $(p,f) \in \mathcal{P} \times \mathcal{F}$ by i) the most fractional participating schedule, ii) the number of non-integral columns, iii) the amount of energy recharged, iv) charging speed of f, and v) the lowest vehicle index.

4.4.2. Partial pricing

Schedules generated in the same iteration often show similar charging patterns due to shared dual variables, overlapping time windows, and equal energy prices, such that charger capacity constraints limit the number of simultaneously generated schedules that can be part of the same basic solution. Hence, many of the schedules generated do not contribute to the convergence of the dual variables and the lower bound, especially when considering large fleets and a high number of low-capacity chargers such that limiting the number of vehicles priced in each iteration often reduces total runtime, even if this requires additional iterations of the column generation procedure.

We implement this so-called *partial pricing* approach as follows: After solving the RMP, we solve the arising pricing subproblems in a round-robin fashion until a total number of ν schedules with negative reduced costs have been generated or all subproblems have been considered. Our termination criterion remains unchanged, that is, we still terminate the column generation procedure when no subproblem produces a schedule with negative reduced cost. The main factor that drives the computational effectiveness of this approach is the number of columns to generate, i.e., ν. Our experiments show that aligning ν with the minimum charger capacity performs best.

One of the major drawbacks of partial pricing is that the lower bound generated at each iteration is relatively weak (cf. Desrosiers and Lübbecke 2005). We address this issue by periodically performing a full iteration, i.e., temporarily set $\nu = |\mathcal{K}|$.

4.4.3. Primal heuristic

We utilize a primal heuristic to quickly find upper bounds, thus speeding up the solution procedure by allowing to prune nodes of the B&B tree early. To this end, we use a diving heuristic that explores an auxiliary branch-and-bound tree in a depth-first fashion, branching on the variables x^k_ω of the extensive formulation (IP 2). At each node of this auxiliary B&B tree, the algorithm forces some fractional x^k_ω to one until an integral or infeasible solution is found. We rely on strong branching to boost the success rate and solution quality of this diving algorithm: at each node explored in the diving phase, we bound the impact of fixing a candidate schedule by solving the accordingly modified RMP. We then fix the schedule promising the most improvement based on the derived lower bounds.

As this procedure is expensive for large numbers of schedules, our algorithm considers only a small subset of schedules at each node. We select these from the column set at the current node ($\tilde{\mathcal{A}}^N$) according to a roulette wheel criterion based on *dissimilarity* and *quality*. For this purpose, we define the distance $d(\omega_1, \omega_2)$ between two columns $\omega_1, \omega_2 \in \tilde{\mathcal{A}}^N_k$ as the hamming distance between charger allocation matrices $\{\boldsymbol{A}^{\omega_1}_{p,f} \mid p \in \mathcal{P}, f \in \mathcal{F}\}$ and $\{\boldsymbol{A}^{\omega_2}_{p,f} \mid p \in \mathcal{P}, f \in \mathcal{F}\}$. Accordingly, the distance between two schedules is inversely proportional to the number of shared charging operations between the two schedules. For each vehicle without an assigned schedule, we then build a pool of columns Θ_k as follows: Let $r_Q(\omega)$ be the *rank* of ω in $\tilde{\mathcal{A}}^N_k$ according to its schedule's cost. Furthermore, let $D(\omega, \Theta_k)$ be the average distance to all $\omega' \in \Theta_k$ or 0 if $\Theta_k = \emptyset$. Again, $r_D(\omega, \Theta_k)$ gives the rank of ω according to $D(\cdot)$. We then build Θ_k greedily such that Equation (14) is maximized:

$$F(\omega, \Theta_k) := \alpha r_Q(\omega) + (1-\alpha) r_D(\omega, \Theta_k). \tag{14}$$

We stop once the column pool reaches the desired size ($\bar{\Theta}_k$).

We execute this primal heuristic on every node of the B&B tree until the algorithm finds an integral solution. We switch to a periodic evaluation strategy at this point and run our heuristic only on every $|\mathcal{K}|^{\text{th}}$ node.

4.4.4. Node selection strategy

We place the nodes generated by our branching procedure into a node queue. The order of this queue dictates the order in which open nodes are solved and hence influences the performance of the B&P algorithm. Our node selection strategy combines the benefits of the two most common branching strategies, namely *depth-first*, which prioritizes nodes with the highest distance from the root node, and *best-bound-first*, which selects nodes according to their lower bounds (cf. Wolsey 1998), in a two-stage approach. We first select nodes according to the depth-first strategy, breaking ties by vehicle index, and switch to best-bound-first when we find a feasible integer solution during branching or using our primal heuristic. Here, we order nodes with equal lower bounds by node depth, prioritizing nodes deeper in the tree. We resolve any remaining ties by picking the leftmost node.

4.4.5. Extensions to related problem settings

One of the major assumptions of our problem setting is the fixed service operation assignment, which limits our algorithm to CSP. Hence, the following section discusses how our algorithm can be extended to related problem settings that relax this assumption and suggest problem settings that certain algorithmic components, such as our labeling algorithm, can be applied to.

Treating service operation assignment as a decision requires several modifications to our algorithm: first, each vehicle now serves only a subset of service operations, which replaces convexity constraints (2c) with a covering constraint for each service operation. This transforms our label-setting algorithm into a label-correcting algorithm as the corresponding dual variables may be positive, and requires changes to the pricing network design. Specifically, in this problem setting, each network must contain service arcs for each service operation. Finally, an additional branching rule that deals with fractional solutions caused by the covering constraints is necessary. We note that this extended formulation would likely require additional algorithmic components to remain scaleable, e.g., in the form of valid inequalities. Other components of our B&P algorithm, e.g., the primal heuristic, require no further modification. In fact, our primal heuristic is straightforwardly amendable to arbitrary B&P algorithms.

Finally, we like to note that the core concept of our labeling algorithm can be used in any problem setting with non-linear, convex trade-off functions. Here, notable examples are the time-dependent vehicle routing problem with variable speed and vehicle routing problems with speed-dependent energy consumption.

4.5. Design of experiments

The aim of our computational study is twofold: a first set of numerical experiments validates the correctness, investigates the performance, and analyzes the scalability of our B&P algorithm. Specifically, we (i) benchmark our algorithm against the MIP-based formulation introduced in Appendix E, (ii) assess the scalability of our algorithm with respect to various instance parameters, such as fleet size, planning horizon length, and charger contention, and (iii) analyze the impact of our individual algorithmic components, i.e., partial pricing, node selection, and the primal heuristic. Our numerical experiments rely on instances generated using a parameterized random instance generation procedure, which is detailed in Appendix F. We will provide the specific parameter settings for each experiment when discussing the respective results. Additionally, we host the code sources at `https://github.com/tumBAIS/`.

Second, we assess the impact of integrated charge and service operation scheduling in a potential real-world scenario. Here, we analyze to which extent flexible service operations improve overall vehicle utilization, investigate how service flexibility affects the amount of charging infrastructure required, and assess the impact of energy prices on the cost savings obtainable through integrated charge and service scheduling. To keep this paper concise, we refer to Appendix F for details on the instance derivation of our numerical experiments and focus on the design of our managerial study in the following:

Our managerial study captures the planning problem of a logistics service provider (LSP) supplying retail stores in an urban area over a planning horizon of two days. We assume a fleet of sixteen vehicles, each operating three shifts a day, i.e., one night, one morning, and one afternoon shift. Each shift comprises a delivery tour that takes six hours and consumes 15 kWh of energy, such that the battery is depleted once a day. We randomly distribute operation departure times such that vehicles spend a minimum of one hour before each operation at the depot. We extend this slack to two hours for the first operation assigned to each vehicle and distribute earliest and latest departure times symmetrically around the departure time of the case without any service flexibility. Each vehicle is equipped with a 45 kWh battery, purchased at a price of €5.406. This corresponds to an average price per kWh of $120/kWh (BloombergNEF 2021) at an exchange rate of 0.877€/$. We use the battery wear data provided in Han, Han, and Aki (2014) calibrated according to our battery price to derive the WDF (cf. Table 3a). Energy prices are based on the hourly day-ahead spot market price for the 16.01.2021 and 17.01.2021 (ENTSOE 2021), linearly interpolated to fit 30-minute time steps. We scale the reported prices according to the average European energy price of €0.2127 in 2020 (Eurostat 2021). As we aim to capture potential charging trade-offs during off-service periods, our planning horizon starts with the previous day's end-of-operations (22:00), assuming an initially empty battery. The depot is equipped with a set of six DC fast chargers. Additionally, each vehicle may be charged using its on-board charger. We refer to Table 3b for details on the charging functions used.

4.6. Results and discussion

We conducted all of our experiments on a standard desktop computer equipped with an `Intel(R) Core(TM) i9-9900, 3.1 GHz CPU` and `16 GB` of `RAM`, running `Ubuntu 20.04`. We have implemented the B&P algorithm in `Python (3.8.11)` using IBM CPLEX (Version 20.01) to solve the RMP. The pricing problem is implemented in `C++` (GCC 11.1.0). We considered solutions with a relative gap

	Fleet size	Planning horizon length (days)	Time window length (periods)	Charger count	Total charger capacity	Segment count (Υ)	Segment count (Φ_f)
Small	3	1	6	1	1	3	3
Base	12	2	4	2	6	4	3
Min	12	1	0	1	6	2	2
Max	68	5	8	6	12	8	8
Step	8	1	1	1	1	1	1

Table 2: Instance generation parameter values used in the numerical experiments.

(a) Considered WDF.

SoC	Cost	Unit cost
25%	1.59€	0.14€
50%	3.30€	0.15€
75%	5.20€	0.17€
100%	7.79€	0.23€
Avg.	-	0.17€

(b) Considered charging functions.

Fast		Slow	
Time (minutes)	SoC	Time (minutes)	SoC
72.32	34.90	435	45.0
92.6	42.49		
120	45.00		
Avg. rate	22.5 kW/h	Avg. Rate	6.21 kW/h
Full charge	2 h	Full charge	7.25 h

Table 3: Wear-cost density function and charging infrastructure considered in the case study.

of less than 0.0001 optimal and ran all our experiments in a single thread. We further limited the computational time to 3600 seconds. Setup time is negligible and thus not reported. We refer to the online Appendix for instances and detailed results.

4.6.1. Computational performance

We benchmark our algorithm against the MIP proposed in Appendix E on a set of 50 small instances, generated according to Appendix F and the parameter values listed as *small* in Table 2. Table 4 summarizes our results and compares runtime, bounds, gap, size of the B&B tree, the total number of instances, and the number of instances solved to optimality. As can be seen, the proposed B&P algorithm outperforms the MIP on all instances, proving optimality in less than a second on average.

In the remainder of this section, we focus on the performance of our B&P algorithm on larger instances. Here, we generate a total of 25 instances according to Appendix F, varying each parameter in Table 2 separately, such that our study comprises a total of 1300 runs. Figure 6 summarizes our results on these.

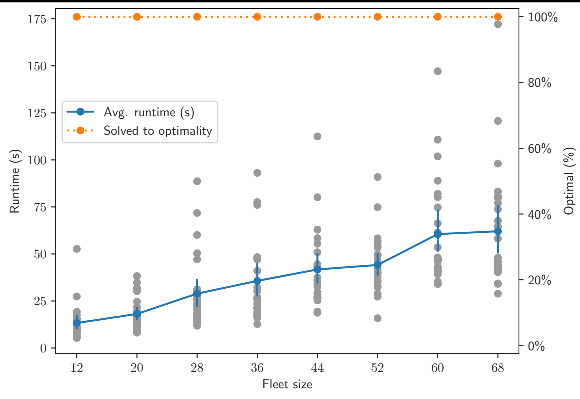
Impact of fleet size (Figure 6a): Our results indicate a linear increase in runtime with increasing fleet size, which results from the number of subproblems scaling linearly with the number of vehicles. Note that we keep the ratio of vehicles to charger capacity constant in this experiment to isolate the effect of increasing fleet size.

Impact of planning horizon length (Figure 6b): Our results indicate a strong correlation between planning horizon length and the average runtime. We can attribute this to the computational complexity of the pricing subproblem: longer planning horizons increase the network size linearly,

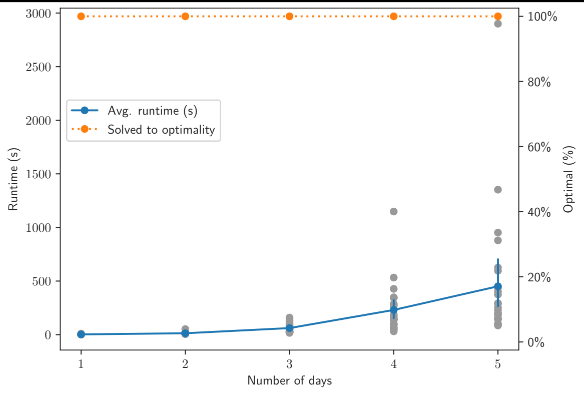
Algorithm	Avg. t[s]	Avg. obj.	Avg. LB	Avg. #nodes	#optimal	#unsolved	#total
Branch & Price	0.85	396.56	396.56	4.98	50	0	50
MIP	3600.00	397.46	114.74	2183022.26	1	2	50

Abbreviations hold as follows: t[s] - runtime in seconds, obj. - objective value, LB - lower bound, #nodes - size of the B&B tree, #optimal - number of optimally solved solutions, #unsolved - number of instances where no incumbent was identified, #total - total number of instances. Averages do not include unsolved instances.

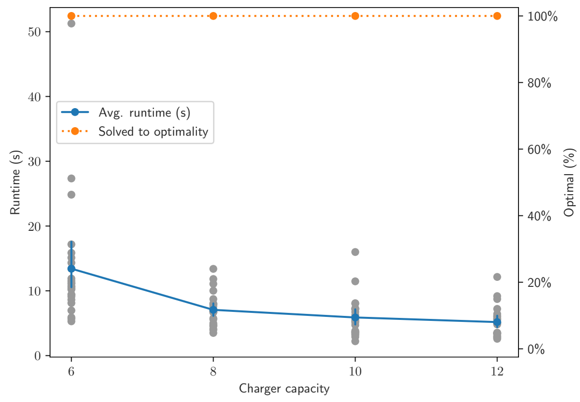
Table 4: Aggregated results for the small benchmark instances.



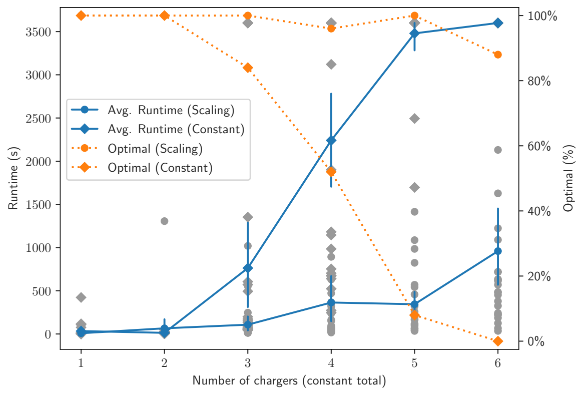
(a) Runtime for varying fleet sizes.



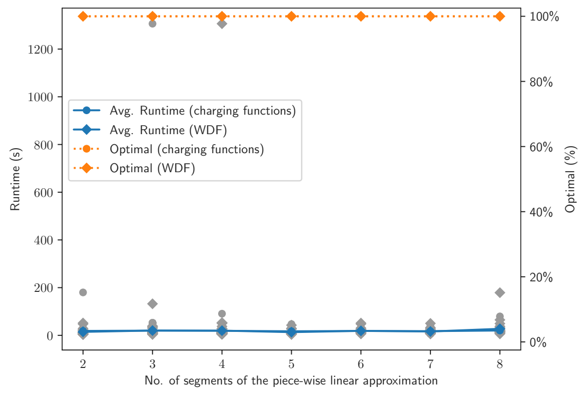
(b) Runtime for varying planning horizon length.



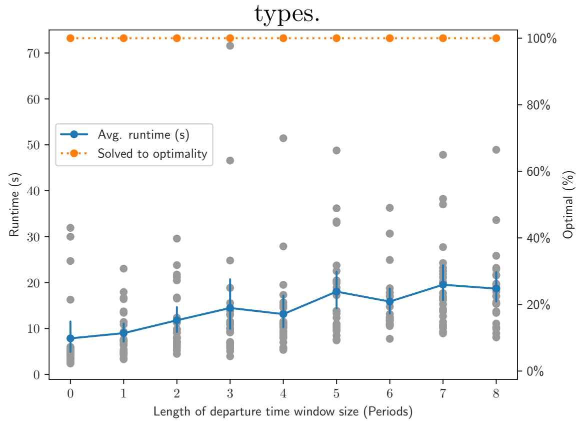
(c) Runtime for varying charger capacities.



(d) Runtime for varying numbers of distinct charger types.



(e) Runtime for varying degree of approximation quality.



(f) Runtime for varying departure time window size.

Note. Each gray dot represents a specific instance. The solid blue line corresponds to the average runtime in seconds. The dashed orange line shows the percentage of instances where the algorithm did not prove optimality within 3600 seconds.

Figure 6: Results for the large benchmark instances.

such that the number of generated labels increases exponentially in the worst case. Additionally, longer planning horizons increase the probability of charger conflicts such that the number of branches increases and more iterations are necessary to solve the individual B&B nodes.

Impact of charger capacity (Figure 6c): Here, the runtime decreases with increasing charger capacity. This can be attributed to lower charger contention, which accelerates the convergence of both the branch and bound and column generation procedures.

Impact of the number of charger types (Figure 6d): We consider a *scaling* and a *constant* setting to isolate the effect of the number of charger types on runtime. In the scaling setting, we increase the total charger capacity proportional to the number of chargers added, such that adding a charger always increases the total number of available charging spots. In the constant setting, we instead distribute the base case charger capacity across available chargers such that the total capacity remains constant. In both experiments, the average runtime increases with the number of chargers available as the size of the pricing network increases. When keeping the total capacity constant, increasing the number of chargers also increases charger contention as vehicles compete for more chargers with lower individual capacities. Together, these effects cause an exponential runtime increase. Increasing the charger capacity proportional to the number of chargers weakens both effects. Here, runtime scales linearly with the number of chargers.

Impact of approximation quality (Figure 6e): This experiment assesses the effect of increasing the number of breakpoints, and thus the approximation quality, of our piecewise linear functions. Recall that the number of created labels increases with the number of breakpoints of both the WDF and the charging functions (cf. Sections 4.3.5 & 4.3.6). Nevertheless, the runtime of our algorithm varies only slightly with increasing approximation quality. This underlines the strength of our dominance criteria.

Impact of departure time window size (Figure 6f): By design of the time-expanded network, one would expect a substantial increase in computation time with increasing departure time window length: longer time windows add additional arcs and thus increase the number of feasible paths through the network. Nevertheless, our algorithm scales well to longer time windows, and we observe only a slight increase in runtime.

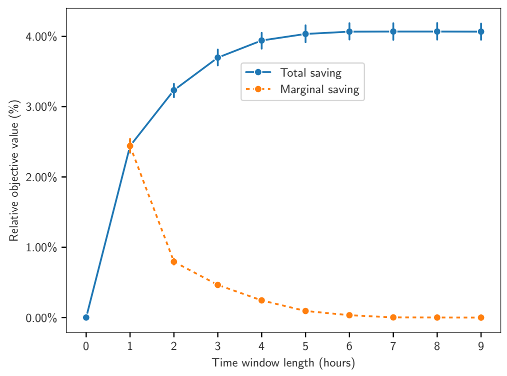
In conclusion, our algorithm manages to reliably solve instances with 68 vehicles or a planning horizon of 5 days within an hour, allowing for day-ahead planning in practice.

Our last numerical experiment analyzes the impact of various B&P-related algorithmic enhancements detailed in Section 4.4, i.e., partial pricing, node selection, and the primal heuristic. For this purpose, we benchmark five different algorithmic configurations (cf. Table 5) on a set of 50 instances sampled

Algorithm	PP	PH	NS	Avg. t[s]	Rel. t[s]	Avg. #nodes	Rel. #nodes
A_1			FIFO	397.30	100.00%	7.46	100.00%
A_2	✓		FIFO	308.55	77.66%	6.42	86.06%
A_3		✓	FIFO	336.93	84.80%	4.48	60.05%
A_4			Two-Stage	367.58	92.52%	6.74	90.35%
A_5	✓	✓	Two-Stage	224.39	56.48%	2.9	38.87%

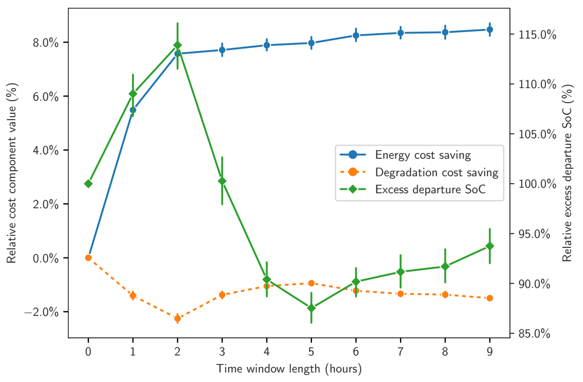
Abbreviations hold as follows: PP - partial pricing, PH - primal heuristic, NS - node selection rule, Avg. t[s] - average runtime in seconds, Rel. t[s] - relative runtime compared to A_1, #nodes - average number of B&B nodes solved, Rel. t[s] - relative number of B&B nodes solved compared to A_1.

Table 5: Analysis of the contribution of individual components



(a) Relative objective value savings.

Note. The solid blue line shows the total objective value saving relative to the static scenario. The dashed orange line shows the marginal saving.



(b) Cost components of relative savings.

Note. The solid blue and dashed orange lines show the average energy and degradation cost savings compared to the static scenario. The green line shows the average excess departure SoC, i.e., the charge upon departure beyond $\Delta\mathtt{SoC}_\vartheta$ averaged over all operations $\vartheta \in \Theta$, relative to the static scenario.

Figure 7: Results of the basecase experiment.

from those solved to optimality in our scalability experiment (cf. Table 2). Our results show that the configurations utilizing our algorithmic enhancements outperform the unmodified B&P algorithm on average. Here, the impact of partial pricing (A_2) is strongest with a runtime improvement of 22.34%, followed by our primal heuristic (A_3) with an improvement of 15.20%, and our node selection rule (A_4) with an improvement of 7.48%. The effect of these algorithmic components is additive: the configuration that combines all components performs best, reducing runtime by 43.52% on average.

4.6.2. Managerial study

Impact on total cost savings This study aims to derive managerial insights on the impact of service operation flexibility on total cost savings. For this purpose, we have run the developed algorithm on a set of 50 realistic instances derived according to Section 4.5 with varying degrees of schedule flexibility, resulting in a total of 500 runs. Figures 7a and 7b summarize our results.

Figure 7a shows the average total and marginal objective value savings of instances with varying time window lengths compared to the same instances with static departure times. We observe that flexible service operations have an overall positive impact on the objective value. Specifically, allowing a service time window of one hour already yields a cost saving of 2.5%. The marginal saving decreases sharply with increasing flexibility, such that one hour of flexibility already exploits 50% of the savings potential, while four hours of flexibility exploit 80%. The relative saving converges to 5% at nine hours of flexibility.

Result 1. Flexible service operations have a positive impact on the total cost. The obtainable savings increase with increasing flexibility, converging to maximum cost savings of 5%. Marginal savings decrease with increasing time windows, such that one hour of flexibility already exploits 50% of the savings potential, while four hours of flexibility yield 80% of the obtainable savings.

Figure 7b shows the impact of increasingly flexible service operations on the individual cost components and excess departure SoC. Here, we observe a trade-off between battery degradation and energy cost. Specifically, higher schedule flexibility allows utilizing periods with energy prices cheap enough to outweigh additional battery degradation costs caused by cycling the battery at higher SoC levels, such that energy costs decrease while average excess departure SoC, and thus battery degradation costs, increase.

Result 2. Dynamic service operations allow utilizing the trade-off between battery degradation cost and energy prices.

Impact of charger capacity We slightly adapt our instance generation procedure and limit charging operations to the fast charger (Table 3b) to isolate the impact of integrated charge and service operation scheduling on charging infrastructure utilization. Again, we generate 50 base instances on which we then vary departure time window length and charger capacity, resulting in a total of 8.000 instances. Figures 8 & 9 illustrate the results of our charger capacity analysis.

Figure 8 shows the percentage of feasible instances for varying charger capacities and time window lengths. Our scenario requires a minimum charger capacity of seven to support fleet operations when planning service and charging operations separately. Integrated planning of charge and service operations lowers the required charger capacity from seven to three, such that only 86% of the original charger capacity is necessary with one hour of flexibility. A departure time window of three hours, which amounts to half of the shift length, reduces the required capacity to 57%. Further doubling the departure time window length yields an additional reduction of 14% to 43% of the basecase's capacity.

Result 3. Integrated planning of charge and service operations reduces the amount of charging infrastructure required for fleet operation but shows decreasing marginal benefits: one hour of flexibility reduces the number of chargers required by 14%, three hours by 43%, and six hours by 57%.

Figure 9 shows the total operational costs for varying charger capacities and time window lengths. As we can see, both adding additional chargers and increasing service flexibility reduces the total operational costs. Here, integrated planning has a stronger effect than increasing charger capacity. Specifically, a departure time window of one hour already outperforms the static scenario, even when doubling the number of available chargers such that a dedicated charger is available for each vehicle. Moreover, with a charger capacity of eight, the total cost of static operations, two, and five hours of planning flexibility amounts to €481.63, €464.21 (3.62% saving), and €459.50 (4.59% saving), respectively. Increasing the charger capacity by four lowers these costs to €479.96, €463.65 (3.40% saving), and €456.75 (4.83% saving), respectively. In comparison, the cost saving of adding four additional chargers to a scenario with eight chargers amounts to 0.34%, 0.12%, and 0.60% for static operations, two, and five hours of planning flexibility, respectively.

RESULT 4. Increasing either service operation flexibility or charger capacity lowers operational costs. Here, increasing flexibility yields higher savings than additional investments into charging infrastructure.

Generally, increasing the number of chargers available offsets total savings, such that the number of chargers available limits the maximum cost savings obtainable through integrated planning: with eight chargers, average savings converge to 4.62%, with 12 chargers, the maximum saving obtainable is 4.84%, and reaches 5.37% when each vehicle has a dedicated charging spot.

RESULT 5. Charger capacity offsets the cost savings obtainable and thus limits the maximum cost savings of integrated charge and service operation planning. With the minimum number of chargers required in the static scenario, savings peak at 4.62%; increasing the charger capacity to 150% and 200% further increases peak savings to 4.84% and 5.37%, respectively.

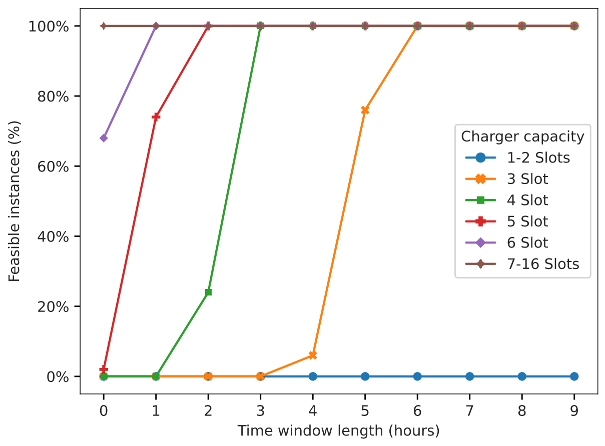
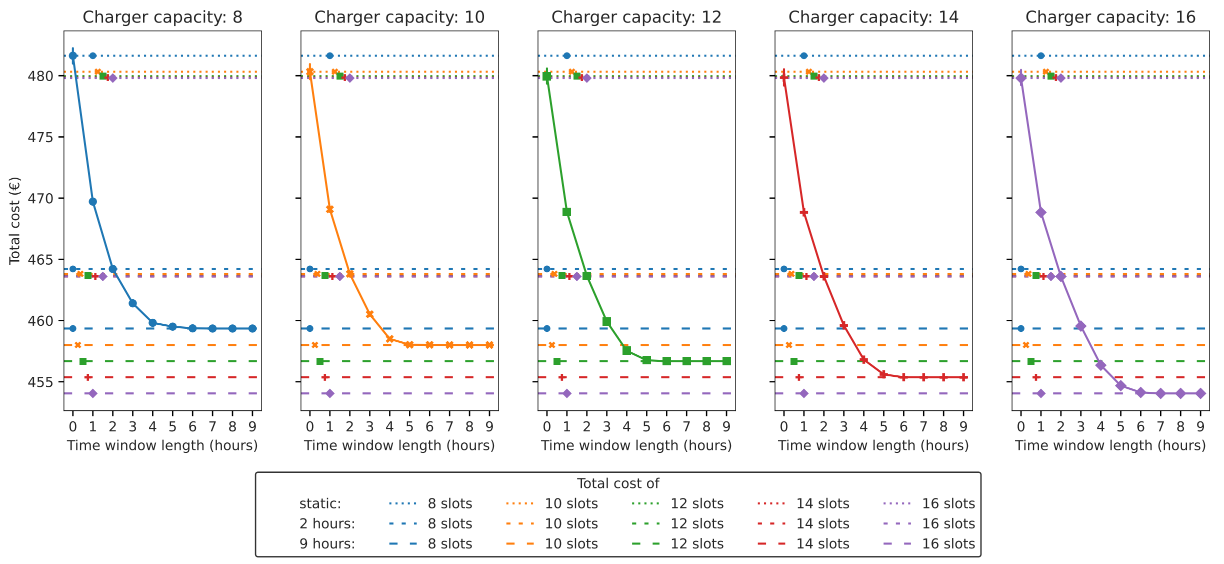


Figure 8: Percentage of feasible instances for varying charger capacity and time window length.

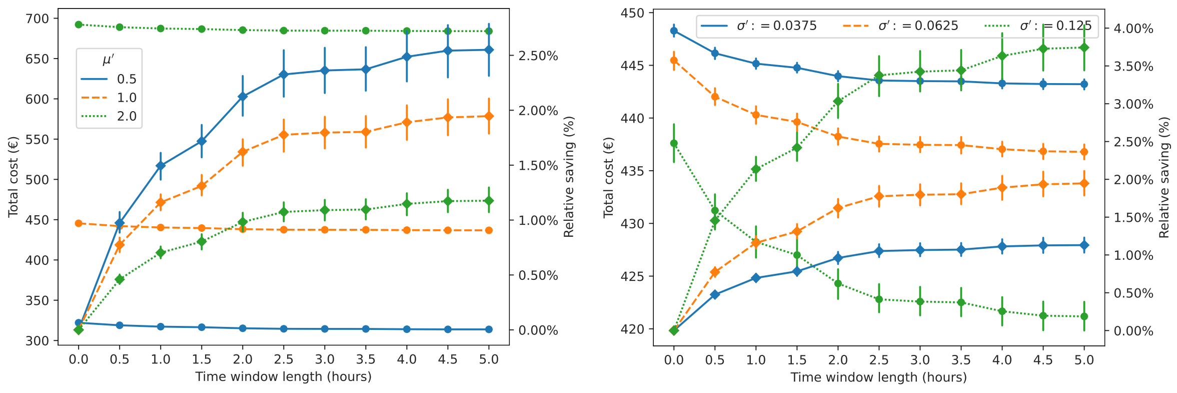


Note. The dotted and dashed lines give the objective for different charger capacities assuming static time windows, a time window size of two hours, and nine hours, respectively.

Figure 9: Objective value for varying charger capacity and time window length.

Impact of energy price distribution We modify the energy price distribution considered in our basecase to investigate its effect on the cost savings obtainable through integrated charge and service operation scheduling. Specifically, we draw energy prices from a normal distribution $\mathcal{N}(\mu' \cdot \overline{\Upsilon}, \sigma' \cdot \overline{\Upsilon})$, where $\overline{\Upsilon}$ gives the average unit wear cost of our WDF (cf. Table 3a). We consider scenarios with high ($\mu' := 2.0$), medium ($\mu' := 1.0$), and low ($\mu' := 0.5$) energy rates. Analogously, we compare TOU plans with high ($\sigma' := 0.1250$), medium ($\sigma' := 0.0625$), and low ($\sigma' := 0.0375$) variance. We generate instances with departure time windows of up to five hours in a full factorial design, such that our study comprises a total of 4.950 runs.

Figure 10 compares the cost savings relative to statically scheduled service operations across time windows of varying size and different values for μ' and σ', respectively. The mean energy price has an offsetting effect on the total cost. Specifically, we observe an average 56.59% increase in total cost when doubling energy prices. Energy rates discounted by 50% yield average savings of 28.23%. Regarding the relative cost saving of adding additional flexibility, we observe the opposite effect. Here, relative savings increase with lower mean energy price: with two hours of flexibility, the high, medium, and low rate plans provide an average relative saving of 0.98%, 1.62%, and 2.12%, respectively. At five hours, the average relative savings amounts to 1.18%, 1.95%, and 2.55%, respectively. The marginal saving is unaffected by mean energy price: high, medium, and low rates each reach 80% of the maximum saving at a time window length of two hours. This is related to the higher impact of battery degradation in scenarios with a low mean energy price.



(a) Impact of mean energy price ($\sigma' := 0.0625$). **(b)** Impact of energy price variance ($\mu' := 1.0$).

Note. Lines with round markers show the total cost. Lines with diamond markers show relative savings.

Figure 10: Results of the energy price experiment.

Result 6. The mean energy price offsets total cost. The impact of integrated charge and service operations planning is most pronounced when battery degradation costs are relatively high.

Concerning the energy price standard deviation, we observe two effects: first, TOU plans with highly variable energy prices lead to greater cost savings in the considered scenario. Specifically, the TOU plans with high, medium, and low variance converge to a saving of 3.74%, 1.95%, and 1.12%, respectively, such that the savings potential increases threefold under highly variable energy prices compared to a rate plan with low variance. Second, the relative savings of additional service flexibility increase with increasing variance. In other words, the higher the energy price variance, the higher the impact of service flexibility.

Result 7. Integrated planning of charge and service operations provides the largest savings in scenarios with highly variable energy prices. Specifically, doubling and quadrupling the energy price variance roughly doubles and triples relative savings.

4.7. Conclusion

We presented a novel charge- and service operation scheduling problem where a fleet of electric vehicles fulfills a set of service operations under the assumption of limited charging station capacity, variable energy prices, battery degradation, and non-linear charging behavior. We developed an exact algorithm based on B&P to solve the proposed problem. A novel labeling algorithm with efficient dominance criteria, a primal heuristic, and a problem-specific branching rule establish the efficiency of our algorithm, which we demonstrated in numerical experiments. This numerical study asserts the competitiveness of our algorithm through a benchmark against an equivalent mixed-integer formulation, showing that our algorithm significantly outperforms commercial solvers. This study further shows the algorithm's scalability to instances of larger size, optimally solving instances with planning horizons of 5 days or 68 vehicles within the hour, allowing for day-ahead planning in practice. We further derived several managerial insights concerning the impact of service flexibility. Specifically, we find that integrated scheduling of charge and service operations allows to better utilize the trade-off between battery degradation costs and energy price, such that cost savings of up to 5% can be realized. Moreover, service flexibility reduces charger contention, allowing to reduce the number of chargers installed by up to 57%. Finally, we analyze the impact of different TOU plans on the benefit of flexible service operations. Here, we find that integrated charge and service operation scheduling performs best in scenarios with highly variable energy rates.

The methodology developed in this work serves as a starting point for future research on charge scheduling problems. Here, potential avenues include extending the B&P algorithm to a Branch-Cut-and-Price framework, i.e., by deriving problem-specific cutting planes, heuristic approaches, and integrating charge scheduling into a vehicle routing or scheduling problem.

Acknowledgments

This work was supported by the German Federal Ministry for Economic Affairs and Energy within the project MILAS (01MV21020B).

References

Abdelwahed A, van den Berg PL, Brandt T, Collins J, Ketter W, 2020 *Evaluating and Optimizing Opportunity Fast-Charging Schedules in Transit Battery Electric Bus Networks. Transportation Science* 54(6):1601–1615.

Adler J, Mirchandani PB, 2016 *The vehicle scheduling problem for fleets with alternative-fuel vehicles. Transportation Science* 51(2):441–456.

Alvo M, Angulo G, Klapp MA, 2021 *An exact solution approach for an electric bus dispatch problem. Transportation Research Part E: Logistics and Transportation Review* 156:102528.

Batsyn MV, Batsyna EK, Bychkov IS, Pardalos PM, 2021 *Vehicle assignment in site-dependent vehicle routing problems with split deliveries. Operational Research* 21(1):399–423.

Baum M, Dibbelt J, Gemsa A, Wagner D, Zündorf T, 2019 *Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles. Transportation Science* 53(6):1627–1655.

BloombergNEF, 2021 *Long-term electric vehicle outlook.* `https://www.bloomberg.com/news/terminal/QUF0OCDWRGGH`, last accessed: 01.10.2022.

Boland N, Hewitt M, Marshall L, Savelsbergh M, 2019 *The price of discretizing time: a study in service network design. EURO Journal on Transportation and Logistics* 8(2):195–216.

Boland N, Hewitt M, Marshall L, Savelsbergh M, 2020 *The continuous-time service network design problem. Operations Research* 65(5):1303–1321.

Bruglieri M, Mancini S, Pisacane O, 2019 *The green vehicle routing problem with capacitated alternative fuel stations. Computers & Operations Research* 112:104759.

Clark D, 2019 *Delivering shipment zero, a vision for net zero carbon shipments.* `https://blog.aboutamazon.com/sustainability/delivering-shipment-zero-a-vision-for-net-zero-carbon-shipments`, last accessed: 24.06.2019.

Conrad RG, Figliozzi MA, 2011 *The recharging vehicle routing problem.* Doolen T, Van Aken E, eds., *Proceedings of the 2011 Industrial Engineering Research Conference*, 1–8 (Reno, NV).

Dabia S, Ropke S, van Woensel T, De Kok T, 2013 *Branch and Price for the Time-Dependent Vehicle Routing Problem with Time Windows. Transportation Science* 47(3):380–396.

Desaulniers G, Errico F, Irnich S, Schneider M, 2016 *Exact algorithms for electric vehicle-routing problems with time windows. Operations Research* 64(6):1388–1405.

Desrosiers J, Lübbecke ME, 2005 *A Primer in Column Generation*, 1–32 (Boston, MA: Springer US), ISBN 978-0-387-25486-9.

DPDHL, 2017 *Mission 2050: Zero emissions.* `https://www.dpdhl.com/content/dam/dpdhl/en/media-center/responsibility/dpdhl-flyer-gogreen-zero-emissions.pdf`, accessed: 25.06.2019.

EEA, 2018 *EEA report no 13/2018.* Technical report, European Environment Agency.

ENTSOE, 2021 *Spot market day-ahead prices.* `https://transparency.entsoe.eu/transmission-domain/r2/dayAheadPrices/show`, last accessed 17.01.2021.

Erdoğan S, Miller-Hooks E, 2012 *A green vehicle routing problem. Transportation Research Part E: Logistics and Transportation Review* 48(1):100–114.

Eurostat, 2021 *Electricity Prices by type of user.* `https://strom-report.de/electricity-prices-europe/`, online data code: TEN00117, last accessed: 10.01.2022.

Foster BA, Ryan DM, 1976 *An integer programming approach to the vehicle scheduling problem. Operational Research Quarterly (1970-1977)* 27(2):367.

Franco A, 2015 *Rechargeable lithium batteries: from fundamentals to applications* (Elsevier), ISBN 978-1-78242-090-3.

Froger A, Jabali O, Mendoza JE, Laporte G, 2022 *The Electric Vehicle Routing Problem with Capacitated Charging Stations. Transportation Science* 56(2):460–482.

Froger A, Mendoza J, Jabali O, Laporte G, 2019 *Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. Computers & Operations Research* 104:256–294.

Griffin L, 2021 *ALtogether ElectricAL.* `https://www.addisonlee.com/addlib/addison-lee-fully-electric-fleet-by-2023/`, last accessed: 01.10.2022.

Guo F, Zhang J, Huang Z, Huang W, 2022 *Simultaneous charging station location-routing problem for electric vehicles: Effect of nonlinear partial charging and battery degradation. Energy* 250:123724.

Han S, Han S, Aki H, 2014 *A practical battery wear model for electric vehicle charging applications. Applied Energy* 113(113):1100–1108.

Juan A, Méndez C, Faulin J, Armas J, Grasman S, 2016 *Electric Vehicles in Logistics and Transportation: A Survey on Emerging Environmental, Strategic, and Operational Challenges. Energies* 9:86.

Kok AL, Hans EW, Schutten JMJ, 2011 *Optimizing departure times in vehicle routes. European Journal of Operational Research* 210(3):579–587.

Kullman ND, Froger A, Mendoza JE, Goodson JC, 2021 *frvcpy: An Open-Source Solver for the Fixed Route Vehicle Charging Problem. INFORMS Journal on Computing* 33(4):1277–1283.

Lam E, Desaulniers G, Stuckey PJ, 2022 *Branch-and-cut-and-price for the Electric Vehicle Routing Problem with Time Windows, Piecewise-Linear Recharging and Capacitated Recharging Stations. Computers & Operations Research* 145:105870.

Lee C, 2020 *An exact algorithm for the electric-vehicle routing problem with nonlinear charging time. Journal of the Operational Research Society* 71(1):1–24.

Liang Y, Dabia S, Luo Z, 2021 *The Electric Vehicle Routing Problem with Nonlinear Charging Functions. arXiv:2108.01273 [cs]* .

Lin B, Ghaddar B, Nathwani J, 2021 *Electric Vehicle Routing and Charging/Discharging under Time-Variant Electricity Prices. Transportation Research Part C: Emerging Technologies* 130:103285, arXiv: 2012.09357.

Luo Z, Qin H, Zhu W, Lim A, 2017 *Branch and Price and Cut for the Split-Delivery Vehicle Routing Problem with Time Windows and Linear Weight-Related Cost. Transportation Science* 51(2):668–687.

Lyft, 2021 *Leading the Transition to Zero Emissions: Our Commitment to 100% Electric Vehicles by 2030.* `https://www.lyft.com/blog/posts/leading-the-transition-to-zero-emissions`, last accessed: 10.01.2022.

Marra F, Fawzy YT, Bülo T, Blažic B, 2012 *Energy storage options for voltage support in low-voltage grids with high penetration of photovoltaic. 2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, 1–7.

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

Olsen N, 2020 *A literature overview on scheduling electric vehicles in public transport and location planning of the charging infrastructure* (Freie Universität Berlin).

OpenEI, 2022 *Utility rate database.* `https://apps.openei.org/USURDB/`, last accessed: 01.07.2022.

Parmentier A, Martinelli R, Vidal T, 2021 *Mobility-on-Demand with Electric Vehicles: Scalable Route and Recharging Planning through Column Generation. arXiv:2104.03823 [math]* .

Pelletier S, Jabali O, Laporte G, 2018 *Charge scheduling for electric freight vehicles. Transportation Research Part B: Methodological* 115(115):246–269.

Pelletier S, Jabali O, Laporte G, Veneroni M, 2017 *Battery degradation and behaviour for electric vehicles: Review and numerical analyses of several models. Transportation Research Part B: Methodological* 103:158–187.

Reniers JM, Mulder G, Howey DA, 2019 *Review and performance comparison of mechanical-chemical degradation models for lithium-ion batteries. Journal of The Electrochemical Society* 166(14):A3189–A3200.

Rodriguez F, Hildermeier J, Jahn A, 2020 *Electrifying eu city logistics.* Technical report.

Sassi O, Oulamara A, 2014 *Simultaneous electric vehicles scheduling and optimal charging in the business context: Case study. IET Conference Proceedings*, volume 5, 6.3–6.3(1) (Institution of Engineering and Technology).

Sassi O, Oulamara A, 2016 *Electric vehicle scheduling and optimal charging problem: complexity, exact and heuristic approaches. International Journal of Production Research* 55(2):519–535.

Schiffer M, Klein PS, Walther G, Laporte G, 2021 *Integrated planning for electric commercial vehicle fleets: A case study for retail mid-haul logistics networks. European Journal of Operational Research* 291(3):944–960.

Schiffer M, Schneider M, Walther G, Laporte G, 2019 *Vehicle routing and location-routing with intermediate stops: A review. Transportation Science* 53(2):319–343.

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

Schneider M, Stenger A, Goeke D, 2014 *The electric vehicle-routing problem with time windows and recharging stations. Transportation Science* 48(4):500–520.

Stavropoulou F, 2022 *The Consistent Vehicle Routing Problem with heterogeneous fleet. Computers & Operations Research* 140:105644.

Taefi TT, 2016 *Viability of electric vehicles in combined day and night delivery: a total cost of ownership example in germany. European Journal of Transport & Infrastructure Research* 16(4):512–553.

Tremblay O, Dessaint L, Dekkiche A, 2007 *A generic battery model for the dynamic simulation of hybrid electric vehicles. 2007 IEEE Vehicle Power and Propulsion Conference*, 284–289.

van Kooten Niekerk ME, van den Akker JM, Hoogeveen JA, 2017 *Scheduling electric vehicles. Public Transport* 9(1):155–176.

Wen M, Linde E, Ropke S, Mirchandani P, Larsen A, 2016 *An adaptive large neighborhood search heuristic for the Electric Vehicle Scheduling Problem. Computers & Operations Research* 76:73–83.

Wolsey LA, 1998 *Integer programming*, volume 52 (John Wiley & Sons), 1st edition.

Yao E, Liu T, Lu T, Yang Y, 2020 *Optimization of electric vehicle scheduling with multiple vehicle types in public transport. Sustainable Cities and Society* 52.

Zang D, Ling J, Wei Z, Tang K, Cheng J, 2019 *Long-term traffic speed prediction based on multiscale spatio-temporal feature learning network. IEEE Transactions on Intelligent Transportation Systems* 20:3700–3709.

Zang Y, Wang M, Qi M, 2022 *A column generation tailored to electric vehicle routing problem with nonlinear battery depreciation. Computers & Operations Research* 137:105527.

Appendix A: Counterexample

Straightforwardly adopting the labeling algorithms proposed in Baum et al. (2019) and Froger et al. (2019) to our problem setting yields an algorithm with a slightly modified label representation. Specifically, cost profiles track only the *last* visited station in these works. Accordingly, intermediate charging is not possible such that realizing a charging opportunity at i when propagating ℓ along (i,j) creates new labels according to the station replacement operation (Section 4.3.5) only. Specifically, the algorithm creates a set of new labels $\{\ell \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c \mid \forall c \in \mathcal{B}(\psi_\ell)\} \cup \{\ell \underset{(i,j)}{\leftarrow} /\}$ based on the breakpoints of cost profile ψ_ℓ at target vertex j.

The following example illustrates a case where this approach does not yield an optimal solution: consider a planning horizon of two periods $\mathcal{P}_1, \mathcal{P}_2$ of duration $\xi = 5$ with energy prices $e_{\mathcal{P}_1} = 10c$ and $e_{\mathcal{P}_2} = c$, a single service operation with consumption q, and a single charger f with constant charging rate $\frac{q}{8}$. For the sake of simplicity, we ignore battery degradation in this example and assume zero fixed costs for each arc in the time-expanded network. The optimal path through this network spends $3 \cdot 10c$ on charging at vertex $(\mathcal{P}_1, f)$ and $5 \cdot c$ at $(\mathcal{P}_2, f)$, such that the minimal cost is $3 \cdot 10c + 5 \cdot c$. According to (Baum et al. 2019, Froger et al. 2019), a visit to vertex $(\mathcal{P}_1, f)$ generates a single label with cost profile $\psi_{\ell_1}(c) = \frac{c}{10} \forall c \in [0,5]$. Realizing the second charging opportunity at vertex $(\mathcal{P}_2, f)$ then creates two labels, one for each breakpoint of ψ_{ℓ_1}, i.e., $\ell_2 := \ell_1 \underset{(i,j)}{\overset{f(i)}{\leftarrow}} 0$ and $\ell_3 := \ell_1 \underset{(i,j)}{\overset{f(i)}{\leftarrow}} 5 \cdot 10c$. Here, ℓ_2 is infeasible as $q_{\max}(\psi_{\ell_2}) = \frac{5}{8} q < q$ and ℓ_3 is not optimal: $\psi_{\ell_3}^{-1}(q) = 5 \cdot 10c + 3c > 3 \cdot 10c + 5c$.

Appendix B: Proofs

PROPOSITION 4. *The slope of a cost profile $\psi_{\ell'}$ obtained from $\ell' := \ell \underset{(i,j)}{\leftarrow} \tau$ can only be negative at $c \in [c_{\min}(\psi_{\ell'}), c_{\max}(\psi_{\ell'})]$ if for $q' := \Phi_{f(i)}(\Phi_{f(i)}^{-1}(\psi_{\ell'}^{-1}(c)) - \tau)$, $\angle_{\psi_\ell^{-1}}(q') < \angle_{\psi_{i,j}^{-1}}(q')$ holds.*

Proof: we first observe that Term 12.1 is never negative as $\Phi_{f(i)}$ is piecewise linear and concave, which, together with $\Phi_{f(i)}^{-1}(q) - \tau \leq \Phi_{f(i)}^{-1}(q)$, implies $\angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q) - \tau) \geq \angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q))$:

$$\angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q) - \tau) \cdot \angle_{\Phi_{f(i)}^{-1}}(q) = \frac{\angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q) - \tau)}{\angle_{\Phi_{f(i)}}(\Phi_{f(i)}^{-1}(q))} \geq 1.$$

As $\angle_{\psi_{i,j}^{-1}}$ is non-negative, we have $\angle_{\psi_{\ell'}^{-1}}(q') < 0$, which implies $\angle_{\psi_\ell^{-1}}(q') < \angle_{\psi_{i,j}^{-1}}(q')$. □

PROPOSITION 5. *For any c on a decreasing segment $[c', c' + \epsilon]$ with $\epsilon > 0$ of some cost profile $\psi_{\ell'}$ obtained from $\ell' := \ell \underset{(i,j)}{\leftarrow} \tau$, there exists some $\ell'' \in \mathcal{L}_j \setminus \{\ell'\}$ such that $\psi_{\ell''}(c) \geq \psi_{\ell'}(c)$.*

Proof: let there be some c where $\psi_{\ell'}$ is decreasing. Further let $q := \psi_{\ell'}^{-1}(c)$. Since $\psi_{\ell'}$ is decreasing on $[c', c' + \epsilon]$, we have $c_{\min}(\psi_{\ell'}) < c < c_{\max}(\psi_{\ell'})$ such that there exists some $\ell'' := \ell \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c''$ with $q_{\max}(\psi_{\ell''}) = q$. As we have $\angle_{\psi_\ell^{-1}}(q') < \angle_{\psi_{i,j}^{-1}}(q')$ for $q' := \psi_{\ell''}^{-1}(c'')$ (Proposition 4), we can apply the same reasoning as in the proof of Theorem 2 (Case 2.2) and argue that there exists some set of labels $\mathcal{L}' \subseteq \mathcal{L}_j \setminus \{\ell'\}$ such that $\mathcal{L}' \geq \psi_{\ell'}(c)$.

$\square$

THEOREM 2. *For any charging arc (i,j) and label $\ell_i \in \mathcal{L}_i$, the set of labels*

$$\mathcal{L}' := \{\ell_j, \ell_i \underset{(i,j)}{\leftarrow} \xi\} \cup \{\ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c \mid \forall c \in \mathcal{B}(\psi_{\ell_i})\},$$

with $\ell_j := \ell_i \underset{(i,j)}{\leftarrow} /$ dominates any label

$$\ell' \in \{\ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c \mid c \in [c_{\min}(\psi_{\ell_i}), c_{\max}(\psi_{\ell_i})]\} \cup \{\ell_i \underset{(i,j)}{\leftarrow} \tau \mid 0 \leq \tau \leq \xi\} \setminus \mathcal{L}'.$$

Proof: assume the contrary, i.e., that there exists some ℓ' not dominated by $\mathcal{L}'$. Then there exists some $c \in \mathbb{R}$ such that $\psi_{\ell'}(c) > \max_{\ell'' \in \mathcal{L}'}\{\psi_{\ell''}(c)\}$.

Case 1: $c < c_{\min}(\psi_{\ell'})$.

Then $\psi_{\ell'}(c) = -\infty \leq \max_{\psi_{\ell''} \in \mathcal{L}'}\{\ell''(c)\}$ holds by definition.

Case 2: $c_{\min}(\psi_{\ell'}) \leq c \leq c_{\max}(\psi_{\ell'})$.

W.l.o.g., we assume $\ell' = \ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c'$ for some c' and let $c'_j := c' + c_{i,j}$ and $q' := \psi_{\ell_i}(c') = \psi_{\ell_j}(c'_j)$. Recall from Section 4.3.5 that

$$\begin{aligned}
\psi_{\ell'}(c) &= \psi_{\ell_i}(c') + c_{i,j} + \Delta c^{-1}_{p(i),f(i)}\langle q' \rangle (c - c_{i,j} - c') \\
&= \psi_{\ell_j}(c'_j) + \Delta c^{-1}_{p(i),f(i)}\langle q' \rangle (c - c_{i,j} - c') \\
&= \psi_{\ell_j}(c'_j) + \Delta c^{-1}_{p(i),f(i)}\langle q' \rangle (c - c'_j).
\end{aligned}$$

Case 2.1: $\angle_{\psi_{\ell_j}}(c'_j) \geq \angle_{\psi_{\overrightarrow{i,j}}}(c'_j)$.

Let $\ell'' := \ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c''$, with $c'' = \min\{c \mid c \in \mathcal{B}(\psi_{\ell_i}) \wedge c \geq c'\}$, and let $c''_j := c'' + c_{i,j}$ such that

$$\begin{aligned}
\psi_{\ell''}(c) &= \psi_{\ell_i}(c') + c_{i,j} + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - c_{i,j} - c'') \\
&= \psi_{\ell_j}(c''_j) + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - c_{i,j} - c'') \\
&= \psi_{\ell_j}(c''_j) + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - c''_j).
\end{aligned}$$

We argue that ℓ'' exists and $\ell'' \in \mathcal{L}'$ holds since $c' \in [c_{\min}(\psi_{\ell_i}), c_{\max}(\psi_{\ell_i})]$.

Case 2.1.1: $c_{\min}(\psi_{\ell'}) \leq c \leq c_{\min}(\psi_{\ell''})$.

Then we have $c_{\min}(\psi_{\ell_j}) \leq c < c_{\min}(\psi_{\ell''}) \leq c_{\max}(\psi_{\ell_j})$. From the definition of ℓ'', it further follows that $\angle_{\psi_{\ell_j}}$ is constant on $[c'_j, c]$. Hence, we get

$$\psi_{\ell_j}(c) = \psi_{\ell_j}(c'_j) + \psi_{\ell_j}(c) - \psi_{\ell_j}(c'_j)$$

$$
\begin{aligned}
&= \psi_{\ell_j}(c'_j) + \angle_{\psi_{\ell_j}}(c'_j) \cdot (c - c'_j) \\
&\overset{(\star)}{\geq} \psi_{\ell_j}(c'_j) + \angle_{\psi_{i,j}}(c'_j) \cdot (c - c'_j) \\
&= \psi_{\ell'}(c),
\end{aligned}
$$

which contradicts our assumption. Here, $(\star)$ holds as $\angle_{\psi_{\ell_j}}(c'_j) \geq \angle_{\psi_{\overrightarrow{i,j}}}(c'_j)$.

Case 2.1.2 : $c_{\min}(\psi_{\ell''}) < c \leq c_{\max}(\psi_{\ell'})$.

We note that $\Delta q := \psi_{\ell''}(c''_j) - \psi_{\ell'}(c''_j) \geq 0$ holds by Case 2.1.1, such that it holds that $\Delta c := \psi_{\ell'}^{-1}(q'') - c''_j \geq 0$. Hence, for $c < c''_j + \Delta c$, we have $\psi_{\ell''}(c) \geq \psi_{\ell'}(c)$ since $\Delta c^{-1}_{p(i),f(i)}\langle q \rangle(\cdot)$ are concave for $q \in [\mathtt{SoC}_{\min}, \mathtt{SoC}_{\max}]$ (cf. Section 4.3.5). Using the same argument for inequality $(\star)$, we get the following for $c \geq c''_j + \Delta c$:

$$
\begin{aligned}
\psi_{\ell'}(c) &= q' + \Delta c^{-1}_{p(i),f(i)}\langle q' \rangle (c - c'_j) \\
&= q'' + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - (c''_j + \Delta c)) \\
&\overset{(\star)}{\leq} q'' + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - c''_j) \\
&= \psi_{\ell''}(c),
\end{aligned}
$$

which contradicts the assumption. See Figure 11a for an illustration.

Case 2.2: $\angle_{\psi_{\ell_j}}(c'_j) < \angle_{\psi_{\overrightarrow{i,j}}}(c'_j)$.

Let $\ell'' := \ell_i \overset{f(i)}{\underset{(i,j)}{\leftarrow}} c''$, with $c'' = \max\{c \mid c \in \mathcal{B}(\psi_{\ell_i}) \wedge c \leq c'\}$, and let $c''_j := c'' + c_{i,j}$ such that $\psi_{\ell''}(c) = \psi_{\ell_j}(c''_j) + \Delta c^{-1}_{p(i),f(i)}\langle q'' \rangle (c - c''_j)$.

Case 2.2.1: $\psi_{\ell'}(c) \leq q_{\max}(\psi_{\ell''})$.

It follows from the definition of ℓ'' that $\angle_{\psi_{\ell_j}}$ is constant on $[c'_j, c''_j]$. Then, analogous to Case 2.1.1, we have

$$
\begin{aligned}
\psi_{\ell''}(c'_j) &= \psi_{\ell''}(c''_j) + \psi_{\ell''}(c'_j) - \psi_{\ell''}(c''_j) \\
&\overset{(\star)}{\geq} \psi_{\ell''}(c''_j) + \angle_{\psi_{\overrightarrow{i,j}}}(c'_j) \cdot (c'_j - c''_j) \\
&\overset{(\star\star)}{\geq} \psi_{\ell''}(c''_j) + \angle_{\psi_{\ell_j}}(c'_j) \cdot (c'_j - c''_j) \\
&\overset{(\star\star\star)}{=} \psi_{\ell'}(c''_j) + \angle_{\psi_{\ell_j}}(c'_j) \cdot (c'_j - c''_j) \\
&= \psi_{\ell'}(c'_j).
\end{aligned}
$$

Here, $(\star)$ follows from the concavity of $\psi_{\overrightarrow{i,j}}$, i.e., $\angle_{\psi_{\overrightarrow{i,j}}}(c''_j) \geq \angle_{\psi_{\overrightarrow{i,j}}}(c'_j)$, $(\star\star)$ follows from $\angle_{\psi_{\ell_j}}(c'_j) < \angle_{\psi_{i,j}}(c'_j)$, and $(\star\star\star)$ holds since $\angle_{\psi_{\ell_j}}$ is constant on $[c''_j, c'_j]$ by the definition of c'_j and c''_j.

Concluding, $\Delta q := \psi_{\ell''}(c'_j) - q' \geq 0$ holds, which implies $\Delta c := \psi_{\ell'}^{-1}(\psi_{\ell''}(c'_j)) - \psi_{\ell'}^{-1}(q') \geq 0$, such that, analogous to Case 2.1.2, we get $\psi_{\ell''}(c) \geq \psi_{\ell'}(c)$ for $c + \Delta c < c'_j$ due to the concavity of $\Delta c_{p(i),f(i)}^{-1}\langle q\rangle(\cdot)$ (cf. Section 4.3.5). Furthermore, for $c + \Delta c \geq c'_j$, it holds that

$$\begin{aligned}
\psi_{\ell''}(c) &= q'' + \Delta c_{p(i),f(i)}^{-1}\langle q''\rangle(c - c'_j) \\
&= q' + \Delta c_{p(i),f(i)}^{-1}\langle q'\rangle(c - c'_j + \Delta c) \\
&\geq q' + \Delta c_{p(i),f(i)}^{-1}\langle q'\rangle(c - c'_j) \\
&= \psi_{\ell'}(c),
\end{aligned}$$

which contradicts the assumption. See Figure 11b for an illustration.

Case 2.2.2: $\psi_{\ell'}(c) > q_{\max}(\psi_{\ell''})$.

Let $\ell_\xi := \ell_i \underset{(i,j)}{\leftarrow} \xi$ and $q := \psi_{\ell'}(c)$. Further let $\ell''' := \ell_i \underset{(i,j)}{\overset{f(i)}{\leftarrow}} c'''$ such that $q_{\max}(\psi_{\ell'''}) = q$. The existence of ℓ''' follows straightforwardly from $c \leq c_{\max}(\psi_{\ell'})$ and the existence of ℓ'. Let $c'''_j := c''' + c_{i,j}$ and $q''' := \psi_{\ell'''}(c'''_j)$. Note that $c'' \leq c''' \leq c'$, and thus $c''_j \leq c'''_j \leq c'_j$, since the definition of c'' implies that $\angle_{\psi_{\ell_i}}$ is constant on $[c'', c']$. Hence, analogous to Case 2.2.1, we get $\psi_{\ell'''}(c) \geq \psi_{\ell'}(c)$. With this in mind, we recall from Equation 8 that

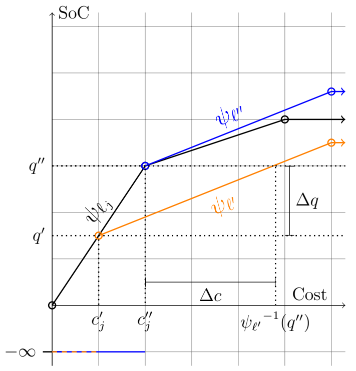
$$\begin{aligned}
\psi_{\ell'''}^{-1}(q) = \psi_{\ell'''}^{-1}(q_{\max}(\psi_{\ell'''})) &= c'''_j + \Delta c_{p(i),f(i)}\langle q'''\rangle(\Phi_{f(i)}(q''', \xi) - q''') \\
&= \psi_{\ell_j}^{-1}(q''') + \Delta c_{p(i),f(i)}\langle q'''\rangle(\Phi_{f(i)}(q''', \xi) - q''') \\
&= \psi_{\ell_i}^{-1}(q''') + c_{i,j} + \Delta c_{p(i),f(i)}\langle q'''\rangle(\Phi_{f(i)}(q''', \xi) - q''').
\end{aligned}$$

Recall that $q := \psi_{\ell'}(c) = \Phi_{f(i)}(q''', \xi)$, such that substituting q yields

$$\begin{aligned}
&\psi_{\ell_i}^{-1}(q''') + c_{i,j} + \Delta c_{p(i),f(i)}\langle q'''\rangle(\Phi_{f(i)}(q''', \xi) - q''') \\
&= \psi_{\ell_i}^{-1}(q''') + c_{i,j} + \Delta c_{p(i),f(i)}\langle q'''\rangle(q - q''') \\
&\overset{(\star)}{=} \psi_{\ell_i}^{-1}(q''') + c_{i,j} + (\Delta c_{p(i),f(i)}\langle 0\rangle(q) - \Delta c_{p(i),f(i)}\langle 0\rangle(q''')) \\
&= \psi_{\ell_i}^{-1}(q''') + c_{i,j} + (\psi_{i,j}^{-1}(q) - \psi_{i,j}^{-1}(q''')) \\
&= \psi_{\ell_\xi}^{-1}(q).
\end{aligned}$$

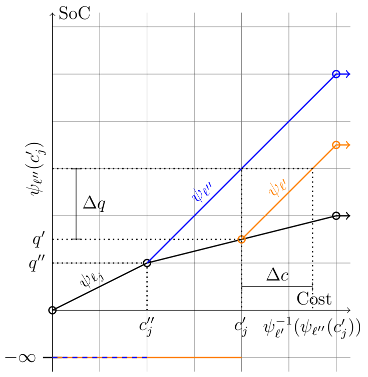
Here, $(\star)$ holds since

$$\begin{aligned}
\Delta c_{p(i),f(i)}\langle q'''\rangle(q - q''') &= e_p \cdot (q - q''') + \Upsilon(q''', q''' + q - q''') \\
&= e_p \cdot (q - q''') + \Upsilon(q''' + q - q''') - \Upsilon(q''')
\end{aligned}$$



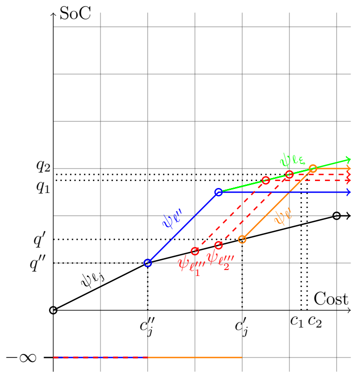
(a) $\angle_{\psi_{\ell_j}}(c_j') \geq \angle_{\psi_{i,j}}(c_j')$ (Case 2.1).

Note. Δc corresponds to the additional cost incurred by using vertex i to recharge up to the SoC at which ℓ'' stops charging using the charging opportunity captured by the previous label. As the previous charging opportunity provides a lower price, Δc is positive.



(b) $\angle_{\psi_{\ell_j}}(c_j') < \angle_{\psi_{i,j}}(c_j')$, $\psi_{\ell'}(c) \leq q_{\max}(\psi_{\ell''})$ (Case 2.2.1).

Note. Here, Δc corresponds to the cost saved by using i to charge up to the arrival SoC of ℓ' at j. Δc is positive as station i provides a lower charging price in this case.



(c) $\angle_{\psi_{\ell_j}}(c_j') < \angle_{\psi_{i,j}}(c_j')$, $\psi_{\ell'}(c) > q_{\max}(\psi_{\ell''})$ (Case 2.2.2).

Note. c_1 and c_2 provide example values for c. The red dashed lines illustrate the corresponding cost profiles $\psi_{\ell_1'''}$ and $\psi_{\ell_2'''}$, respectively. The green profile illustrates charging intermediately. Note that, for the sake of simplicity, we show only the segment of ψ_{ℓ_ξ} relevant to this proof.

Figure 11: Illustration of cases 2.1 and 2.2 in the proof of Theorem 2.

$$
\begin{aligned}
&= e_p \cdot (q - q''') + \Upsilon(q) - \Upsilon(q''') \\
&= (e_p \cdot q + \Upsilon(q) - \Upsilon(0)) - (e_p \cdot q''' + \Upsilon(q''') - \Upsilon(0)) \\
&= (e_p \cdot q + \Upsilon(0,q)) - (e_p \cdot q''' + \Upsilon(0,q''')) \\
&= \Delta c_{p(i),f(i)}\langle 0\rangle(q) - \Delta c_{p(i),f(i)}\langle 0\rangle(q''').
\end{aligned}
$$

Concluding, we have $\psi_{\ell_\xi}^{-1}(q) = \psi_{\ell'''}^{-1}(q)$ and $\psi_{\ell'''}(c) \geq \psi_{\ell'}(c)$ such that $\psi_{\ell_\xi}^{-1}(q) = \psi_{\ell'''}^{-1}(q) \leq \psi_{\ell'}^{-1}(q)$, which contradicts the assumption. See Figure 11c for an illustration.

Case 3: $c \geq c_{\max}(\psi_{\ell'})$.

Case 2 implies that $\exists \ell'' \in \mathcal{L}' : \psi_{\ell''}(c_{\max}(\psi_{\ell'})) \geq q_{\max}(\psi_{\ell'})$. Hence, as $\psi_{\ell'}(c) = q_{\max}(\psi_{\ell'})$ by definition, we get $\psi_{\ell''}(c_{\max}(\psi_{\ell'})) \geq q_{\max}(\psi_{\ell'})$, which contradicts the assumption.

Concluding, for any $c \in \mathbb{R}$, there exists a $\ell'' \in \mathcal{L}'$, such that $\psi_{\ell''}(c) \geq \psi_{\ell'}(c)$. Hence, $\max_{\ell'' \in \mathcal{L}'}\{\ell''(c)\} \geq \psi_{\ell'}(c)$, and Theorem 2 holds. □

Proposition 6. *Let σ be a fractional basic feasible solution to the RMP. Then it holds that $\exists (p,f) \in \mathcal{P} \times \mathcal{F}$ such that $\sum_{x_\omega^k \in \sigma} x_\omega^k \boldsymbol{A}_{p,f}^\omega = C_f$, and there exist at least two $x_\omega^k \in \sigma$ with $0 < x_\omega^k < 1$.*

Proof: Assume the contrary, i.e., that there exists schedules $\omega_1,\ldots,\omega_n$ in basic feasible $\sigma=(x_{\omega_1}^{k_1},\ldots,x_{\omega_n}^{k_n})$ with $0<x_\omega^k<1$, and $\sum_{x_\omega^k\in\sigma} x_\omega^k \boldsymbol{A}_{p,f}^\omega < C_f$. Let $k'\in\mathcal{K}$ such that $\exists \omega_i,\omega_j\in\mathcal{A}_{k'}$ with $i\neq j$ and $0<x_{\omega_i}^{k'}\leq x_{\omega_j}^{k'}<1$ in σ. Here, convexity Constraints (2c) ensure the existence of such ω_i and ω_j. As charger capacity Constraints (2b) are non-binding, there exists $\epsilon>0$ such that $\sigma'=(x_{\omega_1}^{k_1},\ldots,x_{\omega_i}^{k'}+\epsilon,\ldots,x_{\omega_j}^{k'}-\epsilon,\ldots,x_{\omega_n}^{k_n})$ is feasible. This implies that σ is not basic and thus contradicts the assumption. □

Appendix C: Fundamentals

The following sections detail the methodology behind and the derivation of charging functions Φ_f and WDF $\widetilde{\Upsilon}$: we first provide an overview of the electro-chemical fundamentals of the charging process and discuss a charging scheme that prevents overcharging and thus critically damaging the battery's internals in Section C.1. Afterwards, we show how to model this charging scheme in functions Φ_f (Section C.2). Finally, Section C.3 details how we capture battery health considerations in our optimization problem. For an in-depth review of battery modeling for ECVs in particular and electro-chemical cells in general, we refer to Pelletier et al. (2017) and Franco (2015).

C.1. The Constant Current-Constant Voltage charging scheme

A battery's capacity can be measured in several units: Ampere-hours, Coulombs, and kWh, each proper in different application cases. To avoid confusion arising from handling these technicalities in the remainder of this section, we use the concept of SoC, which expresses the battery's unit-independent state of charge relative to its nominal capacity, i.e., a SoC of 0% corresponds to an empty and a SoC of 100% to a full battery.

Most electric vehicles use Lithium-ion (Li-Ion) batteries for energy storage. These are commonly charged with a constant current - constant voltage (CC-CV) charging scheme to prevent critically damaging the battery's internals by overcharging. This process is best understood using the battery model developed in Tremblay, Dessaint, and Dekkiche (2007) and Zang et al. (2019), which is specifically tailored to ECV applications. Tremblay, Dessaint, and Dekkiche model the battery as a controlled voltage source in series with a resistor (see Figure 12), which allows expressing the dynamics of the charging process as:

$$V_{term}(\texttt{SoC}) := V_{OC}(\texttt{SoC}) + R\cdot I. \tag{15}$$

Equation (15) states the relationship between terminal voltage ($V_{term}(\texttt{SoC})$), which refers to the voltage measured across the battery's terminals during the charging process, the open-circuit voltage ($V_{OC}(\texttt{SoC})$), which corresponds to the terminal voltage in a disconnected state, and the charging current I. The battery's internal resistance is denoted by R and varies with several exogenous factors, e.g., temperature, load, and battery age. For the sake of simplicity, we assume this resistance to be constant (cf. Pelletier et al. (2017)).

The open circuit voltage, $V_{OC}(\texttt{SoC})$, increases (non-linearly) with the battery's SoC and is often used as an indicator of the battery's charge level. In practice, $V_{OC}(\texttt{SoC})$ is often approximated using Equation (16), parameterized with experimental data (cf. Marra et al. 2012, Pelletier et al. 2017):

$$V_{OC}(\texttt{SoC}(t)) := E_0 - \frac{K}{\texttt{SoC}(t)} + A\exp(-BQ(1-\texttt{SoC}(t))). \tag{16}$$

Here, E_0 denotes the battery's constant voltage, Q corresponds to the battery'sä capacity in ampere-hours, A and B are parameters, and K is the polarization voltage. To prevent damaging the battery's electrodes, the charging current I and the terminal voltage $V_{term}(\texttt{SoC})$ must remain within charger and battery-dependent bounds $I_{\max}$ and $V_{term}^{\max}$, respectively. Usually, the maximum terminal voltage $V_{term}^{\max}$ lies well below $V_{term}(100\%) = V_{OC}(100\%) + R \cdot I_{\max}$ and is hence exceeded before the battery is fully charged. To respect this voltage threshold, the CC-CV charging scheme is as follows: first the charging current I is held constant at $I_{\max}$ in the *CC* charging phase until $V_{term}(\texttt{SoC})$ reaches $V_{term}^{\max}$, at which point the *CV* charging phase begins. Here, the charging current I is steadily reduced such that $V_{term}(\texttt{SoC})$ remains at but does not exceed $V_{term}^{\max}$. The constant voltage (CV) phase ends when the charging current drops below manufacturer recommendations. The SoC evolution over time during CC-CV charging is thus non-linear and concave.

C.2. Modeling CC-CV charging

We capture the charger-specific non-linear behavior of the CC-CV charging process in charging functions $\Phi_f(\tau)$, which map the time spent charging at charger f to the resulting SoC when charging with an initially empty battery. In what follows, we describe how to derive $\Phi_f(\tau)$ from battery and charger specifications. Let $I_{\max}^f$ be the maximum charging current of charger f and τ_{CV}^f be the point in time where the CV phase of charger f begins. We can then express Φ_f using auxiliary functions

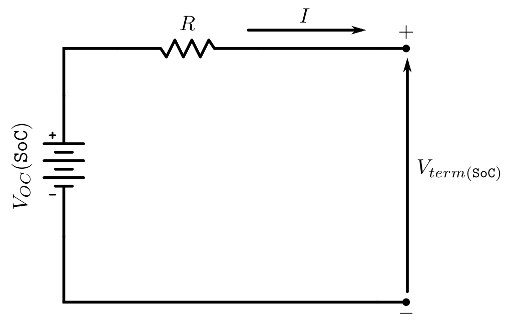


Figure 12: Circuit diagram of the battery model developed in Tremblay, Dessaint, and Dekkiche (2007)

$\Phi_f^{CC}(\tau)$ and $\Phi_f^{CV}(\tau)$, which correspond to the CC and CV phases of the charging process, respectively, in Equations (17a)-(17d):

$$\Phi_f^{CC}(\tau) := \Phi_f(0) + \frac{I_{\max}^f \cdot \tau}{Q}, \tag{17a}$$

$$V_{CC} := V_{OC}(\mathtt{SoC}(\tau)) + R \cdot 3600 \cdot Q \cdot \frac{\partial \Phi_f^{CV}(\tau)}{\partial \tau}, \tag{17b}$$

$$\frac{\partial \Phi_f^{CV}(\tau)}{\partial \tau} := \frac{V_{CC} - V_{OC}(\Phi_f^{CV}(\tau))}{R \cdot 3600 \cdot Q}, \tag{17c}$$

$$\widetilde{\Phi_f}(\tau) := \begin{cases} \Phi_f^{CC}(\tau) & \text{if } \tau \leq \tau_{CV}^f \\ \Phi_f^{CV}(\tau) & \text{otherwise.} \end{cases} \tag{17d}$$

Solving Equations (17a)-(17d), we obtain an accurate model of the charging process, which we can easily incorporate into our planning problem. We note that $\widetilde{\Phi_f}$ is concave for realistic charger models and, without loss of generality, extend the definitions of $\widetilde{\Phi_f}$ and $\widetilde{\Phi_f}^{-1}$ to avoid edge cases in the main body of this work. Equations (18) and (19) state the final definitions.

$$\Phi_f(\tau) := \begin{cases} \widetilde{\Phi_f} & \text{if } \tau \leq 0 \\ \widetilde{\Phi_f}(\tau_{\max}) & \text{if } \tau \geq \tau_{\max} \\ \widetilde{\Phi_f}(\tau) & \text{otherwise.} \end{cases} \tag{18}$$

$$\Phi_f^{-1}(\beta) := \begin{cases} 0 & \text{if } \beta \leq \widetilde{\Phi_f}(0) \\ \tau_{\max} & \text{if } \beta \geq \widetilde{\Phi_f}(\tau_{\max}) \\ \widetilde{\Phi_f}^{-1}(\beta) & \text{otherwise.} \end{cases} \tag{19}$$

Here, $\tau_{\max}$ represents the point in time at which the charging current falls below manufacturer recommendations. We assume that $\tau_{\max}$ is finite, such that there exists some $\tau_{\max} > 0$ with $\Phi_f(\tau_{\max}) = \mathtt{SoC}_{\max}$. Finally, we note that Φ_f are continuous functions and that the main body of this work, in line with Pelletier et al. (2017), Montoya et al. (2017), and Froger et al. (2019), instead uses Φ_f to refer to the piecewise linear approximation of Φ_f.

C.3. Battery degradation

The CC-CV charging scheme prevents critically damaging a battery through overcharging. However, overcharging is not the only cause of accelerated degradation. More precisely, the magnitude of battery degradation resulting from (dis-)charging depends on (endogenous) factors such as cycle depth, charging current, and residual SoC. Hence, different charge scheduling decisions have different impacts on battery life. In fact, there exists a trade-off between utilization and battery degradation, that an operator can use to her advantage. Capturing this trade-off requires quantifying the impact of charge scheduling decisions on battery health in an analytical model. Such models take one of two approaches: they either model battery wear based on the underlying electro-chemical processes, or pursue an empirical approach based on experimental data (cf. Reniers, Mulder, and Howey 2019). As data required to parameterize the former is often unavailable and may vary between individual cells, we rely on an empirical approach to quantify battery degradation. To this end, we follow the approach from Pelletier, Jabali, and Laporte (2018) and base our model on the work of Han, Han, and Aki (2014).

Han, Han, and Aki (2014) relate battery price to cycle life specifications supplied by manufacturers, specifically to the depth of discharge - achievable cycle count (DOD-ACC) curve. Each point on the DOD-ACC curve, $ACC(D)$, corresponds to the number of cycles achievable before the battery becomes unusable when it is cycled at the respective depth of discharge (DoD). For instance, $ACC(20\%) = 2500$ indicates that the battery can be discharged from 100% to 80% and then recharged back to 100% SoC 2500 times before capacity and power fade render it ineffective.

The DOD-ACC function establishes a relationship between battery life and price: dividing the battery price by the total energy transferred over $ACC(D)$ cycles gives the average wear cost of (dis-)charging when cycling the battery at a DoD of D, denoted $AWC(D)$. However, the average wear cost function is only of limited use as it is only valid if the battery is always cycled in the same fashion, i.e., from 100% SoC to $1-D$ and back to 100%. Han, Han, and Aki (2014) address this issue by combining n (equidistant) points on the DOD-ACC curve according to the following methodology: let $S := [S_1, \ldots, S_n]$ be the SoC values corresponding to the given DOD-ACC points and let

$$Q(s) := ACC(1-s) \cdot 2 \cdot (1-s) \cdot C \tag{20}$$

denote the total amount of energy transferred when cycling a battery with capacity C to a SoC of s. For the highest SoC segment S_n, the battery price c_{bat} must equal the wear cost $\widetilde{\Upsilon}(S_n)$ of energy charged on the segment $[S_n, 100\%]$, multiplied by the total amount of energy transferred over the battery's lifespan when cycling at $D = 1 - S_n$. Equation (21) formalizes this relationship:

$$c_{\text{bat}} = Q(S_n) \cdot \widetilde{\Upsilon}(S_n). \tag{21}$$

Each charging cycle at a DoD of $1 - S_{n-1}$ also cycles the battery at $1 - S_n$. Hence, the average wear cost must be at least $\widetilde{\Upsilon}(S_n)$. Formally, we have $AWC(1 - S_{n-1}) = \widetilde{\Upsilon}(S_n) + \epsilon$ for some $\epsilon > 0$. We thus have $AWC(1 - S_i) = \sum_{j=i}^{n} \widetilde{\Upsilon}(S_j)$ by induction, i.e., the average wear cost corresponds to the sum of wear costs incurred on all utilized segments. Hence, we can generalize Equation (21) to Equation (22):

$$c_{\text{bat}} = Q(S_i) \cdot (\sum_{j=i}^{n} \widetilde{\Upsilon}(S_j)). \tag{22}$$

Equation (22) yields a linear equation system of size n, which can be solved for $\widetilde{\Upsilon}(s)$ at discrete points $s \in \{S_1, \ldots, S_n\}$ (cf. Han, Han, and Aki 2014). Setting $\widetilde{\Upsilon}(s) := \widetilde{\Upsilon}(S_i)$ for $s \in [S_i, S_{i+1}], 0 \leq i < n$ finally yields a piecewise constant function, called the *wear density function*, which gives the unit cost of charging at a certain SoC. We utilize this function to compute the *(cumulative) wear density function*, denoted by $\Upsilon : \texttt{SoC} \mapsto \texttt{cost}$. Specifically, we obtain Υ by integrating $\widetilde{\Upsilon}$, which yields a convex piecewise linear function stating the total cost of charging an initially empty battery up to a certain SoC.

Appendix D: Implementation details

Algorithm 1: Label setting search

```
1  Initialization:
2  $\mathcal{L}^{uns}_{s^-} := \{\ell_{s^-}\}$;
3  $\mathcal{Q} := \{s^-\}$;
4  while $\mathcal{Q}$.notEmpty() do
5      $i := \mathcal{Q}$.pop();
6      $\ell :=$ extract_min($\mathcal{L}^{uns}_i$);
7      if $i = s^+$ then
8          return $c_{\min}(\psi_\ell)$;
9      if $\exists \ell' \in \mathcal{L}^{set}_i, \ell' \succeq \ell$ then
10         continue;
11     for $(i,j) \in \mathcal{E}^k$ do
12         if $(i,j) \notin \mathcal{E}^k_{\mathcal{F}}$ then
13             $\mathcal{L}_{new} := \{\ell \underset{(i,j)}{\leftarrow} /\}$;
14         else
               // Track charging decisions in period $p(i)$
15             $\mathcal{L}_{new} := \{\ell \overset{f(i)}{\underset{(i,j)}{\leftarrow}} c \mid c \in \mathcal{B}(\psi_\ell)\}$;
               // Charge for $\tau = \xi$ in $p(i)$
16             $\mathcal{L}_{new} := \mathcal{L}_{new} \cup \{\ell \underset{(i,j)}{\leftarrow} \xi\}$;
17         remove_set_dominated($\mathcal{L}_{new}$);
           // Enqueue feasible labels
18         for $\ell' \in \mathcal{L}_{new}$ do
19             if feasible($\ell'$) then
20                 insert($\mathcal{L}^{uns}_j, \ell'$);
21     insert($\mathcal{L}^{set}_i, \ell$);
22 return infeasible;
```

Algorithm 1 details our label-setting search procedure. The algorithm relies on several data structures: first, it maintains a set of settled ($\mathcal{L}^{set}_v$) and unsettled ($\mathcal{L}^{uns}_v$) labels for each vertex $v \in \mathcal{V}^k$. These keep track of already developed paths and collect candidates for expansion, respectively. To establish the label-setting property of our algorithm, we store vertices v with unsettled labels, i.e., potential candidates for expansion, in a priority queue $\mathcal{Q}$. This queue orders vertices according to the cost of the cheapest unsettled label at the respective vertex, formally, $\min_{\ell \in \mathcal{L}_v^{uns}} c_{\min}(\psi_\ell) \geq \min_{\ell \in \mathcal{L}_{v'}^{uns}} c_{\min}(\psi_\ell) \Rightarrow v \succeq_{\mathcal{Q}} v'$ for $v, v' \in \mathcal{Q}$. We break ties according to the vertex's period index in descending order such that we prefer vertices closer to the sink node.

We initialize $\mathcal{Q}$ with the source vertex and root label (Lines 2-3). The main body of the algorithm (Lines 4-21) iteratively extracts labels from $\mathcal{Q}$ (Lines 5-6) and propagates these along all adjacent arcs (Lines 11-16). We propagate extracted labels $\ell \in \mathcal{L}_i$ along idle and service arcs (i, j) according to $c_{i,j}$, $q_{i,j}$, and $\vartheta_{i,j}$. Charging arcs require special treatment: here, it is possible to either commit to a charging decision at a previously visited station vertex i' to then track charging trade-offs at station vertex i, or to commit to charging at i, continuing to track decisions at i'. Line 15 handles the former case and creates a label for each non-dominated charging decision at the station vertex i' tracked by ℓ so far, i.e., fixes the amount of charge replenished at i', and thus the arrival SoC at i to some value. Line 16 handles the latter case, i.e., spawns a label that charges at i *without* forcing a decision at the tracked station i'. This fixes the amount of time spent charging at i to some value. Note that ignoring the charging opportunity at i corresponds to visiting the respective garage vertex and is thus not explicitly considered. We prune the set of generated labels according to our set-based dominance criterion (cf. Definition 2) in Line 17.

Lines 18-20 insert feasible labels into $\mathcal{L}_j^{uns}$, i.e., track potential candidates for expansion at vertex j, updating the vertex queue accordingly. Finally, we settle the original label at vertex i for future dominance checks. The algorithm terminates when a label is extracted at the sink (Line 8), or no unsettled labels remain.

We implement $\mathcal{L}_i^{uns}$ as a min-heap and key labels by minimum cost. We postpone pairwise dominance checks against already settled labels to label extraction (Line 9). This lazy approach serves two purposes: first, it avoids superfluous dominance checks for labels never considered during the search; second, it delays dominance checks as much as possible to maximize the number of candidates for domination.

We further rely on two techniques to speed up dominance checks: first, we keep $\mathcal{L}_i^{set}$ sorted by maximum reachable SoC. This allows skipping superfluous dominance checks against settled labels with a lower maximum SoC. Second, we maintain a hash table of settled and unsettled labels at each vertex. We probe this hash table and abort the dominance check if an equivalent label is found. These strategies minimize the number of (explicit) dominance checks required to maintain our dominance invariant.

$\mathcal{V}^k$	set of vertices
$\mathcal{V}^k_{\mathcal{F}}$	set of charger vertices
$\mathcal{V}^k_f$	set of vertices associated with charger f
$\delta^+(v)$	set of incoming arcs at vertex v
$\delta^-(v)$	set of outgoing arcs at vertex v
$\mathcal{E}^k_{\mathcal{F}}$	set of charging arcs
$\mathcal{E}^k_{\Theta_k}$	set of service arcs
e_p	energy cost in period p
$q_{i,j}$	charge consumption of arc (i,j)
$c_{i,j}$	cost of arc (i,j)
$\mathcal{F}$	set of chargers
C_f	charger capacity for $f \in \mathcal{F}$
$\mathcal{B}(f)$	breakpoints of the linearized charging function
$q_{f,b}$	SoC associated with breakpoint $b \in \mathcal{B}(f)$
$t_{f,b}$	Time associated with breakpoint $b \in \mathcal{B}(f)$
$\mathcal{B}(\Upsilon)$	breakpoints of the (cumulative) wear density function (WDF)
$q_{\Upsilon,b}$	SoC associated with breakpoint $b \in \mathcal{B}(\Upsilon)$
$c_{\Upsilon,b}$	Costs per kWh associated with breakpoint $b \in \mathcal{B}(\Upsilon)$
$\mathcal{P}$	set of periods in the planning horizon
$\mathcal{K}$	set of vehicles
$\texttt{SoC}_{\max}$	maximum battery charge level (SoC)
$\texttt{SoC}_{\min}$	minimum battery charge level (SoC)
β^k_v	SoC with which vehicle k arrives at vertex v
γ^k_v	SoC charged/discharged at vertex v
ρ^k_v	battery wear cost incurred by charging at v
$x^k_{i,j}$	binary variable, indicating whether vehicle k traverses arc (i,j) ($x^k_{i,j}=1$) or not ($x^k_{i,j}=0$)
$\lambda^{k,in}_{v,b}$	Convex multipliers binding entry SoC to $\Phi_{f(v)}$
$\lambda^{k,out}_{v,b}$	Convex multipliers binding exit SoC to $\Phi_{f(v)}$
$\mu^{k,in}_{v,b}$	Convex multipliers binding entry SoC to Υ
$\mu^{k,out}_{v,b}$	Convex multipliers binding exit SoC to Υ

Table 6: Parameters and variables of the compact formulation.

Appendix E: Compact formulation

In the following, we model our planning problem as a mixed integer program, which we state as a shortest path problem on the time-expanded network presented in Section 4.3.1. MIP 1 comprises the following decision variables: binary variable $x^k_{i,j}$ indicates that vehicle k traverses arc (i,j). Variables $\lambda^{k,in}_{v,b}$ and $\lambda^{k,out}_{v,b}$ model charging operations as piecewise linear functions. To this end, continuous variables $\lambda^{k,in}_{v,b}$ and $\lambda^{k,out}_{v,b}$ are convex multipliers associated with the breakpoints of the respective piecewise linear charging functions, i.e., give the contribution of each breakpoint to the function value. We establish a Special-Ordered Set of Type 2 (SOS2) relationship between variables associated with the same vertex and vehicle. Variables $\mu^{k,in}_{v,b}$ and $\mu^{k,out}_{v,b}$ model the WDF analogously. We use continuous variables β^k_v and γ^k_v to track the arrival SoC and total amount of energy replenished by vehicle k at vertex $v \in \mathcal{V}^k$. With the notation summarized in Table 6, our MIP is as follows.

$$\min \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}^k_{\mathcal{F}}} \gamma^k_v \cdot e_{p(v)} + \rho^k_v \tag{1.1}$$

$$\sum_{(i,j)\in\delta^-(s^-)} x^k_{i,j} = 1 \qquad k\in\mathcal{K} \tag{1.2}$$

$$\sum_{(i,j)\in\mathcal{E}^k_{\vartheta}} x^k_{(i,j)} \geq 1 \qquad \vartheta\in\Theta_k, k\in\mathcal{K} \tag{1.3}$$

$$\sum_{(i,j)\in\delta^+(v)} x^k_{i,j} - \sum_{(i,j)\in\delta^-(v)} x^k_{i,j} = 0 \qquad v\in\mathcal{V}^k\setminus\{s^-,s^+\}, k\in\mathcal{K} \tag{1.4}$$

$$\beta^k_i + q_{i,j} + \gamma^k_i \geq \beta^k_j - (1-x^k_{i,j})\cdot \mathtt{SoC}_{\max} \qquad \forall (i,j)\in\mathcal{E}^k, k\in\mathcal{K} \tag{1.5}$$

$$\beta^k_i + q_{i,j} + \gamma^k_i \leq \beta^k_j + (1-x^k_{i,j})\cdot \mathtt{SoC}_{\max} \qquad \forall (i,j)\in\mathcal{E}^k, k\in\mathcal{K} \tag{1.6}$$

$$\mathtt{SoC}_{\min} \leq \beta^k_v \leq \mathtt{SoC}_{\max} \qquad v\in\mathcal{V}^k, k\in\mathcal{K} \tag{1.7}$$

$$\beta^k_{s^-} = \mathtt{SoC}_{\min} \qquad k\in\mathcal{K} \tag{1.8}$$

$$\sum_{b\in\mathcal{B}(f(v))} \lambda^{k,in}_{v,b}\cdot q_{f(v),b} = \beta^k_v \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.9}$$

$$\sum_{b\in\mathcal{B}(f(v))} \lambda^{k,in}_{v,b} = 1 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.10}$$

$$\sum_{b\in\mathcal{B}(f(v))} \lambda^{k,out}_{v,b}\cdot q_{f(v),b} - \sum_{b\in\mathcal{B}(f(v))} \lambda^{k,in}_{v,b}\cdot q_{f(v),b} = \gamma^k_v \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.11}$$

$$\sum_{b\in\mathcal{B}(f(v))} \lambda^{k,out}_{v,b} = 1 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.12}$$

$$\sum_{b\in\mathcal{B}(f(v))} \lambda^{k,out}_{v,b}\cdot t_{f(v),b} - \sum_{b\in\mathcal{B}(f(v))} \lambda^{k,in}_{v,b}\cdot t_{f(v),b} \leq \xi \cdot \sum_{(i,j)\in\delta^-(v)} x^k_{i,j} \qquad v\in\mathcal{V}^k_{f}, k\in\mathcal{K} \tag{1.13}$$

$$\sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,in}_{v,b}\cdot q_{\Upsilon,b} = \beta^k_v \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.14}$$

$$\sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,in}_{v,b} = 1 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.15}$$

$$\sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,out}_{v,b}\cdot q_{\Upsilon,b} - \sum_{b\in\mathcal{B}(f(i))} \mu^{k,in}_{v,b}\cdot q_{\Upsilon,b} = \gamma^k_v \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.16}$$

$$\sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,out}_{v,b} = 1 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.17}$$

$$\sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,out}_{v,b}\cdot c_{\Upsilon,b} - \sum_{b\in\mathcal{B}(\Upsilon)} \mu^{k,in}_{v,b}\cdot c_{\Upsilon,b} = \rho^k_v \qquad v\in\mathcal{V}^k_{f}, k\in\mathcal{K} \tag{1.18}$$

$$\sum_{k\in\mathcal{K}} \sum_{(i,j)\in\delta^-(v)} x^k_{i,j} \leq C_{f(v)} \qquad \forall v\in\bigcup_{k\in\mathcal{K}} \mathcal{V}^k_{\mathcal{F}} \tag{1.19}$$

$$x^k_{i,j}\in\{0,1\} \qquad (i,j)\in\mathcal{E}, k\in\mathcal{K} \tag{1.20}$$

$$\beta^k_v, \gamma^k_v, \rho^k_v \geq 0 \qquad v\in\mathcal{V}^k, k\in\mathcal{K} \tag{1.21}$$

$$\forall b\in\mathcal{B}(f(v)) : \lambda^{k,in}_{v,b}, \lambda^{k,out}_{v,b} \in SOS2 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.22}$$

$$\forall b\in\mathcal{B}(\Upsilon) : \mu^{k,in}_{v,b}, \mu^{k,out}_{v,b} \in SOS2 \qquad v\in\mathcal{V}^k_{\mathcal{F}}, k\in\mathcal{K} \tag{1.23}$$

The objective function (1.1) minimizes the total cost of the charging schedule, i.e., the sum of energy costs and battery wear incurred. Constraints (1.2) and (1.3) enforce an outgoing and incoming arc at the source and sink nodes, respectively. Constraints (1.4) set up flow conservation on all other vertices. Constraints (1.5)-(1.6) propagate the SoC. The concaveness of the WDF requires strict equality. Consumption and charging operations are captured with $q_{i,j}$ and γ_i^k, respectively. Constraints (1.7) ensure that manufacturer SoC bounds are respected. Constraints (1.8) initialize $\beta_{s^-}^k$. Constraints (1.9-1.12) model charging operations as piecewise linear functions. Constraints (1.13) limit the maximum SoC rechargeable at station nodes, i.e., ensure that the charging rate is respected, and establish a link between $x_{i,j}^k$ and γ_v^k, such that charging can occur only if the station node is visited. Constraints (1.14)-(1.18) model battery degradation similarly. Constraints (1.19) limit the number of simultaneous charging operations at each charger. Finally, Constraints (1.20)-(1.23) state the domain of the decision variables and establish SOS2 sets.

Appendix F: Benchmark instance generation

We choose a discretization step size of 30 minutes and draw period energy prices uniformly from the interval $[0.5, 1.0]$ (€). We further assume a battery capacity of 80 kWh with $\texttt{SoC}_{\min} := 0\%$ and $\texttt{SoC}_{\max} := 100\%$. To avoid biases in our numerical study, we generate charging functions Φ_f and the WDF randomly according to the following procedure: Given parameters n, $\chi_{\min}, \chi_{\max}$, and ν, which correspond to the number of segments, minimum slope, maximum slope, and upper bound respectively, we assign a random weight w_i, and a slope drawn from interval $[\chi_{\min}, \chi_{\max}]$ to each segment i. We then sort the segments by slope in descending (ascending) order such that the resulting function is convex (concave) for WDF and charging functions, respectively. We transform the piecewise linear functions generated in this fashion to valid WDF and charging functions by scaling each segment i to span $\nu \cdot \frac{w_i}{\sum_{j \in [1,n]} w_j}$ on the SoC and time axes. We generate the WDF with $\chi_{\min} := 0.1$, $\chi_{\max} := 0.8$, and $\nu := 80$. We use durations $\nu := [150, 60, 90, 120, 75, 135]$ to generate chargers, such that fully charging the battery using the i^{th} charger takes ν_i minutes. Unless otherwise specified, we distribute charger capacity evenly across all available chargers. Finally, we generate a set of three operations for each day and vehicle. Each service operation consumes 50% of the battery capacity, leading to a total discharge of 120 kWh per day and vehicle. We distribute operation departure times randomly such that vehicles spend a minimum of one hour before each operation at the depot, and center the departure time windows according to the static case. To ensure that the generated instances are comparable, we use independently seeded random engines for each parameter, such that the set of service operations of a one day instance is a subset of the service plan of the two and three-day instances generated from the same seed.

Appendix G: Data used in the example

SoC	Cost	Unit cost
0	0	-
2	1	0.5
7	7	1.2

Table 7: Wear density function.

Arc	Consumption ($q_{i,j}$)	Fixed cost ($c_{i,j}$)	Energy cost at origin
(s^-, v_f)	0	2	0
(v_f, v_2)	0	0	2.5
(v_2, v_3)	1.5	0	0
(v_3, v_g)	0	0	0
(v_g, v_5)	0	0	0.75
$(v_5, s^+)(1)$	0	3.5	0
$(v_5, s^+)(2)$	0	4.25	0

Table 8: Arcs of the time expanded network.

f			g		
Time	SoC	Rate	Time	SoC	Rate
0	0	-	0	0	-
6	2	0.3333	2.5	2	0.8
24.5	7	0.2707	12.25	7	0.5128

Table 9: The charging functions used.

f			g		
Cost	SoC	Rate	Cost	SoC	Rate
0	0	-	0	0	-
6	2	0.3333	2.5	2	0.8
24.5	7	0.2702	12.25	7	0.5128

Table 10: The station cost profiles.

Cost profile	Segments			
$\psi_{\ell_{s^-}}$	$[-\infty, 0.0) \rightarrow [-\infty, -\infty]$	$[0.0, \infty) \rightarrow [0.0, 0.0]$		
$\psi_{\ell_{v_f}}$	$[-\infty, 2.0) \rightarrow [-\infty, -\infty]$	$[2.0, \infty) \rightarrow [0.0, 0.0]$		
$\psi_{\ell_{v_2}}$	$[-\infty, 2.0) \rightarrow [-\infty, -\infty]$	$[2.0, 8.0) \rightarrow [0.0, 2.0]$	$[8.0, 11.7) \rightarrow [2.0, 3.0)$	$[11.7, \infty] \rightarrow [3.0, 3.0]$
$\psi_{\ell_{v_3}}$	$[-\infty, 6.5) \rightarrow [-\infty, -\infty]$	$[6.5, 8.0) \rightarrow [0.0, 0.5]$	$[8.0, 11.7) \rightarrow [0.5, 1.5)$	$[11.7, \infty] \rightarrow [1.5, 1.5]$
$\psi_{\ell_{v_g}}$	$[-\infty, 6.5) \rightarrow [-\infty, -\infty]$	$[6.5, 8.0) \rightarrow [0.0, 0.5]$	$[8.0, 11.7) \rightarrow [0.5, 1.5)$	$[11.7, \infty] \rightarrow [1.5, 1.5]$
$\psi_{\ell^1_{v_5}}$	$[-\infty, 6.5) \rightarrow [-\infty, -\infty]$	$[6.5, 9.0) \rightarrow [0.0, 2.0]$	$[9.0, 12.9) \rightarrow [2.0, 4.0)$	$[12.9, \infty] \rightarrow [4.0, 4.0]$
$\psi_{\ell^2_{v_5}}$	$[-\infty, 12.9) \rightarrow [-\infty, -\infty]$	$[12.9, 14.75) \rightarrow [4.0, 4.5]$	$[14.75, 19.15) \rightarrow [4.5, 5.5)$	$[14.75, \infty] \rightarrow [5.5, 5.5]$
$\psi_{\ell^3_{v_5}}$	$[-\infty, 8.0) \rightarrow [-\infty, -\infty]$	$[8.0, 9.875) \rightarrow [0.5, 2.0]$	$[9.875, 14.75) \rightarrow [2.0, 4.5)$	$[14.75, \infty] \rightarrow [4.5, 4.5]$
$\psi_{\ell^4_{v_5}}$	$[-\infty, 11.7) \rightarrow [-\infty, -\infty]$	$[11.7, 12.325) \rightarrow [1.5, 2.0]$	$[12.325, 19.15) \rightarrow [2.0, 5.5)$	$[12.325, \infty] \rightarrow [5.5, 5.5]$
		$q_{(v_5, s^+)} = 4.25$		
$\psi_{\ell^1_{s^+}}$	$[-\infty, 11.925) \rightarrow [-\infty, -\infty]$	$[11.925, 12.9) \rightarrow [0.0, 0.5]$	$[12.9, \infty) \rightarrow [0.5, 0.5)$	
$\psi_{\ell^2_{s^+}}$	$[-\infty, 12.9) \rightarrow [-\infty, -\infty]$	$[12.9, 14.75) \rightarrow [0.5, 1.0]$	$[14.75, 19.15) \rightarrow [1.0, 2.0)$	$[14.75, \infty] \rightarrow [2.0, 2.0]$
		$q_{(v_5, s^+)} = 3.5$		
$\psi_{\ell^2_{s^+}}$	$[-\infty, 13.825) \rightarrow [-\infty, -\infty]$	$[13.825, 14.75) \rightarrow [0.0, 0.25]$	$[14.75, 19.15) \rightarrow [0.25, 1.25)$	$[14.75, \infty] \rightarrow [1.25, 1.25]$
$\psi_{\ell^3_{s^+}}$	$[-\infty, 14.2625) \rightarrow [-\infty, -\infty]$	$[14.2625, 14.75) \rightarrow [0.0, 0.25]$		$[14.75, \infty] \rightarrow [0.25, 0.25]$

Table 11: Cost profiles created in the example.

5 RoutingBlocks: An open-source Python package for Vehicle Routing Problems with Intermediate Stops

This chapter is based on an article published as:

Klein P. S., Schiffer M. (2024) RoutingBlocks: An Open-Source Python Package for Vehicle Routing Problems with Intermediate Stops. *INFORMS Journal on Computing* (Articles in Advance).

Patrick S. Klein
School of Management, Technical University of Munich, Munich, Germany,
patrick.sean.klein@tum.de

Maximilian Schiffer
School of Management & Munich Data Science Institute, Technical University of Munich, Munich, Germany,
schiffer@tum.de

We introduce `RoutingBlocks`, a versatile open-source Python package designed to simplify the development of algorithms for vehicle routing problems with intermediate stops (VRPIS). The package offers a variety of modular algorithmic components and optimized data structures, crafted specifically to address key challenges of VRPIS, such as a lack of efficient exact move evaluations and difficult station visit decisions. By employing a unified solution and instance representation that abstracts problem-specific behavior, e.g., constraint checking, move evaluation, and cost computation, into well-defined interfaces, `RoutingBlocks` maintains a clear separation between algorithmic components and specific problem configurations, thus allowing to apply the same algorithm to a variety of problem settings. Leveraging an efficient C++ implementation for performance-critical core elements, such as move evaluation and local search operators, `RoutingBlocks` combines the high performance of C++ with the user-friendliness and adaptability of Python, thereby streamlining the development of effective metaheuristic algorithms. As a result, researchers using `RoutingBlocks` can focus on their algorithms' core features, allocating more resources to innovation and advancement in the VRPIS domain.

Key words: Vehicle routing, metaheuristic algorithms, Python, open-source software

5.1. Introduction

Vehicle routing problems with intermediate stops (VRPIS) have become ubiquitous in transport optimization when focusing on recently evolving planning problems, e.g., the deployment of electric vehicles or innovative urban logitics concepts where city freighters perform last-mile delivery, replenishing freight at micro depots (cf. Schiffer et al. 2019). In VRPIS, vehicles perform *intermediate stops* at *intraroute facilities* to replenish a resource, e.g., energy or freight, to keep a vehicle operational while fulfilling its main task. Here, an intraroute facility differs from a conventional depot as it is on the same echelon as the customers, e.g., a charging station or a micro depot.

During the last decade, we have seen steep progress on developing state-of-the-art open source meta-heuristic and exact frameworks for classical vehicle routing problem (VRP) variants such as the capacitated vehicle routing problem (CVRP) and the vehicle routing problem with time windows (VRPTW) (see, e.g., Accorsi and Vigo 2021, Vidal 2022). Contrarily, the progress on state-of-the-art algorithms for VRPIS variants appears to emerge rather slowly. In the case of exact algorithms, this is likely caused by a lack of VRPIS-specific algorithms (cf. Schiffer et al. 2019). For metaheuristic algorithms, this can partially be attributed to missing state-of-the-art open-source implementations tailored to VRPIS.

In fact, extending metaheuristics or open source libraries designed for classical VRP variants to VRPIS is anything but straightforward for several reasons: first, classical algorithms rely heavily on local search, which is insufficient for accommodating intermediate stops as it focuses on cost-minimization. Indeed, inserting intermediate stops often causes a detour which is orthogonal to the goal of local search. Second, existing algorithms rely on efficient, exact evaluation of local search moves in amortized constant time. However, for many well-known VRPIS, e.g., electric vehicle routing problems (EVRPs), no such evaluation functions exist such that move evaluation remains expensive or approximate (Montoya et al. 2017, Schiffer and Walther 2018, Erdelić and Carić 2022). Third, different VRPIS variants often show unique problem characteristics, e.g. w.r.t. replenishment behavior (quantitiy-dependent vs. quantity-independent), replenishment and resource trade-offs, and synchronization constraints (cf., Schiffer et al. 2019), that require specialized algorithmic components. Accordingly, designing a library that is amendable to a broad range of VRPIS is a challenge in itself. In fact, even for mainstream VRPs, existing libraries (Groer 2009, Rasku and Kwon 2023) and open sourced metaheuristics (Vidal 2022) struggle with this generalization task and surpass it by either modeling the union of all problem-specific characteristics[1] (Groer 2009, Vidal 2022), or by not providing a generalized interface for, e.g., cost evaluation and constraint checking (Rasku and Kwon 2023). Both of these approaches have their respective drawbacks: considering the union of all problem-specific characteristics makes the implementation more challenging and requires additional cognitive effort on the side of the library user. Not providing a central interface on the other hand limits the composeability and interoperability of the implemented algorithms, which often use different solution representations and evaluation procedures.

Against this background, we provide an open-source Python package—`RoutingBlocks`—that eases the design and implementation of algorithms for VRPIS. Here, `RoutingBlocks`

[1] using dummy values for problems that only require a subset of these

specifically focusses on metaheuristics to best align with existing research in the area and remain amendable to broad range of applications, e.g., dynamic settings. `RoutingBlocks` breaks with the design of existing open-source VRP packages to account for the peculiarities highlighted above, and, instead of implementing a single, monolithic algorithm, provides a library of modular algorithmic components and efficient data structures. We specifically tailor these to handle some of the fundamental challenges that arise when solving VRPIS. The modular design of our package significantly eases the development of metaheuristics for VRPIS, allowing researchers to focus on implementing the primary algorithmic flow by composing the provided components, with an easy option to add specialized supplementary components if necessary. Ultimately, researchers can focus on the core aspects of their algorithms when utilizing our package, using more of their resources to facilitate innovation and progress in the field of VRPIS.

The remainder of this paper is structured as follows. Section 5.2 gives a general definition of the class of VRPIS considered when developing the `RoutingBlocks` package. Section 5.3 then provides an overview on the architecture and design of `RoutingBlocks` and briefly reviews the package's algorithmic components. Section 5.4 provides information on obtaining the package, before Section 5.5 benchmarks an algorithm implemented using `RoutingBlocks` against state-of-the-art monolithic designs. Section 5.6 concludes the paper. We note that this paper details the design of Routingblocks and supplements the documentation[2], which provides a hands-on introduction to the package in addition to several example implementations.

5.2. Vehicle routing problems with intermediate stops

VRPIS expand upon classic VRPs by considering one or multiple additional resources consumed when traveling along arcs and servicing customers. Unlike resources in classical VRPs, e.g., time, these resources do not increase or decrease monotonously. Instead, they can be replenished at intermediate facilities, typically by facing a trade-off between different resources. An example of a VRPIS is the electric vehicle routing problem with time windows (EVRP-TW), where a fleet of electric vehicles with limited battery capacity must serve a set of customers within specified time windows. In this case, the consumed additional resource

[2] `https://routingblocks.readthedocs.io/`

is the battery's state of charge, which can be replenished by charging at designated charging stations, consuming time in the process.

Formally, VRPIS are defined on directed complete graphs, which consist of vertices representing locations that vehicles can visit during their routes, specifically depots, customers, and intermediate stops. Arcs connect these vertices, and both arcs and vertices have resource consumption functions that determine the consumption of each considered resource incurred by traversing the arc or visiting the vertex. These resource consumption functions can be arbitrary and are often non-linear, e.g., in some EVRP variants. For a formal definition of VRPIS, we refer the interested reader to Schiffer et al. (2017).

5.3. The `RoutingBlocks` package

`RoutingBlocks` is an open-source Python package that eases the implementation of algorithms for VRPIS by providing a diverse set of modular algorithmic components and efficient data structures (see Figure 1), designed specifically to tackle some of the fundamental challenges of VRPIS, e.g., inexact move evaluation and difficult decisions around embedding visits to intraroute replenishment facilities. These components serve as building blocks for problem-specific algorithms and include a local search solver (Sections 5.3.1 and 5.3.2), an exact labelling algorithm (Section 5.3.3), components for metaheuristic algorithms (Section 5.3.4), and auxiliary algorithms and data structures, e.g., move caches (Section 5.3.5). Each of these can be extended through well-defined interfaces to support a wide range of algorithmic designs.

As illustrated in Figure 1, `RoutingBlocks` consists of three modules: *algorithms and data structures*, *core components*, and *problem-specific interfaces*. Algorithms and data structures implement components commonly used in metaheuristics for VRPIS and utilize a common solution and instance representation provided by the core components. These abstract problem-specific behavior, such as constraint checking, move evaluation, and cost computation, such that algorithms remain decoupled from specific problem settings. Users can incorporate problem-specific behavior by implementing the provided interfaces.

`RoutingBlocks` builds on an efficient C++-based implementation of runtime-critical core components. This integration of C++ and Python combines the high performance of C++ with the ease of use and flexibility of Python, allowing rapid development of efficient metaheuristic

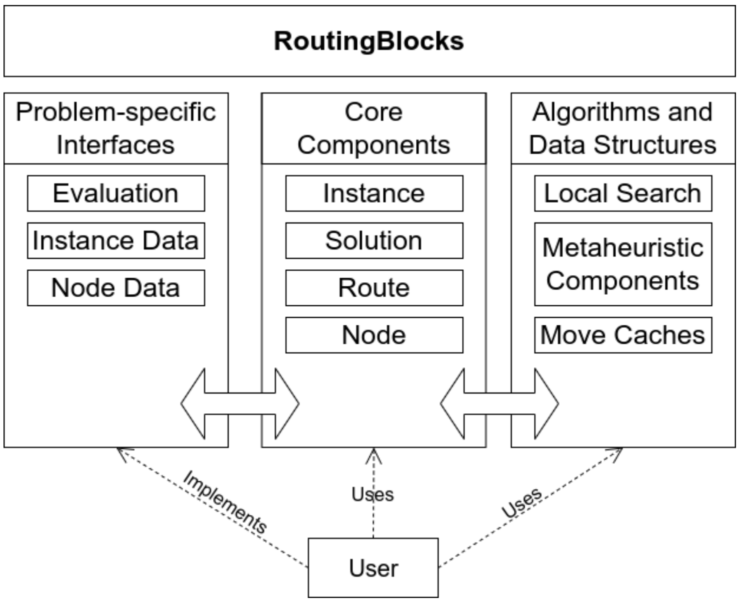


Figure 1 Architecture of the `RoutingBlocks` package.

algorithms. Users can thus implement the core logic of their algorithms in Python, potentially capitalizing on existing high-quality implementations of other complex algorithmic components (see, e.g., Kullman et al. (2021)) and leveraging robust tooling (Peng and Murray 2022, Mendoza et al. 2014). Beyond this, `RoutingBlocks` supports using any of the provided components straight from C++ as well. Accordingly, our package supports customization of the provided algorithmic components using C++, Python, or a combination of the two.

In what follows, we briefly review the design of our core components, i.e., the instance and solution representations, as well as the algorithms and data structures our package provides. We refer to `https://github.com/tumBAIS/RoutingBlocks` for comprehensive documentation of the individual components and an example that illustrates how to compose these to a full algorithm.

5.3.1. Core components

Core components capture the fundamental concepts shared by all VRPIS, such as instance graphs, routes, and solutions. By providing abstract interfaces, they effectively conceal problem-specific data, e.g., customer time windows, and behavior, e.g., cost evaluation procedures, from the provided data structures and algorithms.

5.3.1. Instance Graph Similar to regular VRPs, VRPIS are almost exclusively modeled on graphs, which we represent using the instance class. This class contains a set of vertices, that correspond to locations vehicles can visit during their routes. We divide vertices into three groups: depots, customers and intermediate stops. The class also contains a set of arcs, represented as a dense two-dimensional matrix, where each row corresponds to the set of arcs that originate from a particular vertex.

The instance class assumes only minimal knowledge about concrete vertices and arcs. Specifically, vertices carry only an identifier and a type (i.e., station, depot, or customer), and arcs do not carry any information at all. Instead, these provide type-erased slots for user-defined problem-specific data, e.g., time windows, demand, and distance. Problem-specific details thus remain opaque to the instance but are retrievable by specialized algorithms.

5.3.2. Solution Representation Our framework provides a rich, intuitive solution representation comprising *solution*, *route*, and *node* classes, organized hierarchically: a solution comprises a set of routes, which each manage a sequence of nodes. Here, nodes represent visits to vertices and carry data used for cost calculation, constraint checking, and efficient move evaluation (cf. Section 5.3.2). Clearly, the structure of this data and how it is utilized remains highly problem-specific and thus has to remain opaque to route and solution classes. These operations are instead implemented in a problem-specific *evaluation* class (cf. Section 5.3.3). This reduces the responsibility of route and solution classes to managing the sequences of nodes and set of routes, respectively. Specifically, they provide interfaces to insert, remove, or exchange arbitrary segments of nodes, guaranteeing that information on feasibility, costs, and the data associated with partial routes, i.e., what is carried in node classes, is maintained at all times. This simplifies the implementation of specific algorithms, and further allows to abstract from problem-specific characteristics by default.

Similar to Vidal (2022), we assign monotonic *timestamps* to routes, which can be utilized to implement caching strategies. Specifically, we guarantee that routes with matching timestamps are equal with respect to the sequence of nodes visited. We achieve this with a simple global counter: each time a route is modified, we set it's timestamp to the current value of the counter, which is incremented subsequently.

5.3.3. Evaluation The evaluation class provides interfaces to i) assess the cost of a (partial) route, ii) verify the feasibility of a (partial) route, and iii) maintain the information contained within node structures. It's interfaces are implemented by the user, such that the class has access to data that is opaque to instance, solution, route, and node classes. The design of this interface follows the concept of propagation functions and route segment concatenation introduced in Vidal et al. (2014), which we further detail in Section 5.3.2.

5.3.2. Data structures and algorithms

5.3.1. Local Search Local search is a core component of almost every state-of-the-art metaheuristic designed for VRPs. The main idea of this algorithm is to iteratively improve an initial solution by making small, localized modifications, so called *moves*, to the current solution until reaching a local optimum, i.e., a state where none of the defined moves further improve the solution. This seemingly simple concept allows for a lot of customization, such that the specific design of the local search procedure varies significantly between algorithms. Common design decisions include i) the selection of neighborhood operators, which define the considered moves, ii) pivoting rules, which dictate a strategy for selecting moves, e.g., best and first improvement, and iii) speed-up techniques such as granular neighborhoods.

`RoutingBlocks` provides a local search component that supports a variety of local search designs. Here, we deviate from generator-arc based move definitions (see, e.g., Vidal 2022). Specifically, in generator-arc based designs, each move is uniquely defined by a single arc. This allows to define operators as single, stateless functions that take an arc to be evaluated as input argument, thus allowing for a clear separation of concerns: operators simply evaluate or apply a move, and a local search solver steers the search procedure by controlling which arcs are passed to the respective operators. Unfortunately, this generator-arc based design is not well-suited for VRPIS, where widely used operators, e.g., the *station-in* operator (Schneider et al. 2014) that attempts to insert detours to intermediate replenishment facilities into an existing route, may assess several moves per arc, and arcs can appear an arbitrary number of times within the same solution. We achieve similar benefits as generator-arc based designs by treating operators as iterators over the neighborhood of a solution. Specifically, on each invocation, an operator generates the (next) improving move, returning control to the callee. These moves remain opaque to the local search solver and simply provide apply and evaluation interfaces. This provides the flexibility required to support arbitrary operators while still allowing the local search solver to control the search. Accordingly, implementing design decisions such as pivoting rules and neighborhood composition strategies (e.g., variable neighborhood descent) in `RoutingBlocks` does not require implementation on an operator-to-operator basis.

`RoutingBlocks` provides well defined interfaces for general moves and operators, which can be implemented in Python or C++. It additionally provides specializations of general operator and move interfaces to support generator-arc based implementations, such that porting existing implementations of local search operators to `RoutingBlocks` is straightforward. `RoutingBlocks` further implements a set of well-known operators commonly used in VRPIS, e.g., various swap- and relocation-based neighborhoods (*n-m-exchange*), *TwoOpt**, and *Station-In/Out*, out of the box.

5.3.2. Move evaluation Computationally efficient evaluation of moves is crucial to establish the performance of local search based metaheuristics. Here, many algorithms rely on the observation that any edge- or node-exchange based move can be expressed as a concatenation of route sub-sequences (Kindervater and Savelsbergh 2003). Hence, an evaluation function that takes a list of node sequences as arguments can leverage pre-computed information on route sub-sequences to achieve computationally efficient move evaluation. For instance, removing node x_i from route $[x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n]$ can be expressed as concatenating routes $[x_1, \ldots, x_{i-1}]$ and $[x_{i+1}, \ldots, x_n]$, potentially making use of the *forward* information stored in x_{i-1} and the *backward* information stored in x_{i+1}. This idea has been generalized in Vidal et al. (2014). `RoutingBlocks`'s interface follows their design and exposes five functions to be implemented by the user: i) `initialize_forward(`x`)` and `initialize_backward(`x`)`, which initialize forward and backward information stored on node x, respectively, ii) `propagate_forward([`$\ldots, x_i$`], `y`)` and `propagate_backward(`y`, [`$x_j, \ldots$`])`, which extend the forward and backward information stored for routes $[\ldots, x_i]$ and $[x_j, \ldots]$ to $[\ldots, x_i, y]$ and $[y, x_j, \ldots]$ for some node y, respectively, and iii) `evaluate(`$\sigma_1, \ldots, \sigma_n$`)`, which evaluates the cost of concatenating an arbitrary number of route segments σ.

A major challenge in many VRPIS, e.g., (non-linear) EVRP-TW, is that no efficient *exact* move evaluation functions are known. A common approach to dealing with this issue is to combine efficient approximate concatenation operators with exact move evaluation functions (Montoya et al. 2017, Froger et al. 2022, Erdelić and Carić 2022). Here, the approximate operator ideally provides an upper bound on the improvement obtained from applying a move.

This allows to eagerly discard non-improving moves, such that the potentially expensive exact move evaluation function needs to be applied to only a subset of potential moves.

`RoutingBlocks`'s local search solver supports this approach by default and allows to verify improving moves using a second evaluation function. In general, any component provided by `RoutingBlocks` allows to switch evaluation implementations at any point in time.

5.3.3. Labeling algorithm Many VRPIS rely on exact algorithms to embed detours to intraroute facilities at optimal locations (Schiffer et al. 2017, Erdelić and Carić 2022). A specific approach that was proven to work well in practice models this detour embedding problem as a resource-constrained shortest path problem (CSP) on an auxilliary graph, solved for each route in the solution independently. This graph comprises a vertex for each non-station node in the route, and allows detours to these by inserting copies of each intraroute facility between two consecutive non-station vertices. We refer to Schiffer and Walther (2018) for a comprehensive description of this procedure.

`RoutingBlocks` provides an implementation of this algorithm that takes care of label management, dominance-based path pruning, graph building, and other boilerplate tasks while providing an interface to abstract problem-specific functionality, i.e., label representation, resource extension functions, dominance, and preprocessing. The design of this interface bases on the abstractions introduced in Irnich (2008).

5.3.4. Metaheuristic components Broady speaking, Metaheuristics are solution methods that guide local improvement based procedures to allow escaping from local optima. `RoutingBlocks` provides a set of data structures and boilerplate required to implement common metaheuristic algorithms. These include components that provide deterministic sources of randomness, several *destroy* operators that remove vertices from a solution, *repair operators* that insert vertices into a solution, an (adaptive) priority list to *learn* to dynamically rate operators based on past performance, and a solver that applies operators sequentially, i.e., removes and (re-)inserts a subset of vertices. Specifically, `RoutingBlocks` provides several generic destroy and repair operators known to work well for VRPIS out-of-the-box: *RandomInsertion*, *BestInsertion* (Ropke and Pisinger 2006), *WorstRemoval* (Ropke and Pisinger 2006), *RelatedRemoval* (Pisinger and Ropke 2007), *ShawRemoval* (Shaw 1997), and *StationVicinityRemoval* (Goeke and Schneider 2015). These remain extensible, e.g., the acceptance criterion of best sequential insertion is customizable, such that N-best-insertion and best-insertion-with-blinks can be implemented straightforwardly. We provide interfaces to allow user-defined destroy and repair operators, implemented in either C++ or Python.

Our examples[3] showcase how to utilize these components to build well-known metaheuristic algorithms, e.g. *adaptive large neighborhood search (ALNS) (cf., Ropke and Pisinger 2006).*

5.3.5. Auxiliary data structures We provide several auxiliary data structures, such as insertion and removal caches, to significantly reduce the development overhead of implementing custom algorithms. Similar to the concept of static move descriptors (Zachariadis and Kiranoudis 2010, Beek et al. 2018), these caches track changes to solutions, avoiding potentially expensive re-evaluations of moves unaffected by a specific modification. Our implementation of this caching mechanism is much simpler than the one proposed in Zachariadis and Kiranoudis (2010) and Beek et al. (2018): as in Vidal (2022), we rely on route timestamps (cf. Section 5.3.2) to determine if a move has to be re-evaluated or not. The reason for this is twofold: first, maintaining complicated static move descriptor data structures can entail significant computational overhead especially in VRPIS, where the number of nodes in a solution may change dynamically. Second, this simple timestamp-based solution does not rely on any problem-specific information.

The benefit of these move caches is especially pronounced when used in algorithms implemented in Python. Here, the use of caches, which are implemented in native code, allows to bundle evaluation operations and thus minimizes the number of calls between Python and the C++ extension module.

5.4. Distribution

The `RoutingBlocks` package is available via `pip` through `PyPI` (`https://pypi.org/project/routingblocks`). It's source code, documentation, and examples can be found at `https://github.com/tumBAIS/RoutingBlocks`. The repository further provides information on contributing to the project, e.g., by reporting bugs or supplying additional algorithmic components.

[3] https://routingblocks.readthedocs.io/en/latest/examples.html

5.5. Benchmarks

We have implemented a simple metaheuristic algorithm for the electric vehicle routing problem with time windows and partial recharging (EVRP-TW-PR) using `RoutingBlocks` to show that the performance of algorithms developed with `RoutingBlocks` remains comparable to specialized, monolithic implementations, i.e., the one provided in Erdelić and Carić (2022). This particular problem setting is particularly suitable to showcase some of `RoutingBlocks'` core features, such as two-stage move evaluation, complex guiding metaheuristics, and exact subcomponents. Moreover, the algorithm proposed in Erdelić and Carić (2022) is similar to our proposed algorithm: It bases on ALNS, relies on an exact dynamic programming based station placement procedure, and uses an approximate evaluation function. Note that the purpose of this benchmark is to assess the performance of `RoutingBlocks` and not to provide new best known solutions. We thus did not conduct sensitivity analyses on the components used or tune our algorithm's hyperparameters. We refer to Erdelić and Carić (2022) for further information on the considered instances. The source code of our implementation is available on `https://github.com/tumBAIS/RoutingBlocks` and may serve as a reference implementation, showcasing how to utilize the package. Beyond this, up-to-date benchmarks that verify the performance of our core algorithms compared to pure Python-based implementations are part of our CI pipeline and available on `https://github.com/tumBAIS/RoutingBlocks`.

We conducted all of our experiments on a standard desktop computer equipped with an `Intel(R) Core(TM) i9-9900, 3.1 GHz CPU` and `16 GB` of `RAM`, running `Ubuntu 20.04` and Python `3.8.11`. We limit the total number of iterations of our ALNS algorithm to 5000, and the number of iterations without improvement to 1100. Table 1 summarizes the results of our benchmark compared to two state-of-the-art monolithic algorithms implemented in native code, i.e., C# and C++ (cf. Schiffer et al. 2017, Erdelić and Carić 2022). We report detailed statistics in the Appendix.

The solution quality achieved with a `RoutingBlocks`-based implementation remains compareable to that of monolithic algorithms, maintaining a gap of 0.44% to the best known solution on average. When comparing runtime, we see that `RoutingBlocks` is 14.86% slower than a `C#`-based implementation (Erdelić and Carić 2022), and 277.93% slower than a pure C++ implementation Schiffer et al. (2017). Despite these performance differences, the efficiency of `RoutingBlocks` is well within the bounds expected for a generic library, making it a suitable choice for prototyping metaheuristic algorithms.

n	$n^{\texttt{RB}}$	n^{e}	n^{s}	$n^{\texttt{RB}}_{\texttt{BKS}}$	$n^{e}_{\texttt{BKS}}$	$n^{s}_{\texttt{BKS}}$	$\overline{\Delta^{\texttt{RB}}_{BKS}}$ [%]	$\overline{\Delta^{e}_{BKS}}$ [%]	$\overline{\Delta^{s}_{BKS}}$ [%]	$\overline{\tau^{\texttt{RB}}}$ [s]	$\overline{\tau^{e}}$ [s]	$\overline{\tau^{s}}$ [s]
56	54	56	42	24	49	29	0.43	0.04	0.13	615.21	535.61	162.78

Abbreviations hold as follows: n - total number of instances, n° - number of instances where the respective algorithm matched the number of vehicles in the BKS, $n^{\circ}_{\texttt{BKS}}$ - number of instances where the respective algorithm matched the BKS, $\overline{\Delta^{\circ}_{BKS}}$ - average gap of instances with the same number of vehicles as the BKS expressed in percent, $\overline{\tau^{\circ}}$ - average runtime on instances with the same number of vehicles as the BKS in seconds. `RB` corresponds to `RoutingBlocks`, e to the algorithm of Erdelić and Carić (2022), and s to the algorithm of Schiffer et al. (2017). We caclulate percentage gaps as $\Delta^{a}_{\texttt{BKS}} = \frac{v^{a} - v_{\texttt{BKS}}}{v_{\texttt{BKS}}} \cdot 100\%$.

Table 1 Aggregated results of our benchmark against Schiffer et al. (2017) and Erdelić and Carić (2022).

5.6. Conclusion

We introduced `RoutingBlocks`, a modular Python package for the implementation of algorithms for VRPIS. `RoutingBlocks` provides a diverse set of algorithmic components and efficient data structures designed specifically to tackle some of the fundamental challenges of VRPIS, such as inexact move evaluation and complex station visit decisions. These serve as building blocks for problem-specific algorithms, streamlining the development process for algorithms addressing VRPIS and related problems. `RoutingBlocks` provides a scalable C++-based implementation as a rich Python API, combining the high performance and scalability of C++ with the ease of use and flexibility of Python. This allows the rapid development of scalable, problem-specific algorithms that can further leverage existing high-quality implementations of complex algorithms (see, e.g., Kullman et al. 2021), and leverage robust tooling (Peng and Murray 2022, Mendoza et al. 2014). In accordance with this approach, `RoutingBlocks` supports customization of the provided algorithmic components using C++, Python, or a combination of both, catering to diverse algorithmic requirements. We provide `RoutingBlocks` under a permissive open-source license, ensuring that it remains relevant and useful to the community over time. Ultimately, we hope that `RoutingBlocks` will serve as a valuable resource for anyone working on the challenging and important class of VRPIS.

Appendix A: Detailed results

Instance	`RoutingBlocks`			Erdelić and Carić (2022)			Schiffer et al. (2017)		
	σ	K	τ $[s]$	σ	K	τ $[s]$	σ	K	τ $[s]$
c101	1043.38	12	382.17	1043.38	12	220.20	1043.38	12	64.20
c102	1074.07	10	350.14	1058.67	10	190.20	1027.80	11	121.20
c103	973.92	10	654.90	971.19	10	248.40	973.63	10	127.80
c104	892.33	10	1070.25	884.38	10	454.20	884.48	10	135.00
c105	1064.66	10	739.17	1064.66	10	143.40	1023.52	11	81.00
c106	1091.14	10	574.49	1061.61	10	151.80	1009.26	11	58.80
c107	1088.43	10	482.79	1046.50	10	158.40	1010.91	11	54.60
c108	1055.70	10	607.18	1022.93	10	274.80	1030.48	10	105.00
c109	954.77	10	491.64	940.38	10	268.80	940.38	10	148.20
c201	629.95	4	136.11	629.95	4	49.20	629.95	4	25.20
c202	629.95	4	160.79	629.95	4	139.20	629.95	4	91.80
c203	629.95	4	207.14	629.95	4	155.40	629.95	4	199.20
c204	628.91	4	251.22	628.91	4	310.20	628.91	4	196.20
c205	629.95	4	188.13	629.95	4	63.60	629.95	4	46.80
c206	629.95	4	220.71	629.95	4	77.40	629.95	4	46.80
c207	629.95	4	233.37	629.95	4	96.60	629.95	4	72.60
c208	629.95	4	217.99	629.95	4	78.60	629.95	4	66.00
r101	1639.68	17	1815.91	1624.89	17	371.40	1618.89	18	161.40
r102	1454.92	15	1461.45	1454.53	15	441.60	1426.82	16	244.20
r103	1211.40	13	1375.46	1304.24	12	496.80	1211.99	13	358.80
r104	1056.69	11	1013.35	1051.41	11	706.20	1051.41	11	249.00
r105	1347.80	14	1234.39	1347.80	14	324.00	1347.80	14	181.20
r106	1263.13	13	1495.94	1263.13	13	510.00	1265.18	13	237.00
r107	1108.47	11	681.00	1104.51	11	635.40	1108.23	12	150.00
r108	1021.47	11	422.24	1030.44	10	696.00	1020.52	11	200.40
r109	1185.40	12	952.50	1176.69	12	502.80	1176.69	12	191.40
r110	1069.18	11	1396.62	1067.11	11	495.00	1068.87	11	226.20
r111	1073.15	11	692.80	1076.15	11	489.00	1070.97	12	241.20
r112	1001.79	11	929.92	1001.79	11	448.20	1001.79	11	204.00
r201	1255.81	3	560.94	1255.81	3	627.00	1255.81	3	88.80
r202	1051.46	3	274.08	1052.52	3	685.80	1051.46	3	151.80
r203	895.54	3	538.69	895.54	3	1144.20	895.96	3	314.40
r204	779.57	2	333.15	779.49	2	1325.40	779.49	2	428.40
r205	987.36	3	281.68	987.22	3	558.60	987.22	3	175.80
r206	925.37	3	546.18	922.08	3	701.40	922.08	3	211.20
r207	843.20	2	244.77	845.19	2	960.60	843.20	2	156.60
r208	737.30	2	275.65	736.46	2	1436.40	736.12	2	228.60
r209	871.55	3	245.37	863.17	3	668.40	863.17	3	115.20
r210	844.71	3	260.75	844.71	3	835.20	844.86	3	215.40
r211	823.82	2	548.54	825.25	2	1164.60	826.26	2	163.80
rc101	1662.16	15	590.50	1661.53	15	352.80	1648.99	16	133.80
rc102	1510.16	14	992.38	1510.16	14	461.40	1510.16	14	112.20
rc103	1361.68	12	986.65	1359.34	12	455.40	1304.10	13	181.20
rc104	1175.06	11	1118.39	1174.32	11	575.40	1175.06	11	155.40
rc105	1473.34	13	1448.29	1471.80	13	269.40	1449.53	14	179.40
rc106	1386.25	13	484.94	1391.23	13	333.60	1385.96	13	116.40
rc107	1250.80	11	524.47	1244.37	11	368.40	1248.11	12	186.00
rc108	1154.14	11	565.44	1154.14	11	541.20	1154.14	11	135.00
rc201	1443.13	4	322.21	1433.57	4	474.00	1443.13	4	84.00
rc202	1406.18	3	274.24	1403.67	3	848.40	1407.05	3	180.60
rc203	1054.91	3	327.32	1054.91	3	1264.20	1054.91	3	319.80
rc204	886.57	3	447.36	884.75	3	1237.20	884.75	3	256.80
rc205	1256.54	3	515.78	1238.46	3	962.40	1255.84	3	143.40
rc206	1184.61	3	371.69	1197.60	3	639.60	1184.61	3	100.80
rc207	990.59	3	407.76	978.30	3	892.20	992.16	3	134.40
rc208	838.86	3	524.60	833.12	3	1014.00	833.12	3	161.40

Abbreviations hold as follows: σ - objective value, K - number of vehicles used, τ $[s]$ - runtime in seconds.

Table 2 Detailed results of our benchmark against Schiffer et al. (2017) and Erdelić and Carić (2022).

References

Accorsi L, Vigo D (2021) A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science* 55(4):832–856.

Beek O, Raa B, Dullaert W, Vigo D (2018) An Efficient Implementation of a Static Move Descriptor-based Local Search Heuristic. *Computers & Operations Research* 94:1–10, ISSN 0305-0548.

Christiaens J, Vanden Berghe G (2020) Slack Induction by String Removals for Vehicle Routing Problems. *Transportation Science* 54(2):417–433, ISSN 0041-1655.

Erdelić T, Carić T (2022) Goods delivery with electric vehicles: Electric vehicle routing optimization with time windows and partial or full recharge. *Energies* 15(1):285.

Froger A, Jabali O, Mendoza JE, Laporte G (2022) The Electric Vehicle Routing Problem with Capacitated Charging Stations. *Transportation Science* 56(2):460–482, ISSN 0041-1655.

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

Groer C (2009) Vrph. `https://github.com/coin-or/VRPH`.

Irnich S (2008) A Unified Modeling and Solution Framework for Vehicle Routing and Local Search-Based Metaheuristics. *INFORMS Journal on Computing* 20(2):270–287, ISSN 1091-9856.

Kindervater GAP, Savelsbergh MWP (2003) 10. vehicle routing: handling edge exchanges. *Local Search in Combinatorial Optimization*, 337–360 (Princeton University Press).

Kullman ND, Froger A, Mendoza JE, Goodson JC (2021) frvcpy: An open-source solver for the fixed route vehicle charging problem. *INFORMS Journal on Computing* .

Mendoza J, Guéret C, Hoskins M, Lobit H, Pillac V, Vidal T, Vigo D (2014) Vrp-rep: a vehicle routing community repository.

Montoya A, Guéret C, Mendoza JE, Villegas JG (2017) The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological* 103:87–110, ISSN 0191-2615.

Peng L, Murray C (2022) VeRoViz: A vehicle routing visualization toolkit. *INFORMS Journal on Computing* 34(4):1842–1848.

Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. *Computers & Operations Research* 34(8):2403–2435.

Rasku J, Kwon C (2023) VeRyPy. `https://github.com/yorak/VeRyPy`.

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

Schiffer M, Klein P, Schneider M, Walther G (2017) A solution framework for a class of vehicle routing problems with intermediate stops. *Working Paper OM-DPO 01/2017* .

Schiffer M, Schneider M, Walther G, Laporte G (2019) Vehicle routing and location routing with intermediate stops: A review. *Transportation Science* 53(2):319–343.

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

Schneider M, Stenger A, Goeke D (2014) The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science* 48(4):500–520.

Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK* 46.

Vidal T (2022) Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research* 140:105643, ISSN 0305-0548.

Vidal T, Crainic TG, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research* 234(3):658–673, ISSN 0377-2217.

Zachariadis EE, Kiranoudis CT (2010) A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research* 37(12):2089–2105, ISSN 0305-0548.

6 Conclusion

Author: Patrick Sean Klein

6.1. Main contributions

This thesis aimed to provide logistics service providers (LSPs) with the tools and methodology necessary to address complex optimization challenges emerging in today's last-mile distribution. For this purpose, I developed algorithmic solutions, complemented by practical contributions in the form of open-source software. This combination of fundamental theoretical research with tangible, practical tools is not only essential for ethical and reproducible science but is also pivotal in bridging the gap between theory and practice. Chapters 3 and 4 focused on theoretical contributions, developing algorithmic solutions for dynamic vehicle dispatching and electric fleet management, respectively. Chapter 5 introduced the practical contribution of this thesis, namely, the Python package `RoutingBlocks`. This package and all other open-source implementations of the Thesis' contributions are available on `https://github.com/tumBAIS`.

The first theoretical contribution of this thesis addressed dynamic route planning and dispatching. Specifically, it considered a Dynamic Vehicle Routing Problem (DVRP) where customer requests were not known in advance but gradually revealed over the span of the planning horizon. This setting required operators to dispatch vehicles in an online fashion, carefully balancing the benefits of immediately serving a specific customer against bundling it with so far unrevealed requests. Successfully tackling this problem required solving two challenging problems: first, determining the optimal timing for serving a request, and second, efficiently assembling served requests into routes. These challenges, while interrelated, required fundamentally different methodological approaches. The first challenge is a predictive task, typically addressed using Machine Learning (ML), whereas the second involves solving a Combinatorial Optimization (CO) problem. The major methodological contribution of this thesis was an approach that integrates these orthogonal methodologies in a single Machine Learning enriched Combinatorial Optimization (ML-CO) pipeline, allowing the ML model to leverage the combinatorial structure present in the CO problem. For this purpose, the CO component was reformulated as a prize-collecting Vehicle Routing Problem (VRP). This transformed the problem of deciding whether to serve a request at a particular time to anticipating the cost of serving an order in the future, moving the dispatching decision into the CO problem. This transformation ultimately allowed to compute approximate gradients over the CO algorithm, thus enabling end-to-end learning. This innovative approach not only advanced the state-of-the-art in DVRP but also made a significant contribution to the broader ML-CO domain. Specifically, this thesis introduced the first ML-CO pipeline that incorporated a metaheuristic component in the CO layer, showcasing how to carefully design a metaheuristic that allows the computation of meaningful approximate gradients in integrated pipelines. Extensive numerical experiments demonstrated the performance of this approach: it outperformed monte-carlo and rolling-horizon policies by 1.57% and 7.12% in terms of objective value while only taking 1.67% and 15.00% of the runtime, respectively. The approach further outperformed state-of-the-art reinforcement learning and classic two-stage non-integrated approaches, winning first prize in the EURO Meets NeurIPS 2022 Vehicle Routing Competition.

The second theoretical contribution of this thesis dealt with an integrated charging and service operation scheduling problem for fleets of electric vehicles. It enhanced the state-of-the-art by jointly scheduling charging and service operations while accounting for realistic battery behavior, i.e., non-linear charging and battery deterioration, variable energy prices, and charging infrastructure capacity constraints. Methodologically, this thesis approached the problem by decomposing it into master and subproblems, which were then solved using column generation within a Branch and Price (B&P) framework. In this setup, the master problem created fleet schedules according to infrastructure capacity constraints, drawing from a pool of promising vehicle schedules generated by the individual subproblems. This approach entailed two major algorithmic challenges: first, solving the subproblems required to consider non-linear trade-offs involving battery deterioration, charging costs, charger capacity, and the scheduling of service operations. To address this challenge, the thesis developed a novel label-setting shortest-path algorithm based on a function-based label representation. This approach enabled optimal decision-making even in continuous-time settings, significantly advancing the state-of-the-art in fixed-route vehicle charging problems. Second, capacity constraints caused a large fan-out in the branch-and-bound tree, threatening scalability. The B&P algorithm employed efficient branching and pruning strategies to address this issue. These strategies encompassed a problem-specific branching rule, a primal heuristic, and partial pricing, all of which collectively contributed to effectively managing the algorithm's scalability. A comprehensive numerical study showed the efficiency of the developed algorithm on large-scale instances. An ablation analysis further quantified the contribution of each of the developed branching and pruning strategies, providing a deeper understanding of the individual and collective impacts of these strategies on the overall effectiveness of the algorithm. Beyond these theoretical findings, this thesis contributes an analysis of the impact of jointly scheduling charging and service operations, focusing on i) the benefit of an integrated approach to charge and service operation scheduling, ii) the impact of infrastructure capacity on operational costs, and iii) the impact of considering variable energy prices. Specifically, integrating charging and service operation scheduling lowered the amount of charging infrastructure required by up to 57% and reduced operational costs by up to 5%. Here, both the degree of service schedule flexibility and the energy price distribution significantly impacted operational cost savings.

This thesis further made a practical contribution by providing a software package developed to ease the implementation of advanced routing algorithms, called `RoutingBlocks`. This library serves as a tool to enable researchers and practitioners to leverage advanced routing algorithms in their research or as part of their software infrastructure. For such a library to be effectively used in practice, it needs to be adaptable to a variety of unique, case-specific problem settings and needs to support numerous algorithmic approaches, including genetic algorithms, neighborhood search, and simulated annealing, all while maintaining computational efficiency. Balancing these diverse and sometimes conflicting requirements is undoubtedly a challenging task. `RoutingBlocks` tackled this challenge by providing a modular algorithmic framework that allows the composition of tailored solution methods from optimized, state-of-the-art algorithmic components and data structures. Specifically, the package achieved a clear separation between algorithmic components and problem configurations by employing a unified solution and instance representation that abstracts problem-specific behavior, e.g., constraint checking, move evaluation, and cost computation, thus allowing to apply the same algorithm to various problem settings. Each algorithmic component was thoroughly documented and showcased in several tutorials, ensuring the package's accessibility - a key for its application in educational and practical settings. To maintain high performance while adhering to this modular approach, `RoutingBlocks` employs a low-level, `C++`-based implementation. To shield users from the complexities of low-level programming, such as explicit memory management and lifetime issues, `RoutingBlocks` concealed its high-performance native components behind a user-friendly and accessible Python interface. A numerical benchmark against two monolithic state-of-the-art algorithms implemented in native code demonstrated the versatility and applicability of this approach. Here, `RoutingBlocks` achieved comparable performance, maintaining a gap of 0.44% w.r.t. objective value, while being only 14.86% slower than a state-of-the-art native implementation. This is well within the bounds expected for a generic library exposed in a high-level dynamic language such as Python.

In summary, this thesis contributed new methods, algorithms, and software that effectively address challenging optimization problems in modern last-mile logistics. Specifically, it tackled a dynamic route planning and dispatching problem, proposed efficient algorithms to design charging schedules for fleets of electric vehicles, and developed a software package to ease the implementation of advanced vehicle routing algorithms.

6.2. Limitations and Perspectives

The work presented in this thesis provides comprehensive algorithmic frameworks and modular tooling, establishing a solid foundation for future research to build upon in these areas. Nevertheless, some problem cases remained out of scope.

First, building on the practical applications of the dynamic route planning and dispatching problem considered in this thesis, future research could explore its applicability in real-world settings beyond the one considered in the EURO Meets NeurIPS Vehicle Routing Competition. For example, studies could examine the model's effectiveness in environments with rapid changes in demand or in scenarios with fleet size constraints. Furthermore, reusing the developed approach for dynamic slot pricing problems presents an exciting research avenue. From a methodological perspective, the developed ML-CO pipeline raises several foundational questions. For instance, investigating the extent to which the performance of the anticipative strategy, the quality of heuristic solutions, and the predictive capability of the ML model impact performance promises general insights into the principles of ML-CO pipelines. Such an investigation could deepen our understanding of ML-CO pipelines, further advancing the state-of-the-art in this area.

Second, while the integrated charging and service scheduling problem allows continuous-time charging, it is modeled on a discrete time horizon. Relaxing this common assumption may lead to more efficient schedules, potentially further reducing the amount of charging infrastructure required. This likely requires switching to an event-based formulation, but future work can still leverage the label-setting shortest-path algorithm developed for the discrete-time setting. Moreover, extending the problem setting to integrate vehicle assignment-, routing-, or rostering decisions could further improve short- and long-term cost savings. Such extensions can leverage the speedup techniques proposed for the B&P and shortest-path algorithms. Methodologically, further improving the scalability of the B&P algorithm, e.g., through deriving problem-specific cutting planes, developing heuristic approaches, or formulating lower bounds, presents promising research avenues.

Third, with the state-of-the-art in VRPs advancing rapidly, continuously extending `RoutingBlocks` with new algorithms and data structures remains a significant under-taking. Beyond this, further performance improvements, e.g., by embedding a domain-specific modeling language, provide exciting avenues for future development and software-focused research. Ideally, these advancements will help to bridge the performance gap between `RoutingBlocks` and native implementations.