Heuristic and exact algorithms for solving the electric autonomous dial-a-ride problem

Yue Su

▶ **To cite this version:**

Yue Su. Heuristic and exact algorithms for solving the electric autonomous dial-a-ride problem. Operations Research [math.OC]. Université Paris-Saclay, 2023. English. NNT : 2023UPAST092 . tel-04497526v2

HAL Id: tel-04497526
https://theses.hal.science/tel-04497526v2

Submitted on 23 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université PARIS-SACLAY

Heuristic and Exact Algorithms for Solving the Electric Autonomous Dial-A-Ride Problem

Algorithmes heuristiques et exacts pour résoudre le problème du dial-a-ride avec les véhicules électriques autonomes

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 573 interfaces : matériaux, systèmes, usages (INTERFACES)
Spécialité de doctorat : Informatique
Graduate School : Sciences de l'ingénierie et des systèmes. Référent : CentralSupélec

Thèse préparée dans l'unité de recherche **Laboratoire de Genie Industriel (Université Paris-Saclay, CentraleSupélec)**, sous la direction de **Jakob PUCHINGER**, Professeur, le co-encadrement de **Nicolas DUPIN**, Maître de conférences

Thèse soutenue à Paris-Saclay, le 05 juillet 2023, par

Yue SU

Composition du jury

Membres du jury avec voix délibérative

Oualid JOUINI Professeur, CentraleSupélec	Président
An CARIS Professeure, Hasselt University	Rapporteure & Examinatrice
Fabien LÉHUÉDÉ Professeur, IMT Atlantique	Rapporteur & Examinateur
Sophie N. PARRAGH Professeure, Johannes-Kepler Universität Linz	Examinatrice
Claudia ARCHETTI Professeure, ESSEC	Examinatrice

THESE DE DOCTORAT
NNT : 2023UPAST092

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre : Algorithmes heuristiques et exacts pour résoudre le problème du dial-a-ride avec les véhicules électriques autonomes

Mots clés : Le problème du dial-a-ride ; Branch-and-price ; Deterministic annealing ; Véhicules électriques autonomes

Résumé : Cette thèse propose des algorithmes heuristiques et exacts efficaces pour résoudre le problème E-ADARP (Electric Autonomous Dial-A-Ride Problem), qui consiste à concevoir un ensemble d'itinéraires à coût minimum qui répond à toutes les demandes des clients pour une flotte de véhicules électriques autonomes (EAV). L'E-ADARP présente deux caractéristiques importantes : (i) l'emploi des EAVs et une politique de recharge partielle ; (ii) la fonction objectif de somme pondérée qui minimise le temps de trajet total et le temps de trajet total excédentaire de l'utilisateur. Dans cette thèse, nous proposons d'abord un algorithme de recuit déterministe (DA). La recharge partielle (i) est gérée par un schéma d'évaluation d'itinéraire exact de complexité temporelle linéaire. Pour aborder (ii), nous proposons une nouvelle méthode qui permet des calculs efficaces du temps de parcours minimal de l'utilisateur en introduisant une représentation fragmentée des chemins. Pour valider les performances de l'algorithme DA, nous comparons les résultats de notre algorithme aux résultats de l'algorithme Branch-and-Cut (B&C) sur les instances existantes. Notre algorithme DA fournit 25 nouvelles meilleures solutions et 45 solutions égales pour 84 instances existantes. Nous établissons de nouvelles instances avec jusqu'à 8 véhicules et 96 requêtes, et nous fournissons 19 nouvelles solutions pour ces instances. Ensuite, nous présentons un algorithme CG, qui est intégré dans le schéma Branch-and-price (B&P) pour résoudre exactement l'E-ADARP. Notre algorithme CG s'appuie sur un algorithme d'étiquetage efficace pour générer des colonnes avec des coûts réduits négatifs. Dans l'extension des étiquettes, le principal défi consiste à déterminer tous les horaires optimaux de temps de trajet en excès d'utilisateur. Pour résoudre ce problème, nous appliquons la représentation basée sur les fragments et proposons une nouvelle approche pour extraire les fragments des arcs tout en garantissant l'optimalité de l'excès de temps de parcours de l'utilisateur. Nous construisons ensuite un nouveau graphe qui préserve toutes les routes réalisables du graphe original en énumérant tous les fragments réalisables, en les extrayant en arcs et en les connectant les uns aux autres, aux dépôts et aux stations de recharge de manière réalisable. Sur le nouveau graphe, nous appliquons des règles de dominance fortes et des vérifications de faisabilité à temps constant pour calculer efficacement les chemins les plus courts. Dans les expériences, nous résolvons 71 instances sur 84 optimalement, améliorons 30 bornes inférieures et générons 41 nouvelles meilleures solutions sur des instances précédemment résolues et non résolues. Enfin, nous étudions le bi-objectif E-ADARP (BO-EADARP), qui traite le temps de trajet total et le temps de trajet total de l'utilisateur excédentaire comme des objectifs distincts. Pour aborder le BO-EADARP, nous introduisons deux algorithmes de recherche dans l'espace de critères et un algorithme de recherche dans l'espace de décision (l'algorithme BOBP). Nous appliquons ces algorithmes pour résoudre le BO-EADARP sur des instances de petite à moyenne taille avec différentes restrictions de batterie minimales. Parmi les trois algorithmes, l'algorithme BOBP s'avère être le plus efficace. Nous analysons ensuite les solutions efficaces sous différentes contraintes énergétiques. Nos observations révèlent une augmentation notable des temps de trajet totaux pour les solutions efficaces obtenues avec des restrictions d'énergie plus élevées, tandis que les temps de trajet excédentaires totaux correspondants restent stables. Pour chaque niveau de restriction d'énergie, les solutions efficaces obtenues offrent des informations managériales pour les fournisseurs de services rentables et non rentables : un objectif peut être amélioré significativement avec une légère augmentation de l'autre.

Title : Heuristic and exact algorithms for solving the electric autonomous dial-a-ride problem

Keywords : Dial-a-ride problem ; Branch-and-price ; Deterministic annealing ; Electric autonomous vehicles

Abstract : This thesis proposes highly-efficient heuristic and exact algorithms to solve the Electric Autonomous Dial-A-Ride Problem (E-ADARP), which consists in designing a set of minimum-cost routes that accommodates all customer requests for a fleet of Electric Autonomous Vehicles (EAVs). The E-ADARP has two important features : (i) the employment of EAVs and a partial recharging policy; (ii) the weighted-sum objective function that minimizes the total travel time and the total excess user ride time. In this thesis, we first propose a Deterministic Annealing (DA) algorithm to solve the E-ADARP. Partial recharging (i) is handled by an exact route evaluation scheme of linear time complexity. To tackle (ii), we propose a new method that allows effective computations of minimum excess user ride time by introducing a fragment-based representation of paths. To validate the performance of the DA algorithm, we compare our algorithm results to the best-reported Branch-and-Cut (B&C) algorithm results on existing instances. Our DA algorithm provides 25 new best solutions and 45 equal solutions for 84 existing instances. To test the algorithm's performance on larger-sized instances, we establish new instances with up to 8 vehicles and 96 requests, and we provide 19 new solutions for these instances. Then, we present a highly efficient CG algorithm, which is integrated into the Branch-and-price (B&P) scheme to solve the E-ADARP exactly. Our CG algorithm relies on an effective labeling algorithm to generate columns with negative reduced costs. In the extension of labels, the key challenge is determining all excess-user-ride-time optimal schedules to ensure finding the minimum-negative-reduced-cost route. To handle this issue, we apply the fragment-based representation and propose a novel approach to abstract fragments to arcs while ensuring excess-user-ride-time optimality. We then construct a new graph that preserves all feasible routes of the original graph by enumerating all feasible fragments, abstracting them to arcs, and connecting them with each other, depots, and recharging stations in a feasible way. On the new graph, we apply strong dominance rules and constant-time feasibility checks to compute the shortest paths efficiently. In the computational experiments, we solve 71 out of 84 instances optimally, improve 30 previously-reported lower bounds, and generate 41 new best solutions on previously solved and unsolved instances. Finally, we investigate the Bi-objective E-ADARP (BO-EADARP), which treats the total travel time and the total excess user ride time as separate objectives. To tackle the BO-EADARP, we introduce two criterion space search algorithms and a decision space search algorithm (i.e., the BOBP algorithm). In the computational experiments, we apply these algorithms to solve the BO-EADARP on small-to-medium-sized instances with different minimum battery restrictions. Among the three algorithms, the BOBP algorithm proves to be the most efficient. We then analyze the efficient solutions under different energy restrictions. Our observations reveal a noticeable increase in total travel times for the obtained efficient solutions with higher energy restrictions, while the corresponding total excess user ride times remain stable. For each level of energy restriction, the obtained efficient solutions offer managerial insights for profitable and non-profitable service providers : one objective can be improved significantly with a slight increase of the other.

Acknowledgements

On 17th October 2019, I arrived in Paris to begin my PhD studies. At this point, completing my thesis and becoming a doctor in the field of operations research, I would like to express my sincere gratitude to everyone who has supported, encouraged, and accompanied me during my thesis.

When came to my laboratory (Laboratoire de Genie Industriel), I was warmly welcomed by my colleagues and was impressed by the nice atmosphere and the diversity of disciplines. I would like to thank my advisor, Prof. Jakob Puchinger, who has given me this opportunity to work with him and be a part of the LGI OM team. Along with all the guidance and support during my research, he also offered me a research visit to Johannes-Kepler Universität Linz and launched the research collaboration with Prof. Sophie N. Parragh in the third year of my PhD. I feel very lucky to have him as my PhD advisor, and I enjoyed every discussion with him. I would also like to thank my co-advisor Dr. Nicolas Dupin, who gave me guides at the beginning stage of each work. Finally, I would like to express my gratitude to Prof. Sophie N. Parrgah for introducing me to her group and welcoming me during my research visit. I was deeply impressed by the warm atmosphere in her group, and we have various activities every week (e.g., hiking, visiting, eating...). At Linz University, I met many interesting and dynamic colleagues (Markus, Martin, Manuel, Xeni, An, Sabine, Elli, Kubra, Oryan) and had many precious experiences. During the collaboration, Sophie gave me a lot of constructive comments and helped me to enhance my work. Her expertise and patience have been invaluable to me and have played a crucial role in the success of this thesis.

My thesis is built upon the solid foundation of my family and friends. I am deeply thankful to my family, who are always my strongest backup force. Three months after I began my PhD, the covid started, and I could not go home to see my family physically. However, I have video phone calls every week with my mom, and she always encourages me when I feel down. During the phone call, I also chatted with my dad and Grandma Liu. Each time after the call, I came back to my studies with full energy. I would also like to thank my boyfriend Hui Yang, who gave me many inspirations in my studies. He always helped me without asking for any returns. Without his support, I could not have had such a fruitful and productive PhD life. My thesis was also composed of numerous lucky encounters. Many thanks to Delphine and Suzanne for all the administrative help. I was so happy to meet Tarek, who is also a big fan of manga and games. I am so excited and happy to share with you every manga and game I like. Thanks to all my friends in Anthropolis chair, most especially to Tjark and Mariana. Thanks to my best friend Yuting, we have

known each other there has been almost ten years. Very lucky that we were both accepted at the same time by the PhD program of CentraleSupélec and have you accompanied and supported me during my thesis. Thanks to all my friends and colleagues at LGI, who have greatly enriched my experiences.

Finally, I would like to thank all the jury members, Prof. Fabien LÉHUÉDÉ, Prof. An CARIS, Prof. Claudia ARCHETTI, Prof. Sophie N. PARRAGH, and Prof. Oualid JOUINI, who have provided me with a lot of valuable feedback and suggestions. Their insights and guidance were instrumental in helping me to improve the manuscript and develop future directions.

Now, a new adventure as a postdoc researcher is in front of me. Here, I would like to cite a famous Chinese verse to close my PhD chapter: 长风破浪会有时，直挂云帆济沧海！ (A time will come to ride the wind and cleave the waves, I'll set my cloud-white sail and cross the sea which raves.)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Dial-A-Ride Problem (DARP) consists in designing minimum-cost routes by scheduling a fleet of vehicles to serve a set of customers who specify their origins and destinations (Cordeau & Laporte, 2007). For each customer request, a time window is defined on either the origin or the destination. The DARP was first introduced in the context of providing door-to-door service for handicapped individuals, e.g., Madsen et al. (1995); Toth & Vigo (1996). In recent years, the concept of the DARP has been extended to adopt requests from normal individuals and provide them with ride-sharing services. Many demand-responsive systems have been constructed, such as the mobile-based app of BlaBlaCar in France and Didi Hitch in China (Jin et al., 2018). With the booming of on-demand ride-sharing services, considerable attention has arisen to solving the DARP and its variants (Cordeau & Laporte, 2007; Ho et al., 2018). The DARP is a generalization of several NP-hard problems such as the Pickup and Delivery Vehicle Routing Problem (PDPVRP) and the Vehicle Routing Problem with Time Windows (VRPTW) and is therefore very difficult to solve to optimality using exact methods. Consequently, only a few studies have proposed exact methods, e.g., Cordeau (2006); Braekers et al. (2014); Gschwind & Irnich (2015); Ropke et al. (2007); Parragh (2011); Braekers & Kovacs (2016); Qu & Bard (2015) and most studies propose heuristic and metaheuristic methods that return high-quality local optimum quickly (e.g., Cordeau & Laporte (2003); Ropke & Pisinger (2006); Parragh et al. (2010); Masmoudi et al. (2017); Detti et al. (2017)). The DARP is even more challenging than the PDPVRP and the VRPTW, as user inconvenience needs to be considered while minimizing the operational cost (Cordeau & Laporte, 2003). The classical DARP model imposes a maximum user ride time constraint for every user request to maintain a certain level of service quality. Due to this constraint in combination with time windows, scheduling service start times as early as possible does not necessarily result in a feasible schedule for a given sequence of pickup and delivery locations, given that one exists. On the contrary, allowing delays in the service start time may help to eliminate unnecessary waiting time for succeeding nodes and, as such, reduce the user ride time. Heuristic solution methods for the DARP usually invoke the "*eight-step*" procedure of Cordeau & Laporte (2003), which composes a feasible schedule with the latest possible service start time at the origin depot and subsequently delays service start times at pickup nodes

using the notion of forward time slack (Savelsbergh, 1992) while respecting maximum user ride time constraints. As a result, the complexity of the route evaluation in the DARP rises substantially.

The Electric Autonomous DARP (E-ADARP) was first introduced by Bongiovanni et al. (2019) and has been developed to incorporate various real-life settings. Different from the classical DARP, which assumes vehicles to have homogeneous vehicle capacity (e.g., Cordeau (2006); Cordeau & Laporte (2003)), the E-ADARP accounts for heterogeneous vehicles with different capacities to better accommodate diverse user requests. Furthermore, the E-ADARP includes multiple origin and destination depots in the graph, enabling vehicles to start and end their routes at different depots. Apart from these fundamental characteristics, the E-ADARP also considers practical aspects associated with the electric and autonomous nature of the vehicles. Firstly, vehicles are allowed to perform partial recharging at recharging stations, which is a more feasible alternative to full recharging (e.g., Schneider et al. (2014); Hiermann et al. (2016)). This strategy grants vehicles greater flexibility in charging operations and has demonstrated efficacy in enhancing solution quality (Keskin & Çatay (2016)). Secondly, given that the E-ADARP employs autonomous vehicles that need to be continuously relocated during service, it removes the need to predefine destination depots and enables non-stop vehicle operations. This characteristic is more practical and differentiates the E-ADARP from most of the DARP literature (e.g., Braekers et al. (2014); Parragh (2011)), where maximum route duration constraints are considered, and destination depots are predefined.

The objective of the E-ADARP is to design a set of minimum-cost routes for a fleet of Electric Autonomous Vehicles (EAVs) by scheduling them to provide ride-sharing services for customers that specify their origins and destinations. Although the E-ADARP shares some of the constraints of the classical DARP (e.g., maximum user ride time, time window constraints), the E-ADARP is different from the classical DARP in two aspects: (i) the employment of EAVs and a partial recharging policy, and (ii) a weighted-sum objective that minimizes both total travel time and total excess user ride time; The first aspect (i) requires checking battery feasibility for a given route and implies other important features of the E-ADARP: (a) partial recharging is allowed en route, and (b) the maximum route duration constraints no longer exist due to the autonomy of vehicles. Allowing partial recharging introduces a trade-off between the time window and battery constraints: although longer recharging extends the driving range, it may also lead to time-window infeasibility for later nodes. The second aspect (ii) allows quantifying the user inconvenience directly. By incorporating excess user ride time into the objective, the E-ADARP offers the opportunity to enhance service quality by minimizing the excess user ride time without introducing additional operational costs. However, solving the E-ADARP becomes more complex, as it requires determining schedules with minimal total excess user ride time for a feasible solution. Other problem-specific constraints also increase the complexity of solving the E-ADARP. These constraints include a minimum battery level that must be maintained at the end of the route as well as limited visits to each recharging station.

In this thesis, we propose efficient heuristic and exact algorithms (i.e., Deterministic Annealing and Branch-and-price algorithms) to solve the static version of the E-ADARP. In solving the static E-ADARP, we develop efficient

methods that can exactly determine excess-user-ride-time optimal schedules for a given E-ADARP path. These methods are interesting for decision-makers seeking to enhance service quality by strategically scheduling vehicles. After validating the efficiency of our proposed algorithms, we extend our investigation to the bi-objective version of the E-ADARP. This variant considers total travel time and total excess user ride time as two distinct objectives. Our objective in this part is to analyze the trade-off between operational costs, represented by total travel time, and service quality, measured by total excess user ride time. By examining this trade-off, we aim to provide a full picture of this trade-off. This analysis is beneficial for decision-makers in both profitable service providers (e.g., Uber and Didi) and non-profitable organizations (e.g., the Red Cross). Managerial insights from the analysis enable decision-makers to make more informed and reliable choices based on their specific priorities.

The remainder of this chapter is organized as follows. In Section 1.1, we present a comprehensive literature review on (1) exact and approximate approaches that solve the classical DARPs, (2) related literature of the DARP with Electric Vehicles (EVs) and Electric Vehicle Routing Problems (E-VRPs), (3) feasibility checking in DARPs and excess user ride time minimization, and (4) methods for solving multi-objective problems. Then, we conclude existing research gaps and the motivation of the thesis. Section 1.2 discusses the challenges in solving the E-ADARP and highlights the contributions of the thesis. Section 1.3 outlines the structure of the thesis and the organization of the following chapters.

1.1 Literature Review

The E-ADARP can be regarded as a combination of the classical DARP and the Electric Vehicle Routing Problems (E-VRPs). However, it is distinct from these two contexts in the following aspects:

1. the application of Electric Autonomous Vehicles (EAVs) in the vehicle fleet and partial recharging performed at recharging stations;

2. a weighted sum objective function minimizing total travel time and total excess user ride time.

This section first reviews the representative works of exact and heuristic solution methods to tackle the classical DARP and its variants. Then, we systematically review the literature related to DARPs with EVs and E-VRPs. More specifically, we focus on reviewing the works that consider the partial recharging policy. The third part of this section is an overview of feasibility checking and total/excess user ride time minimization in the DARPs. We first review representative works on the feasibility check for a given DARP route, especially those that focus on evaluating maximum user ride time feasibility. We then review DARP-related articles that specifically focus on user ride time minimization. The last part of this section is a comprehensive review of multi-objective optimization, and we mainly focus on state-of-the-art methods of bi-objective optimization. Related works on multi-objective DARP are analyzed. This section is closed with the conclusion of the review and the motivation of the thesis.

1.1.1 Exact and approximate approaches to tackle DARPs

There are two main streams of solution methods to solve the DARP, they are exact and approximate approaches. Exact approaches mainly take the structure of the branch-and-bound (B&B) tree to obtain the global optimum. Due to the complexity of the problem as well as the solving mechanism of the B&B (i.e., obtain global optimum by enumerating all possible solutions), exact approaches can only solve instances of limited size to optimality within considerable computational time. Considering the $\mathcal{NP}$-hardness of DARPs, most research attention has been concentrated on developing highly-efficient approximate approaches such as heuristic and meta-heuristic algorithms to solve larger-sized instances and return high-quality local optimums within reasonable computational time.

Exact algorithms in the literature of DARPs can be classified as Branch-and-Cut (B&C), Branch-and-Price (B&P), and Branch-and-Price-and-Cut (B&P&C) algorithms. The basic idea of a B&C algorithm is to generate valid cutting planes to tighten the LP relaxations of the problem in the B&B tree. These additional cuts lead to stronger lower bounds and increase the chances that an integer solution is found. The first B&C algorithm to solve the classical DARP is proposed by Cordeau (2006), where the author presented a three-index compact formulation for the DARP and introduced several families of valid inequalities. The proposed B&C algorithm can solve instances with up to 4 vehicles and 48 requests. Based on the three-index formulation in Cordeau (2006), Ropke et al. (2007) introduced two tighter formulations of the DARP. Three new classes of valid inequalities, as well as some previously identified valid inequalities for the DARP and the VRPTW, are added. The largest instances that can be solved optimally with the proposed B&C algorithm contains 8 vehicles and 96 requests. These identified valid inequalities in Cordeau (2006); Ropke et al. (2007) have also been applied in the following studies, e.g., Parragh (2011); Braekers et al. (2014); Braekers & Kovacs (2016); Liu et al. (2015). These works define new variants of the DARP and supplement problem-specific cuts to solve the defined problem variants. Different from B&C algorithms, the basic idea of the B&P algorithm is to first reformulate the compact formulation into a master problem (usually formulated as set partitioning problem) and pricing subproblems (can be formulated as shortest path problem with resource constraints) by Dantzig-Wolfe decomposition (Lübbecke & Desrosiers (2005)). At each node of the B&B tree, the CG algorithm is then invoked to obtain tightened lower bounds of the original problem by iteratively solving the restricted master problem and pricing subproblems. When applying the CG algorithm, the solving efficiency of pricing subproblems plays the key role in the CG efficiency and the overall performance of a B&P algorithm. In the literature, pricing subproblems are usually solved by dynamic programming (e.g., Garaix et al. (2011)). Also, pricing subproblems can be solved with a hybridized framework (e.g., Parragh et al. (2012, 2015)), where heuristic and exact algorithms are both used to find negative-reduced-cost columns. Finally, the B&P&C algorithms take the structure of B&P and integrates valid cutting planes in the B&P tree to further enhance the obtained lower bounds. A few existing literature implement the B&P&C algorithms to solve the DARP and its variants, e.g., Qu & Bard (2015); Gschwind & Irnich (2015); Ropke & Cordeau (2009), where they observe that adding cuts can effectively reduce the number of nodes

explored in the B&B tree.

Heuristic and metaheuristic methods are extensively used to solve the DARP and its variants in the literature. Among literature that proposes heuristic or metaheuristic methods, local search-based metaheuristics take a large portion. The idea of local search-based metaheuristics is to generate new solutions by exploring the neighborhood of the current solution through local search operators. By designing efficient local search operators, these metaheuristics can effectively explore the search space and identify high-quality solutions in short computational time. The most-used local-search-based metaheuritics in the literature are: tabu search (TS) (e.g., Cordeau & Laporte (2003); Kirchler & Calvo (2013); Detti et al. (2017); Paquette et al. (2013); Ho & Haugland (2011); Guerriero et al. (2013)), simulated annealing (SA) and deterministic annealing (DA) (e.g., Braekers et al. (2014); Su et al. (2023); Reinhardt et al. (2013)), variable neighborhood search (VNS) (e.g.,Parragh et al. (2010, 2012, 2015); Parragh (2011); Detti et al. (2017)), and large neighborhood search (LNS) and adaptive large neighborhood search (ALNS) (e.g., Ropke & Pisinger (2006); Gschwind & Drexl (2019); Molenbruch et al. (2017); Braekers & Kovacs (2016); Masson et al. (2014); Qu & Bard (2013)). Recently, a growing number of researches consider the hybridization of metaheuristic algorithms with other methods such as mathematical programming approaches or other metaheuristic algorithms. Through combining different techniques appropriately, the performance of individual techniques is enhanced by leveraging the strengths of each. For example, some researches hybridize one metaheuristic into another (e.g., Masmoudi et al. (2016, 2017); Chevrier et al. (2012)). In these hybrid algorithms, local search-based metaheuristics (e.g., TS and SA) are combined with population-based metaheuristics (e.g., genetic algorithm (GA)). These hybrid algorithms take advantages of both types of algorithms by using local search to explore the search space around a single solution and population-based metaheuristics to evolve the population over time. By combining these two approaches, hybrid algorithms can achieve better performance than using either algorithm alone. Another way to hybridize metaheuristics is to execute them sequentially. An example is presented in Parragh et al. (2009), where the authors apply a path relinking algorithm after a VNS algorithm. The idea of path relinking algorithm is to combine two elite solutions (e.g., the solutions from VNS, one solution called the “start solution”, the other is “target solution”) and to generate a even better solution from these two solutions.

1.1.2 Related literature of DARPs with EVs and E-VRPs

Several articles have investigated the impact of EVs on the DARP and Masmoudi et al. (2018) is the first work that introduces DARP with EVs. In their work, EVs are recharged through battery swapping and are assumed to have a constant recharging time. The authors use a realistic energy consumption model to formulate the problem and introduce three enhanced Evolutionary VNS (EVO-VNS) algorithm variants, which can solve instances with up to three vehicles and 18 requests. Bongiovanni et al. (2019) considers EAVs in the DARP and introduces the E-ADARP. Partial recharging is allowed when vehicles visit recharging stations, and the authors impose a minimum battery level constraint for the vehicle’s State of Charge (SoC) at the destination depot. The minimum battery level is formulated

as γQ, where γ is the ratio of the minimum battery level to total battery capacity, and Q is the total battery capacity. Three different γ values are analyzed, i.e., $\gamma \in \{0.1, 0.4, 0.7\}$, meaning that 10%, 40%, and 70% of the total battery capacity must be maintained at the destination depot. Solving the problem becomes more difficult when γ increases. The authors formulate the problem into a three-index and a two-index model and introduce new valid inequalities in a Branch-and-Cut (B&C) algorithm. When $\gamma = 0.1, 0.4$, the proposed B&C algorithm obtains optimal solutions for 42 out of 56 instances. However, when $\gamma = 0.7$, the B&C algorithm cannot solve 9 out of 28 instances feasibly, even with a two-hour run time. The largest instance that can be solved optimally by the B&C algorithm contains 5 vehicles and 40 requests. Recently, Bongiovanni et al. (2022b) have proposed a Machine Learning-based Large Neighborhood Search (MLNS) to solve the dynamic version of the E-ADARP. The proposed approach is a two-phase metaheuristic that sequential solves static E-ADARP subproblems. However, its performance on the previously defined static E-ADARP instances is not reported. Different from our algorithm, the authors focus on selecting destroy-repair operators at each iteration by a machine learning approach, which is trained offline on a large dataset produced through simulation.

The E-VRP was first introduced by Conrad & Figliozzi (2011), and extensive works have been conducted in the field of E-VRPs in recent years, e.g., Erdoğan & Miller-Hooks (2012); Schneider et al. (2014); Goeke & Schneider (2015); Hiermann et al. (2016, 2019). Among them, Erdoğan & Miller-Hooks (2012) is the first to propose a Green VRP (G-VRP) using alternative fuel vehicles. These vehicles are allowed to visit a set of recharging stations during vehicle trips. The authors adapt two constructive heuristics to obtain feasible solutions and they further enhance these heuristics by applying local search. However, the proposed model does not consider capacity restrictions and time window constraints. Schneider et al. (2014) propose a more comprehensive model named the Electric Vehicle Routing Problem with Time Windows (E-VRPTW). They extend the work of Erdoğan & Miller-Hooks (2012) by using electric vehicles and considering limited vehicle capacity and specified customer time windows. They apply a Variable Neighborhood Search (VNS) algorithm hybridized by Tabu Search in local search to address E-VRPTW. The recharging stations are inserted or removed by a specific operator, and the recharged energy is assumed to be linear with the recharging time. They apply a full recharging policy on each visit to a recharging station. All the vehicles are assumed to be identical in terms of vehicle and battery capacity. Goeke & Schneider (2015) extend the homogeneous E-VRPTW by considering a mixed fleet of electric and conventional vehicles. A realistic energy consumption model that integrates speed, load, and road gradient is employed. To address the problem, they propose an ALNS algorithm using a surrogate function to evaluate violations efficiently. Hiermann et al. (2016) extend the work of Goeke & Schneider (2015) by taking into account the heterogeneous aspect (i.e., fleet composition). They solve the problem by ALNS and determine the positions of recharging stations via a labeling algorithm. The recharging policy considered is also full recharging with a constant recharging rate. Hiermann et al. (2019) extend their previous work (Hiermann et al., 2016) by considering partial recharging for a mixed fleet of conventional, plug-in hybrid, and electric vehicles. The engine mode selection for plug-in hybrid vehicles is considered as a decision variable in their

study. A layered optimization algorithm is presented. This algorithm combines labeling techniques and a greedy route evaluation policy to calculate the amount of energy required to be charged and determine the engine mode and energy types. This algorithm is finally hybridized with a set partitioning problem to generate better solutions from obtained routes. Except for Hiermann et al. (2019), many works also relax the full recharging assumption to a partial recharging policy, e.g.,(Bruglieri et al., 2015; Keskin & Çatay, 2016; Desaulniers et al., 2016; Duman et al., 2021; Ceselli et al., 2021). The majority of these articles develop meta-heuristic algorithms to solve the problem, e.g., Felipe et al. (2014); Hiermann et al. (2019); Montoya et al. (2017); Froger et al. (2017), while some of them propose exact methods. The representative work is Desaulniers et al. (2016), where the authors investigate four E-VRP variants: E-VRP with full recharging plus single/multiple visits to recharging stations and E-VRP with partial recharging plus single/multiple visits to recharging stations. For each variant, customized mono- and bi-directional labeling algorithms are proposed to solve the CG subproblems within a B&P framework. More recently, Lam et al. (2022) investigates a more practical case of E-VRPTW in which the availability of chargers at the recharging stations is considered. They propose a B&C&P algorithm that is capable of solving instances with up to 100 customers.

1.1.3 Feasibility checking in DARPs and related literature that minimizes total or excess user ride time

As for feasibility checks for a given DARP route, the "*eight-step*" method introduced in Cordeau & Laporte (2003) is extensively applied. However, "*eight-step*" method has a worst-case time complexity of $\mathcal{O}(r^2)$ (r is the number of vertices in the route) and would generate considerable computational time in the case of voluminous feasibility checks are needed (e.g., when applying a heuristic method). To improve the efficiency in feasibility checking for a DARP route, several works have been conducted. In Hunsaker & Savelsbergh (2002), the authors propose a feasibility-checking heuristic of linear time complexity in the DARP with maximum waiting time constraints. Their proposed heuristic constitutes three passes to check the feasibility. However, this method cannot guarantee finding a feasible schedule in case one exists. In this case, this algorithm returns an incorrect infeasibility declaration. This issue is handled in Haugland & Ho (2010), where an exact procedure of feasibility checking for a DARP route is proposed. The worst-time complexity of their procedure is of $\mathcal{O}(r \log r)$. Tang et al. (2010) proposes an algorithm of $\mathcal{O}(r^2)$ worst-time complexity by gradually constructing the schedule on the original route. Then, Firat & Woeginger (2011) propose another idea to handle the feasibility issue. They rewrite the scheduling problem considered in the DARP as a system of difference constraints. Then, the feasibility checking is converted to decide whether a digraph associated with the defined system has a negative-weight cycle. The cycle detection process can be fulfilled in linear time after reformulating such a system over an appropriate set of variables. Gschwind (2019) proposes a route feasibility check procedure for the synchronized pickup and delivery problem (SPDP), which was first introduced by Gschwind (2015). Besides an upper bound on ride time, the SPDP has imposed a lower bound on it. Therefore, the

SPDP has minimum and maximum ride time constraints. In Gschwind (2019), the author adapted the algorithms of Tang et al. (2010) and Firat & Woeginger (2011) in the feasibility checking of an SPDP route. From the computational results, the adapted algorithm of Tang et al. (2010) outperforms the one from Firat & Woeginger (2011) averagely. Recently, Gschwind & Drexl (2019) proposes an ALNS algorithm equipped with an amortized constant time to check the insertion feasibility of a request into a given feasible route. They make use of the auxiliary data calculated from preprocessing work in all feasibility checks. Consequently, the checking is independent of the number of requests in the route and only takes only constant time for a given route. However, the preprocessing work seems to be time-consuming and has a time complexity of $\mathcal{O}(r^3)$ for a route of r vertices to generate necessary auxiliary data. To conclude, many literature has been proposed to handle the feasibility issue in a more efficient way for a given complete route. However, in the extension of a given partial path, these above literature is not appropriate for checking the extension feasibility. Gschwind & Irnich (2015) is the only work that handles the time-window and ride-time constraints jointly in the labeling algorithm for solving the CG subproblems. This is the first time that the schedule feasibility issue is tackled in the extension of a partial path. In their proposed labeling algorithm, they introduce additional resource attribute named "latest possible delivery time", which is formulated a function of the service start time t at the current node where all possible time schedules of all open requests along the partial path are considered. This labeling algorithm results in a labeling-based feasibility check of $\mathcal{O}(rM)$ time complexity, where M is the maximum number of open requests along a feasible path.

All the above-mentioned literature only improves the feasibility checking efficiency of maximum user ride time constraints for a given DARP route/path. However, these state-of-the-art methods cannot guarantee to minimize the excess user ride time for a given DARP route/path. In fact, to minimize excess user ride time for a given DARP/path is a more intricate task, as we need to consider all feasible schedules and pick up the ones who have the minimum excess user ride time. There are several examples where a service-quality-oriented objective (e.g., user ride time) is considered in the context of DARP (e.g., Parragh et al. (2009); Parragh (2011); Lehuédé et al. (2014); Molenbruch et al. (2017); Bongiovanni et al. (2022a)). Among them, Lehuédé et al. (2014) considers the average excess user ride time as one of the objectives and Parragh et al. (2009); Molenbruch et al. (2017); Bongiovanni et al. (2022a) consider total user ride time/total excess user ride time as an objective to be minimized. In Lehuédé et al. (2014), the authors apply a scheduling algorithm based on the "eight-step" method for generated routes. In the work of Parragh et al. (2009), a two-phase heuristic method is developed. A set of efficient solutions is constructed, minimizing a weighted sum of total distance traveled and mean user ride time under different weight combinations. In the route evaluation, the authors point out that the "*eight-step*" method of Cordeau & Laporte (2003) does not aim to minimize the total user ride time. An increase in user ride time may happen when delaying the service start time at destination nodes. Therefore, they improve the original scheme of the "*eight-step*" method by adapting the computation of forward time slack to avoid any increase in excess user ride time for requests served on a route. The resulting scheme is more restrictive in terms of feasibility and may lead to incorrect infeasibility declaration. This drawback is tackled in the

scheduling heuristic proposed by Molenbruch et al. (2017). The heuristic starts by constructing a schedule (which may be infeasible) by setting the excess ride time of each request to its lower bound. Then, it gradually removes the infeasibility by shifting the service start time at some nodes while minimizing excess user ride time. However, the developed scheduling procedures in Parragh et al. (2009) and Molenbruch et al. (2017) are not proven optimal to minimize user ride time for a given route. Bongiovanni et al. (2022a) first proposes an exact scheduling procedure that can minimize the excess user ride time for a path without charging stations in polynomial time. Then, the authors extend the proposed scheduling procedure in the E-ADARP by integrating a battery management heuristic. However, the obtained schedules for an E-ADARP route are no longer exact as the excess-time optimal schedules may not be battery-feasible. The reported results show that on the investigated instances, the proposed scheduling procedure does not produce incorrect infeasible declarations, while others (i.e., Cordeau & Laporte (2003), Parragh et al. (2009)) do. To the best of our knowledge, no work in the literature can handle excess user ride time minimization exactly for a given E-ADARP route or in the extension of an E-ADARP partial path.

1.1.4 Related literature of bi-objective optimization

In the context of solving multiobjective mixed integer programming (MOMIP) problems, exact algorithms can be divided into two classes: criterion space search algorithms (e.g., (Haimes, 1971; Chalmet et al., 1986; Ralphs et al., 2006; Parragh et al., 2009; Boland et al., 2015; Glize et al., 2022)) and decision space search algorithms (e.g., (Mavrotas & Diakoulaki, 1998; Masin & Bukchin, 2008; Sourd & Spanjaard, 2008; Vincent et al., 2013; Stidsen et al., 2014; Parragh & Tricoire, 2019)). In this part, we focus on reviewing methods to solve bi-objective mixed integer programming (BOMIP) problems.

Criterion space search algorithms work in the objective space. Pareto optimal solutions are found in the process of solving a sequence of single-objective problems with single-objective algorithms (e.g, (Boland et al., 2015)). One of the most popular criterion space search algorithms to handle bi-objective optimization problem is the ϵ-constraint method, which was first introduced by Haimes (1971). The main idea of the ϵ-constraint method is to keep one objective in the objective function and formulate another objective as a constraint binding with ϵ values. In every iteration, the mono-objective problem is solved and ϵ values are updated to improve the quality of the solution (i.e., the second objective values). This method has the advantage of easy implementation while being very effective, and has therefore been widely applied in the literature to solve bi-objective integer programs (e.g., Kirlik & Say1n (2014); Lokman & Köksalan (2013); Laumanns et al. (2006); Ozlen et al. (2014)). Another method to find all non-dominated points in criterion space is called two-phase method (e.g.,Visée et al. (1998),Pedersen et al. (2008),Przybylski et al. (2008),Przybylski et al. (2010),Özpeynirci & Köksalan (2010), Parragh et al. (2009)). The two-phase method was originally introduced by Ulungu (1993) to solve a bi-objective assignment problem by dividing the search for efficient points in criterion space into two phases. The first phase of the two-phase method corresponds to find all extreme supported efficient points. Usually, the weighted-sum algorithm (also called “dichotomic method”) of Aneja & Nair

(1979) is used in this stage. After finding all extreme supported efficient points, the second phase aims to find other efficient points (including non-extreme supported efficient and non-supported efficient points)in the triangles defined by two consecutive extreme supported efficient points. Recently, Boland et al. (2015) introduced a balanced box method to generate all efficient points of a bi-objective integer program by iteratively exploring rectangle search areas defined by newly-found efficient points in the criterion space. Another contribution of this work is the introduction of a bi-directional ϵ-constraint method. In the bi-directional ϵ-constraint method, previously generated feasible solutions that are provided to the integer programming in the later iterations are getting better and better, which benefits the computational efficiency compared to the single-directional one. They also provide several enhancements to accelerate the balanced box method as well as other criterion space search algorithms (i.e., those in Haimes (1971); Ralphs et al. (2006); Chalmet et al. (1986)). From the experimental results, the proposed balanced box method outperforms the bi-directional ϵ-constraint method, the enhanced augmented weighted Tchebycheff method, and the enhanced perpendicular search method with regards to the computational time and the percentage of solving instances. More recently, Glize et al. (2022) proposes an ϵ-constraint column generation-and-enumeration algorithm. Five acceleration mechanisms are introduced to speed up the solving process of each single-objective problem with the column generation-and-enumeration algorithm. The resulting algorithm is proved to be more efficient than the bi-objective branch-and-price algorithm proposed in Parragh & Tricoire (2019) in solving Bi-Objective Team Orienteering Problem (BOTOPTW) with Time Windows. This method is also the first exact method that solves the Bi-Objective Vehicle Routing Problem with Time Windows (BOVRPTW).

Decision space search algorithms work in the space of decision variables. These algorithms can be regarded as generalizations of Branch-and-Bound (B&B) algorithms in the context of MOMIP, where a bound set instead of a single numerical value is generated when solving each node. As bounding procedures is the key ingredient of B&B algorithms, state-of-the-art algorithms mainly focus on this element to enhance the overall efficiency of algorithms. In the works of Mavrotas & Diakoulaki (1998, 2005), the authors provided B&B algorithms to find efficient solutions for multiple objectives mixed 0-1 problem (MOM01P), where they fathom a node if the ideal point of the node (they work on a maximization problem) is dominated by some elements in the lower bound set that is composed of non-dominated points. Vincent et al. (2013) enhance the bounding procedure of Mavrotas & Diakoulaki (1998, 2005) by comparing the bound set on each node of the B&B tree instead of the ideal point. Sourd & Spanjaard (2008) defines separating hyperplane h between upper and lower bound sets to judge whether a node can be fathomed in a general B&B framework. The main idea is first to construct a set L' which is dominated by the lower bound set L. Then, L' is used to define the function h and to check whether a node can be fathomed. Another idea of bounding procedure in the state-of-the-art B&B algorithms (e.g, Masin & Bukchin (2008)) is to calculate a single lower bound with a surrogate objective function and decide whether the B&B node can be fathomed with regards to the numerical value of this lower bound. However, this idea is only illustrated via a small scheduling example and no computational results have been reported. Based on all the above literature, Stidsen et al. (2014) propose a B&B

algorithm that can find a complete set of efficient points for a certain class of BOMIP problems. In the bounding procedure, the authors propose to partition objective space into slices (this is done by adding constraints), then explore each slice independently. From their computational results, they observed that this strategy could allow more nodes to be fathomed and will not significantly increase the computational time. Also, they propose to improve the branching procedure (called "Pareto branching") by using bounds from enumerated feasible solutions to create nodes. Extending the idea of Pareto branching from Stidsen et al. (2014), Parragh & Tricoire (2019) introduced a generic B&B algorithm based on a problem-independent branching rule to solve the BOTOPTW. In the B&B algorithm, they calculate the lower bound set at each node of the B&B search tree by column generation. Several improvements derived from the integrality of objective functions are proposed to accelerate the algorithm. In the computational experiments, the proposed B&B algorithm is compared to several criterion space search algorithms (i.e., algorithms of Haimes (1971), Boland et al. (2015)) and has been proven to be the most efficient method among all considered algorithms in solving the BOTOPTW.

In the context of multi-objective DARP, a few works have been conducted to analyze the trade-off between user inconvenience and operational cost. In the work of Parragh et al. (2009), a two-phase heuristic method is developed. A set of efficient solutions is constructed, minimizing a weighted sum of total distance traveled and mean user ride time for different weight combinations. To validate the performance of the proposed heuristic, $\epsilon-$constraint method is used to generate the complete set of efficient points on small instances. The approximated Pareto Front generated by a heuristic method is compared to the generated set of efficient points. The proposed two-path method is shown to be efficient in generating high-quality approximations of the true Pareto front. Following Parragh et al. (2009), Parragh (2011) further considered a weighted-sum objective function consisting of the total routing cost and wait time for all vehicles with passengers aboard. As in this paper, the objective function included a time-dependent criterion, which resulted in a highly-constrained model, medium-sized instances with up to 4 vehicles and 40 customers can be solved. Paquette et al. (2013) integrate tabu search with a reference point method, computing distances to an ideal point overall objectives. A set of supported efficient solutions is constructed by applying a weighted-sum objective function in which the search is guided by adapting weights dynamically. Total routing cost, user waiting time, and user ride time are minimized. Recently, in the work of Molenbruch et al. (2017), the total user ride time is set as the second objective and a new scheduling heuristic is proposed to construct the schedule that minimizes the total user ride time. The multi-directional local search embedded variable neighborhood descent framework is proposed. The problem tackled in this work is characterized by combination restrictions, which prevent some users from being transported together and limit the set of drivers to which users may be assigned.

1.1.5 Conclusion and motivation

From our review, our first conclusion is that the effect of electric vehicles on the DARP has rarely been investigated in the previous literature. Bongiovanni et al. (2019) is the only work that conducts a comprehensive study and proposes

an exact method (i.e., the B&C algorithm) to optimize the static version of the DARP with EVs. However, the proposed B&C algorithm requires important run-times and has difficulties obtaining optimal/feasible solutions within the given time limit for medium-to-large-sized instances, which limits its application in practice. The second limitation from Bongiovanni et al. (2019) is that there are significant gaps between their best-found upper and lower bounds for medium-to-large-sized instances.

Our second conclusion is that the minimization of total/excess user ride time has rarely been studied in the literature. This issue is related to optimizing a scheduling problem that is derived from the DARP. Bongiovanni et al. (2022a) seems to be the only work that proposes an exact scheduling procedure that obtains the excess-user-ride-time optimal schedule for a DARP route. However, their extended scheduling procedure to handle the E-ADARP turns out to be a heuristic. Hence, the extended scheduling procedure cannot guarantee to obtain the excess-user-ride-time optimal schedule for an E-ADARP route. In addition, none of the existing literature is appropriate to derive exact and efficient excess-user-ride time optimization in the extension of an E-ADARP partial path. Therefore, two limitations from the literature are observed: (1) no existing work can calculate exactly the minimum excess user ride time for a given E-ADARP route; (2) no existing work can guarantee excess-user-ride-time optimality in the extension of an E-ADARP partial path.

Our last conclusion is that few works have been conducted in the context of multiple-objective DARP, and no work exists for the E-ADARP considering multiple objectives. In the transportation of users, there is a fundamental trade-off between service quality and operational efficiency: users may not be transported directly to their destinations in the optimal operational plan. With the development of ride-sharing services, investigating this fundamental trade-off becomes crucial in finding a good balance between human and economic perspectives. Therefore, we conclude that it is of great value to consider the conflicting interests of service providers and users in the objective function and investigate the Bi-objective E-ADARP (hereafter BO-EADARP). Solving the BO-EADARP would help to provide decision-makers with the full picture of possible efficient transportation plans and select one(s) according to their priorities and preferences.

The above limitations in the literature motivate us to:

1. Develop an exact and efficient method to calculate minimum excess user ride time for a given E-ADARP route;

2. Based on the above point, propose an efficient metaheuristic algorithm that can provide high-quality solutions for E-ADARP instances within reasonable computational time;

3. Develop an exact and efficient method that can guarantee excess-user-ride-time optimality in the extension of an E-ADARP partial path;

4. Based on the above point, propose an efficient CG algorithm, including a labeling algorithm for solving the pricing problem to provide tighter lower bounds and improved upper bounds;

5. Based on the developed CG algorithm, develop the B&P algorithm to obtain optimal solutions for the E-ADARP;

6. Extend the single-objective B&P algorithm to solve the BO-EADARP and analyze the generated Pareto optimal (also called “efficient”) solutions.

1.2 Challenges and Contributions

Several challenges exist in the way of developing highly-efficient heuristics and exact methods to solve the E-ADARP. These challenges derive from the problem-specific features of the E-ADARP (i.e., partial recharging, excess-user-ride-time minimization) as well as the nature of methods (i.e., local search, label extension). Here we summarize these challenges as follows.

1. The efficiency of a metaheuristic largely depends on its neighborhood search mechanisms, which perform a large number of evaluations. In the case of the DARP, these are route evaluations and cost computations. These two tasks are more complicated in the E-ADARP than in the DARP, as we allow partial recharging and minimize total excess user ride time for a given route. Existing scheduling procedures only obtain the approximation of minimum excess user ride time, which may deteriorate the solution quality and mislead search direction. Moreover, these procedures are time-consuming when applied in a metaheuristic as they are usually of quadratic time complexity and may introduce numerous repeated computations. Lastly, the battery constraints and a partial recharging policy increase the complexity of route evaluation in the E-ADARP.

2. The CG performance largely depends on the efficiency of the labeling algorithm to solve the E-ADARP subproblems, where one must determine an excess-user-ride-time optimal schedule from battery-feasible schedules during label extension. This issue complicates solving the E-ADARP subproblems and cannot be handled exactly by existing DARP feasibility check methods (Gschwind & Irnich, 2015; Gschwind & Drexl, 2019) and scheduling procedures (Parragh et al., 2009; Molenbruch et al., 2017; Bongiovanni et al., 2022a).

3. The BO-EADARP is much harder to be solved exactly than the single-objective E-ADARP, as one must fully explore the bi-dimension search area in order to demonstrate the completeness of the Pareto front. The existing literature (Parragh et al. (2009); Molenbruch et al. (2017)) handles multi-objective DARPs by developing efficient heuristics to calculate a high-quality approximation of the Pareto front. To the best of our knowledge, no current work exists for solving the bi-objective DARP with EVs. In the context of bi-objective optimization, some works (e.g., Glize et al. (2022)) implement exact approaches (e.g., ϵ-constraint method) that can efficiently handle bi-objective problems (e.g., BO-VRP, BOTOPTW). However, as these analyzed problems are much simpler than the BO-EADARP, the existing methods are not guaranteed to be efficient in solving the BO-EADARP.

   The contributions of this thesis are summarized as follows.

(a) We propose a new approach that efficiently computes minimum excess user ride time by introducing a fragment-based representation of paths. Then, we apply an exact route evaluation scheme that executes feasibility checking in linear time. Combining these two methods, we propose an exact and efficient optimization of excess user ride time for an E-ADARP route.

(b) We adapt a Deterministic Annealing (DA) algorithm to tackle the E-ADARP by integrating the proposed excess user ride time optimization method. To the best of our knowledge, this is the first time an exact excess user ride time optimization has been developed for computing locally optimal solutions within an algorithm for solving the E-ADARP. This method allows computing the minimum excess user ride time for a feasible E-ADARP route in linear time after preprocessing.

(c) We demonstrate the performance of the proposed DA algorithm through extensive numerical experiments. On the previously solved instances, the DA algorithm improves the solution quality by 0.16% on average. We provide the best solutions for 70 out of 84 instances, among which 25 are new best solutions. To further test our algorithm in solving large-scale instances, we construct new benchmark instances with up to 8 vehicles and 96 requests, and we provide 19 new solutions on newly-introduced instances. We also extend the E-ADARP model to investigate the effects of allowing unlimited visits to recharging stations. The major difficulties for local search introduced by highly-constrained instances are lessened considering this more realistic situation.

(d) To handle the excess-user-ride-time optimality in the extension of an E-ADARP partial path, we take advantage of the fragment-based representation defined in (a). On each fragment, we apply a new approach to determine the minimum excess user ride time and abstract the fragment to an arc while guaranteeing excess-user-ride-time optimality.

(e) We construct a new graph that preserves all feasible routes of the original one and ensure excess-user-ride-time optimality on each arc of the new graph. We develop an efficient labeling algorithm with strong dominance rules on the new graph.

(f) The numerical results demonstrate the superiority of our CG algorithm over the state-of-the-art methods, with our algorithm being able to provide 40 new best solutions, 29 tighter lower bounds, and 17 new lower bounds for previously solved and unsolved instances. We prove optimality for 66 out of 84 instances without branching. For other instances, very small average gaps of 0.07% between lower bounds and the best-known upper bounds are observed. In addition, our algorithm can easily be adapted to tackle another problem variant where unlimited visits to each recharging station are allowed. The proposed labeling algorithm can also serve as the first exact scheduling procedure that generates excess-user-ride-time optimal schedules for a feasible E-ADARP route.

(g) We integrate the CG algorithm into the B&P framework, and we further solve 5 additional instances

optimally, obtain 6 tighter lower bounds and generate 3 additional new best solutions. Benefiting from the good quality of lower bounds obtained at the root node, our B&P algorithm searches only a few nodes of the search tree to close the gaps. Compared with the best-reported B&C results in Bongiovanni et al. (2019), we finally solve 71 out of 84 instances optimally within the two-hour time limit, while the B&C algorithm can only solve 49 instances optimally. In addition, we obtain 26 new best solutions and 54 equal solutions and enhanced 30 lower bounds. The average computational time of our B&P algorithm decreases by 16% compared with that of the B&C algorithm. On larger-sized instances (i.e., type-r instances), we obtain 16 new best solutions, compared with the existing results of Su et al. (2023). Therefore, the superiority of our B&P algorithm upon the existing exact method in the literature to solve the E-ADARP has been proved.

(h) To solve the BO-EADARP, we implement three different algorithms, among which two are criterion space search algorithms (i.e., ϵ-constraint method and balanced box method) and one is a decision space search algorithm (i.e., the BOBP). In the computational experiments, we solve the BO-EADARP with three different algorithms on small-to-medium-sized instances. From our results, the BOBP algorithm is the most efficient algorithm, as it generates the highest number of efficient solutions in the least average computational time. Then, we analyze the obtained efficient solutions in order to investigate the fundamental trade-off between service quality and operational efficiency. The obtained efficient solutions offer practical interests for different service providers: for profitable service providers (e.g., Uber, Didi), it is possible to significantly improve service quality while keeping near-optimal operational costs; for non-profitable service providers (e.g., Red Cross), the operational costs can be largely reduced while maintaining high service quality.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2 provides the problem definition and the notations of sets, parameters, and variables (Section 2.1). It also discusses the objective function and constraints of the E-ADARP (Section 2.2 and 2.3). Then, the compact formulation of the E-ADARP is introduced (Section 2.4). This chapter ends with a discussion on the effect of relaxing charging visits per recharging station (Section 2.5).

Chapter 3 presents the DA algorithm to solve the E-ADARP. To overcome the issues mentioned in challenge 1, we first propose an exact method of linear time complexity to compute the cost and evaluate the feasibility of an E-ADARP route based on battery-restricted fragments (Section 3.1). Repeated computations are avoided via fragment enumeration in the preprocessing phase (Section 3.2.4). These methods pave the way for an efficient DA algorithm (Section 3.2) and yield high-quality solutions for all instances. Finally, we conduct extensive experiments with our proposed DA algorithm (Section 3.3).

Chapter 4 presents the CG algorithm relying on a highly-efficient labeling algorithm to solve the E-ADARP.

First, the extended formulation of the E-ADARP and description of the CG subproblem is introduced (Section 4.1). To ensure the excess-user-ride-time optimality during label extension, we construct a new sparser graph, where excess-user-ride-time optimality is guaranteed on each arc (Section 4.2.2 to 4.2.3). We define a labeling algorithm on this new graph and handle battery feasibility by tailored REFs. Its efficiency is ensured by strong dominance rules and constant-time feasibility checks (Section 4.2.4). Cutting planes are introduced to enhance the lower bounds obtained from the CG algorithm. Also, we implement the B&P algorithm to yield optimal solutions and compare the obtained B&P results with the best-reported B&C results (Section 4.3). The last part of this chapter (Section 4.4) includes extensive computational experiments that investigate effects of allowing multiple visits per recharging station.

Chapter 5 tackles the BO-EADARP. First, we introduce notations that are often used in multi-objective optimization in Section 5.1. Then, we implement two criterion space search algorithms in Section 5.2. These algorithms include the ϵ-constraint method and the balanced-box method. In Section 5.3, we propose a generalized B&P algorithm that extends the single-objective B&P algorithm to the bi-objective context. Section 5.4 presents computational experiments with these algorithms on small-to-medium-sized instances under different minimum battery restrictions (i.e., $\gamma = 0.1, 0.4, 0.7$) and analyzes the obtained efficient solutions from a managerial perspective.

Chapter 6 concludes the thesis and discusses future works.

The chapters of this thesis are based the following papers:

- Chapter 3 Y. Su, N. Dupin, J. Puchinger (2023). "A deterministic annealing local search for the electric autonomous dial-a-ride problem". *European Journal of Operational Research*, accepted.
- Chapter 4 Y. Su, N. Dupin, S. N. Parragh, J. Puchinger (2023). "A column generation approach for the electric autonomous dial-a-ride problem". *Transportation Research Part B: Methodological*, in revision.
- Chapter 5 Y. Su, S. N. Parragh, N. Dupin, J. Puchinger (2023). "The bi-objective electric autonomous dial-a-ride problem". Working paper.

Some elements of the research included in this thesis have been presented in conferences. They are:

- Y. Su, N. Dupin, S. N. Parragh, J. Puchinger. A column generation approach for the electric autonomous dial-a-ride problem. Nominated as one of the 6 finalists for the "Best student article prize" at 24ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Rennes, France, 2023.
- Y. Su, S. N. Parragh, N. Dupin, J. Puchinger. The bi-objective electric autonomous dial-a-ride problem. Presented at 24ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Rennes, France, 2023.
- Y. Su, N. Dupin, J. Puchinger. A column-generation based heuristic for the electric autonomous dial-a-ride problem. Presented at ODYSSEUS 2021, Tangier, Morocco, 2022.

- Y. Su, N. Dupin, J. Puchinger. A column-generation based heuristic for the electric autonomous dial-a-ride problem. Presented at 23ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Lyon, France, 2022.
- Y. Su, J. Puchinger, N. Dupin. A deterministic annealing local search for the electric autonomous dial-a-ride problem. Presented at 31st European Conference on Operational Research (EURO 2021), Athens, Greece, 2021.
- Y. Su, N. Dupin, J. Puchinger. A deterministic annealing local search for the electric autonomous dial-a-ride problem. Presented at 22ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Mulhouse, France, 2021.

Chapter 2

Problem Statement

In this chapter, we first present the mathematical notations of the E-ADARP that were originally introduced in Bongiovanni et al. (2019) and are used throughout the thesis. Then, we present the objective function and constraints of the E-ADARP. Also, we introduce the mathematical formulation of the E-ADARP as a MILP. The final part discusses the practical interests of extending the original problem to allow unlimited visits to recharging stations.

2.1 Notation and problem statement

The problem is defined on a complete directed graph $G = (V, A)$, where V represents the set of vertices and A is the set of arcs, i.e., $A = \{(i, j) : i, j \in V, i \neq j\}$. Set V can be further partitioned into several subsets, i.e., $V = N \cup S \cup O \cup F$, where N represents the set of all customers, S is the set of recharging stations, O and F denote the set of origin depots and destination depots, respectively. The set of all pickup vertices is denoted as $P = \{1, \cdots, i, \cdots, n\}$ and the set of all drop-off vertices is denoted as $D = \{n+1, \cdots, n+i, \cdots, 2n\}$. The union of P and D is N, i.e., $N = P \cup D$. Each customer request is a pair $(i, n+i)$ for $i \in P$ and the maximum ride time for users associated with request i is assumed to be m_i. A time window is defined on each node $i \in V$, denoted as $[e_i, l_i]$, in which e_i and l_i represent the earliest and latest time at which the vehicle starts its service, respectively. A load q_i and a service duration s_i is also associated for each node $i \in V$. For pickup node $i \in P$, q_i is positive. For the corresponding drop-off node $n+i$, we have $q_{n+i} = -q_i$. For other nodes $j \in O \cup F \cup S$, q_j and s_j are equal to zero. In this thesis, we tackle the static E-ADARP (i.e., all the customer requests are known at the beginning of the planning horizon T_p).

Each vehicle $k \in K$ must start with an origin depot $o \in O$ and end with a destination depot $f \in F$. In this study, the number of origin depots is equal to the number of vehicles, i.e., $|O| = |K|$, as in Bongiovanni et al. (2019). However, the set of destination depots can be larger than the set of origin depots, namely, $|F| \geqslant |O|$, which means a vehicle can select a depot from F at the end of the route. An E-ADARP *route* is defined as a path in graph G originating from

the origin depot and terminating in the destination depot, and passing through the pickup, drop-off, and charging station (if required) locations, which satisfies pairing and precedence, load, battery, time window, and maximum user ride time constraints. The E-ADARP consists in designing K routes, one for each vehicle, so that all customer nodes are visited exactly once, each recharging station and destination depot is visited at most once, and the weighted-sum objective function (presented in Section 2.2) is minimized. For unemployed vehicles, they travel directly from their designated origin depot to a destination depot. Vehicles are assumed to be heterogeneous in terms of their maximum vehicle capacities C_k and homogeneous in terms of battery capacities (denoted as Q).

The travel time on each arc $(i, j) \in A$ is denoted as $t_{i,j}$ and the battery consumption is denoted as $b_{i,j}$. We assume that $b_{i,j}$ is proportional to $t_{i,j}$ and we have $b_{i,j} = \beta t_{i,j}$, with β being the energy discharging rate. When a vehicle recharges at a recharging station, the energy recharged is proportional to the time spent at the facilities. The recharging rate is denoted as α. Energy units are converted to time units by defining $h_{i,j} = b_{i,j}/\alpha$. Then, the battery consumption $b_{i,j}$ on arc (i, j) is converted to the time needed for recharging this amount of energy. Similarly, we can also convert the current energy level to the time needed to recharge to this energy level. Let H denote the time required to recharge from zero to full battery capacity Q. Partial recharging is allowed while a vehicle visits recharging stations, and a minimum battery level γQ must be respected at destination depots. The triangle inequality is assumed to hold for travel times and battery consumption.

Figure 2.1 presents a solution of an E-ADARP instance that includes 4 vehicles and 16 requests. Each request contains the pickup node (denoted as $i+$) and the corresponding drop-off node $i-$. If minimum battery level constraints are not satisfied, vehicles must make detours to recharging stations before returning to destination depots. Each vehicle starts from a different origin depot and returns to a different destination depot. Each recharging station is visited at most once and no passenger is onboard when recharging.

2.2 Objectives of the E-ADARP

Different from most works in the context of the DARP, the objective of the E-ADARP is expressed as a weighted sum of the total travel time of all the vehicles and the excess user ride time of all the users. Taking excess user ride time into the objective allows quantifying the user inconvenience directly. Also, it might help to improve the service quality by minimizing the excess user ride time at no additional operational cost if considered in a strictly lexicographic way. Related works considering the inclusion of user ride time in the objective function are Parragh et al. (2009) and Molenbruch et al. (2017). The objective function is formulated as:

$$\min w_1 \sum_{k \in K} \sum_{i,j \in V} t_{i,j} x_{i,j}^k + w_2 \sum_{i \in P} R_i \tag{2.1}$$

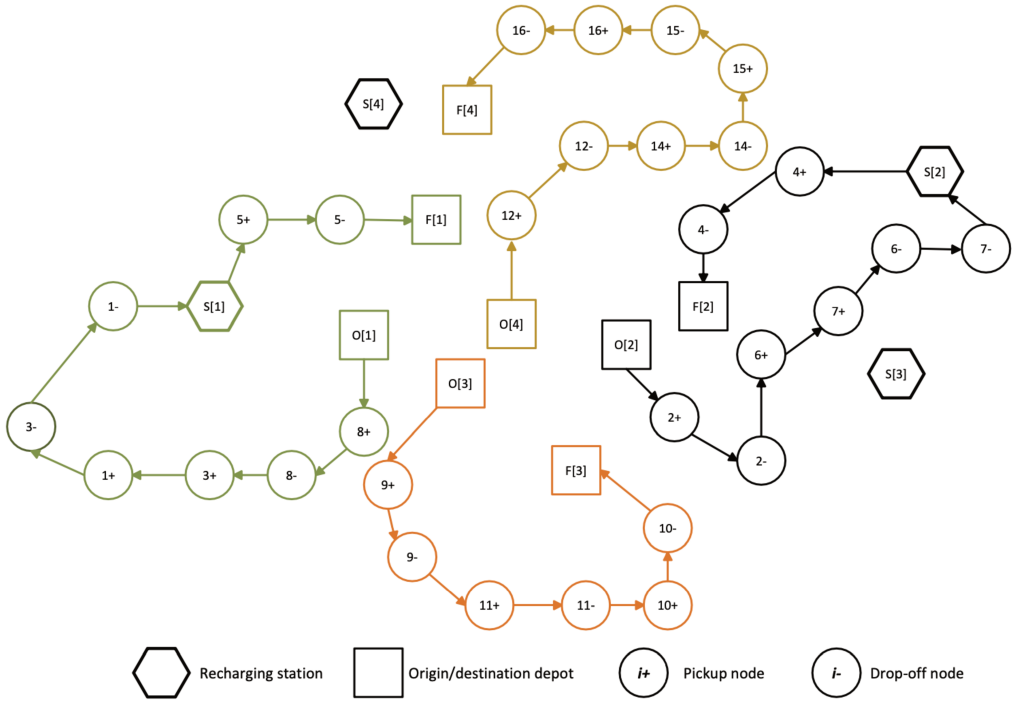


Figure 2.1 A solution of an E-ADARP instance that contains 4 vehicles and 16 requests

where $x_{i,j}^k$ is a binary decision variable which denotes whether vehicle k travels from node i to j, w_1 and w_2 are weight factors. R_i is the excess user ride time of request $i \in P$ and is formulated as the difference between the actual ride time $T_{n+i}^k - (T_i^k + s_i)$ and direct travel time $t_{i,n+i}$ and T_i^k is the service start time at node i.

2.3 Constraints of the E-ADARP

The E-ADARP consists of the following features that are different from the typical DARPs:

1) Battery limitation and minimum battery level restriction, which introduce the detour to recharging stations;

2) We allow partial recharging at recharging stations, and the recharging time must be determined;

3) Vehicles locate at different origin depots and select the destination depot from a set of destination depots;

4) Maximum route duration constraints are removed due to the autonomy of vehicles.

A solution of the E-ADARP is a set of $|K|$ routes and it is called "feasible" when all the following constraints are satisfied:

1. Every route starts from an origin depot and ends at a destination depot;

2. For each request, its corresponding pickup and drop-off node belong to the same route, and the pickup node is visited before its drop-off node;
3. User nodes and origin depots are visited exactly once, while each destination depot and recharging station is visited at most once;
4. The maximum vehicle capacity must be respected at each node;
5. Each node is visited within its time window $[e_i, l_i]$ where $i \in V$. Vehicle can arrive earlier than e_i but cannot arrive later than l_i. In the first case, waiting time occurs at i.
6. The maximum user ride time is not exceeded for any of the users;
7. The battery level at the destination depot must be at least equal to the minimal battery level;
8. The battery levels at any nodes of a route can not exceed the battery capacity and cannot be negative;
9. The recharging station can only be visited when there is no passenger on board;
10. Each recharging station $s \in S$ can only be visited at most once by all vehicles.

An E-ADARP route is called "feasible" if the above constraints, except for constraints (3) and (10), are fulfilled. Also, it should be noted that one can allow multiple visits to a recharging station by replicating the set of recharging stations, as in Bongiovanni et al. (2019).

2.4 Mathematical formulation of the E-ADARP

In this section, we present the mathematical formulation of the E-ADARP, which was initially introduced by Bongiovanni et al. (2019).

$$\min w_1 \sum_{k \in K} \sum_{i,j \in V} t_{i,j} x_{i,j}^k + w_2 \sum_{i \in P} R_i \tag{2.2}$$

subject to:

$$\sum_{j \in P \cup S \cup F} x_{o^k,j}^k = 1, \forall k \in K \tag{2.3}$$

$$\sum_{j \in F} \sum_{i \in D \cup S \cup \{o^k\}} x_{i,j}^k = 1, \forall k \in K \tag{2.4}$$

$$\sum_{k \in K} \sum_{i \in D \cup S \cup \{o^k\}} x_{i,j}^k \leqslant 1, \forall j \in F \cup S \tag{2.5}$$

$$\sum_{j\in V, j\neq i} x_{i,j}^k - \sum_{j\in V, j\neq i} x_{j,i}^k = 0, \forall k \in K, i \in N \cup S \tag{2.6}$$

$$\sum_{k\in K} \sum_{i\in N, j\neq i} x_{i,j}^k = 1, \forall i \in P \tag{2.7}$$

$$\sum_{j\in N, j\neq i} x_{i,j}^k - \sum_{j\in N, j\neq n+i} x_{j,n+i}^k = 0, \forall k \in K, i \in P \tag{2.8}$$

$$T_i^k + s_i + t_{i,n+i} \leqslant T_{n+i}^k, \forall k \in K, i \in P \tag{2.9}$$

$$e_i \leqslant T_i^k \leqslant l_i, \forall k \in K, i \in V \tag{2.10}$$

$$T_i^k + t_{i,j} + s_i - M_{i,j}(1 - x_{i,j}^k) \leqslant T_j^k, \forall k \in K, i \in V, j \in V, i \neq j | M_{i,j} > 0 \tag{2.11}$$

$$T_{n+i}^k - T_i^k - s_i \leqslant m_i, \forall k \in K, i \in P \tag{2.12}$$

$$R_i \geqslant T_{n+i}^k - T_i^k - s_i - t_{i,n+i}, \forall k \in K, i \in P \tag{2.13}$$

$$L_i^k + l_j - G_{i,j}^k(1 - x_{i,j}^k) \leqslant L_j^k, \forall k \in K, i \in V, j \in V, i \neq j \tag{2.14}$$

$$L_i^k + l_j + G_{i,j}^k(1 - x_{i,j}^k) \geqslant L_j^k, \forall k \in K, i \in V, j \in V, i \neq j \tag{2.15}$$

$$L_i^k \geqslant \max\{0, l_i\}, \forall k \in K, i \in N \tag{2.16}$$

$$L_i^k \leqslant \min\{C^k, C^k + l_i\}, \forall k \in K, i \in N \tag{2.17}$$

$$L_i^k = 0, \forall k \in K, i \in o^k \cup F \cup S \tag{2.18}$$

$$B_i^k = B_0^k, \forall k \in K, i \in o^k \tag{2.19}$$

$$B_j^k \leqslant B_i^k - b_{i,j} + Q(1 - x_{i,j}^k), \forall k \in K, i \in V \setminus S, j \in V \setminus \{o^k\}, i \neq j \tag{2.20}$$

$$B_j^k \geqslant B_i^k - b_{i,j} - Q(1 - x_{i,j}^k), \forall k \in K, i \in V \setminus S, j \in V \setminus \{o^k\}, i \neq j \tag{2.21}$$

$$B_j^k \leqslant B_s^k + \alpha E_s^k - b_{s,j} + Q(1 - x_{s,j}^k), \forall k \in K, s \in S, j \in P \cup F \cup S, s \neq j \tag{2.22}$$

$$B_j^k \geqslant B_s^k + \alpha E_s^k - b_{s,j} - Q(1 - x_{s,j}^k), \forall k \in K, s \in S, j \in P \cup F \cup S, s \neq j \tag{2.23}$$

$$Q \geqslant B_s^k + \alpha E_s^k, \forall k \in K, s \in S \tag{2.24}$$

$$B_i^k \geqslant \gamma Q, \forall k \in K, i \in F \tag{2.25}$$

$$E_s^k \leqslant T_i^k - t_{s,i} - T_s^k + M_{s,i}^k \left(1 - x_{s,i}^k\right), \quad \forall s \in S, i \in P \cup S \cup F, k \in K, i \neq s \tag{2.26}$$

$$E_s^k \geqslant T_i^k - t_{s,i} - T_s^k - M_{s,i}^k \left(1 - x_{s,i}^k\right), \quad \forall s \in S, i \in P \cup S \cup F, k \in K, i \neq s \tag{2.27}$$

$$x_{i,j}^k \in \{0, 1\}, \forall k \in K, i \in V, j \in V \tag{2.28}$$

$$B_i^k \geqslant 0, \forall k \in K, i \in V \tag{2.29}$$

$$E_s^k \geqslant 0, \forall k \in K, s \in S \tag{2.30}$$

Constraints (2.3) and (2.4) ensure that all vehicles start from their origin depots and end at a selected destination depot. Constraints (2.5) guarantee that each destination depot and recharging station is visited at most once. Constraints (2.6) are flow conservation constraints. Constraints (2.7) restrict the visit at each pickup node to exactly once. The pair and precedence constraints of pickup and drop-off nodes are ensured by constraints (2.8) and (2.9).

Constraints (2.10) and (2.11) are time window constraints. It should be noted that in Bongiovanni et al. (2019), the "big M" values are not calculated correctly. We present a more in-depth analysis in Appendix A to illustrate how the "big M" values should be set correctly. Constraints (2.12) are maximum user ride time constraints, and constraints (2.13) calculate the excess user ride time for request i. Constraints (2.14) to (2.18) are added to ensure the vehicle load will not exceed the maximum vehicle capacity. Constraints (2.19) set the initial battery level for vehicle k at the origin depot o^k. Constraints (2.20) and (2.21) calculate the battery levels of vehicles from $i \in V \setminus S$ to $j \in V \setminus \{o^k\}$. Constraints (2.22) and (2.23) track the battery levels of vehicles that leave from a recharging station $s \in S$ to the next node j. The maximum battery capacity is ensured in constraints (2.24). Constraints (2.25) restrict the minimum battery level that vehicle must satisfy when returning to the destination depot. Constraints (2.26) and (2.27) set lower and upper bounds for recharging time at a recharging station.

Table 2.1 summarizes all the notations and definitions for sets, parameters, and decision variables that will be used throughout the thesis.

2.5 Multiple visits to a recharging station?

Each E-ADARP instance of Bongiovanni et al. (2019) only contains a few recharging stations. In Bongiovanni et al. (2019), they first restrict the visit to the recharging station to at most one visit. This assumption is not practical in real life and increases the complexity of solving the E-ADARP, especially in the scenario of high battery restriction at the end of the route (i.e., $\gamma = 0.7$). Then, they investigate the effect of allowing multiple visits to recharging stations by replicating set S. Allowing multiple visits per station seems to be more practical, as it accounts for the physical limitations of recharging infrastructure (e.g., the limited number of charging ports) and vehicles can visit the same recharging station if it allows fewer detours. However, Bongiovanni et al. (2019) uses a predefined maximum number of visits (denoted as n_{as}) to a recharging station, and their proposed algorithm cannot solve all instances feasibly within their maximum algorithm capacity. The maximum value of n_{as} that can be solved by their proposed algorithm is $n_{as} = 3$. In this thesis, we relax this constraint and allow unlimited visits to the recharging stations in Chapter 3 and 4 (the corresponding sections are Section 3.3.4 and Section 4.4.2). By treating recharging stations as unlimited resources (i.e., represented by $n_{as} = \infty$), our proposed algorithms obtain feasible solutions for all instances, where most of them are solved optimally. Then, with obtained solutions, we calculate a more reliable value of n_{as} by analyzing all the best-obtained solutions for all instances. Our obtained n_{as} ensures solving different types of instances feasibly. Experiments with setting $n_{as} = 2, 3$ are also conducted and compared to the best-obtained results of Bongiovanni et al. (2019).

Table 2.1 Notations of the E-ADARP

Sets	Definitions
$N = \{1, \cdots, n, n+1, \cdots, 2n\}$	Set of pickup and drop-off nodes
$P = \{1, \cdots, i, \cdots, n\}$	Set of pickup nodes
$D = \{n+1, \cdots, n+i, \cdots, 2n\}$	Set of drop-off nodes
$K = \{1, \cdots, k\}$	Set of available vehicles
$O = \{o_1, o_2, \cdots, o_k\}$	Set of origin depots
$F = \{f_1, f_2, \cdots, f_h\}$	Set of all available destination depots (supposing the total number is h)
$S = \{s_1, s_2, \cdots, s_g\}$	Set of recharging stations (supposing the total number is g)
$V = N \cup S \cup O \cup F$	Set of all nodes
Parameters	**Definitions**
$t_{i,j}$	Travel time from location $i \in V$ to location $j \in V$
$b_{i,j}$	Battery consumption from location $i \in V$ to location $j \in V$
$h_{i,j}$	The time needed for recharging $b_{i,j}, i, j \in V$
e_i	Earliest time at which service can begin at $i \in V$
l_i	Latest time at which service can begin at $i \in V$
s_i	Service duration at $i \in V$
q_i	Change in load at $i \in N$
m_i	Maximum user ride time for request $i \in P$
C_k	The vehicle capacity of vehicle k
Q	The battery capacity
H	Recharging time required to recharge from zero to Q
B_{o_k}	Initial battery capacity of vehicle k
α	The recharged energy per time unit
β	The discharged energy per time unit
T_p	Planning horizon
γ	The ratio of minimum battery level at destination depot to Q
w_1, w_2	Weight factors for total travel time and total excess user ride time
Decision Variables	**Definitions**
$x_{i,j}^k$	$x_{i,j}^k = 1$ if vehicle k travels from location i to $j \in V$, 0 otherwise
T_i^k	Service start time of vehicle k at location $i \in V$
L_i^k	Load of vehicle k at location $i \in V$
B_i^k	Battery load of vehicle k at location $i \in V$
E_s^k	Charging time of vehicle k at charging station $s \in S$
R_i	Excess user ride time of passenger $i \in P$

Chapter 3

A Deterministic Annealing Local Search for the Electric Autonomous Dial-A-Ride Problem

This chapter is based on our published journal paper “A deterministic annealing local search for the electric autonomous dial-a-ride problem”, by European Journal of Operational Research. In this chapter, we propose a Deterministic Annealing (DA) algorithm to solve the E-ADARP and provide the first heuristic results for the static version of the E-ADARP. The two challenges of the E-ADARP presented in the first chapter are handled exactly in the DA algorithm: Partial recharging (i) is handled by an exact route evaluation scheme of linear time complexity. The minimum excess user ride time in objective function (ii) is calculated by a new method that allows effective computations of minimum excess user ride time by introducing a fragment-based representation of paths. Combining these two methods, we propose an exact and efficient optimization of excess user ride time for a generated E-ADARP route. The remainder of this chapter is organized as follows. Section 3.1 introduces the fragment-based representation of paths and the method to minimize total excess user ride time. A novel route evaluation scheme of linear time complexity is then described. Based on Section 3.1, Section 3.2 presents the framework of the proposed DA algorithm and its main ingredients. In Section 3.3, we conduct extensive computational experiments to demonstrate the performance of the proposed DA algorithm. This chapter ends in Section 3.4 with a summary of the results and contributions, closing with discussions of future extensions.

3.1 Excess User Ride Time Optimization

The idea of our excess user ride time optimization method is as follows. We first introduce a fragment-based representation of paths, which extends the one proposed in Rist & Forbes (2021) by additionally considering battery constraints for ensuring overall route feasibility in terms of energy consumption. Based on this representation of paths, each E-ADARP route can be represented by a series of battery-restricted fragments (see Definition 1). Then, we prove in Theorem 1 that the minimum total excess user ride time for a feasible route can be determined by summing the minimum excess user ride time of each battery-restricted fragment. Following this idea, we enumerate all the feasible battery-restricted fragments and calculate their minimum excess user ride times in the preprocessing phase (shown in Section 3.2.4). With all the feasible fragments obtained as well as their minimum excess user ride time, we only need to check the feasibility of the route, which is realized via an exact route evaluation scheme of linear time complexity.

3.1.1 Representation of paths

The most important characteristic of the E-ADARP is the incorporation of total excess user ride time in the objective function as well as the maximum user ride time in the constraints. Usually, the maximum user ride time constraints can be tackled by calculating forward time slack and delaying the service start time at some nodes (e.g., Cordeau & Laporte (2003), Kirchler & Calvo (2013), Parragh et al. (2009)). To minimize the total excess user ride time, we declare one important point: total excess user ride time can only be minimized when vehicles finish their delivery (i.e., no open request on the path). We then introduce battery-restricted fragments:

Definition 1 (Battery-restricted fragment). Assume that $\mathcal{F} = (i_1, i_2, \cdots, i_k)$ is a sequence of pickup and drop-off nodes, where the vehicle arrives empty at i_1 and leaves empty at i_k and has passenger(s) on board at other nodes. Then, we call $\mathcal{F}$ a battery-restricted fragment if there exists a feasible route of the form:

$$(o, s_{i_1}, \cdots, s_{i_v}, \overbrace{i_1, i_2, \cdots, i_k}^{\mathcal{F}}, s_{i_{v+1}}, \cdots, s_{i_m}, f) \tag{3.1}$$

where $s_{i_1}, \cdots, s_{i_v}, s_{i_{v+1}}, \cdots, s_{i_m} (v, m \geqslant 0)$ are recharging stations, $o \in O$, and $f \in F$.

It should be noted that if no recharging station is required in the route of Definition 1, i.e., $v = m = 0$ in Equation (3.1), the battery-restricted fragment is equivalent to a fragment defined in Rist & Forbes (2021). Figure 3.1 presents an example of a feasible route that consists of two battery-restricted fragments, i.e., $\mathcal{F}_1 = \{1+, 2+, 1-, 2-\}$ and $\mathcal{F}_2 = \{3+, 3-\}$. Note that $\mathcal{F}_1 \cup \mathcal{F}_2$ is not a battery-restricted fragment, as the vehicle becomes empty at intermediate node 2-. Based on this definition, each E-ADARP route can be regarded as the concatenation of several battery-restricted fragments, recharging stations (if required), an origin depot, and a destination depot.

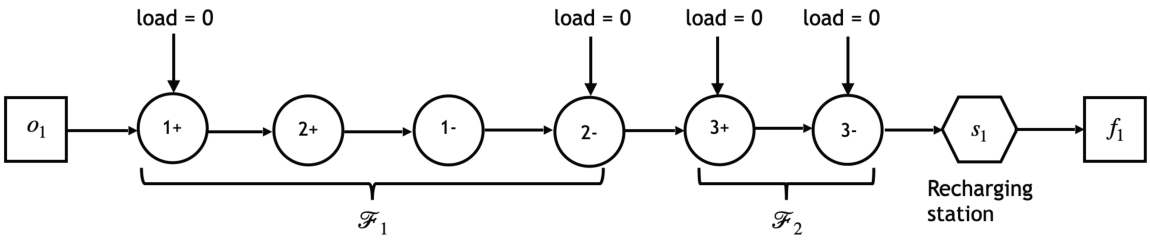


Figure 3.1 Example of battery-restricted fragments

Clearly, on each battery-restricted fragment (hereinafter referred to as "fragment"), the minimum excess user ride time can be calculated exactly. We prove in the next section (Theorem 1) that the minimum excess user ride time of route $\mathcal{R}$ can be calculated by summing the minimum excess user ride time on each fragment $\mathcal{F}_i \subseteq \mathcal{R}$. Then, we only focus on optimizing excess user ride time for each fragment.

3.1.2 Excess user ride time optimization for a fragment

Let $EU_{min}(\mathcal{R})$ and $EU_{min}(\mathcal{F})$ be the minimum excess user ride over route $\mathcal{R}$ and fragment $\mathcal{F}$, respectively. We have the following Theorem.

Theorem 1. If $\mathcal{R}$ is a feasible route and $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n$ are all the fragments on $\mathcal{R}$, then we have $EU_{min}(\mathcal{R}) = EU_{min}(\mathcal{F}_1) + EU_{min}(\mathcal{F}_2) + \cdots + EU_{min}(\mathcal{F}_n)$

Proof. In this proof, a schedule is called "optimal" if it has minimal excess user ride time.

Assuming that $\mathcal{T} = [\cdots, T_v, \cdots]_{v \in \mathcal{R}}$ is an optimal schedule of route $\mathcal{R}$, T_v is the service start time at node v, and the arrival time of node v is: $arr_v = T_{v-1} + t_{v-1,v} + s_{v-1}$. To prove the theorem, it is enough to show that for each fragment $\mathcal{F}_i \subseteq \mathcal{R}$, the restricted schedule $\mathcal{T}|_{\mathcal{F}_i} = [\cdots, T_v, \cdots]_{v \in \mathcal{F}_i}$ over $\mathcal{F}_i$ is also an optimal schedule for $\mathcal{F}_i$. To simplify the notation, we denote $\mathcal{T}|_{\mathcal{F}_i}$ as $\mathcal{T}_i$. Our proof consists of two different cases :

1. $arr_v = T_v$ for all $v \in \mathcal{F}_i$ and node v is not the start node of $\mathcal{F}_i$. In this case, the vehicle starts service at its arrival on each node in $\mathcal{F}_i$. Clearly, $\mathcal{T}_i$ is also an optimal schedule over $\mathcal{F}_i$ as the waiting time on $\mathcal{F}_i$ is zero, proof is finished;

2. $arr_v < T_v$ for some $v \in \mathcal{F}_i$ and node v is not the start node of $\mathcal{F}_i$. In this case, waiting time is generated at some nodes. Let $v_1 \in \mathcal{F}_i$ be the first node such that $arr_{v_1} < T_{v_1}$ and $v_2 \in \mathcal{F}_i$ be the last node such that $arr_{v_2} < T_{v_2}$. Then we derive the following properties of $\mathcal{T}_i$:

   (i) $T_{v_0} = l_{v_0}$ for some $v_0 <$ [1] $v_1, v_0 \in \mathcal{F}_i$.

   If not, we have $\Delta_1 = min\{T_{v_1} - arr_{v_1}, \{l_v - T_v\}_{v<v_1, v \in \mathcal{F}_i}\} > 0$. We can obtain a new feasible schedule $\mathcal{T}_1$ by delaying the service start times of all nodes in $\mathcal{F}_i$ that are before node v_1 (i.e., node v such that

[1] we say $v_0 < v_1$ if v_0 is a node before v_1 in the route and $v_0 \neq v_1$.

$v < v_1, v \in \mathcal{F}_i$) to $T'_v = T_v + \Delta_1$. The excess user ride time of $\mathcal{T}_1$ is at least Δ_1 smaller than $\mathcal{T}$. It contradicts to our assumption that $\mathcal{T}$ is an optimal schedule (for example, see Example 1);

(ii) $T_{v_3} = e_{v_3}$ for some $v_3 \geqslant v_2, v_3 \in \mathcal{F}_i$.

If not, we have $\Delta_2 = min\{T_{v_2} - arr_{v_2}, \{T_v - e_v\}_{v \geqslant v_2, v \in \mathcal{F}_i}\} > 0$. We can obtain a new feasible schedule $\mathcal{T}_2$ by moving forward the service start times of all nodes in $\mathcal{F}_i$ that are after node v_2 (i.e., node v such that $v \geqslant v_2, v \in \mathcal{F}_i$) to $T''_v = T_v - \Delta_2$. The excess user ride time of $\mathcal{T}_2$ is at least Δ_2 smaller than $\mathcal{T}$. It contradicts to our assumption that $\mathcal{T}$ is an optimal schedule (for example, see Example 2);

Based on (i) and (ii), assuming that v_s, v_e are the first and the last node of $\mathcal{F}_i$, we derive that all the feasible schedules for $\mathcal{F}_i$ must satisfy the following two points:

(iii) Since we have $arr_v = T_v$ for all $v < v_0 < v_1$ and $T_{v_0} = l_{v_0}$, any feasible schedules over $\mathcal{F}_i$ could not begin service at v_s later than T_{v_s} (T_{v_s} is the latest possible service start time at v_s). Otherwise, it will surpass the latest time window l_{v_0} at node v_0;

(iv) Since we have $arr_v = T_v$ for all $v_2 \leqslant v_3 < v$ and $T_{v_3} = e_{v_3}$, any feasible schedules over $\mathcal{F}_i$ could not arrive at v_e earlier than arr_{v_e}.

Assuming that $\mathcal{T}_i^* = [\cdots, T_v^*, \cdots]_{v \in \mathcal{F}_i}$ is an optimal schedule of $\mathcal{F}_i$, and the arrival time at v is $arr_v^* = T_{v-1}^* + t_{v-1,v} + s_{v-1}$. Now, we prove that the excess user ride time of $\mathcal{T}_i$ is the same as $\mathcal{T}_i^*$ using the above properties. Note that we are still under the condition that $arr_v < T_v$ for some $v \in \mathcal{F}_i$.

According to (iii) and (iv), we have $T_{v_s}^* \leqslant T_{v_s}, arr_{v_e}^* \geqslant arr_{v_e}$ for an optimal schedule $\mathcal{T}_i^*$ over $\mathcal{F}_i$. Clearly, $\mathcal{T}_i^*$ satisfies $EU_{min}(\mathcal{T}_i^*) \leq EU_{min}(\mathcal{T}_i)$. Next, we will prove that $EU_{min}(\mathcal{T}_i^*) = EU_{min}(\mathcal{T}_i)$. Then, we prove $\mathcal{T}_i$ is an optimal schedule over $\mathcal{F}_i$. Our proof contains two cases:

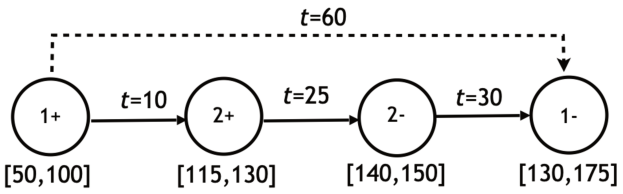
(a) If $arr_v^* = T_v^*$ for all $v \in \mathcal{F}_i$: As we have $T_{v_s}^* \leqslant T_{v_s}$, then $arr_{v_e}^* \leqslant arr_{v_e}$. Therefore, we derive that $arr_{v_e}^* = arr_{v_e}$, $T_{v_s}^* = T_{v_s}$. As we assume in the condition that $arr_v^* = T_v^*$ for all $v \in \mathcal{F}_i$, we must have $T_v = T_v^*$ for all $k \in \mathcal{F}_i$. It contradicts to our assumptions that $arr_v < T_v$ for some $v \in \mathcal{F}_i$. Therefore, this case will not happen;

(b) If $arr_v^* < T_v^*$ for some $v \in \mathcal{F}_i$: Then we can prove the same result as in (i) (ii) and (iii) for T_v^* in the same manner. Then $T_{v_s} \leqslant T_{v_s}^*, arr_{v_e} \geqslant arr_{v_e}^*$ and thus we derive $T_{v_s} = T_{v_s}^*, arr_{v_e} = arr_{v_e}^*$. Then we have $EU_{min}(\mathcal{T}_i^*) = EU_{min}(\mathcal{T}_i)$. Otherwise, if $EU_{min}(\mathcal{T}_i^*) < EU_{min}(\mathcal{T}_i)$, we can obtain a new feasible schedule $\mathcal{T}'$ over $\mathcal{R}$ from $\mathcal{T}$ by replacing $\mathcal{T}_i$ to $\mathcal{T}_i^*$, and $\mathcal{T}'$ has smaller excess user ride time than $\mathcal{T}$, which is a contradiction!

□

For the sake of illustration, we take Example 1 and 2 to explain point (i) and (ii) of case 2, respectively.

Example 1. Consider a fragment $\mathcal{F} = \{1+, 2+, 2-, 1-\}$, the time window on each node, and the travel time for each arc is shown in Figure 3.2. The direct travel time from node 1+ to node 1- is shown on the dashed line. Assume that the service time at each node is equal to zero and each request includes one passenger to be transported. We present two sets of service start times in the table, one is called schedule $\mathcal{B}$, and the other is called schedule $\mathcal{A}$. Also, we present the excess user ride times that are calculated from schedule $\mathcal{B}$ in the row named $R_{\mathcal{B}}$, and these are calculated from schedule $\mathcal{A}$ in the row named $R_{\mathcal{A}}$. We denote the service start time of schedule $\mathcal{B}$ at node v as $\mathcal{B}_v$. Note that schedule $\mathcal{A}$ is an **optimal schedule**.

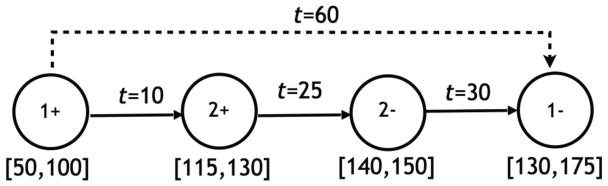


v	1+	2+	2−	1−
$\mathcal{B}$	90	115 (arrive at 100)	140	170
$R_{\mathcal{B}}$	-	-	0	170-90-60
$\mathcal{A}$	100	115 (arrive at 110)	140	170
$R_{\mathcal{A}}$	-	-	0	170-100-60

Figure 3.2 Example of case 2 (i)

In this example, node 2+ is the first node such that $arr_{2+} < \mathcal{A}_{2+}$ (i.e., node v_1 in the proof) and node 1+ is a node before node 2+ with $\mathcal{A}_{1+} = l_{1+}$ (i.e., node v_0 in the proof). Therefore, schedule $\mathcal{B}$ is a **conflicting schedule of case 2 (i)**. Schedule $\mathcal{B}$ is not excess-user-ride-time optimal, as one can further reduce excess user ride time of request 1 by delaying the service start time at node 1+ by $\Delta_1 = \min\{\mathcal{B}_{2+} - arr_{2+}, l_{1+} - \mathcal{B}_{1+}\} = \min\{115 - 100, 100 - 90\}$, as schedule $\mathcal{A}$ does.

Example 2 (Example 1 continued)**.** We take the same problem settings as in Example 1 to illustrate point (ii) of case 2. Note that schedule $\mathcal{A}$ is an **optimal schedule**.



v	1+	2+	2−	1−
$\mathscr{B}'$	100	115 (arrive at 110)	145 (arrive at 140)	175
$R_{\mathscr{B}'}$	-	-	145-115-25	175-100-60
$\mathscr{A}$	100	115 (arrive at 110)	140	170
$R_{\mathscr{A}}$	-	-	0	170-100-60

Figure 3.3 Example of case 2 (ii)

In this example, node 2+ is the last node such that $arr_{2+} < \mathcal{A}_{2+}$ (i.e., node v_2 in the proof) and node 2- is a node after node 2+ with $\mathcal{A}_{2+} = e_{2+}$ (i.e., node v_3 in the proof). Therefore, schedule $\mathcal{B}'$ is a **conflicting schedule of case 2 (ii)**. Schedule $\mathcal{B}'$ is not an optimal schedule, as one can further reduce excess user ride times of request 1 and request 2 by moving forward the service start time at node 2- by:

$$\Delta_2 = \min\{\mathcal{B}'_{2+} - arr_{2+}, \{\mathcal{B}'_{2-} - e_{2-}, \mathcal{B}'_{1-} - e_{1-}\}\} = \min\{115 - 110, \{145 - 140, 175 - 130\}\}.$$

Based on Theorem 1, we convert the optimization of total excess user ride time for route $\mathcal{R}$ to the optimization of excess user ride time on its fragments $\mathcal{F} \subseteq \mathcal{R}$. Clearly, we can calculate the minimum excess user ride time directly if no waiting time is generated on a fragment. In the case of waiting time generated, one can compute the minimum excess user ride time if fragment $\mathcal{F}$ only contains a direct trip from one pickup node to the corresponding drop-off node. In the case that $\mathcal{F}$ contains two or more requests and waiting time generates for some $i \in \mathcal{F}$, the minimization of excess user ride time for $\mathcal{F}$ is equivalent to minimize a weighted sum of waiting time along $\mathcal{F}$, where weight factors are vehicle load at nodes with waiting time. To obtain the minimum excess user ride time, we resort to solving a Linear Program (LP), which is presented as follows.

Let $P_{\mathcal{F}}$ denote the set of requests served on a fragment $\mathcal{F}$:

$$\min \sum_{i \in P_{\mathcal{F}}} R_i \tag{3.2}$$

s.t.

$$T_i + s_i + t_{i,j} \leqslant T_j, \quad \forall i \in \mathcal{F}, \quad idx_j = idx_i + 1, idx_i \neq |\mathcal{F}| \tag{3.3}$$

$$T_{n+i} - (T_i + s_i) \leqslant m_i, \quad \forall i \in P_{\mathcal{F}} \tag{3.4}$$

$$T_{n+i} - T_i - s_i - t_{i,n+i} \leqslant R_i, \quad \forall i \in P_{\mathcal{F}} \tag{3.5}$$

$$e_i \leqslant T_i \leqslant l_i, \quad \forall i \in \mathcal{F} \tag{3.6}$$

$$R_i \geqslant 0, \quad \forall i \in P_{\mathcal{F}} \tag{3.7}$$

where T_i denotes the service start time at node i, idx_i is the index of node i on the fragment. The objective function is to minimize the total excess user ride time of $\mathcal{F}$. Constraints (3.3) are time window constraints. Constraints (3.4) and constraints (3.5) are user ride time constraints.

Note that we ensure the maximum user ride time and vehicle capacity constraints when we generate fragments (will be explained in Section 3.2.4). If a route $\mathcal{R}$ contains an infeasible fragment, it is discarded directly without further evaluation.

3.1.3 Exact route evaluation scheme of linear time complexity

One challenge of the E-ADARP is tackling the trade-off between recharging time and time window constraints. A longer recharging time will extend the driving range and is beneficial to meet the energy restriction at the destination depot. However, the vehicle risks violating the time window constraints for the succeeding nodes. These two aspects interact, and it is hard to check the feasibility of a generated route (denoted as $\mathcal{R}$). We construct an exact route evaluation scheme of linear time complexity based on the forward labeling algorithm of Desaulniers et al. (2016). To the best of our knowledge, it is the first time an exact route evaluation scheme is developed to handle the DARP with EVs.

Given a route $\mathcal{R}$, we associate each node $i \in \mathcal{R}$ with a label $L_i := \{(T_i^{rch_s})_{s \in S}, T_i^{tMin}, T_i^{tMax}, T_i^{rtMax}\}$ including four resource attributes. We denote $\mathcal{P}_i$ as the partial path from the first node of $\mathcal{R}$ until node i. The definition of each resource attribute is shown as follows:

1. $T_i^{rch_s}$: The number of times recharging station $s \in S$ is visited along $\mathcal{P}_i$;
2. T_i^{tMin}: The earliest service start time at vertex i assuming that, if a recharging station is visited prior to i along $\mathcal{P}_i$, a minimum recharge (ensuring the battery feasibility up to i) is performed;
3. T_i^{tMax}: The earliest service start time at vertex i assuming that, if a recharging station is visited prior to i along $\mathcal{P}_i$, a maximum recharge (ensuring the time-window feasibility up to i) is performed;
4. T_i^{rtMax}: The maximum recharging time required to fully recharge at vertex i assuming that, if a recharging station is visited prior to i along $\mathcal{P}_i$, a minimum recharge (ensuring the battery feasibility up to i) is performed;

The initial label is defined as $\{(\overbrace{0, \cdots, 0}^{|S| \text{ times}}), 0, 0, 0\}$. We compute the succeeding label L_j from the previous label L_i by Resource Extension Functions (REFs):

$$T_j^{rch_s} = T_i^{rch_s} + \begin{cases} 1, & \text{if } j = s \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

$$T_j^{tMin} = \begin{cases} \max\{e_j, T_i^{tMin} + t_{i,j} + s_i\}, & \text{if } T_i^{rch} = \emptyset \\ \max\{e_j, T_i^{tMin} + t_{i,j} + s_i\} + Z_{i,j}, & \text{otherwise} \end{cases} \tag{3.9}$$

$$T_j^{tMax} = \begin{cases} \min\{l_j, \max\{e_j, T_i^{tMin} + T_i^{rtMax} + t_{i,j} + s_i\}\}, & \text{if } i \in S \\ \min\{l_j, \max\{e_j, T_i^{tMax} + t_{i,j} + s_i\}\}, & \text{otherwise} \end{cases} \tag{3.10}$$

$$T_j^{rtMax} = \begin{cases} T_i^{rtMax} + h_{i,j}, & \text{if } T_i^{rch} = \emptyset \\ \min\{H, \max\{0, T_i^{rtMax} - S_{i,j}\} + h_{i,j}\}, & \text{otherwise} \end{cases} \tag{3.11}$$

where:

$$S_{i,j}(T_i^{tMin}, T_i^{tMax}, T_i^{rtMax}) = \begin{cases} \max\{0, \min\{e_j - T_i^{tMin} - t_{i,j} - s_i, T_i^{rtMax}\}\}, & \text{if } i \in S \\ \max\{0, \min\{e_j - T_i^{tMin} - t_{i,j} - s_i, T_i^{tMax} - T_i^{tMin}\}\}, & \text{otherwise} \end{cases} \tag{3.12}$$

$$Z_{i,j}(T_i^{tMin}, T_i^{tMax}, T_i^{rtMax}) = \max\{0, \max\{0, T_i^{rtMax} - S_{i,j}(T_i^{tMin}, T_i^{tMax}, T_i^{rtMax})\} + h_{i,j} - H\} \tag{3.13}$$

The $S_{i,j}$ is the slack time between the earliest time window e_j at j and the earliest arrival time to j. If i is a recharging station, $S_{i,j}$ is the maximum amount of recharging time that can be performed at i, namely $T_i^{tMax} - T_i^{tMin}$. $Z_{i,j}$ is the minimum recharging time required to keep battery feasibility accounting for the available slack at the previous recharging station.

According to Desaulniers et al. (2016), we have following proposition:

Proposition 1. The route $\mathcal{R}$ is feasible if and only of $\forall j \in R$, the label L_j satisfies:

$$T_j^{tMin} \leqslant l_j, \quad T_j^{tMin} \leqslant T_j^{tMax}, \quad T_j^{rch_s} \leqslant 1, \quad T_j^{rtMax} \leqslant \begin{cases} (1-\gamma)H, & j \in F \\ H, & \text{otherwise} \end{cases}$$

Clearly, the feasibility checking algorithm is of linear time complexity with respect to the length of the input route. After checking the feasibility, the total cost of route $\mathcal{R}$ is obtained by summing the travel time of arcs and the excess user ride time of fragments, recalling Theorem 1.

3.2 Deterministic Annealing Algorithm for the E-ADARP

Based on Section 3.1.2 and Section 3.1.3, we establish a DA algorithm that ensures minimal excess user ride time for a generated solution and integrates an exact route evaluation. Different types of local search operators are embedded in the proposed DA algorithm to solve the E-ADARP.

DA was first introduced by Dueck & Scheuer (1990) as a variant of simulated annealing. Recent research shows that DA can obtain near-optimal or optimal solutions for a series of vehicle routing problems (Bräysy et al., 2008; Braekers et al., 2014). To the best of our knowledge, the only paper that implements DA to solve the DARP is that of Braekers et al. (2014). Applying DA algorithm provides several advantages, and the most important one is its easy parameter tuning process, as the DA algorithm mainly relies on a single parameter. In addition, the DA algorithm is proved to be very efficient in solving the typical DARP. However, Braekers et al. (2014) considers a single-objective case in the DARP. To solve the E-ADARP, we adapt the DA algorithm to accommodate problem-specific features of the E-ADARP by integrating the proposed excess user ride time optimization approach.

The framework for the proposed DA algorithm is depicted in Algorithm 1. The algorithm input is an initial solution x_{init} constructed by a parallel insertion heuristic (presented in Section 3.2.1) and the initial settings of DA-related parameters. These parameters include: (i) a maximal number of iterations N_{iter}; (ii) the initial and maximal temperature Θ_{max}; (iii) restart parameter n_{imp}. It should be mentioned that the initial solution x_{init} is feasible for the E-ADARP constraints, except that only a subset of requests may be served. The solution cost of the initial solution is denoted as $c(x)$, and the number of requests served in the initial solution is updated to N_{req} so that a lexicographic optimization considers cost comparison in $c(x)$ values only if it does not worsen the number of requests served. A list of indexed operators $opt_1, \ldots, opt_z$ are operated sequentially in each DA iteration (presented in Section 3.2.3).

There are two steps in the algorithm: local search and threshold update. At the beginning of the algorithm, the threshold value Θ is set to Θ_{max}, and the best solution x_b and current solution x is initialized to an initial solution x_{init}. During the local search process, local search operators are applied to alter the current solution. In the next step, the threshold value is updated and restarted when the value is negative.

In the local search process, we first remove the existing recharging stations on the current route and then generate a random neighborhood solution x' from the current solution x by applying different operators. In the case of neighborhood solution x' satisfies $c(x') < c(x) + \Theta$ but violates battery constraints, we call an insertion algorithm to repair x' by inserting recharging stations at proper places (presented in Section 3.2.2). Solution x' is accepted to become the new current solution when the number of assigned requests increases or the total cost is less than that of the current solution plus the threshold value Θ.

In the threshold update process, when no new global best solution is found, Θ is reduced by $\Theta_{max}/\Theta_{red}$, where Θ_{red} is a predefined parameter. To ensure that Θ is always non-negative, we reset Θ to $r \times \Theta_{max}$, with r a random number generated between zero and one whenever Θ becomes negative. The search is restarted from x_b when no

Algorithm 1 DA Algorithm for the E-ADARP

Input: Initial solution x_{init}, initial values of N_{iter}, Θ_{max}, and n_{imp}. Θ is set to Θ_{max}
Output: Best solution x_b found by our algorithm;

while $iter \leqslant N_{iter}$ **do**
  $i_{imp} \leftarrow i_{imp} + 1$;
  for $j = 1 \rightarrow z - 1$ **do**
    Apply local search operator opt_j on x to obtain neighborhood solution x';
    if $c(x') < c(x) + \Theta$ **then**
      $x \leftarrow x'$;
    end if
  end for
  if $N_{req} < n$ **then**
    Apply opt_z operator to add request to generate neighborhood solution x';
    Update the number of requests served in x' as N'_{req};
  end if
  if $(c(x') < c(x_b)$ **and** $N'_{req} = N_{req})$ **or** $N'_{req} > N_{req}$ **then**
    $x_b \leftarrow x'$
    $i_{imp} \leftarrow 0$
  else
    $\Theta \leftarrow \Theta - \Theta_{max}/\Theta_{red}$
    if $\Theta < 0$ **then**
      $r \leftarrow$ random number between 0 and 1
      $\Theta \leftarrow r \times \Theta_{max}$
      if $i_{imp} > n_{imp}$ **then**
        $x \leftarrow x_b$
        $i_{imp} \leftarrow 0$
      end if
    end if
  end if
  $iter \leftarrow iter + 1$
end while
return x_b

improvement is found in n_{imp} iterations and Θ becomes negative.

3.2.1 Parallel insertion heuristic

While in most of the literature, the initial solution is often generated randomly, we construct our initial solution by a parallel insertion algorithm considering the time window and spatial closeness, as in Masmoudi et al. (2017). First, we sort all the requests $(i, n+i), i \in P$ in increasing order based on e_i. Then, we randomly initialize k routes $\{\mathcal{R}_1, \cdots, \mathcal{R}_k\}$ ($0 < k \leqslant K$ with K being the number of total vehicles). Each of the k first requests in the sorted request list is assigned randomly to different routes. These requests are deleted from the list of requests.

Then, we sort the route list $\{\mathcal{R}_1, \cdots, \mathcal{R}_k\}$ in increasing order with regards to the distance between the last node of the analyzed route and the pickup node of the first request remaining in the request list. The first request is assigned to the first route in the route list. To insert the selected request, we enumerate all the possible insertion positions and insert the corresponding pickup node and drop-off node in a feasible way on this route. If this request cannot be inserted feasibly, then we move to the second route. This process is repeated until this request is inserted or all the routes are analyzed. If this request cannot be inserted in any of the existing routes, we move to the second request in the list and repeat the above process. After this process, if some requests are still not assigned, a new route is activated, and the above process will be repeated. The algorithm terminates when the request list is empty or the existing requests in the list cannot be inserted into any of the routes in a feasible way.

3.2.2 Recharging station insertion for a given route

If a route $\mathcal{R} \in x'$ only violates the battery constraints and neighborhood solution x' has $c(x') < c(x) + T$, we insert a/several recharging station(s) to repair $\mathcal{R}$. First, we create two empty sets, one is to store repaired route candidates (called "list of feasible routes"), the other is to store potential route candidates (called "list of candidate routes"). For each possible insertion position, we select a random recharging station from the set of available stations to insert. We do not consider inserting the best station (e.g, the closest one), as we may have other battery-infeasible routes in $\mathcal{R}$, which requires visiting this recharging station to be repaired. If a feasible route is generated after insertion, we add it to the list of feasible routes. Otherwise, we store this route in the list of candidate routes. Suppose the route is still infeasible after trying all the possible insertion positions. In that case, we move to the next iteration to insert another recharging station for all the possible positions of all the candidate routes. The algorithm returns the repaired minimum-cost feasible route if $\mathcal{R}$ can be repaired or an empty set otherwise. For acceleration, we only consider repairing the route containing less than N_{rch} recharging stations and we take $N_{rch} = \lceil |S|/2 \rceil$.

3.2.3 Local search

We design seven operators (i.e., $opt_1, \cdots, opt_7$ in Algorithm 1) to improve the initial solution generated from the constructive heuristic. Among them, three are intra-route operators (i.e., *ex-pickup*, *ex-dropoff*, and *ex-2-neighbor*), three are inter-route operators (i.e., *2-opt*, *relocate*, and *exchange*). The last operator named *add-request* is applied in each iteration on neighborhood solution x', which is generated after applying $opt_1, \cdots, opt_5$, if there exists un-served requests.

Intra-route operators

Ex-pickup operator swaps the positions of two consecutive nodes $(i+, j+)$, where node $i+$ is a pick-up node and node $j+$ is not the corresponding drop-off node. An example is shown in Figure 3.4. In each iteration, one pick-up node is selected randomly. If the successor of this pick-up node does not correspond to its drop-off node, then the two positions are exchanged.

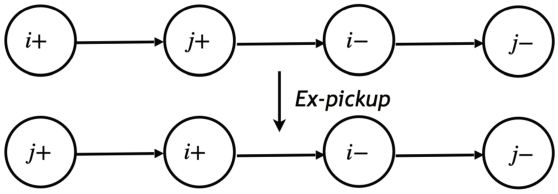


Figure 3.4 Ex-pickup operator example.

Ex-dropoff operator creates a neighborhood solution by swapping the positions of two consecutive nodes $(j+, i-)$, where point $i-$ is a drop-off node and point $j+$ is not the corresponding pick-up node. Figure 3.5 shows an example of how *ex-dropoff* works. In each iteration, one drop-off node is selected randomly, if the precedent node of this drop-off node does not correspond to its pick-up node, then the two positions are exchanged.

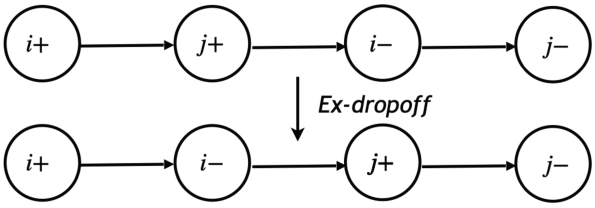


Figure 3.5 Ex-dropoff operator example.

There is another situation shown in Figure 3.6, where the successor of pick-up node $i+$ is its drop-off $i-$, and the predecessor of drop-off node $j-$ is its corresponding pick-up $j+$, but we can still exchange i- and $j+$ to create a new neighborhood solution. This operation is realized by *ex-2-neighbor* operator.

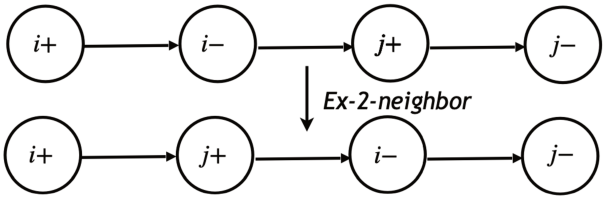


Figure 3.6 Ex-2-neighbor operator example.

Inter-route operators

Inspired by state-of-the-art heuristic methods, we apply three widely-used inter-route operators to generate neighbors of the current solution, as in Braekers et al. (2014). *Two-opt* operator selects two random routes and splits each route into two parts by a randomly selected zero-split node i such that $i \in D \cup S$. Then, the first part of the first route is connected with the second part of the second route and the first part of the second route is connected with the second part of the first route. Note that *2-opt* is able to realize the exchange of several requests at one iteration. Figure 3.7 is an example of how *2-opt* operator works.

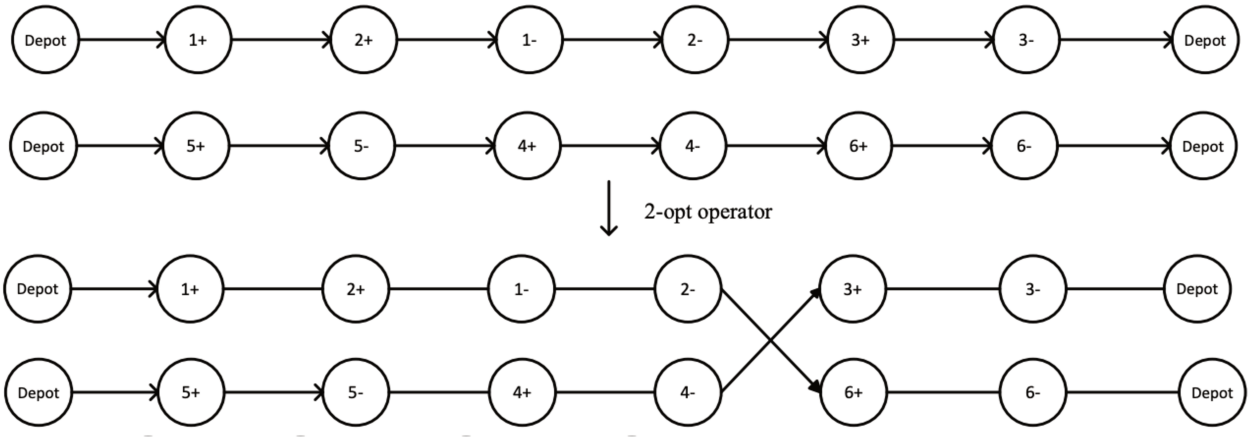


Figure 3.7 2-opt operator example

Relocate operator randomly removes one request from a random route and re-inserts the request at the best position of another route. The best position means the position that brings the least increase on solution cost after inserting the selected request. A simple example is shown in Figure 3.8, where a request $(2+, 2-)$ is removed from the first route and reinserted into the second route at the best positions.

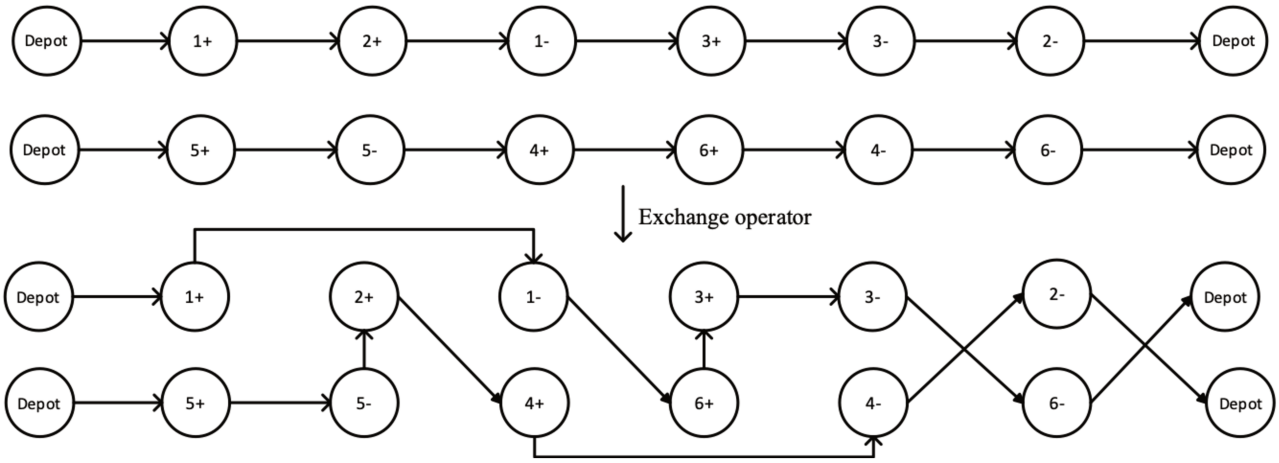


Figure 3.9 Exchange operator example

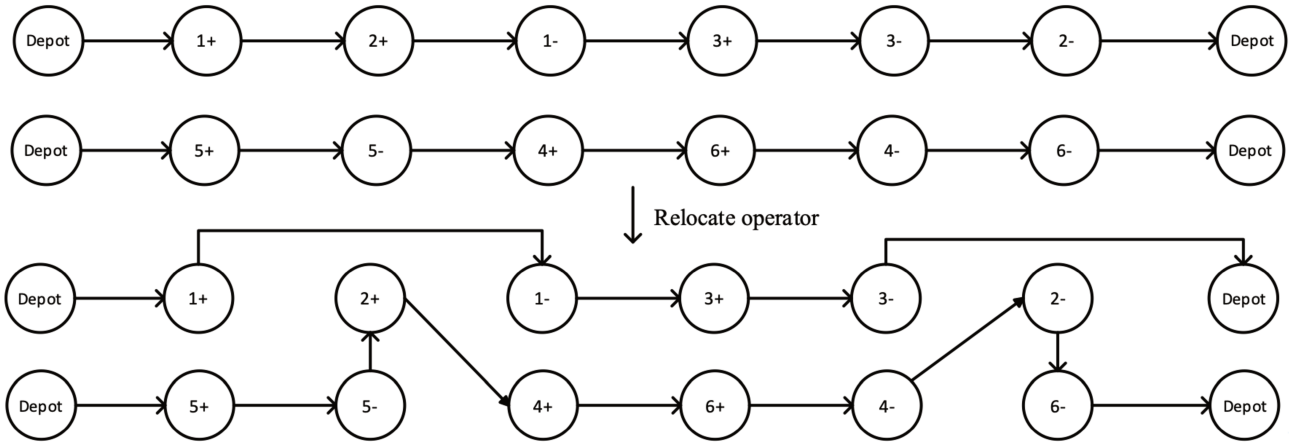


Figure 3.8 Relocate operator example

Exchange operator swaps two random requests of two randomly-selected routes. The selected requests are re-inserted into the best position of the other route.

Insertion operator

Add-request operator is applied in each iteration when there exist uninserted requests for current solution x. This operator tries to insert one uninserted request into a random route of x. When all the requests are served in x, this operator will no longer be applied. Figure 3.10 describes how *add-request* adds uninserted request $(h+, h-)$ on a route.

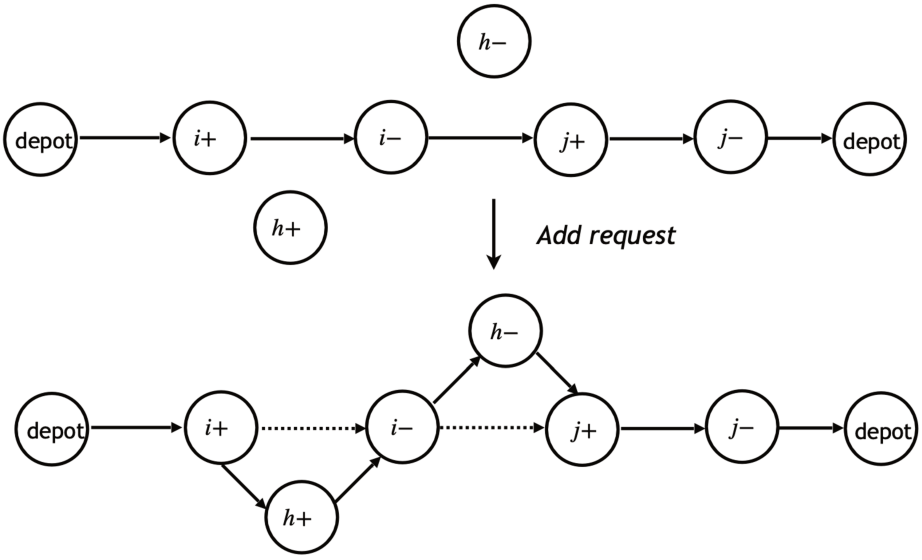


Figure 3.10 Add-request operator example

3.2.4 Implementation details

This section presents the preprocessing works and the algorithm implementation details for allowing multiple/unlimited visits to recharging stations. The preprocessing works include: time window tightening, arc elimination, and fragment enumeration.

Preprocessing works

We first introduce two traditional methods introduced by Cordeau (2006), which includes time window tightening and arc elimination. Then, we introduce fragments enumeration method.

Time window tightening is executed as:

- For $i \in P$, e_i is set to $\max\{e_i, e_{n+i} - m_i - s_i\}$ and $l_i = \min\{l_{n+i} - t_{i,n+i} - s_i, l_i\}$;
- For $i \in D$, $e_{n+i} = \max\{e_{n+i}, e_i + t_{i,n+i} + s_i\}$, and $l_{n+i} = \min\{l_i + m_i + s_i, l_{n+i}\}$.
- For $s \in S$, the time window can be tightened by considering the travel time from the origin depot to recharging station and from recharging station to the destination depot. The earliest time to start service at charging station s is set to $\min\{e_j + t_{j,s}\}, \forall j \in O$; the latest time at charging station s to start service at recharging station is $\max\{T_p - t_{s,j}\}, \forall j \in F$;
- For $i \in O \cup F$, the earliest time window e_i is set to $\max\{0, \min\{e_j - t_{i,j}\}\}, \forall j \in P$, and $l_i = \min\{l_i, \max\{l_j + s_i + t_{j,i}\}\}, \forall j \in D$.

It should be noted that we tighten the value of e_i for node $i \in P$ by considering the earliest possible service start time at node $i \in P$ for arriving at the corresponding drop-off node $n+i$ at e_{n+i}. Hence, $e_i = \max\{e_i, e_{n+i} - m_i - s_i\}$. Similarly, we tighten $l_i, i \in P$ by considering the latest possible service start time at node $i \in P$ for arriving at node $n+i$ at l_{n+i}, i.e., $l_i = \min\{l_{n+i} - t_{i,n+i} - s_i, l_i\}$. The arc elimination process follows the method of Cordeau (2006). We reduce the number of arcs in the graph by removing arcs that will not lead to a feasible solution.

We further accelerate computations by enumerating all feasible fragments before computation, as in Alyasiry et al. (2019), Rist & Forbes (2021). This method simplifies route evaluation and avoids recalculations as we only need to query information from each fragment. We enumerate all the feasible fragments with depth-first search and calculate their minimum excess user ride time. Then, the total excess user ride time of a route $\mathcal{R}$ can be calculated by summing $EU_{min}(\mathcal{F}), \mathcal{F} \subseteq \mathcal{R}$, recalling Theorem 1.

To generate all feasible fragments, we start from each pickup node and extend it node by node respecting time window, capacity, battery, and maximum user ride time constraints. We assume that the vehicle starts from each pickup node with a full battery level. The maximum user ride time, vehicle capacity constraints are checked during the extension process. For each node on a fragment, it must have a positive battery level.

Note that if a fragment only contains one request, we calculate the excess user ride time directly and check the maximum user ride time constraints. If a fragment contains two or more requests, we resort to a LP solver (Gurobi) to solve the LP model (shown in 3.1.2) and check the maximum user ride time constraints. For each feasible fragment, the obtained minimum excess user ride time value is recorded. We conduct a preliminary test and provide details for fragment enumeration on each instance as shown in Table B.1 in Appendix B. In Table B.1, N_{frag} denotes the number of fragments generated, Leg_{avg} and Leg_{max} denote the average and maximum length of fragments, respectively. N_{LP} represents the number of time LP is solved, and CPU is the total computational time for enumeration in seconds.

For all the instances, the fragment enumeration can be fulfilled in a matter of seconds. In the computational experiments, we report the CPU time, which includes the computational time for performing all the preprocessing works in Section 3.3.

Adapt DA algorithm to allow multiple visits to each recharging station

Different from the model of Bongiovanni et al. (2019), we allow multiple visits at each recharging station without the need to replicate set S. In the case of $n_{as} = 2, 3$, we replicate the recharging station set S to allow at most two and at most three visits per station. All the ingredients remain the same in these two cases. In the case of $n_{as} = \infty$, we remove the feasibility checking rule $T_j^{rch_s} \leqslant 1$ to allow one route visiting multiple times for a station. When selecting a recharging station to insert in a route, we relax the set of available recharging stations to S. This operation allows inserting a recharging station that has already been used in other routes.

3.3 Computational Experiments and Results

In this section, we conduct extensive numerical experiments and analyze the results. All algorithms are coded in Julia 1.7.2 and are performed on a standard PC with an Intel Xeon Gold 6230 20C at 2.1GHz. This section is organized as follows. The benchmark instances for the computational experiments and abbreviations used in the Tables are introduced in the first part. Then, a sensitivity analysis is conducted to find good parameter settings for the proposed DA algorithm in Section 3.3.2. After ensuring the robustness of parameters and operators, we validate the performance of the proposed algorithm on the standard E-ADARP instances compared to the state-of-the-art results in Section 3.3.3. Section 3.3.4 investigates the effect of allowing multiple visits to recharging stations.

3.3.1 Benchmark instances and abbreviations

This section presents the benchmark instances used to test the algorithm performance, their characteristics, and the notations for the computational experiments.

Benchmark Instances

Instances are named following the pattern xK-n-γ, where K is the number of vehicles, n is the number of requests, and $\gamma \in \{0.1, 0.4, 0.7\}$. Three sets of instances are considered in the experiments, which differentiate by $x \in \{a, u, r\}$:

- "a" denotes the standard DARP benchmark instance set from Cordeau (2006) extended with features of electric vehicles and recharging stations by Bongiovanni et al. (2019). To simplify, we call them type-a instances. For type-a instances, the number of vehicles is in the range $2 \leqslant K \leqslant 5$, and the number of requests is in the range $16 \leqslant n \leqslant 50$.
- "u" denotes instances based on the ride-sharing data from Uber Technologies (instance name starts with "u") that were adopted from Bongiovanni et al. (2019). To simplify, we call them type-u instances. For type-u instances, the number of vehicles is in the range $2 \leqslant K \leqslant 5$, and the number of requests is in the range $16 \leqslant n \leqslant 50$, as in type-a instances.
- "r" denotes larger DARP benchmark instances built from Ropke et al. (2007) using the same extension rules to have E-ADARP instances from DARP instances. To simplify, we call them type-r instances. For type-r instances, the number of vehicles is in the range $5 \leqslant K \leqslant 8$ and the number of requests is in the range $60 \leqslant n \leqslant 96$.

Type-a instances are supplemented with recharging station ID, vehicle capacity, battery capacity, the final state of charge requirement, recharging rates, and discharging rates. The same operation is applied to type-r instances to generate a large-scale set of instances. The vehicle capacity is set to three passengers, and the maximum user ride time is 30 minutes. As in Bongiovanni et al. (2019), recharging rates and discharging rates are all set to 0.055KWh

per minute according to the design parameter of EAVs provided in: https://www.hevs.ch/media/document/1/fiche-technique-navettes-autonomes.pdf. The efficient battery capacity is set to 14.85 KWh, and the vehicle can approximately visit 20 nodes without recharging.

The ride-sharing dataset of Uber is obtained from the link: https://github.com/dima42/uber-gps-analysis/tree/master/gpsdata. Type-u instances are created by extracting origin/destination locations from GPS logs in the city of San Francisco (CA, USA) and applying Dijkstra's shortest path algorithm to calculate the travel time matrix with a constant speed setting (i.e., 35km/h). Recharging station positions can be obtained through Alternative Fueling Station Locator from Alternative Fuels Data Center (AFDC). For a more detailed description of instances development, the interested reader can refer to Bongiovanni et al. (2019). The preprocessed data that extract requests information from the raw data provided by Uber Technologies are published on the website (https://luts.epfl.ch/wpcontent/uploads/2019/03/e_ADARP_archive.zip). The vehicle capacity is set to three passengers, and the maximum user ride time is 8 minutes.

Following Bongiovanni et al. (2019), we consider three different γ values, i.e., $\gamma \in \{0.1, 0.4, 0.7\}$, representing different minimal battery restrictions at the destination depot. For weight factors, we take $w_1 = 0.75$ and $w_2 = 0.25$.

Abbreviations in the tables

The DA algorithm has deterministic rules to accept a solution and the sequence of neighborhoods, which is contrary to Simulated Annealing. There remains a randomized part in the selection of neighboring solutions. Unless indicated, we perform 50 runs on each instance with different seeds to analyze the statistical distribution of the solution quality.

For each instance, we present the following values:

- BC' is the cost of best solutions from B&C algorithm reported in Bongiovanni et al. (2019);
- BC is the cost of best solutions found by the proposed DA algorithm over 50 runs;
- AC is the average-cost solution found by the proposed DA algorithm over the 50 runs.
- $Q1$ is the middle number between the best-obtained solution and the median of all the solutions over 50 runs;
- $Q3$ is the middle number between the median of all the solutions over 50 runs and the worst solutions yielded;

To analyze the distribution of the solution found for the 50 runs, we calculate solutions gaps to BC'. Assuming a solution with value v (v could be BC, $Q1$, $Q3$), we compute its gap to BC' by:

$$gap = \frac{v - BC'}{BC'} \times 100\%$$

Note that type-r instances for the E-ADARP are studied here for the first time, we therefore replace BC' with BC in the above formula to analyze the gaps of $Q1/AC/Q3$ to BC.

We present the following average values to analyze the consistency of the proposed DA algorithm:

- $Q1\%$ is the average gap to BC' of the first quartile value over the different runs;
- $Q3\%$ is the average gap to BC' of the third quartile value over the different runs;
- $BC\%$ is the average gap of BC to BC' over the different runs;
- $AC\%$ is the average gap of AC to BC' over the different runs;
- FeasRatio is the ratio of feasible solutions found among all the solutions generated by DA algorithm;
- CPU is the average computational time of the DA algorithm (preprocessing time is included) in seconds;
- CPU′ is the computational time of the B&C algorithm reported in Bongiovanni et al. (2019) in seconds;
- NC (Not Calculable) means that there are unsolved instances under the analyzed parameter and we cannot calculate gaps.
- NA (Not Available) indicates that corresponding value (e.g., BC, BC') is not available as the analyzed algorithm cannot provide a feasible solution.
- A dash "–" indicates that the DA algorithm finds new best solutions on a previously unsolved instance and we cannot calculate the gap.

In Section 3.3.4, we present DA algorithm results when allowing multiple visits to each recharging station. To distinguish, subscripts "2", "3", and "∞" are added to BC, AC, and CPU to denote $n_{as} = 2, 3, \infty$, respectively. As Bongiovanni et al. (2019) provides results on type-u instances with $n_{as} = 2, 3$, we add their reported results in the column named BC'_2 and BC'_3 of Table 3.8 and compare our DA algorithm results to theirs.

3.3.2 Parameter tuning for the DA algorithm

The performance of the proposed algorithm depends on several parameters that must be set in advance. To ensure the algorithm's performance, we first identify robust parameter settings. We analyze different settings of parameters on the type-a instance set, as it contains instances of different sizes and is enough to select good parameters. For a comprehensive overview, we take into account different scenarios, i.e., $\gamma = 0.1, 0.4, 0.7$, for each parameter setting.

The DA-related parameters are:

- Number of iterations N_{iter} ;
- Maximum threshold value Θ_{max};
- Threshold reduction value Θ_{red};

- Restart parameter n_{imp}.

To avoid re-tuning Θ_{max} when using different instances, we use a relative value for Θ_{max}. The maximum threshold value is expressed as the product of the average distance between two nodes in the studied graph (denoted $\bar{c}$) and a predefined parameter θ_{max}, that is $\Theta_{max} = \bar{c} \times \theta_{max}$, where θ_{max} is initially set to 1.5. For other parameters like Θ_{red} and n_{imp}, we take the same settings as in Braekers et al. (2014): $\Theta_{red} = 300$ and $n_{imp} = 50$.

Sensitivity analysis and parameter tuning for θ_{max}

The sensitivity analysis results for θ_{max} under $\gamma = 0.1, 0.4, 0.7$ are shown Table 3.1, and we test seven values for θ_{max}. For each value of θ_{max}, we perform ten runs on each instance and iterate the proposed algorithm 10000 times for each run. Under each energy restriction, we report $BC\%$, $AC\%$, $Q1\%$, $Q3\%$ over ten runs for the analyzed θ_{max} value. For the scenario of $\gamma = 0.7$, we report FeasRatio and average CPU time. We present detailed results on each instance under different settings of θ_{max} in Table B.2 and B.3 in Appendix B.

Table 3.1 Sensitivity analysis for θ_{max} under different γ cases on type-a instances

θ_{max}	0.6	0.9	1.2	1.5	1.8	2.1	2.4
$\gamma = 0.1$							
$BC\%$	0.11%	0.10%	0.19%	0.32%	0.29%	0.28%	0.51%
$AC\%$	0.49%	0.53%	0.74%	0.83%	0.82%	0.94%	1.07%
$Q1\%$	0.23%	0.30%	0.43%	0.54%	0.56%	0.66%	0.81%
$Q3\%$	0.66%	0.73%	0.90%	0.93%	1.04%	1.22%	1.28%
FeasRatio	140/140	140/140	140/140	140/140	140/140	140/140	140/140
CPU (s)	83.93	77.43	78.52	80.09	81.16	82.12	83.42
$\gamma = 0.4$							
$BC\%$	0.19%	0.27%	0.27%	0.40%	0.49%	0.70%	0.63%
$AC\%$	NC	0.68%	0.79%	0.95%	1.18%	1.36%	1.54%
$Q1\%$	0.31%	0.49%	0.57%	0.65%	0.84%	0.96%	1.11%
$Q3\%$	0.72%	0.84%	0.97%	1.21%	1.5%	1.68%	1.83%
FeasRatio	139/140	140/140	140/140	140/140	140/140	140/140	140/140
CPU (s)	121.34	116.97	119.03	121.72	122.97	125.65	127.80
$\gamma = 0.7$							
FeasRatio	85/140	106/140	106/140	108/140	112/140	105/140	106/140
CPU (s)	227.05	201.68	206.06	212.5	215.86	221.04	222.31

Contribution of local search operators

As the algorithm largely relies on local search operators, their usefulness is verified. In this part, we analyze the contribution of local search operators to improve the solution quality. The effectiveness of each local search operator is presented, and the results of six different algorithm configurations are shown in Table 3.2. In each of these configurations, one operator is excluded from the algorithm, and we run each algorithm configuration ten times, with each run iterating the respective algorithm 10000 times. We calculate the average solution gap of $BC\%$, $AC\%$, $Q1\%$, and $Q3\%$. Results for different algorithm configurations setting the previously selected parameter values ($\theta_{max} = 0.9$) are summarized in Table 3.2. For the scenario $\gamma = 0.7$, we report CPU times and FeasRatio.

Table 3.2 Experimental results when removing a single operator: *Ex-pickup*, *Ex-dropoff*, *Ex-2-neighbor*, *Relocate*, *Exchange*, and *2-opt*

Removing	None	*Ex-pickup*	*Ex-dropoff*	*Ex-2-neighbor*	*Relocate*	*Exchange*	*2-opt*
$\gamma = 0.1$							
$BC\%$	0.10%	0.14%	0.23%	0.19%	0.25%	0.38%	2.64%
$AC\%$	0.52%	0.52%	0.55%	0.56%	1.16%	0.68%	5.60%
$Q1\%$	0.30%	0.40%	0.40%	0.44%	0.79%	0.51%	3.76%
$Q3\%$	0.73%	0.74%	0.90%	0.79%	1.64%	1.00%	6.19%
FeasRatio	140/140	140/140	140/140	140/140	139/140	140/140	140/140
CPU (s)	77.43	74.88	71.41	88.97	57.53	79.51	68.92
$\gamma = 0.4$							
$BC\%$	0.27%	0.27%	0.27%	0.38%	0.38%	0.27%	2.56%
$AC\%$	0.68%	0.73%	0.74%	0.78%	1.15%	0.84%	4.92%
$Q1\%$	0.49%	0.51%	0.49%	0.64%	0.86%	0.63%	3.66%
$Q3\%$	0.84%	0.93%	1.22%	1.06%	1.63%	1.10%	6.03%
FeasRatio	140/140	140/140	140/140	139/140	136/140	140/140	140/140
CPU (s)	116.97	109.24	106.25	134.29	81.92	115.52	105.08
$\gamma = 0.7$							
FeasRatio	106/140	96/140	106/140	90/140	86/140	97/140	74/140
CPU (s)	201.68	191.54	185.69	237.5	137.17	210.65	182.31

We can find that each operator performs very well in improving the solution quality, especially the *2-opt* operator. Additionally, the *relocate* and *2-opt* operator contributes to provide more feasible solutions in the case of $\gamma = 0.4, 0.7$. Therefore, it is necessary to include these operators in local search. As for *add-request*, it is essential for inserting requests that are not served in the current solution. From the above analysis, the usefulness of each local search operator is proved.

Sensitivity analysis on number of iterations

Then, we conduct the sensitivity analysis for the number of iterations N_{iter}. To identify a good N_{iter}, we conduct experiments with all the energy-level restrictions on type-a instances. We test ten values of N_{iter}, and report $BC\%$, $AC\%$, $Q1\%$, $Q3\%$ over ten runs. For the scenario of $\gamma = 0.7$, as different settings of N_{iter} result in a different number of feasible solutions, we compare FeasRatio. The results are shown in Table 3.3.

From Table 3.3, we observe that the values of $BC\%$, $AC\%$, $Q1\%$, $Q3\%$ are improved with more iterations. Among ten values of N_{iter}, 10000 iterations provide us with the best solution quality. We therefore set N_{iter} to 10000 to conduct experiments. The performance of DA is also demonstrated as small results dispersion is found under all the values of N_{iter}. Moreover, we also notice that the computational time grows approximately linearly with the number of iterations, which is a computational advantage compared with the B&C algorithm.

Note that choosing $N_{iter} = 8000$ or $N_{iter} = 9000$ slightly degrades the performances. With such parameters, the computational time will be decreased. Choosing $N_{iter} = 10000$ is more robust, especially keeping in mind the evaluation of larger type-r instances.

Table 3.3 Statistical comparison of DA performance under different iteration times for all γ values on type-a instances

N_{iter}	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Low energy restriction $\gamma = 0.1$										
$BC\%$	0.60%	0.44%	0.35%	0.31%	0.20%	0.17%	0.14%	0.13%	0.11%	0.10%
$AC\%$	NC	NC	0.95%	0.82%	0.73%	0.68%	0.63%	0.59%	0.56%	0.53%
$Q1\%$	1.42%	0.82%	0.61%	0.52%	0.45%	0.42%	0.36%	0.34%	0.31%	0.30%
$Q3\%$	2.35%	1.69%	1.12%	1.02%	0.96%	0.88%	0.83%	0.79%	0.76%	0.73%
FeasRatio	138/140	139/140	140/140	140/140	140/140	140/140	140/140	140/140	140/140	140/140
CPU (s)	10.15	17.5	24.86	32.43	39.92	47.45	54.92	62.38	69.79	77.43
Medium energy restriction $\gamma = 0.4$										
$BC\%$	1.07%	0.72%	0.57%	0.48%	0.40%	0.37%	0.34%	0.31%	0.30%	0.27%
$AC\%$	NC	NC	NC	1.17%	1.03%	0.90%	0.84%	0.78%	0.73%	0.68%
$Q1\%$	1.69%	1.18%	0.96%	0.82%	0.72%	0.66%	0.63%	0.57%	0.54%	0.49%
$Q3\%$	2.98%	2.09%	1.61%	1.39%	1.22%	1.14%	1.08%	0.99%	0.87%	0.84%
FeasRatio	138/140	139/140	139/140	140/140	140/140	140/140	140/140	140/140	140/140	140/140
CPU (s)	14.45	25.82	37.21	48.62	59.94	71.29	82.74	94.12	105.7	116.97
High energy restriction $\gamma = 0.7$										
FeasRatio	79/140	88/140	94/140	95/140	96/140	97/140	100/140	102/140	103/140	106/140
CPU (s)	21.94	41.83	61.7	81.88	101.73	121.63	141.56	161.64	181.63	201.68

3.3.3 DA algorithm performance on the E-ADARP instances

In this section, we present the performance of our DA algorithm after tuning parameters from the previous section. Table 3.4, Table 3.5, and Table 3.6 present our DA algorithm results on type-a, -u, and -r instances under $\gamma = 0.1, 0.4, 0.7$, respectively. In each table, we report the values of BC, AC, $Q1$, $Q3$, and their corresponding gaps with BC' (presented in the column named "BC'"). If we obtain better solutions than the best-reported results of Bongiovanni et al. (2019), we mark them in bold with an asterisk. We mark our solutions in bold if they are equal to those reported in Bongiovanni et al. (2019).

It should be noted that we find strictly better integer solutions than the reported optimal results of Bongiovanni et al. (2019) in case of $\gamma = 0.4, 0.7$. The reason is that in the model of Bongiovanni et al. (2019), the employed "big M" values were not correctly computed. We refer to Appendix A for a more in-depth analysis and how the "big M" values should be set correctly. To distinguish these incorrect results, we mark them in italics in the column of "BC'" and mark our obtained solutions in bold with double stars. The corresponding $BC\%$ values are therefore negative.

Type-a instances results under different energy restrictions

We first conduct experiments on type-a instances considering different scenarios $\gamma = 0.1, 0.4, 0.7$. A higher γ value means a higher minimum battery level that vehicles must keep when returning to the destination depot. Recalling that each recharging station can only be visited at most once. The E-ADARP model is more constrained with an increasing γ. In Table 3.4, we compare our algorithm results to the best reported results in Bongiovanni et al. (2019).

We obtain equal/improved solutions for 36 out of 42 instances. Among them, 13 are the new best solutions. For

some instances, we obtain better solutions than the reported optimal solutions in Bongiovanni et al. (2019), leading to savings on solution cost with up to 1.40%. These instances are: a2-24-0.4, a3-30-0.4, a3-36-0.4, a2-24-0.7, a3-24-0.7, and a4-24-0.7.

In all the scenarios, the proposed DA algorithm has quite small gaps to the best-reported results in Bongiovanni et al. (2019). In case of $\gamma = 0.1, 0.4$, the average $BC\%$ is 0.05% and 0.13% (the worst $BC\%$ is 0.40% and 1.26%), and other values $AC\%$, $Q1\%$, $Q3\%$ are under 2.09%, 2.51%, and 2.79%, respectively. When $\gamma = 0.7$, we consistently provide new solutions for a2-20, a4-32, and a5-40, while B&C cannot solve these instances optimally or feasibly within two hours. Particularly, the generated new solutions on instance a4-32 and a5-40 have a much lower solution cost compared to the former reported best solutions in Bongiovanni et al. (2019), with an average gap of -7.49 % and -5.22%, respectively.

In terms of computational efficiency, the CPU time for the proposed DA algorithm grows approximately in a linear way with sizes of instances. The average CPU time for all instances is 96.71s, and the proposed DA algorithm can efficiently solve large-scale instances within maters of minutes.

Type-u instances results under different energy restrictions

On type-u instances, we conduct experiments under different energy-restriction levels $\gamma = 0.1, 0.4, 0.7$. The results are shown in Table 3.5.

The proposed DA algorithm finds equal solutions for 22 out of 42 instances and finds new best solutions for 12 previously solved and unsolved instances. Particularly, on instance u2-24-0.1, u2-24-0.4, u4-40-0.4, and u3-30-0.7, we find strictly better solutions than the reported optimal solutions in Bongiovanni et al. (2019), which contributes to savings on solution costs with up to 0.43%. In each scenario, our best solutions have quite small gaps to the BC' reported in Bongiovanni et al. (2019) (the worst- and best-case $BC\%$is 1.00% and -0.60%, respectively). We further demonstrate our algorithm consistency via other statistical values ($Q1\%$, $AC\%$, $Q3\%$), as our algorithm continuously finds high-quality solutions with the increasing size of instances. In terms of computational efficiency, solving the problem exactly seems more computationally effective on small-sized instances. The reason is that we fix N_{iter} to 10000 for all instances, whereas the small-sized ones can be solved to their best values (i.e., optimal objective values reported in Bongiovanni et al. (2019)) in much fewer iterations. On medium-to-large-sized instances, using an efficient heuristic (e.g., the proposed DA algorithm) is a more computational appealing option.

Type-r instances results under different energy restrictions

We present our algorithm results on type-r instances in Table 3.6. These results are the first solutions found for these new instances and can serve as benchmark results for future studies.

In scenarios $\gamma = 0.1$ and $\gamma = 0.4$, we find feasible solutions for 19 out of 20 instances, with an average CPU time of 269.71s and 373.89s, respectively. When increasing from $\gamma = 0.1$ to $\gamma = 0.4$, the statistical dispersion also

Table 3.4 Results of the proposed DA algorithm on type-a instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	Proposed DA algorithm, 10000 iterations, 50 runs									Bongiovanni et al.,[a]	
Instance	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
a2-16	**237.38**	0	237.38	0	237.38	0	237.38	0	39.3	**237.38***	1.2
a2-20	**279.08**	0	279.08	0	279.08	0	279.08	0	73.8	**279.08***	4.2
a2-24	**346.21**	0	346.21	0	346.21	0	346.21	0	160.6	**346.21***	9.0
a3-18	**236.82**	0	236.82	0	236.82	0	236.82	0	25.2	**236.82***	4.8
a3-24	**274.80**	0	274.80	0	274.80	0	274.80	0	58.3	**274.80***	13.8
a3-30	**413.27**	0	413.27	0	413.27	0	413.27	0	54.3	**413.27***	102.0
a3-36	**481.17**	0	481.17	0	481.17	0	481.17	0	152.5	**481.17***	106.8
a4-16	**222.49**	0	222.49	0	222.49	0	222.49	0	19.5	**222.49***	3.6
a4-24	**310.84**	0	310.84	0	310.84	0	312.44	0.51%	29.6	**310.84***	31.2
a4-32	**393.96**	0	393.95	0	395.12	0.29%	397.58	0.92%	52.0	**393.96***	612.0
a4-40	**453.84**	0	458.22	0.97%	459.42	1.23%	460.56	1.48%	92.0	**453.84***	517.2
a4-48	555.93	0.25%	560.19	1.02%	561.26	1.21%	562.87	1.50%	141.8	**554.54**	7200.0
a5-40	414.80	0.07%	418.48	0.96%	420.35	1.41%	422.56	1.94%	64.9	**414.51***	1141.8
a5-50	561.41	0.40%	567.82	1.55%	570.58	2.04%	573.51	2.56%	137.3	**559.17**	7200.0
Summary		0.05%		0.32%		0.44%		0.64%	78.6		1210.5
$\gamma = 0.4$	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
a2-16	**237.38**	0	237.38	0	237.38	0	237.38	0	52.9	**237.38***	1.8
a2-20	**280.70**	0	280.70	0	280.70	0	280.70	0	140.7	**280.70***	49.8
a2-24	**347.04****	-0.29%	347.04	-0.29%	347.04	-0.29%	347.04	-0.29%	231.0	*348.04**	25.2
a3-18	**236.82**	0	236.82	0	236.82	0	236.82	0	26.3	**236.82***	4.2
a3-24	**274.80**	0	274.80	0	274.80	0	276.11	0.48%	67.9	**274.80***	16.8
a3-30	**413.34****	-0.01%	413.34	-0.01%	413.34	-0.01%	413.34	-0.01%	88.7	*413.37**	99.0
a3-36	**483.06****	-0.22%	483.83	-0.06%	483.86	-0.06%	485.43	0.27%	157.8	*484.14**	306.6
a4-16	**222.49**	0	222.49	0	222.49	0	222.49	0	19.4	**222.49***	5.4
a4-24	**311.03**	0	311.28	0.08%	311.65	0.20%	313.21	0.70%	32.0	**311.03***	39.6
a4-32	**394.26**	0	395.05	0.20%	397.21	0.75%	400.32	1.54%	63.0	**394.26***	681.6
a4-40	**453.84**	0	457.20	0.74%	459.46	1.24%	461.06	1.59%	116.7	**453.84***	417.6
a4-48	558.11	0.63%	561.40	1.23%	563.47	1.60%	565.35	1.94%	177.5	**554.60**	7200.0
a5-40	416.25	0.42%	418.97	1.08%	420.32	1.40%	422.75	1.99%	72.6	**414.51***	1221.0
a5-50	567.54	1.26%	572.23	2.09%	574.56	2.51%	576.11	2.79%	162.8	**560.50**	7200.0
Summary		0.13%		0.36%		0.52%		0.79%	100.7		1233.5
$\gamma = 0.7$	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
a2-16	**240.66**	0	240.66	0	240.66	0	240.66	0	95.8	**240.66***	5.4
a2-20	**293.27***	–	293.27	–	294.11	–	NA	NA	172.8	NA	7200.0
a2-24	**353.18****	-1.40%	366.49	2.31%	NA	NA	NA	NA	206.6	*358.21**	961.2
a3-18	**240.58**	0	240.58	0	240.58	0	240.58	0	58.3	**240.58***	48.0
a3-24	**275.97****	-0.63%	275.97	-0.63%	277.43	-0.10%	279.13	0.51%	123.7	*277.72**	152.4
a3-30	**424.93***	–	432.29	–	436.20	–	NA	NA	77.7	NA	7200.0
a3-36	**494.04**	0	497.11	0.62%	502.27	1.67%	505.95	2.41%	125.4	**494.04**	7200.0
a4-16	**223.13**	0	223.13	0	223.13	0	223.13	0	31.3	**223.13***	67.2
a4-24	**316.65****	-0.49%	318.21	0	318.31	0.03%	320.87	0.84%	53.7	*318.21**	1834.8
a4-32	**397.87***	-7.49%	401.58	-6.63%	405.85	-5.63%	408.69	-4.97%	71.4	430.07	7200.0
a4-40	**479.02***	–	NA	NA	NA	NA	NA	NA	114.7	NA	7200.0
a4-48	**582.22***	–	610.75	–	NA	NA	NA	NA	164.4	NA	7200.0
a5-40	**424.26***	-5.22%	433.12	-3.24%	436.94	-2.39%	441.15	-1.45%	97.5	447.63	7200.0
a5-50	**603.24***	–	NA	NA	NA	NA	NA	NA	158.4	NA	7200.0
Summary		–		–		–		–	110.8		4333.4

a: Due to incorrect big M values, some of the reported optimal results of Bongiovanni et al. (2019) are higher than our obtained solution values. Those results are highlighted in italics and our obtained results are marked in bold with double stars;

Table 3.5 Results of the proposed DA algorithm on type-u instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	Proposed DA algorithm, 10000 iterations, 50 runs									Bongiovanni et al.,[a]	
Instance	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
u2-16	**57.61**	0	57.61	0	57.61	0	57.61	0	120.1	**57.61***	21.0
u2-20	**55.59**	0	55.59	0	56.34	1.34%	56.34	1.34%	401.8	**55.59***	9.6
u2-24	**90.73****	-0.60%	90.84	-0.47%	90.84	-0.47%	90.98	-0.32%	599.7	*91.27**	432.0
u3-18	**50.74**	0	50.74	0	50.74	0	50.93	0.37%	108.3	**50.74***	10.8
u3-24	**67.56**	0	67.87	0.46%	68.16	0.89%	68.16	0.89%	111.5	**67.56***	130.2
u3-30	**76.75**	0	77.21	0.60%	77.80	1.37%	78.65	2.47%	174.1	**76.75***	438.0
u3-36	104.27	0.22%	104.87	0.79%	105.42	1.33%	106.36	2.23%	420.7	**104.04***	1084.8
u4-16	**53.58**	0	53.58	0	53.58	0	53.58	0	51.4	**53.58***	48.0
u4-24	90.13	0.34%	90.72	1.00%	90.85	1.14%	90.95	1.25%	55.3	**89.83***	13.2
u4-32	**99.29**	0	99.29	0	99.42	0.13%	99.67	0.38%	119.1	**99.29***	1158.6
u4-40	**133.11**	0	134.46	1.02%	135.18	1.55%	136.08	2.23%	154.0	**133.11***	185.4
u4-48	**147.75***	-0.37%	148.87	0.39%	149.69	0.93%	150.42	1.43%	841.0	148.30	7200.0
u5-40	**121.86**	0	123.11	1.03%	123.38	1.25%	124.47	2.14%	113.8	**121.86**	1141.8
u5-50	144.22	0.78%	145.04	1.36%	145.63	1.77%	146.30	2.24%	245.5	**143.10**	7200.0
Summary		0.03%		0.44%		0.80%		1.19%	251.2		1795.1
$\gamma = 0.4$	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
u2-16	**57.65**	0	57.65	0	57.65	0	57.65	0	156.6	**57.65***	25.8
u2-20	**56.34**	0	56.34	0	56.34	0	56.34	0	606.6	**56.34***	12.0
u2-24	**91.24****	-0.43%	91.27	-0.39%	91.72	0.10%	92.06	0.47%	817.8	*91.63**	757.2
u3-18	**50.74**	0	50.74	0	50.74	0	50.99	0.50%	125.0	**50.74***	13.8
u3-24	**67.56**	0	67.87	0.46%	68.16	0.89%	68.16	0.89%	141.0	**67.56***	220.8
u3-30	**76.75**	0	77.12	0.48%	77.93	1.54%	78.65	2.48%	285.8	**76.75***	336.6
u3-36	104.49	0.41%	105.65	1.53%	106.37	2.22%	107.19	3.01%	898.9	**104.06***	2010.0
u4-16	**53.58**	0	53.58	0	53.58	0	53.58	0	60.5	**53.58***	44.4
u4-24	90.72	1.00%	90.72	1.00%	91.00	1.30%	91.12	1.44%	65.6	**89.83***	28.2
u4-32	**99.29**	0	99.29	0	99.42	0.13%	99.90	0.61%	156.3	**99.29***	2667.6
u4-40	**133.78****	-0.10%	135.43	1.14%	135.83	1.44%	136.56	1.98%	303.1	*133.91**	2653.2
u4-48	**148.48***	–	149.86	–	150.81	–	151.77	–	1390.7	NA	7200.0
u5-40	**121.96***	-0.22%	123.08	0.69%	123.63	1.15%	124.42	1.79%	160.8	122.23	7200.0
u5-50	143.68	0.38%	145.66	1.76%	146.60	2.42%	147.15	2.80%	391.5	**143.14**	7200.0
Summary		–		–		–		–	397.2		2169.3
$\gamma = 0.7$	BC	$BC\%$	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)	BC'	CPU′(s)
u2-16	**59.19**	0	59.26	0.11	60.01	1.38	60.19	1.69	419.6	**59.19***	338.4
u2-20	**56.86**	0	58.39	2.69	58.39	2.69	58.88	3.55	1527.6	**56.86***	72.0
u2-24	**92.84***	–	94.33	–	99.38	–	NA	NA	502.5	NA	7200.0
u3-18	**50.99**	0	50.99	0	50.99	0	50.99	0	206.9	**50.99***	24.0
u3-24	**68.39**	0	68.39	0	68.44	0.08%	68.73	0.49%	375.8	**68.39***	400.2
u3-30	**77.94****	-0.26%	78.72	0.74%	79.37	1.57%	79.56	1.81%	1094.8	*78.14**	3401.4
u3-36	106.00	0.20%	106.41	0.59%	107.57	1.68%	107.92	2.01%	1606.4	**105.79**	7200.0
u4-16	**53.87**	0	53.87	0	53.87	0	53.87	0	96.9	**53.87***	88.8
u4-24	90.07	0.12%	90.97	1.12%	90.97	1.12%	90.97	1.12%	254.5	**89.96***	22.8
u4-32	**99.50**	0	100.01	0.51%	101.09	1.60%	101.75	2.26%	325.3	**99.50***	2827.2
u4-40	**136.08***	–	137.65	–	138.98	–	NA	NA	708.0	NA	7200.0
u4-48	**152.58***	–	157.85	–	162.62	–	NA	NA	1958.8	NA	7200.0
u5-40	**123.52***	–	125.30	–	126.10	–	127.08	–	359.6	NA	7200.0
u5-50	**143.51***	-0.59%	148.16	2.64%	149.52	3.58%	152.36	5.54%	922.2	144.36	7200.0
Summary		–		–		–		–	780.1		3598.2

a: Due to incorrect big M values, some of the reported optimal results of Bongiovanni et al. (2019) are higher than our obtained solution values. Those results are highlighted in italics and our obtained results are marked in bold with double stars;

increases, but the dispersion remains quite acceptable. For instance r7-84, most of the runs with $\gamma = 0.4$ do not find a feasible solution. For instance r8-96, our DA algorithm cannot find a feasible solution among 50 runs with $\gamma = 0.4$. These instances seem challenging for future works.

When $\gamma = 0.7$, we found no feasible solution for all the type-r instances, despite 50 runs and 10000 iterations. One reason is that many of these instances are too constrained to be feasible for $\gamma = 0.7$ with the limitation of visiting recharging stations. However, it opens a perspective to prove it using exact methods with lower bounds.

Table 3.6 Results of the proposed DA algorithm with 10000 iterations 50 runs on type-r instances under $\gamma = 0.1, 0.4$

$\gamma = 0.1$	BC	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)
r5-60	691.83	699.93	1.17%	706.20	2.08%	710.43	2.69%	178.44
r6-48	506.72	509.67	0.58%	512.69	1.18%	515.39	1.71%	229.31
r6-60	692.00	696.67	0.67%	700.15	1.18%	703.95	1.73%	127.03
r6-72	777.44	788.12	1.37%	794.69	2.22%	801.87	3.14%	208.39
r7-56	613.10	620.69	1.24%	624.51	1.86%	630.72	2.87%	88.20
r7-70	760.90	772.45	1.52%	778.84	2.36%	786.02	3.30%	209.76
r7-84	889.38	900.34	1.23%	904.88	1.74%	913.88	2.75%	322.66
r8-64	641.99	647.87	0.92%	652.59	1.65%	657.49	2.41%	612.06
r8-80	803.52	820.96	2.17%	828.67	3.13%	834.19	3.82%	357.75
r8-96	1053.11	1069.98	1.60%	1080.80	2.63%	1089.96	3.50%	363.46
Summary			1.25%		2.00%		2.79%	269.71
$\gamma = 0.4$	BC	$Q1$	$Q1\%$	AC	$AC\%$	$Q3$	$Q3\%$	CPU(s)
r5-60	697.97	710.30	1.77%	718.44	2.93%	727.27	4.20%	293.25
r6-48	506.91	509.48	0.51%	514.46	1.49%	517.53	2.10%	257.59
r6-60	694.78	702.67	1.14%	706.07	1.62%	710.80	2.31%	173.43
r6-72	799.60	811.85	1.53%	821.17	2.70%	832.07	4.06%	349.98
r7-56	613.66	620.58	1.13%	624.40	1.75%	627.51	2.26%	99.91
r7-70	766.05	778.70	1.65%	784.54	2.41%	791.07	3.27%	273.52
r7-84	932.12	964.04	3.43%	NA	NA	NA	NA	584.26
r8-64	638.36	649.84	1.80%	652.30	2.18%	657.02	2.92%	641.63
r8-80	811.19	823.70	1.54%	833.05	2.69%	841.76	3.77%	448.14
r8-89	NA	NA	NA	NA	NA	NA	NA	617.17
Summary			NA		NA		NA	373.89

Conclusion of algorithm performance

On both type-a and -u instances, we observe the limit of solving capabilities of the B&C. Even with a time limit of two hours, it is difficult for B&C to solve medium-to-large-sized E-ADARP instances, especially under a high energy restriction. Our DA algorithm can continuously provide high-quality solutions for highly constrained instances within a reasonable computational time. We also show that our DA algorithm can tackle larger-sized instances with up to 8 vehicles and 96 requests. Nineteen type-r instances for $\gamma = 0.1$ and $\gamma = 0.4$ are solved feasibly, and these results are the first solutions found for these new instances, which can serve as a benchmark for future studies. To conclude, the proposed DA algorithm remains highly effective and can provide optimal/near-optimal solutions even facing highly constrained instances. The proposed DA algorithm significantly outperforms the B&C algorithm for medium-to-large-sized instances, and its consistency seems quite acceptable for such difficult instances.

3.3.4 Sensitivity analysis of the maximum number of charging visits per station

As discussed in Section 2.5, the hypothesis of visiting each recharging station at most once is not realistic. We adjust our DA algorithm as mentioned in Section 3.2.4 to allow multiple visits to each recharging station. The adjusted DA algorithm is able to investigate the effect of increasing the value of n_{as} on solution cost and feasibility. Recalling that we analyze four different cases: $n_{as} = 1, 2, 3, \infty$.

For type-a instances, as in the scenario of $\gamma = 0.1$, we obtain optimal solutions for most of the instances, and other instances are solved without visiting recharging stations. Therefore, we focus on scenarios of $\gamma = 0.4, 0.7$ and analyze the effect of allowing multiple visits in these cases. For type-u and -r instances, we conduct experiments with adjusted DA algorithm with $n_{as} = 2, 3, \infty$ under $\gamma \in \{0.1, 0.4, 0.7\}$. The detailed results are presented as below. In Table 3.7 and 3.9, we compare DA algorithm results on each instance with setting $n_{as} = 1, 2, 3, \infty$ and we mark the best one(s) in bold. In Table 3.8, we compare our algorithm results under each setting of n_{as} with the reported results in Bongiovanni et al. (2019). Improved solutions are marked in bold with an asterisk while equal solutions are marked in bold. In the column of BC_{∞}, if the obtained solution is better than other solutions obtained under $n_{as} = 1, 2, 3$, we mark it in bold with double stars. On each instance, the adjusted DA algorithm performs 50 runs with 10000 iterations per run. We report the maximum number of recharging visits experienced on a station (denoted as N^s_{max}) for the best-obtained solution under $n_{as} = \infty$ in the column named "N^s_{max}". In addition, we also report the average number of visited recharging stations under setting $n_{as} = \infty$ in the column of "$\overline{N^s_{avg}}$".

From these results, we observe that the previous difficulties for the DA algorithm to solve the E-ADARP instances are reduced considering multiple visits per station. The major findings are: (1) significant increases on $\overline{N^s_{avg}}$ are observed on all instances with increasing γ value, especially on type-r instances, where the average value of $\overline{N^s_{avg}}$ is tripled when γ changes to 0.7; (2) allowing multiple visits to each recharging station improves the solution quality as we found lower-cost solutions. Particularly, we obtain feasible solutions for all type-r instances under $\gamma = 0.7$ with $n_{as} = 3, \infty$, while no feasible solution is found with $n_{as} = 1$; (3) for type-a and -r instances, relaxing to $n_{as} = \infty$ seems to be more computationally attractive as it does not introduce additional computational time, compared to the results obtained by replicating recharging stations. For type-u instances, having a pre-calculated n_{as} would be more computationally favorable; (4) on average, allowing at-most-two and -three visits per station slightly increases the computational time. Allowing at-most-three visits per station seems to strike a good balance between solution quality and computational time; (5) $n_{as} = 3$ seems to be a good upper bound for solving type-u instances allowing multiple recharging visits, while one needs to set n_{as} to 4 and 7 for type-a and -r instances, respectively. A potential perspective from these results would be to investigate more realistic constraints, e.g., on the capacity of recharging stations, rather than limiting visits to recharging stations in the E-ADARP. Another direction for future studies is to design a heuristic to calculate the upper bound on the total number of recharging visits for a given route.

Table 3.7 Solution quality and performance on type-a instances when increasing the maximum number of charging visits per station

	DA with $n_{as}=1$			DA with $n_{as}=2$			DA with $n_{as}=3$			DA with $n_{as}=\infty$				
$\gamma=0.4$	BC	AC	CPU	BC_2	AC_2	CPU_2	BC_3	AC_3	CPU_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
a2-16	**237.38**	237.38	52.85	**237.38**	237.38	52.65	**237.38**	237.38	53.07	**237.38**	237.38	50.65	1	1.0
a2-20	**280.70**	280.70	140.70	**280.70**	280.70	148.12	**280.70**	280.70	141.97	**280.70**	280.70	144.92	1	1.0
a2-24	347.04	347.04	230.99	**346.28**	346.28	286.96	**346.28**	346.28	284.47	**346.28**	346.28	265.80	2	3.0
a3-18	**236.82**	236.82	26.30	**236.82**	236.82	26.93	**236.82**	236.82	26.43	**236.82**	236.82	25.36	0	0.0
a3-24	**274.80**	274.80	67.85	**274.80**	274.80	71.07	**274.80**	274.80	69.48	**274.80**	274.80	66.66	1	0.8
a3-30	**413.34**	413.34	88.67	**413.34**	413.34	104.70	**413.34**	413.34	106.13	**413.34**	413.34	103.54	1	2.0
a3-36	483.06	483.86	157.79	**481.17**	481.46	255.25	**481.17**	481.17	264.23	**481.17**	481.17	248.95	3	3.0
a4-16	**222.49**	222.49	19.39	**222.49**	222.49	19.71	**222.49**	222.49	19.08	**222.49**	222.49	17.78	0	0.0
a4-24	**311.03**	311.65	31.97	**311.03**	311.65	31.54	**311.03**	311.65	31.15	**311.03**	311.65	29.53	0	0.0
a4-32	**394.26**	397.21	62.95	**394.26**	397.31	65.66	**394.26**	397.21	63.85	**394.26**	397.27	61.71	1	1.0
a4-40	**453.84**	459.46	116.65	**453.84**	459.18	125.28	**453.84**	459.11	116.86	**453.84**	458.74	121.04	1	0.7
a4-48	558.11	563.47	177.51	558.18	564.63	235.32	**557.86**	564.21	238.60	558.96	564.86	231.45	2	2.7
a5-40	416.25	420.32	72.64	415.62	420.09	71.75	**415.43**	420.16	72.01	415.79	419.82	70.78	0	0.1
a5-50	567.54	574.56	162.82	**564.90**	575.04	190.93	567.40	574.64	189.18	567.13	574.28	184.43	1	1.6
Avg			100.65			120.42			119.75			115.90	1.0	1.2
$\gamma=0.7$	BC	AC	CPU	BC_2	AC_2	CPU_2	BC_3	AC_3	CPU_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
a2-16	**240.66**	240.66	95.75	**240.66**	240.66	125.10	**240.66**	240.66	124.90	**240.66**	240.66	119.29	2	3.0
a2-20	293.27	294.11	172.77	286.52	286.52	331.90	**285.86**	285.86	327.01	286.52	288.89	316.22	2	3.6
a2-24	353.18	NA	206.58	352.25	363.17	373.77	**350.49**	361.02	390.86	354.38	374.68	357.33	2	3.9
a3-18	240.58	240.58	58.30	**238.82**	238.82	70.27	**238.82**	238.82	69.52	**238.82**	238.82	65.89	3	4.0
a3-24	275.97	277.43	123.71	**275.20**	275.20	154.90	**275.20**	275.94	155.39	**275.20**	275.20	150.02	2	2.9
a3-30	424.93	436.20	77.73	416.87	417.90	173.80	**415.71**	417.35	176.38	**415.71**	417.07	170.95	3	4.7
a3-36	494.04	502.27	125.42	486.36	487.34	332.47	**484.85**	487.59	350.73	**484.85**	487.91	343.02	3	4.8
a4-16	223.13	223.13	31.32	**222.49**	223.13	33.40	**222.49**	222.49	36.24	**222.49**	222.49	31.37	2	1.7
a4-24	316.65	318.31	53.73	**315.98**	317.99	74.82	**315.98**	317.99	80.77	**315.98**	317.99	70.97	2	2.7
a4-32	397.87	405.85	71.44	395.84	402.85	127.78	**394.99**	402.38	142.98	**394.94**	401.82	123.77	4	3.7
a4-40	479.02	NA	114.74	458.98	467.15	235.88	458.73	465.04	250.11	**458.52**	467.60	226.05	3	4.6
a4-48	582.22	NA	164.39	569.23	576.26	379.04	**566.26**	577.30	434.97	568.08	575.96	403.27	2	5.3
a5-40	424.26	436.94	97.51	417.35	424.29	153.00	**416.89**	423.96	169.49	419.33	425.29	149.77	4	4.3
a5-50	603.24	NA	158.39	583.37	590.81	320.55	**576.54**	589.38	367.00	579.15	588.98	352.73	4	5.7
Avg			110.84			206.19			219.74			205.76	2.7	3.9

Table 3.8 Solution quality and performance on type-u instances when increasing the maximum number of charging visits per station

	DA with $n_{as}=1$				DA with $n_{as}=2$				DA with $n_{as}=3$				DA with $n_{as}=\infty$				
$\gamma=0.1$	BC	AC	CPU	BC'	BC_2	AC_2	CPU_2	BC'_2	BC_3	AC_3	CPU_3	BC'_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
u2-16	**57.61**	57.61	120.06	57.61*	**57.61**	57.61	124.25	57.61*	**57.61**	57.61	126.90	57.61*	**57.61**	57.61	182.64	1	0.8
u2-20	**55.59**	56.34	401.82	55.59*	**55.59**	55.59	421.77	55.59*	**55.59**	55.59	440.65	55.59*	**55.59**	55.59	642.95	1	0.6
u2-24	**90.73***	90.84	599.73	*91.27**	**90.73***	90.73	572.00	*91.27**	**90.73***	90.73	592.75	*91.27**	**90.73**	90.73	1021.42	1	2.9
u3-18	**50.74**	50.74	108.32	50.74*	**50.74**	50.74	111.63	50.74*	**50.74**	50.74	112.69	50.74*	**50.74**	50.74	172.79	0	0.0
u3-24	**67.56**	68.16	111.49	67.56*	**67.56**	68.16	115.43	67.56*	**67.56**	68.16	117.24	67.56*	**67.56**	68.16	173.10	0	0.0
u3-30	**76.75**	77.80	174.11	76.75*	**76.75**	77.55	182.98	76.75*	**76.75**	77.55	168.08	76.75*	**76.75**	77.55	268.96	0	0.3
u3-36	104.27	105.42	420.72	104.04*	104.27	105.45	578.30	104.04*	104.27	106.10	552.64	104.04*	104.27	105.48	775.41	1	1.7
u4-16	**53.58**	53.58	51.37	53.58*	**53.58**	53.58	51.14	53.58*	**53.58**	53.58	49.18	53.58*	**53.58**	53.58	72.84	0	0.0
u4-24	90.13	90.85	55.26	89.83*	89.91	90.85	57.23	89.83*	90.08	90.85	56.87	89.83*	90.08	90.85	79.82	1	0.5
u4-32	**99.29**	99.42	119.12	99.29*	**99.29**	99.42	114.88	99.29*	**99.29**	99.42	118.06	99.29*	**99.29**	99.42	162.58	0	0.4
u4-40	**133.11**	135.18	154.00	133.11*	**133.11**	135.34	163.78	133.11*	133.14	135.21	159.92	133.11*	**133.11**	135.23	216.58	1	1.7
u4-48	**147.75***	149.69	840.96	148.30	**147.73***	149.89	917.71	148.37	**147.43***	149.52	902.77	149.14	**147.33****	149.37	1403.39	2	2.9
u5-40	**121.86**	123.38	113.81	121.86	**121.86**	123.54	116.57	121.86	**121.86**	123.74	118.30	121.86	**121.86**	123.59	149.98	1	0.8
u5-50	144.22	145.63	245.52	143.10	143.27	145.73	258.43	142.83	143.51	145.91	279.38	142.83	**143.14****	146.05	393.68	1	1.5
Avg			251.16				270.43				271.10				408.30	0.7	1.0
$\gamma=0.4$	BC	AC	CPU	BC'	BC_2	AC_2	CPU_2	BC'_2	BC_3	AC_3	CPU_3	BC'_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
u2-16	**57.65**	57.65	156.61	57.65*	**57.65**	57.65	171.29	57.65*	**57.65**	57.65	168.11	57.65*	**57.65**	57.65	276.29	1	2.0
u2-20	**56.34**	56.34	606.64	56.34*	**56.34**	56.34	690.19	56.34*	**56.34**	56.34	682.00	56.34*	**56.34**	56.34	1006.29	1	2.0
u2-24	**91.24***	91.72	817.79	*91.63**	**91.14***	91.43	836.02	*91.27**	**91.14***	91.43	885.85	*91.27**	91.16	91.17	1399.38	2	3.3
u3-18	**50.74**	50.74	124.95	50.74*	**50.74**	50.74	129.61	50.74*	**50.74**	50.74	133.92	50.74*	**50.74**	50.74	213.60	1	1.1
u3-24	**67.56**	68.16	141.01	67.56*	67.86	68.06	145.32	67.56*	67.67	68.16	153.68	67.56*	**67.56**	68.16	214.32	1	1.2
u3-30	**76.75**	77.93	285.81	76.75*	**76.75**	78.13	306.82	76.75*	**76.75**	78.28	298.28	76.75*	**76.75**	77.85	420.10	1	2.1
u3-36	104.49	106.37	898.90	104.06*	**104.06**	106.68	1038.76	104.06*	104.69	106.57	1078.92	104.06*	104.31	106.07	1589.46	2	3.5
u4-16	**53.58**	53.58	60.52	53.58*	**53.58**	53.58	62.49	53.58*	**53.58**	53.58	63.21	53.58*	**53.58**	53.58	85.00	0	0.0
u4-24	90.72	91.00	65.57	89.83*	90.21	90.90	68.48	89.83*	90.13	90.90	70.04	89.83*	**90.08****	90.85	91.67	2	2.1
u4-32	**99.29**	99.42	156.27	99.29*	**99.29**	99.42	166.76	99.29*	**99.29**	99.42	162.08	99.29*	**99.29**	99.42	230.20	1	2.7
u4-40	**133.78***	135.83	303.06	*133.91**	**133.61***	135.75	318.07	*133.68**	134.23	136.16	326.55	134.01	**133.36****	136.19	457.33	2	4.2
u4-48	**148.48***	150.81	1390.74	NA	**148.18***	150.53	1247.04	150.96	**148.23***	150.21	1454.38	150.78	**147.75****	149.71	2050.93	2	4.9
u5-40	**121.96***	123.63	160.80	122.23	**121.96***	123.50	163.39	122.22	**121.96**	123.77	166.90	121.96	**121.96**	123.94	237.16	1	3.3
u5-50	143.68	146.60	391.46	143.14	143.78	146.36	401.78	142.83	143.50	146.21	415.65	143.48	**143.42****	145.65	619.05	1	4.1
Avg			397.15				410.43				432.83				835.06	1.3	2.6
$\gamma=0.7$	BC	AC	CPU	BC'	BC_2	AC_2	CPU_2	BC'_2	BC_3	AC_3	CPU_3	BC'_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
u2-16	**59.19**	60.01	419.57	59.19*	**58.17**	58.17	460.44	58.17*	**58.17**	58.17	530.24	58.17*	58.75	59.46	663.32	2	3.3
u2-20	**56.86**	58.39	1527.60	56.86*	**56.86**	58.03	1561.63	56.86*	**56.86**	57.98	1583.70	56.86*	**56.86**	58.39	2619.96	1	2.8
u2-24	**92.84***	99.38	1065.06	NA	**92.43***	105.67	1307.99	97.50	**92.43***	101.95	1529.29	NA	92.77	100.36	2090.28	2	5.0
u3-18	**50.99**	50.99	206.92	50.99*	**50.99**	50.99	206.48	50.99*	**50.99**	50.99	217.78	50.99*	**50.99**	50.99	301.43	1	3.0
u3-24	**68.39**	68.44	375.75	68.39*	68.24	68.39	389.47	68.06*	68.24	68.51	419.27	68.06*	**68.06****	68.41	544.52	2	3.8
u3-30	**77.94***	79.37	1094.81	*78.14**	**77.94***	79.09	1132.97	78.16	**77.94***	79.02	1293.92	78.16	**77.83****	79.11	1595.22	2	4.2
u3-36	106.00	107.57	1606.43	105.79	**106.39***	107.62	1521.37	107.65	106.39	107.07	1605.03	106.18	**105.98****	106.95	2690.77	2	4.5
u4-16	**53.87**	53.87	96.90	53.87*	**53.87**	53.87	100.33	53.87*	**53.87**	53.87	103.29	53.87*	**53.87**	53.87	133.65	1	3.0
u4-24	90.07	90.97	254.45	89.96*	89.96	90.97	263.46	89.83*	89.91	90.97	282.62	89.83	**89.83****	90.72	375.00	2	3.9
u4-32	**99.50**	101.09	325.31	99.50*	**99.50**	99.95	321.35	99.50*	**99.50**	100.34	342.44	99.50*	**99.50**	100.28	526.67	1	4.6
u4-40	**136.08***	138.98	708.04	NA	**134.98***	138.37	731.95	137.49	**135.38***	138.01	730.23	137.61	**134.94****	136.20	971.29	2	5.5
u4-48	**152.58***	162.62	1958.80	NA	**150.55***	154.19	1962.85	NA	**151.57***	155.36	1955.60	NA	**149.51****	152.90	2907.41	3	6.3
u5-40	**123.52***	126.10	359.59	NA	**124.04***	126.08	385.25	125.14	**123.71***	125.63	401.18	124.18	**123.32****	125.15	506.11	2	5.4
u5-50	**143.51***	149.52	922.19	144.36	**144.24***	148.13	923.51	164.19	**143.51***	148.53	1001.25	144.10	**142.89****	146.10	1165.39	2	6.1
Avg			780.10				804.93				856.84				1220.79	1.8	4.4

Table 3.9 Solution quality and performance on type-r instances when increasing the maximum number of charging visits per station

	DA with $n_{as}=1$			DA with $n_{as}=2$			DA with $n_{as}=3$			DA with $n_{as}=\infty$				
$\gamma=0.1$	BC	AC	CPU	BC_2	AC_2	CPU_2	BC_3	AC_3	CPU_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
r5-60	691.83	706.20	178.44	689.75	703.86	175.42	688.52	706.91	180.86	**687.68**	705.59	171.75	0	0.2
r6-48	506.72	512.69	229.31	**506.45**	513.62	241.23	507.03	513.63	231.32	506.91	514.15	241.89	0	0.0
r6-60	692.00	700.15	127.03	**690.15**	701.15	133.74	692.24	701.86	137.18	691.07	702.09	128.33	0	0.0
r6-72	777.44	794.69	208.39	776.68	795.41	212.78	**775.93**	793.96	208.77	777.46	795.14	210.51	1	0.1
r7-56	**613.10**	624.51	88.20	614.61	623.65	91.27	615.61	623.52	84.50	614.18	622.69	87.32	0	0.0
r7-70	760.90	778.84	209.76	761.16	776.92	212.08	761.25	778.05	202.26	**760.10**	777.10	202.03	0	0.0
r7-84	889.38	904.88	322.66	**884.43**	903.96	318.05	890.47	905.78	339.95	885.89	905.13	300.21	0	0.1
r8-64	641.99	652.59	612.06	**640.05**	653.65	645.07	642.09	653.44	773.82	640.24	653.81	647.97	0	0.0
r8-80	803.52	828.67	357.75	807.04	826.91	366.82	**799.00**	826.71	376.87	804.02	826.92	372.21	0	0.0
r8-96	1053.11	1080.80	363.46	1052.19	1078.29	358.23	1064.64	1081.49	377.77	**1049.98**	1077.21	366.73	0	0.4
Avg			269.71			275.47			291.33			272.90	0.1	0.1
$\gamma=0.4$	BC	AC	CPU	BC_2	AC_2	CPU_2	BC_3	AC_3	CPU_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
r5-60	697.97	718.44	293.25	703.00	721.56	308.94	692.84	710.40	288.01	**691.72**	709.78	285.00	2	3.0
r6-48	506.91	514.46	257.59	**506.45**	511.62	248.38	506.75	511.00	258.81	507.25	514.64	255.83	0	0.1
r6-60	694.78	706.07	173.43	693.80	706.11	175.96	693.03	703.13	174.80	**692.83**	701.86	174.24	1	1.7
r6-72	799.60	821.17	349.98	795.88	814.03	342.96	**776.17**	800.29	336.47	781.22	801.86	342.33	1	3.3
r7-56	613.66	624.40	99.91	**612.76**	625.42	98.97	616.24	623.58	100.81	615.74	623.51	99.11	0	0.2
r7-70	766.05	784.54	273.52	763.46	785.69	275.48	**760.09**	783.13	280.49	761.58	778.04	273.50	1	1.5
r7-84	932.12	NA	584.26	897.50	932.05	488.49	897.34	915.24	446.76	**896.91**	916.23	456.77	3	3.4
r8-64	638.36	652.30	641.63	642.34	652.65	646.45	639.01	652.80	671.52	**637.84**	652.17	719.50	0	0.2
r8-80	811.19	833.05	448.14	816.17	834.80	438.40	**808.14**	828.89	420.03	813.16	829.92	450.94	1	1.1
r8-96	NA	NA	617.17	1089.18	1129.20	588.26	1060.48	1098.13	545.21	**1058.41**	1090.04	564.49	5	4.6
Avg			373.89			361.23			352.29			362.17	1.4	1.9
$\gamma=0.7$	BC	AC	CPU	BC_2	AC_2	CPU_2	BC_3	AC_3	CPU_3	BC_∞	AC_∞	CPU_∞	N^s_{max}	$\overline{N^s_{avg}}$
r5-60	NA	NA	507.76	731.84	770.95	484.01	**704.97**	725.74	483.86	708.54	723.73	492.51	5	6.9
r6-48	NA	NA	502.21	518.87	540.88	507.06	509.80	525.98	486.31	**509.76**	525.10	483.94	3	5.0
r6-60	NA	NA	327.25	716.48	741.76	300.67	700.82	713.33	306.60	**697.57**	711.52	289.76	6	7.0
r6-72	NA	NA	590.56	920.61	NA	605.16	798.26	817.20	561.24	**796.19**	826.48	574.02	4	8.4
r7-56	NA	NA	221.09	644.19	662.06	208.57	**622.66**	640.69	210.29	625.91	641.82	212.05	3	7.0
r7-70	NA	NA	510.60	866.06	NA	507.14	**777.85**	803.20	465.43	781.56	800.35	480.03	6	7.8
r7-84	NA	NA	790.95	NA	NA	753.17	**906.14**	938.15	623.70	915.61	938.49	705.25	6	8.7
r8-64	NA	NA	1207.35	664.02	698.61	1170.20	**647.02**	666.20	1185.16	649.93	668.48	1290.02	7	6.3
r8-80	NA	NA	868.04	966.47	NA	846.51	**829.54**	857.56	707.30	843.26	865.90	744.33	4	8.2
r8-96	NA	NA	860.97	NA	NA	845.14	1105.82	1145.82	646.04	**1097.76**	1136.43	806.99	7	11.2
Avg			638.68			622.76			567.59			607.89	5.1	7.7

3.4 Conclusion

This chapter proposes an efficient DA algorithm to solve the E-ADARP, which aims to minimize a weighted-sum objective, including the total travel time and the total excess user ride time. To minimize the total excess user ride time, we propose a fragment-based representation of paths. A new method is developed upon this representation to calculate the minimum excess user ride time for a given route. Another challenge in solving the E-ADARP involves incorporating the partial recharging at recharging stations, which complicates the feasibility checking of a given route; to resolve this issue, we propose an exact route evaluation scheme of linear time complexity that can accurately handle the effect of allowing partial recharging and validate the feasibility of solutions. These two methods compose an exact and efficient optimization of excess user ride time for an E-ADARP route. To the best of our knowledge, this is the first time that total excess user ride time is optimized in an exact way for the E-ADARP.

In computational experiments, we first prove the effectiveness and accuracy of our DA algorithm compared to the best-reported results of Bongiovanni et al. (2019). On 84 existing E-ADARP instances, our DA algorithm obtains equal solutions for 45 instances and provides better solutions on 25 instances. We also demonstrate that the proposed DA algorithm can consistently provide high-quality solutions in a short computational time. On the previously solved instances, the DA algorithm improves the solution quality by 0.16% on average. On newly introduced large-scale E-ADARP instances, we provide new solutions for 19 instances. These results may serve as benchmark results for future studies. We then extend the E-ADARP model to allow unlimited visits to each recharging station. The previous difficulties for DA local search are lessened under this more realistic situation, and the results are less dispersed than the results of the at-most-one visit to each recharging station. Our extension of the E-ADARP model thus offers a new perspective in proposing a more realistic constraint in the E-ADARP for recharging stations, e.g., considering capacity and scheduling constraints in recharging stations.

Chapter 4

Branch-and-Price Algorithm to Solve the Electric Autonomous Dial-A-Ride Problem

This chapter is based on our journal paper "Column generation for solving the electric autonomous dial-a-ride problem" (arxiv link: https://arxiv.org/pdf/2206.13496.pdf), which is currently in revision. Some preliminary results have been reported at the "best student article prize" session of Roadef conference and at Odysseus conference. In this chapter, we present a highly efficient CG algorithm, which is integrated into the Branch-and-price (B&P) scheme to solve the E-ADARP exactly. The core part of the CG algorithm is to design an effective labeling algorithm that computes the shortest paths efficiently in each iteration. In this process, one must determine an excess-user-ride-time optimal schedule from battery-feasible schedules during label extension. This issue complicates solving the E-ADARP subproblems and cannot be handled exactly by existing DARP feasibility check methods (Gschwind & Irnich, 2015; Gschwind & Drexl, 2019) and scheduling procedures (Parragh et al., 2009; Molenbruch et al., 2017; Bongiovanni et al., 2022a). To handle this issue, we first present a fragment-based representation of paths, which was first introduced by Su et al. (2023). A novel approach is invoked to abstract fragments to arcs while ensuring excess-user-ride-time optimality. We then construct a new graph that preserves all feasible routes of the original graph by enumerating all feasible fragments, abstracting them to arcs, and connecting them with each other, depots, and recharging stations in a feasible way. On the new graph, we apply strong dominance rules and constant-time feasibility checks to compute the shortest paths efficiently. These methods pave the way for a powerful labeling algorithm that ensures excess-user-ride-time optimality in label extension.

This chapter is organized into four sections. In Section 4.1, we present the extended formulation of the E-ADARP and the CG framework. Section 4.2 introduces the proposed labeling algorithm to solve the pricing subproblems of the E-ADARP. Section 4.3 presents the cutting planes that strengthen the lower bounds and the principle of the B&P framework. The results of the proposed CG algorithm and the developed B&P algorithm are presented in Section 4.4.

Finally, Section 4.5 discusses conclusions.

4.1 Extended Formulation of the E-ADARP and CG subproblem

The E-ADARP consists of finding a set of E-ADARP routes for EAVs to transport users with specific origin-destination pairs such that each customer $i \in N$ is visited exactly once, and the weighted sum of total travel time and total excess user ride time is minimized. The definition of an E-ADARP route is as follows:

Definition 2 (E-ADARP route). An E-ADARP route is a path that starts at an origin depot and ends at a destination depot such that the following constraints are satisfied:

1) The route starts at an origin depot and ends at a destination depot and has no cycle;

2) Pairing and precedence constraints for pickup and drop-off nodes.

3) Time window, maximum user ride time, vehicle capacity, and battery capacity constraints;

4) The minimum battery level must be satisfied at the destination depot;

5) At-most-one visit to a recharging station if visited.

We obtain the extended formulation (also called "Master Problem", abbreviated as MP) of the E-ADARP via Dantzig-Wolfe decomposition. The MP is formulated as a set covering problem, where Ω denotes the set of all E-ADARP routes. For each route $\omega \in \Omega$, we define c_ω as the cost of route ω. Note that the route cost accounts for the weighted sum of total travel time and total excess user ride time. We define $\theta_{i\omega}$ as a binary coefficient that equals one if the request i is visited by route ω (zero otherwise). Let y_ω denote a binary variable that equals one if and only if route $\omega \in \Omega$ is included in the solution (0 otherwise). The number of requests to be served is n. To restrict the visits to each recharging station and destination depot, we define another binary coefficient $\phi_{f\omega}$, determining whether the recharging station or destination depot f is visited in route ω. As we have multiple origin depots which may be located at different places and cannot be visited repeatedly by different vehicles, we define $\epsilon_{o\omega}$ to denote whether origin depot o is visited in ω. The objective function of MP is formulated as the total routing cost and we improve the formulation of MP by adding a high penalty P_i for request i not served in the objective function. The benefit of adding penalties is that we can start from a heuristic solution of the E-ADARP (e.g., obtained from the DA algorithm of Su et al. (2023)) that does not include all the requests. Also, we introduce a binary variable, denoted as a_i, to represent whether request i is visited or not. If $a_i = 1$, request i is omitted, otherwise, request i is visited. The set covering problem is formulated as:

$$\min \sum_{\omega \in \Omega} c_\omega y_\omega + \sum_{i \in P} P_i a_i \tag{4.1}$$

subject to:

$$\sum_{\omega \in \Omega} \theta_{i\omega} y_\omega \geqslant 1 - a_i, \quad \forall i \in P \tag{4.2}$$

$$\sum_{\omega \in \Omega} \phi_{f\omega} y_\omega \leqslant 1, \quad \forall f \in S \cup F \tag{4.3}$$

$$\sum_{\omega \in \Omega} \epsilon_{o\omega} y_\omega \leqslant 1, \quad \forall o \in O \tag{4.4}$$

$$\sum_{\omega \in \Omega} y_\omega \leqslant |K| \tag{4.5}$$

$$y_\omega \in \{0, 1\}, \quad \forall \omega \in \Omega \tag{4.6}$$

$$a_i \in \{0, 1\}, \quad \forall i \in P \tag{4.7}$$

Constraints (4.2) and (4.3) restrict the visit to each request, recharging station, and destination depot. Constraints (4.4) and (4.5) guarantee that each origin depot appears at most once in the solution and at most $|K|$ vehicles are used. Due to the large size of Ω, we cannot solve the MP directly. Instead, we solve the linear relaxation of the MP on a subset of set Ω (denoted as Ω'), which we call the continuous Restricted Master Problem (hereafter continuous RMP). The subset Ω' can be generated by CG.

In CG, the continuous RMP and the pricing subproblems are solved iteratively. The subproblems are solved to generate E-ADARP routes with negative reduced costs. These routes are added to Ω', and the continuous RMP will be solved to update the dual variable values. With the renewed dual variable values, the subproblems will be solved again to find E-ADARP routes with negative reduced costs. The iterative solving of the continuous RMP and subproblems ends when no more negative-reduced-cost columns can be found. In this case, the optimal solution of the continuous MP is found. The integer RMP is solved at the end to obtain integer solutions by using all columns generated. Algorithm 2 outlines the CG algorithm framework.

4.1.1 Column Generation Subproblem

As mentioned, we solve the pricing subproblem in order to identify E-ADARP routes with negative reduced cost $\bar{c}_\omega, \omega \in \Omega$. The reduced cost for an E-ADARP route ω is formulated as:

$$c_\omega - \sum_{i \in P} \theta_{i\omega} \lambda_i - \sum_{f \in S \cup F} \phi_{f\omega} \tau_f - \sum_{o \in O} \epsilon_{o\omega} \zeta_o - \kappa \tag{4.8}$$

Algorithm 2 Column Generation Framework

Input: Initial columns pool population, preprocessing work;
Output: Optimal solution of continuous MP and best integer solution;
1: **Repeat**
2: Solve continuous RMP on Ω';
3: Update dual variables and optimal cost from continuous RMP;
4: Solve sub-problem exactly by labeling algorithm with updated dual variable values;
5: **if** objective value of sub-problem is negative **then**
6: Add all columns with negative reduced cost to column pool;
7: **end if**
8: **Until** no more negative-reduced-cost column can be found by labeling algorithm;
9: Solve integer RMP to obtain the best integer solution;
10: **Return** optimal solution of continuous MP and integer solution;

where $\lambda_i, i \in P$, $\tau_f, f \in S \cup F$, and $\zeta_o, o \in O$ are the dual variable values of constraints (4.2), (4.3), and (4.4), respectively. The dual variable values associated with constraint (4.5) is κ.

The objective function of the subproblem is:

$$\underset{\omega \in \Omega}{\text{minimize}}\, \bar{c}_\omega \tag{4.9}$$

4.2 Forward Labeling Algorithm for ESPPRC-MERT

We design a customized forward labeling algorithm to solve the pricing sub-problems, which are formulated as Elementary Shortest Path Problems with Resource Constraints and Minimizing Excess Ride Time (hereafter ESPPRC-MERT). This labeling algorithm extends the one tackling E-VRP subproblems in Desaulniers et al. (2016) by considering the following aspects:

1) The characteristics of the DARP are taken into account (i.e., pickup and delivery);

2) Problem-specific constraints (i.e., minimum-battery-level constraint, maximum user ride time constraint, limited visits to each recharging station) are considered;

3) Minimizing the total excess user ride time for a partial path.

The last point is the most challenging one of solving the ESPPRC-MERT, as the minimum excess user ride time is particularly difficult to be calculated in the extension of labels. This difficulty manifests in two aspects: (1) the minimum excess user ride time can only be determined at nodes where the vehicle has no passenger onboard at arrival/departure; (2) an excess-user-ride-time optimal schedule for a partial path may conflict with time window constraints on succeeding nodes.

To handle this issue, we construct a new sparser graph G_{sp}, where each arc is ensured to be excess-user-ride-time optimal. We propose an efficient labeling algorithm over G_{sp} to compute routes with negative reduced costs. The

construction of G_{sp} takes three steps:(1) we generate all battery-restricted fragments (defined in Definition 3); (2) we abstract each fragment to an arc that ensures excess-user-ride-time optimality; (3) we construct G_{sp} by connecting each transformed arc with depots, recharging stations, and other transformed arcs in a feasible way.

This section is organized as follows: in Section 4.2.1, we introduce a fragment-based representation of paths, which regards fragments as basic components. Then, each partial path is a concatenation of fragments, over which the minimum excess user ride time is determined. In Section 4.2.2, we explain how fragments are abstracted to arcs. This abstraction allows us to design a single REF that represents the extension from the start node to the end node of a fragment. In Section 4.2.3, we enumerate all feasible fragments, abstract fragments to arcs, and connect transformed arcs with each other, depots, and recharging stations in a feasible way to construct G_{sp}. We show in Theorem 3, G_{sp} preserves all feasible routes of the original one and therefore preserves all negative-reduced-cost routes of the original graph. Then, we define labels for nodes on G_{sp} and present their notations and the definitions of their associated resources in Section 4.2.4. The following part includes the label feasibility check, REFs, and dominance rules. The last part presents the extension of our CG algorithm to handle the case of heterogeneous vehicle capacities.

4.2.1 Representation of Partial Paths.

One important characteristic of the ESPPRC-MERT is that the reduced cost incorporates the total excess user ride time, which needs to be minimized along the extension. In the classical representation of a partial path, the path is extended in a node-by-node fashion. As mentioned, in the case of open requests existing on the partial path, we cannot calculate the excess user ride time for these open requests. Hence, for the label associated with this partial path, its reduced cost cannot be determined. In order to calculate the minimum excess user ride time in the extension of a partial path, we extend the partial path with battery-restricted fragments, as in Section 3.1, which generalizes the notion of fragments as proposed in Rist & Forbes (2021) by adding battery constraints in the feasibility check. Here, we recall the definition of battery-restricted fragments:

Definition 3 (Battery-restricted fragment). Assuming that $\mathcal{F} = (i_1, i_2, \cdots, i_k)$ is a sequence of pickup and drop-off nodes, where the vehicle arrives empty at i_1 and leaves empty at i_k and has passenger(s) on board at other nodes. Then, we call $\mathcal{F}$ a battery-restricted fragment if there exists a feasible route of the form:

$$(o, s_{i_1}, \cdots, s_{i_v}, \overbrace{i_1, i_2, \cdots, i_k}^{\mathcal{F}}, s_{i_{v+1}}, \cdots, s_{i_m}, f),$$

where $s_{i_1}, \cdots, s_{i_v}, s_{i_{v+1}}, \cdots, s_{i_m} (v, m \geqslant 0)$ are recharging stations, and $o \in O$, $f \in F$.

Each E-ADARP route can be regarded as the concatenation of an origin depot, battery-restricted fragments (hereinafter referred to as “fragments”), recharging stations (if required), and a destination depot. Clearly, we can

exactly minimize the excess user ride time on each fragment.

4.2.2 Abstracting Fragments to Arcs.

In this section, we show that each fragment $\mathcal{F}$ can be abstracted to an arc that captures all excess-user-ride-time optimal schedules over $\mathcal{F}$. To better illustrate our idea, we take an example as follows:

Example 3. Given a fragment $\mathcal{F} = \{1+, 2+, 1-, 2-\}$, the time window on each node and the travel time for each arc are shown in Figure 4.1. The dashed lines present the direct travel times from pickup nodes to the corresponding drop-off nodes. Assuming that the service time at each node is equal to zero and each request includes one passenger to be transported.

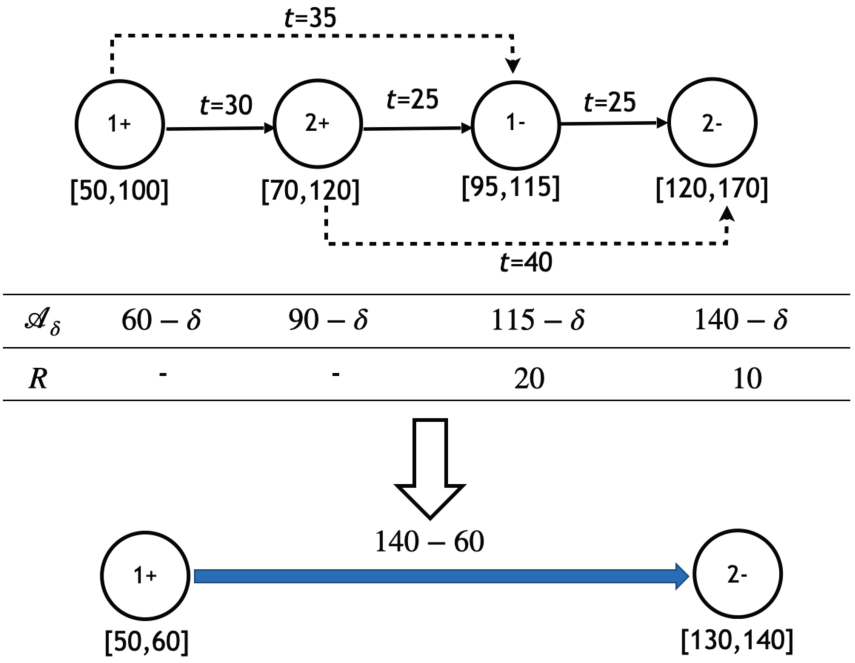


Figure 4.1 Example of abstracting a fragment to an arc

Clearly, any excess-user-ride-time optimal schedules on $\mathcal{F}$ can be represented as $\mathcal{A}_\delta$, where $0 \leqslant \delta \leqslant 10$. These schedules have an identical minimum excess user ride time of 30 minutes. Hence, we can abstract $\mathcal{F}$ to an arc from node 1+ and node 2- with time windows being [50,60] and [130,140]. This arc captures all excess-user-ride-time optimal schedules $\mathcal{A}_\delta$ on $\mathcal{F}$. The travel time from node 1+ to node 2- is presented on the converted arc.

Based on this observation, we present the general method to abstract fragments to arcs. Assuming that a fragment $\mathcal{F}$ is $\{1, 2, \cdots, m\}$, we denote any excess-user-ride-time optimal schedule $\mathcal{A}$ for $\mathcal{F}$ as a set of service start times, namely, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_m)$. Then, to abstract $\mathcal{F}$ to an arc, it is enough to determine all possible values of

$\mathcal{A}_1$ and $\mathcal{A}_m$ for any excess-user-ride-time optimal schedule $\mathcal{A}$ over $\mathcal{F}$. To calculate all possible values of $\mathcal{A}_1$ and $\mathcal{A}_m$, we introduce *vehicle-waiting-time optimal schedules*:

Definition 4. A vehicle-waiting-time optimal schedule $\mathcal{B}$ for a fragment $\mathcal{F}$ is defined as a set of service start times $\mathcal{B}_i$, $i \in \mathcal{F}$ that minimize the sum of vehicle waiting times at each node along $\mathcal{F}$ (i.e., $\sum_{i=2}^{m}[\mathcal{B}_i - (\mathcal{B}_{i-1} + t_{i-1,i} + s_{i-1})]$).

Note that a vehicle-waiting-time optimal schedule is not necessarily an excess-user-ride-time optimal schedule, as the latter one minimizes a weighted sum of waiting times along $\mathcal{F}$, which weight factors are equal to vehicle loads at nodes with waiting time. For a given fragment $\mathcal{F}$, we determine two vehicle-waiting-time optimal schedules:

1. the "latest" vehicle-waiting-time optimal schedule $\mathcal{B}^l$;
2. the "earliest" vehicle-waiting-time optimal schedule $\mathcal{B}^e$.

We show later in Theorem 2 that these schedules $\mathcal{B}^l, \mathcal{B}^e$ determine all possible values of $\mathcal{A}_1$ and $\mathcal{A}_m$ in any excess-user-ride-time optimal schedule $\mathcal{A}$ by the following means: For any excess-user-ride-time optimal schedule $\mathcal{A}$, there exists $\delta^l, \delta^e \geqslant 0$ such that:

$$\mathcal{A}_1 = \mathcal{B}_1^l - \delta^l,\ \mathcal{A}_m = \mathcal{B}_m^l - \delta^l;$$
$$\mathcal{A}_1 = \mathcal{B}_1^e + \delta^e,\ \mathcal{A}_m = \mathcal{B}_m^e + \delta^e;$$

Next, for a given fragment $\mathcal{F} = \{1, 2, \cdots, m\}$, we present the construction scheme for $\mathcal{B}^l$ and $\mathcal{B}^e$.

Construction of $\mathcal{B}^l$ and $\mathcal{B}^e$

The latest vehicle-waiting-time optimal schedule $\mathcal{B}^l$ must obey the following two rules:

1. Starting service as late as possible at the first node in $\mathcal{F}$;
2. Starting service as early as possible at all other nodes in $\mathcal{F}$;

For the first rule, as the vehicle arrives at the first node of $\mathcal{F}$ with no passenger, the delay of service start time at the first node will always help to eliminate unnecessary vehicle waiting time at succeeding nodes.

As for the second rule, when there is/are passenger(s) on board, it is straightforward to start service as early as possible as in this case to reduce vehicle waiting time. In the following part, we will first construct $\mathcal{B}^l$ and then construct $\mathcal{B}^e$.

- **Construct $\mathcal{B}^l$:** Assuming that a fragment $\mathcal{F} = \{1, 2, \cdots, m\}$. Let $\mathcal{B}_i^l$ be the service start time for schedule $\mathcal{B}^l$ at node i. Then the arrival time at each node i is $Arr_i = \mathcal{B}_{i-1}^l + t_{i-1,i} + s_{i-1}, 2 \leqslant i \leqslant m$. The waiting time Δ_i at node i is calculated as $\Delta_i = \mathcal{B}_i^l - Arr_i$. Based on the proposed rules, we define $\mathcal{B}_i^l$ inductively as follows:

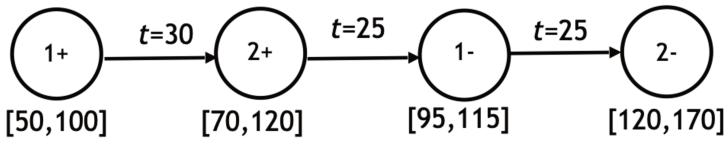
1. $\mathcal{B}_1^l = l_1$;
2. assuming $\mathcal{B}_i^l$ has been defined for $i < v$, we define $\mathcal{B}_v^l$ by:
   (a) if the extension from node $(v-1)$ respects the time window constraint at node v (i.e., $Arr_v \leq l_v$), then we define $\mathcal{B}_v^l = \max\{e_v, Arr_v\}$;
   (b) Otherwise,
   - if $\min_{i<v}\{\mathcal{B}_i^l - e_i\} \geq Arr_v - l_v$, we can update the schedule at nodes $1, \cdots, v-1$ by moving forward $Arr_v - l_v$. Then the extension from node $(v-1)$ to node v will not violate the time window constraint. I.e., we update $\mathcal{B}_i^l$ as $\mathcal{B}_i^l - (Arr_v - l_v)$ for $i = 1, \cdots, v-1$ and define $\mathcal{B}_v^l = l_v$;
   - Otherwise, there is no feasible schedule for $\{1, 2, \cdots, v\}$.

For the last point, the maximum amount of time that can be moved forward from node 1 to node $(v-1)$ is $\min_{i<v}\{\mathcal{B}_i^l - e_i\}$. To satisfy the time window constraint at node v, the minimum amount of time that is required to be moved forward is $Arr_v - l_v$. In the case of $\min_{i<v}\{\mathcal{B}_i^l - e_i\} < Arr_v - l_v$, the time window constraint at node v is never fulfilled.

- **Construct $\mathcal{B}^e$:** After determining the latest vehicle-waiting-time optimal schedule $\mathcal{B}^l$ on $\mathcal{F}$, the earliest vehicle-waiting-time optimal schedule $\mathcal{B}^e$ is determined by moving forward $\mathcal{B}^l$ on $\mathcal{F}$ by the maximum amount of time (denoted as $\overleftarrow{\Delta}$) that will not change the minimum vehicle waiting time. $\overleftarrow{\Delta}$ can be calculated by taking the minimum value among all the $\mathcal{B}_i^l - e_i$, for $i \in \mathcal{F}$. That is: $\overleftarrow{\Delta} = \min_{i\in\mathcal{F}}\{\mathcal{B}_i^l - e_i\}$, and $\mathcal{B}_i^e = \mathcal{B}_i^l - \overleftarrow{\Delta}$, $i \in \mathcal{F}$.

We recall the previous example to present the construction of $\mathcal{B}^l$ and $\mathcal{B}^e$:

Example 4 (Example 3 continued)**.** For the given fragment $\mathcal{F}$, $\mathcal{B}^l$ and $\mathcal{B}^e$ for fragment $\mathcal{F}$ are directly shown in Figure 4.2. In this example, $\overleftarrow{\Delta} = 10$ and $\mathcal{B}_i^e = \mathcal{B}_i^l - 10$, $i \in \mathcal{F}$.



$\mathcal{B}^l$	60	90	115	140
$\mathcal{A}_\delta$	$60-\delta$	$90-\delta$	$115-\delta$	$140-\delta$
$\mathcal{B}^e$	50	80	105	130

Figure 4.2 Example of any excess-user-ride-time optimal schedule on a fragment

Now, we prove that $\mathcal{B}^l, \mathcal{B}^e$ can determine all possible values of $\mathcal{A}_1$ and $\mathcal{A}_m$ of any excess-user-ride-time optimal schedule $\mathcal{A}$.

Theorem 2. Assuming that fragment $\mathcal{F} = \{1, 2, \cdots, m\}$, $\mathcal{A}$ is an **excess-user-ride-time optimal** schedule, $\mathcal{B}^l$ is the constructed latest vehicle-waiting-time optimal schedule over $\mathcal{F}$, and $\mathcal{B}^e$ is the earliest vehicle-waiting-time optimal schedule. Then there exists $\delta^e, \delta^l \geq 0$ such that:

$$\mathcal{A}_1 = \mathcal{B}_1^e + \delta^e,\ \mathcal{A}_m = \mathcal{B}_m^e + \delta^e;$$
$$\mathcal{A}_1 = \mathcal{B}_1^l - \delta^l,\ \mathcal{A}_m = \mathcal{B}_m^l - \delta^l.$$

Proof of Theorem 2. In the case of $\Delta_u(\mathcal{B}^l) = 0$ for all $1 \leq u \leq m$ ($\Delta_i(\cdot)$ is the waiting time at node i according to a given schedule), then the theorem clearly holds. Next, we assume that there exists $1 \leq v \leq m$ such that $\Delta_v(\mathcal{B}^l) \neq 0$. We will show that $\mathcal{A}_1 = \mathcal{B}_1^l, \mathcal{A}_m = \mathcal{B}_m^l$.

The proof contains two parts.

- $\mathcal{A}_1 = \mathcal{B}_1^l$. According to the first construction rule of $\mathcal{B}^l$, we have $\mathcal{B}_1^l \geq \mathcal{A}_1$. Next, we prove $\mathcal{B}_1^l \leq \mathcal{A}_1$ by contradiction.

  Assuming that $\mathcal{B}_1^l > \mathcal{A}_1$, then if $\Delta_2(\mathcal{A}) = 0$, we must have $\mathcal{B}_2^l > \mathcal{A}_2 \geq e_2$, therefore $\Delta_2(\mathcal{B}^l) = 0$ by our construction. Moreover, if $\Delta_3(\mathcal{A}) = 0$, we must have $\mathcal{B}_3^l > \mathcal{A}_3 \geq e_3$, therefore $\Delta_3(\mathcal{B}^l) = 0$.

  Repeat the above process, since $\Delta_v(\mathcal{B}^l) \neq 0$, there exists $v_0 \leq v$ such that: $\mathcal{B}_i^l > \mathcal{A}_i, \Delta_i(\mathcal{A}) = 0$ for $1 \leq i < v_0$, $\Delta_{v_0}(\mathcal{A}) > 0$. Then the total excess user ride time of $\mathcal{A}$ can be further reduced by **delaying** the service start time by:

  $$min\{min_{1 \leq i < v_0}\{\mathcal{B}_i^l - \mathcal{A}_i\}, \Delta_{v_0}(\mathcal{A})\}$$

  in $1, \cdots, v_0 - 1$. $\mathcal{A}$ is not an optimal plan, which is a contradiction!

- $\mathcal{B}_m^l = \mathcal{A}_m$. Since there exists $1 \leq v \leq m$ such that $\Delta_v(\mathcal{B}^l) \neq 0$, we have $\mathcal{B}_v^l = e_v$ at node v. According to the second construction rules of $\mathcal{B}^l$, we also have $\mathcal{B}_m^l \leq \mathcal{A}_m$. Next, we prove $\mathcal{B}_m^l \geq \mathcal{A}_m$ by contradiction.

  Assuming that $\mathcal{B}_m^l < \mathcal{A}_m$, if $\Delta_m(\mathcal{A}) = 0$, we must have $\mathcal{B}_{m-1}^l < \mathcal{A}_{m-1}$. Moreover, if $\Delta_{m-1}(\mathcal{A}) = 0$, we must have $\mathcal{B}_{m-2}^l < \mathcal{A}_{m-2}$. Since $\mathcal{B}_1^l = \mathcal{A}_1$ as we proved above, there must exists $2 \leq v_1 \leq m$ such that: $\mathcal{B}_i^l < \mathcal{A}_i, \Delta_i(\mathcal{A}) = 0$ for $v_1 < i \leq m$ and $e_{v_1} \leq \mathcal{B}_{v_1}^l < \mathcal{A}_{v_1}, \Delta_{v_1}(\mathcal{A}) > 0$. Then the excess user ride time of $\mathcal{A}$ can be further reduced by **moving forward** the service start time by:

  $$min\{min_{v_1 \leq i \leq m}\{\mathcal{A}_i - \mathcal{B}_i^l\}, \Delta_{v_1}(\mathcal{A})\}$$

  in $v_1, \cdots, m$. $\mathcal{A}$ is not an optimal plan, which is a contradiction!

□

Theorem 2 also implies that as soon as the excess-user-ride-time optimal schedule for fragment $\mathcal{F}$ contains waiting time, we have $\mathcal{B}^l = \mathcal{B}^e$ and $\delta^l = \delta^e = 0$. In this case, any excess-user-ride-time optimal schedule (denoted as $\mathcal{A}$) must satisfy $\mathcal{A}_1 = \mathcal{B}_1^l = \mathcal{B}_1^e$ and $\mathcal{A}_m = \mathcal{B}_m^l = \mathcal{B}_m^e$, where node 1 and node m are the first and the last node of $\mathcal{F}$. In the other case, an excess-user-ride-time optimal schedule can be obtained by moving forward(backward) the vehicle-waiting-time optimal schedule $\mathcal{B}^l$ ($\mathcal{B}^e$) by δ, such that $0 \leqslant \delta \leqslant \mathcal{B}_1^l - \mathcal{B}_1^e$.

Abstracting a Fragment to an Arc.

For each fragment $\mathcal{F} = \{1, \cdots, m\}$, assuming $\mathcal{B}^e, \mathcal{B}^l$ are the corresponding earliest and latest vehicle-waiting-time optimal schedules. Based on Theorem 2, restricting time windows at node 1 and node m to $[\mathcal{B}_1^e, \mathcal{B}_1^l]$ and $[\mathcal{B}_m^e, \mathcal{B}_m^l]$ will include all excess-user-ride-time optimal schedules on fragment $\mathcal{F} = \{1, \cdots, m\}$. Then we can abstract $\mathcal{F}$ to an arc $(1, m)$ such that:

1. the total travel time from 1 to m (denoted as $t'_{1,m}$) is $\mathcal{B}_m^l - \mathcal{B}_1^l$;
2. the original time windows of node 1 and node m are restricted to $[\mathcal{B}_1^e, \mathcal{B}_1^l]$ and $[\mathcal{B}_m^e, \mathcal{B}_m^l]$;
3. the battery consumption from 1 to m is $\sum_{i=1}^{m-1} h_{i,i+1}$;

If no waiting time is generated on $\mathcal{F}$, we can calculate the minimum excess user ride time directly. It is also straightforward to compute the minimum excess user ride time for $\mathcal{F}$ that contains one request with waiting time generated. When waiting time is generated on a fragment containing more than two requests, calculating the value of minimum excess user ride time becomes difficult. In this case, we improve the LP model introduced in Section 3.1.2 by setting the service start times at node 1 and node m to $\mathcal{B}_1^l$ and $\mathcal{B}_m^l$, respectively. Then, we solve the second version of LP (LP2) to determine the minimum excess user ride time, as presented in the following.

Let $P_{\mathcal{F}}$ denote all pickup nodes on $\mathcal{F} = \{1, \cdots, m\}$:

$$\min \sum_{i \in P_{\mathcal{F}}} R_i \tag{4.10}$$

s.t.

$$\begin{cases} T_i = \mathcal{B}_1^l, & \text{if } i = 1 \\ T_i = \mathcal{B}_m^l, & \text{if } i = m \\ e_i \leqslant T_i \leqslant l_i, & \text{Otherwise} \end{cases} \tag{4.11}$$

$$T_i + s_i + t_{i,j} \leqslant T_j, \quad \forall i \in \mathcal{F} \setminus \{m\}, \quad idx_j = idx_i + 1 \tag{4.12}$$

$$T_{n+i} - (T_i + s_i) \leqslant m_i, \quad \forall i \in P_{\mathcal{F}} \tag{4.13}$$

$$T_{n+i} - T_i - s_i - t_{i,n+i} \leqslant R_i, \quad \forall i \in P_{\mathcal{F}} \tag{4.14}$$

$$R_i \geqslant 0, \quad \forall i \in P_{\mathcal{F}} \tag{4.15}$$

where idx_i is the index of node i on the segment. The objective function is to minimize the total excess user ride time. Constraints (4.11) to (4.12) are time window constraints where we set the service start time at node 1 and node m to $\mathcal{B}_1^l$ and $\mathcal{B}_1^m$ ($\mathcal{B}^l$ is the latest vehicle-waiting-time optimal schedule), respectively. Constraints (4.13) and constraints (4.14) are user ride time constraints.

4.2.3 Constructing a New Sparser Graph.

In this section, we construct a new sparser graph G_{sp} by two steps: (1) we enumerate all feasible fragments and abstract them to arcs; (2) we connect each transformed arc with depots, recharging stations, and other transformed arcs in a feasible way.

The fragment enumeration is conducted with depth-first search as in Alyasiry et al. (2019). For each feasible fragment, the corresponding restricted time windows and minimum excess user ride times are recorded. To generate all feasible fragments, we assume that the vehicle departs from each pickup node with a full battery level and must respect constraints of maximum user ride time, battery capacity, time window, pairing, precedence, and vehicle capacity. We start from each pickup node and extend it node by node until no more feasible fragment that starts from this pickup node can be generated. By enumerating all feasible fragments before computation, we largely accelerate the labeling algorithm as we only need to query information instead of recalculating. To provide more details, we refer to Table B.1 in Section 3.2.4. For all the instances, the fragment enumeration can be fulfilled in a matter of seconds. In the computational experiments, we report the CPU time, which includes the computational time for fragment enumeration. With the information of all feasible fragments, we abstract fragments to arcs as presented in Section 4.2.2.

Then, we construct G_{sp} by connecting depots, recharging stations, and the start nodes and end nodes of transformed arcs in a feasible way. Details for the connection between nodes are as follows:

1. Each origin depot connects with all start nodes of arcs, recharging stations, and destination depots;
2. Each recharging station connects with start nodes of arcs and destination depots in a feasible way;
3. Each end node of an arc connects with destination depots, recharging stations, and all the start nodes of arcs in a feasible way;

Figure 4.3 shows an example of constructing new arcs in G_{sp}. It should be noted that for two different fragments, even though they have the same start node $i+$ and end node $j-$, we need to treat them as two different arcs in G_{sp} as they represent different fragments consisting of different sequences of nodes, which lead to different restricted

time windows. To distinguish, we make copies (e.g., $i'+$, $j'+$) for node $i+$ and node $j-$ and we generate two arcs $(i+, j-)$ and $(i'+, j'-)$.

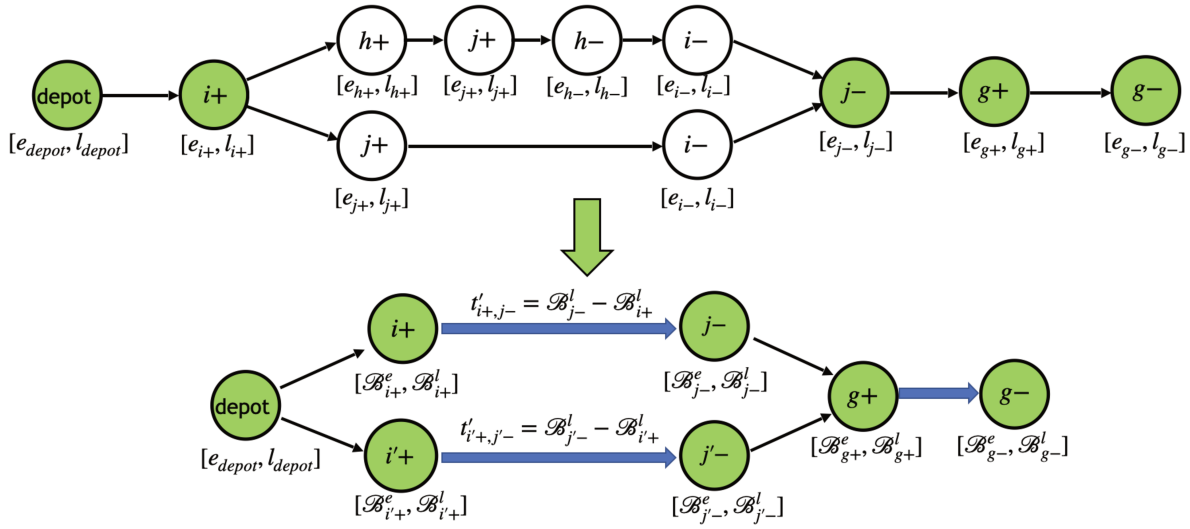


Figure 4.3 Example of constructing arcs in the new graph G_{sp}

Note that our way of constructing the network is completely different from Alyasiry et al. (2019), who build a network by discretizing the time windows of all pickup and delivery nodes with a fixed length of time η. However, we make copies $(i+', j+')$ to distinguish different fragments with the same start/end nodes as they have different restricted time windows.

Next, we work on the new sparser graph G_{sp} instead of G, as we show in Theorem 3 that G_{sp} preserves all feasible routes over G.

Theorem 3. Let $\mathcal{R}$ be a route over graph G, and $\mathcal{R}_{sp}$ be the corresponding route over G_{sp}. Then $\mathcal{R}$ is feasible if and only if $\mathcal{R}_{sp}$ is feasible.

Proof of Theorem 3. Clearly, if $\mathcal{R}_{sp}$ is feasible then $\mathcal{R}$ is feasible. Therefore, we only need to prove the other direction.

Now, we assume that $\mathcal{R}$ is feasible and $\mathcal{A}$ is a **feasible** schedule over $\mathcal{R}$. It is enough to show that there is also a **feasible** schedule over $\mathcal{R}_{sp}$. This is implied by the following lemma:

Lemma 1. For any fragment $\mathcal{F} = \{1, 2, \cdots, m\}$ in $\mathcal{R}$. Let $\mathcal{B}^e, \mathcal{B}^l$ be the constructed vehicle-waiting-time optimal schedule over $\mathcal{F}$. Then there exists $0 \leq \delta \leq \mathcal{B}^l_1 - \mathcal{B}^e_1$ such that $\mathcal{A}_1 \leq \mathcal{B}^l_1 - \delta, \mathcal{B}^l_m - \delta \leq \mathcal{A}_m$.

Then we can obtain a feasible schedule over $\mathcal{R}_{sp}$ by replacing the schedule of every fragment $\mathcal{F} = \{1, 2, \cdots, m\}$ to the schedule $\{\mathcal{B}^l_1 - \delta, \mathcal{B}^l_m - \delta\}$ over the arc generated from $\mathcal{F}$.

□

Proof of Lemma 1. According to the construction rules of $\mathcal{B}^l$, we always have $\mathcal{A}_1 \leq \mathcal{B}^l_1$. Let $\mathcal{B}^\delta$ be the vehicle-waiting-time optimal schedule obtained by moving forward the service begin time by δ from $\mathcal{B}^l$ and $\Delta_v(\mathcal{B}^l)$ denote the waiting

time at node v according to $\mathcal{B}^l$.

In the proof, we show that we can find a δ_0 satisfying $0 \leq \delta_0 \leq \mathcal{B}_1^l - \mathcal{B}_1^e$ such that $\mathcal{A}_1 \leq \mathcal{B}_1^{\delta_0}, \mathcal{B}_m^{\delta_0} \leq \mathcal{A}_m$. There are two cases:

1. If $\Delta_v(\mathcal{B}^l) \neq 0$ for some $1 \leq v \leq m$, then it is enough to take $\delta_0 = 0$ as shown in Theorem 2.

2. If $\Delta_v(\mathcal{B}^l) = 0$ for all nodes on $\mathcal{F}$, let δ_0 be the **maximal** value that satisfies (i) $\mathcal{B}^{\delta_0}$ is feasible and (ii) $\mathcal{A}_1 \leq \mathcal{B}_1^{\delta_0}$. There are two cases:

   (a) If $\mathcal{A}_1 \geqslant \mathcal{B}_1^e$, then we have $\mathcal{A}_1 = \mathcal{B}_1^{\delta_0}$. Since $\Delta_v(\mathcal{B}^l) = 0$ for all nodes, we derive $\mathcal{B}_m^{\delta_0} \leq \mathcal{A}_m$.

   (b) If $\mathcal{A}_1 < \mathcal{B}_1^e$, then we have $\mathcal{A}_1 < \mathcal{B}_1^{\delta_0}$ and $\mathcal{B}^{\delta_0} = \mathcal{B}^e$. By definition of $\mathcal{B}^e$, there must exist a node $u \in \mathcal{F}$ such that $\mathcal{B}_u^e = e_u$. Therefore, we derive $\mathcal{B}_m^{\delta_0} \leq \mathcal{A}_m$ as we have $\Delta_u(\mathcal{B}^l) = 0$ for all nodes.

   Summing up these cases, we can always find δ_0 such that $\mathcal{A}_1 \leq \mathcal{B}_1^{\delta_0}, \mathcal{B}_m^{\delta_0} \leq \mathcal{A}_m$.

□

Then, we design our labeling algorithm on the newly constructed graph.

4.2.4 Labeling Algorithm.

We design a labeling algorithm on the new sparser graph G_{sp}, where excess-user-ride-time optimality is ensured on each arc. The proposed labeling algorithm extends the label at the end of the partial path $\mathcal{P}$. We denote L_i as the label associated with a partial path ends with node i. The forward labeling algorithm extends labels from a source node $o_k \in O$ to a non-predefined sink node $f \in F$. Let a label associated with a partial path $\mathcal{P}$ from o_k to current vertex i be $L_i = \{R_i^{cost}, (R_i^{rch_s})_{s \in S}, R_i^{tMin}, R_i^{tMax}, R_i^{rtMax}, R_i^{req}\}$, the definition of each resource is described as follows:

1. R_i^{cost}: The reduced cost of the partial route until i;

2. $R_i^{rch_s}$: The number of times recharging station $s \in S$ is visited along partial path $\mathcal{P}$;

3. R_i^{tMin}: The earliest service start time at vertex i that considers a minimum recharging time (ensuring the battery feasibility up to vertex i) at the recharging station if a recharging station is visited along the partial path before reaching i;

4. R_i^{tMax}: The earliest service start time at vertex i that considers a maximum recharge time (ensuring the time-window feasibility up to vertex i) at the recharging station if a recharging station is visited along the partial path before reaching i;

5. R_i^{rtMax}: The maximum recharging time required to fully recharge at vertex i. In the case that a recharging station is visited prior to i along $\mathcal{P}$, the vehicle performs a minimum recharge that ensures the battery feasibility up to vertex i;

6. R_i^{req}: The set of unreachable requests until i along partial path $\mathcal{P}$. A request is said to be "unreachable" if time window constraints are violated or this request has been visited.

In case no recharging station is visited on partial path $\mathcal{P}$, the value of R_i^{tMin} is equal to R_i^{tMax}, indicating the earliest service start time at vertex i. R_i^{rtMax} represents the accumulated amount of needed recharging time until i. In the initial label at vertex o_k, $R_{o_k}^{req}$ is an empty set, $R_{o_k}^{tMin}$ and $R_{o_k}^{tMax}$ are equal to e_{o_k}, while all other components are set to zero. We extend a label $L_i = \{R_i^{cost}, (R_j^{rch_s})_{s\in S}, R_i^{tMin}, R_i^{tMax}, R_i^{rtMax}, R_i^{req}\}$ along arc $(i,j) \in A'$ using the following REFs:

$$R_j^{cost} = R_i^{cost} + \bar{c}_{i,j} \tag{4.16}$$

$$R_j^{rch_s} = R_i^{rch_s} + \begin{cases} 1, & \text{if } j = s \\ 0, & \text{otherwise} \end{cases} \tag{4.17}$$

$$R_j^{tMin} = \begin{cases} \max\{\mathcal{B}_j^e, R_i^{tMin} + t'_{i,j}\}, & \text{if } R_i^{rch} = \emptyset \\ \max\{\mathcal{B}_j^e, R_i^{tMin} + t'_{i,j}\} + Z_{i,j}, & \text{otherwise} \end{cases} \tag{4.18}$$

$$R_j^{tMax} = \begin{cases} \min\{\mathcal{B}_j^l, \max\{\mathcal{B}_j^e, R_i^{tMin} + R_i^{rtMax} + t'_{i,j}\}\}, & \text{if } i \in S \\ \min\{\mathcal{B}_j^l, \max\{\mathcal{B}_j^e, R_i^{tMax} + t'_{i,j}\}\}, & \text{otherwise} \end{cases} \tag{4.19}$$

$$R_j^{rtMax} = \begin{cases} R_i^{rtMax} + h'_{i,j}, & \text{if } R_i^{rch} = \emptyset \\ \min\{H, \max\{0, R_i^{rtMax} - S_{i,j}\} + h'_{i,j}\}, & \text{otherwise} \end{cases} \tag{4.20}$$

$$R_j^{req} = R_i^{req} \cup U_n(R_j^{tMin}) \tag{4.21}$$

where in these functions:

$$S_{i,j}(R_i^{tMin}, R_i^{tMax}, R_i^{rtMax}) = \begin{cases} \max\{0, \min\{\mathcal{B}_j^e - R_i^{tMin} - t'_{i,j}, R_i^{rtMax}\}\}, & \text{if } i \in S \\ \max\{0, \min\{\mathcal{B}_j^e - R_i^{tMin} - t'_{i,j}, R_i^{tMax} - R_i^{tMin}\}\}, & \text{otherwise} \end{cases} \tag{4.22}$$

$$Z_{i,j}(R_i^{tMin}, R_i^{tMax}, R_i^{rtMax}) = \max\{0, \max\{0, R_i^{rtMax} - S_{i,j}(R_i^{tMin}, R_i^{tMax}, R_i^{rtMax})\} + h'_{i,j} - H\} \tag{4.23}$$

The $S_{i,j}$ is the slack time between the earliest vehicle-waiting-time optimal service start time $\mathcal{B}_j^e$ at j and the earliest arrival time to j. If i is a recharging station, $S_{i,j}$ may be equal to the maximum possible recharging time R_i^{rtMax} at vertex i, while at other nodes, it may be equal to $R_i^{tMax} - R_i^{tMin}$. $Z_{i,j}$ is the minimum recharging time accounting for the available slack that the previous recharging station must perform to maintain battery feasibility. $U_n(R_j^{tMin})$ is the function to determine the unreachable nodes from j.

An extension feasibility check is performed while extending label L_i to label L_j via arc $(i,j) \in A'$. The feasibility check rules are presented in the following proposition:

Proposition 2. The extension of label L_i to label L_j is feasible if and only of label L_j satisfies:

$$R_j^{tMin} \leqslant \mathcal{B}_j^l, \quad R_j^{tMin} \leqslant R_j^{tMax}, \quad R_j^{req_p} \leqslant 1, \forall p \in P, \quad R_j^{rtMax} \leqslant \begin{cases} (1-\gamma)H, & j \in F \\ H, & \text{otherwise} \end{cases}$$

where $R_j^{req_p}$ is the number of times request p is visited along the partial path.

If j is a recharging station, then constraint (4.24) must be considered:

$$R_j^{rch_j} \leqslant 1 \tag{4.24}$$

In case of a feasibility violation, the corresponding label will be discarded. Also, it should be mentioned that each time when a fragment is added, the visited customers on this fragment need to be checked. If the fragment contains a visited customer, it should be discarded. The maximum user ride time constraints and capacity constraints are checked when generating fragments.

Definition 5. Suppose that $L^k = \{R_k^{cost}, (R_k^{rch_s})_{s\in S}, R_k^{tMin}, R_k^{tMax}, R_k^{rtMax}, R_k^{req}\}, k \in 1,2$ are two labels and the partial path associated to L^1 and L^2 are $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Assuming that $\mathcal{P}_1$, $\mathcal{P}_2$ end at the same node, L^1 dominates L^2 if and only if:

$$R_1^r \leqslant R_2^r, \forall r \in \{cost, rch, tMin\} \tag{4.25}$$

$$R_1^{req} \subseteq R_2^{req} \tag{4.26}$$

$$R_1^{rtMax} - (R_1^{tMax} - R_1^{tMin}) \leqslant R_2^{rtMax} - (R_2^{tMax} - R_2^{tMin}) \tag{4.27}$$

$$R_1^{rtMax} - (R_2^{tMax} - R_1^{tMin}) \leqslant R_2^{rtMax} \tag{4.28}$$

The last two conditions are equivalent to the requirement that: for every service start time $T_2 \in [R_2^{tMin}, R_2^{tMax}]$, there exists a service start time $T_1 \in [R_1^{tMin}, T_2]$ such that $R_1^{rtMax} - (T_1 - R_1^{tMin}) \leqslant R_2^{rtMax} - (T_2 - R_2^{tMin})$. In other words, we can always find a service start time $T_1 \leqslant T_2$ that does not consume more energy.

4.2.5 Consideration of Heterogeneous Vehicle Capacities

Our CG algorithm can be extended to allow heterogeneous vehicle capacities. As in Parragh et al. (2012), we define $\mathcal{T}$ as the set of available vehicle types and classify vehicles with the same capacity into one vehicle type $t \in \mathcal{T}$. Let Ω^t denote the set of feasible routes for vehicles of type $t \in \mathcal{T}$ and $\Omega = \bigcup_{t \in \mathcal{T}} \Omega^t$ is the set of all feasible routes, we only need to reformulate constraint (4.5) in the MP formulation as follows:

$$\sum_{\omega \in \Omega} y_\omega \leqslant M_t, \quad \forall t \in \mathcal{T} \tag{4.29}$$

where M_t is the maximum number of vehicles of type t that can be included in the solution. Then, we obtain $|\mathcal{T}|$ subproblems in the CG, one for each vehicle type. The reduced cost $\bar{c_\omega}$ for a route ω generated for the subproblem t is formulated as:

$$c_\omega - \sum_{i \in P} \theta_{i\omega} \lambda_i - \sum_{f \in S \cup F} \phi_{f\omega} \tau_f - \sum_{o \in O} \epsilon_{o\omega} \zeta_o - \kappa_t \tag{4.30}$$

where κ_t is the dual variable value associated with constraints (4.29) for vehicles of type t. Then, we solve each subproblem $t \in \mathcal{T}$ to generate feasible routes of negative reduced costs.

4.3 Cutting Planes and Branching Strategies

To strengthen the continuous MP formulation, we apply two types of cutting planes for instances that are not solved optimally by CG. The first type of cutting plane is the two-path cut, which was initially proposed by Kohl et al. (1999) for solving VRPTW and is defined as follows. For a subset $W \subseteq N \cup S$, we define the sets of predecessors of W as $\pi(W) = \{i \in P : i + n \in W, i \notin W\}$, the sets of successors of W as $\sigma(W) = \{i + n \in D : i \in W, i + n \notin W\}$, and the flow enter subset $W \subseteq N \cup S$ is:

$$x(W) = \sum_{i \notin W} \sum_{j \in W} x_{i,j} \tag{4.31}$$

where $x_{i,j} = \sum_{k \in K} x_{i,j}^k$ is calculated with the current solution of continuous MP.

We aim at finding the subset W such that $x(W) < 2$ and $k(W) > 1$, where $k(W)$ is the smallest number of

vehicles needed to serve all nodes in W. The corresponding two-path inequality for such a subset W is:

$$\sum_{\omega \in \Omega} n_{\omega}^{W} y_{\omega} \geqslant 2 \tag{4.32}$$

where n_{ω}^{W} is the number of times column ω enters W, i.e., the number of arcs $(i,j) \in \omega$ such that $i \notin W$ and $j \in W$.

To separate two-path inequalities, we adapt the greedy heuristic proposed in Kohl et al. (1999). After identifying set W satisfying $x(W) < 2$, we determine whether there exists an elementary path that serves all nodes in $\pi(W) \cup W \cup \delta(W)$ in a feasible way. If no such path can be found, then the subset W defines a valid two-path inequality (4.32), which is added to the continuous MP formulation. The dual variable associated with inequality (4.32) must be subtracted from $\bar{c}_{\omega}$ for all arcs (i,j) with $i \notin W$ and $j \in W$.

The second type of cutting planes are subset row inequalities which were first introduced by Jepsen et al. (2008) to solve the VRPTW. For a subset $W \subseteq N \cup S$ of three elements, the corresponding subset row inequality is:

$$\sum_{\omega \in \Omega'} m_{\omega}^{W} y_{\omega} \leqslant 1 \tag{4.33}$$

where $m_{\omega}^{W} = \lfloor \beta_{\omega}^{W} / 2 \rfloor$ and β_{ω}^{W} is the number of visits to a customer in W along route ω. For an elementary route ω, we have $m_{\omega}^{W} = 1$ if ω visits two or three customers in W and $m_{\omega}^{W} = 0$ otherwise.

Subset row inequalities are separated if we find no two-path cut from the current solution of continuous MP. We separate subset row inequalities by enumerating all subsets of three customers and checking for each subset if the corresponding inequality is violated. The valid inequalities are added to the continuous MP formulation. Note that the dual variables associated with the cuts cannot be integrated into the reduced cost of the labels. Instead, one additional resource attribute is created to record the number of visits to each subset of customers. Also, the dominance rules are modified, as in Jepsen et al. (2008); Desaulniers et al. (2011). Each time, we identify such cuts and select at most n_{max}^{SRC} cuts according to the rules proposed in Desaulniers et al. (2008). In our case, $n_{max}^{SRC} = 10$.

4.3.1 Branching Strategies

In case of the CG algorithm does not obtain integer solutions, we impose branching strategies on fractional solutions to derive integer solutions. We consider two branching strategies in the branch-and-bound search tree: the first branching strategy branches on the total number of routes (as in Desrochers et al. (1992)), and the second branching strategy branches on the total flow of an arc (as in Desaulniers et al. (2016)). For a fractional solution, we evaluate the branching strategies with the mentioned order and select the first type that can be imposed. When imposing the selected branching strategy, the first type of branching strategy is imposed by adding inequality to the continuous MP. As for the second type of branching strategy, we remove columns that contain incompatible arcs from the column pool and prevent the generation of columns that include these arcs in the labeling algorithm. If we obtain fractional

arc flow for several arcs, the arc with a fractional flow that is closest to 0.5 is selected. Each time when a branching strategy is imposed, we create two branches and explore the branch-and-bound tree in a depth-first fashion.

4.4 Computational Experiments

In this section, we first describe the benchmark instance sets used to examine the performance of the proposed CG and B&P algorithms. Thereafter, we first present the results obtained by our CG algorithm under different minimum battery level restrictions (i.e., $\gamma = 0.1, 0.4, 0.7$). We compare the root node results (i.e., Lagrangian dual bounds and CG primal bounds solving integer RMP on the previously generated columns) to the best-reported B&C results of Bongiovanni et al. (2019). As for large-scale instances, no exact results are reported in Bongiovanni et al. (2019), we compare the CG results with the best heuristic results of Section 3.3. Next, we integrate the CG algorithm into the B&P scheme and report the B&P algorithm results on type-a, -u, and -r instances. The B&P results are compared to the best-reported B&C results of Bongiovanni et al. (2019). Then, we extend the E-ADARP to allow unlimited visits to each recharging station and compare our results of the B&P algorithm to the best-reported results of at-most-one visit to each recharging station to analyze the effect of relaxing the restriction of recharging visits. Finally, we compare the CG and B&P results with DA algorithm results presented in Chapter 3 in terms of solution quality and computational time. Practical insights are drawn for decision-makers to select methods according to their preferences.

All algorithms are coded in Julia 1.7.2 and are performed on a standard PC with an Intel(R) Core(TM) i7-8700 CPU at 3.20 GHz and with 32 Gb of RAM using a single thread only. Other packages used in the program are JuMP 1.0.0 and Gurobi, 0.11.1. It should be noted that the results reported in Bongiovanni et al. (2019) are obtained from a computer with an Intel(R) Core(TM) i7-4790 CPU at 3.60 GHz and with 16 Gb of RAM.

4.4.1 Benchmark Instances

The benchmark instance sets that are considered in Chapter 3 are used in the computational experiments of this chapter. Two are existing benchmark instance sets from Bongiovanni et al. (2019). The third set corresponds to the large-scale E-ADARP instances introduced in Su et al. (2023). All the instances are labeled in the form xk-n, where $x \in \{a, u, r\}$, k is the number of vehicles and n is the number of requests. Here we recall the characteristics of the considered benchmark instance sets are as follows:

- "a" denotes the E-ADARP instances proposed by Bongiovanni et al. (2019) that are extended from the standard DARP benchmark instance set of Cordeau (2006) by supplementing features of electric vehicles and recharging stations. These instances are called type-a instances. For these instances, the number of vehicles is in the range $2 \leqslant k \leqslant 5$, and the number of requests is in the range $16 \leqslant n \leqslant 50$.
- "u" denotes the E-ADARP instances generated by Bongiovanni et al. (2019) that are based on the ride-sharing

data from Uber Technologies. These instances are called type-u instances. For these instances, the number of vehicles is in the range $2 \leqslant k \leqslant 5$ and the number of requests is in the range $16 \leqslant n \leqslant 50$, as in the type-a instances.

- "r" denotes the larger E-ADARP instances adopted from Su et al. (2023). These instances are based on the large-scale instances of Ropke et al. (2007) for the standard DARP and are adapted to the E-ADARP following the same logic as the type-a instances. These instances are called type-r instances. For these instances, the number of vehicles is in the range $5 \leqslant k \leqslant 8$ and the number of requests is in the range $60 \leqslant n \leqslant 96$.

Following Bongiovanni et al. (2019), we set $w_1 = 0.75$, $w_2 = 0.25$. We consider three different γ values, namely, $\gamma \in \{0.1, 0.4, 0.7\}$, representing the low-,medium-, and high-battery-level restriction case, respectively. Higher values of γ result in more tightly constrained instances of the E-ADARP, allowing us to analyze the algorithm's performance from the loosely- to the highly-constrained case.

4.4.2 Computational Results

We first solve the continuous RMP with the proposed CG algorithm. To obtain integer solutions, we use the generated set of columns to solve the integer RMP and report the obtained results. The column pool is initialized by iterating the DA algorithm (see Chapter 3) 500 times and storing all the generated columns. The time limit for solving continuous RMP is set to two hours (the same as in Bongiovanni et al. (2019)). To close the integrality gaps in the case of existing, we tried the following ideas: (1) separating two-path cuts and subset row inequalities and adding them to continuous MP formulation so as to strengthen lower bounds, and (2) implementing the B&P algorithm. Table 4.1, Table 4.2, and Table 4.3 present the obtained CG results under different γ values (i.e., $\gamma = 0.1, 0.4, 0.7$) for type-a, -u, and -r instances, respectively. In each table, the lower bound obtained from solving continuous RMP, the upper bound from solving the integer RMP, and the computation time in seconds is shown. If our CG algorithm obtains fractional solutions at the root node, we first try to close integrality gaps by adding cuts to the continuous MP. Then, the generated lower bounds, upper bounds, and total computational time are reported. We also implement the B&P algorithm to close integrality gaps if the CG does not obtain integer solutions at the root node. We summarize the computational results of the B&P algorithm on type-a, -u, and -r instances in Table 4.4, 4.5, 4.6. The obtained B&P results are compared to the results of the CG algorithm with adding cuts.

The benchmark results from Bongiovanni et al. (2019) are given in the last three columns. The meaning of the abbreviations in the tables is as follows:

1. Obj and Obj_1: The objective values of integer RMP solutions obtained by the CG algorithm without and with cutting planes being added, respectively;
2. LB and LB_1: Lagrangian dual bound values obtained by solving the continuous MP without and with cutting planes being added, respectively;

3. $LB\%$ and $LB_1\%$: the gaps of the LB and LB_1 to the best-obtained integer solution values (denoted as $Best$) among all considered algorithms, respectively;
4. CPU: the CPU time in seconds (contained fragment enumeration and preprocessing time);
5. Obj': the optimal objective values from B&P;
6. LB': the lower bounds obtained from B&P;
7. $LB'\%$: the gaps of LB' to the best-obtained integer solution values (denoted as $Best$) among all considered algorithms;
8. N_{node}: the number of nodes explored in the B&P tree;
9. Obj_2: the best solution found by Bongiovanni et al. (2019);
10. LB_2: the best lower bounds reported in Bongiovanni et al. (2019);
11. $LB_2\%$: the gaps between the LB_2 and $Best$;
12. *BKS*: the best known solutions of the DA algorithm of Su et al. (2023);
13. $\overline{CPU}$: average CPU time (in seconds) per instance;
14. $\overline{LB}\%$ and $\overline{LB_1}\%$: average value of $LB\%$ and $LB_1\%$ per instance, respectively;
15. $\overline{LB_2}\%$: average $LB_2\%$ per instance;
16. NA (Not Available): the considered algorithm does not obtain a feasible integer solution or solve the continuous MP within the given time limit for the respective instance;
17. $\#opt$: number of optimal solutions obtained by the respective algorithm;
18. $\#bestlb$: number of times the respective algorithm provides the best lower bound of all considered algorithms;
19. $\#bestub$: number of times the respective algorithm provides the best integer solution of all considered algorithms;

And:

$$LB\% = \frac{Best - LB}{Best} \times 100\%$$

$LB_1\%$ and $LB_2\%$ are calculated similarly.

In Table 4.1, Table 4.2, and Table 4.3, we report solutions for solving the integer RMP in the first column and the Lagrangian dual bounds (hereafter LB) in the second column. If CG converges within the time limit, the LB values are equal to solutions to continuous RMP. Otherwise, we mark them in italics. If the optimal solution of continuous MP is also an integer solution, then we obtain the optimal solution of MP, and we indicate it by an asterisk in column

"Obj". Otherwise, we try to close the integrality gap by enhancing the continuous MP formulation with cuts. Then, we perform the CG algorithm to solve the continuous MP, considering additional dual variables introduced by cuts. If the remaining integrality gap is closed, we mark the obtained optimal solution with an asterisk in column "Obj_1". Equal/improved integer solutions and LB values compared to those reported in Bongiovanni et al. (2019) are marked in bold.

In the scenario of $\gamma = 0.4$ and $\gamma = 0.7$, we find strictly better integer solutions than the reportedly optimal solution in Bongiovanni et al. (2019). After analyzing their model and parameter settings, we find that in the model of Bongiovanni et al. (2019), the employed "big M" values were not correctly computed. Readers can refer to Appendix A for a more in-depth analysis and how the "big M" values should be set correctly. These problematic results are marked in italics in column "Obj_2", and we add two asterisks to our values in columns "Obj" and "Obj_1" for the concerned instances (e.g.,a2-24-0.4, a3-30-0.4). The corresponding $LB_2\%$ values are therefore negative.

CG Results on type-a instances under different minimum battery level restrictions.

We first conduct experiments on type-a instances considering different minimum battery level restrictions $\gamma = 0.1, 0.4, 0.7$. With rising values of γ, vehicles must have a higher minimum battery level when returning to the depot. Recalling that each recharging station can only be visited at most once. We present our proposed CG algorithm results and compare them to those of Bongiovanni et al. (2019).

Before adding cuts to further enhance lower bounds, our proposed CG algorithm obtains optimal solutions for 28 out of 42 instances. In addition, we provide 25 equal integer solutions and generate 11 better integer solutions for previously solved and unsolved instances. Furthermore, 20 equal lower bounds are generated, and 14 lower bounds reported in Bongiovanni et al. (2019) are improved by 0.93% on average. Then, the remaining integrality gaps are closed to a large extent by adding cuts to the continuous MP formulation. The proposed CG algorithm solves 9 more instances optimally and further improves 6 lower bounds and 4 integer solutions. In total 37 out of 42 instances are solved optimally at the root node, and we improve the reported lower bounds of Bongiovanni et al. (2019) by 1.06% on average.

Another important observation is that, in the case of $\gamma = 0.4, 0.7$, we obtain strictly better optimal solutions than the reportedly optimal solutions in Bongiovanni et al. (2019). The associated instances are a2-24-0.4, a3-30-0.4, a3-36-0.4, a2-24-0.7, a3-24-0.7, and a4-24-0.7. The reason is that in the model of Bongiovanni et al. (2019), the "big M" values were incorrect. We give a detailed explanation in Appendix A.

CG Results on type-u instances under different minimum battery level restrictions.

In this part, we conduct experiments on type-u instances under different minimum battery level restrictions $\gamma = 0.1, 0.4, 0.7$. The results are summarized in Table 4.2.

Without adding cuts to the continuous MP, the proposed CG algorithm solves 22 out of 42 instances optimally

Table 4.1 Root node results of CG with labeling algorithm on type-a instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	CG results				CG with cutting planes results				Bongiovanni et al., (2019) results [a]			
Instance	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**237.38***	**237.38**	0	23.1	**237.38***	**237.38**	0	23.1	**237.38***	**237.38**	0	1.2
a2-20	**279.08***	**279.08**	0	140.6	**279.08***	**279.08**	0	140.6	**279.08***	**279.08**	0	4.2
a2-24	**346.21***	**346.21**	0	274.2	**346.21***	**346.21**	0	274.2	**346.21***	**346.21**	0	9.0
a3-18	**236.82***	**236.82**	0	14.2	**236.82***	**236.82**	0	14.2	**236.82***	**236.82**	0	4.8
a3-24	**274.80***	**274.80**	0	116.4	**274.80***	**274.80**	0	116.4	**274.80***	**274.80**	0	13.8
a3-30	**413.27***	**413.27**	0	351.5	**413.27***	**413.27**	0	351.5	**413.27***	**413.27**	0	102.0
a3-36	**481.17***	**481.17**	0	884.6	**481.17***	**481.17**	0	884.6	**481.17***	**481.17**	0	106.8
a4-16	**222.49**	222.38	0.05%	7.5	**222.49***	**222.49**	0	11.4	**222.49***	**222.49**	0	3.6
a4-24	**310.84***	**310.84**	0	28.8	**310.84***	**310.84**	0	28.8	**310.84***	**310.84**	0	31.2
a4-32	**393.96***	**393.96**	0	311.3	**393.96***	**393.96**	0	311.3	**393.96***	**393.96**	0	612.0
a4-40	**453.84***	**453.84**	0	764.0	**453.84***	**453.84**	0	764.0	**453.84***	**453.84**	0	517.2
a4-48	**554.54***	**554.54**	0	2148.5	**554.54***	**554.54**	0	2148.5	**554.54**	526.96	5.04%	7200.0
a5-40	416.79	413.48	0.25%	318.6	**414.51***	**414.51**	0	805.2	**414.51***	**414.51**	0	1141.8
a5-50	**559.17***	**559.17**	0	1521.6	**559.17***	**559.17**	0	1521.6	**559.17**	531.15	5.01%	7200.0
Avg			**0.02%**	493.2			**0**	528.2			0.72%	1210.5
$\gamma = 0.4$	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**237.38***	**237.38**	0	40.4	**237.38***	**237.38**	0	40.4	**237.38***	**237.38**	0	1.8
a2-20	**280.70***	**280.70**	0	134.2	**280.70***	**280.70**	0	134.2	**280.70***	**280.70**	0	49.8
a2-24	**347.04****	**347.04**	0	535.5	**347.04****	**347.04**	0	535.5	*348.04**	*348.04*	-0.29%	25.2
a3-18	**236.82***	**236.82**	0	22.8	**236.82***	**236.82**	0	22.8	**236.82***	**236.82**	0	4.2
a3-24	**274.80***	**274.80**	0	81.0	**274.80***	**274.80**	0	81.0	**274.80***	**274.80**	0	16.8
a3-30	**413.34****	**413.34**	0	484.6	**413.34****	**413.34**	0	484.6	*413.37**	*413.37*	-0.01%	99.0
a3-36	483.83**	481.85	0.41%	1273.5	**482.75****	**482.64**	0.02%	7200.0	*484.14**	*484.14*	-0.06%	306.6
a4-16	**222.49**	222.38	0.05%	6.5	**222.49***	**222.49**	0	10.1	**222.49***	**222.49**	0	5.4
a4-24	**311.03***	**311.03**	0	43.2	**311.03***	**311.03**	0	43.2	**311.03***	**311.03**	0	39.6
a4-32	**394.26***	**394.26**	0	276.9	**394.26***	**394.26**	0	276.9	**394.26***	**394.26**	0	681.6
a4-40	**453.84***	**453.84**	0	681.8	**453.84***	**453.84**	0	681.8	**453.84***	**453.84**	0	417.6
a4-48	**554.60***	**554.60**	0	6648.4	**554.60***	**554.60**	0	6648.4	**554.60**	529.22	4.58%	7200.0
a5-40	415.25	413.48	0.25%	307.8	**414.51***	**414.51**	0	481.6	**414.51***	**414.51**	0	1221.0
a5-50	**559.51***	**559.51**	0	1984.5	**559.51***	**559.51**	0	1984.5	560.50	528.91	5.47%	7200.0
Avg			**0.05%**	894.4			**0**	1330.3			0.74%	1233.5
$\gamma = 0.7$	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**240.66***	**240.66**	0	56.1	**240.66***	**240.66**	0	56.1	**240.66***	**240.66**	0	5.4
a2-20	**293.27**	**292.73**	0.18%	1108.8	**293.27***	**293.27**	0	1733.8	NA	287.17	2.08%	7200.0
a2-24	**353.18****	**353.18**	0	1057.4	**353.18****	**353.18**	0	1057.4	*358.21**	*358.21*	-1.42%	961.2
a3-18	**240.58***	**240.58**	0	38.6	**240.58***	**240.58**	0	38.6	**240.58***	**240.58**	0	48.0
a3-24	**275.97****	**275.97**	0	175.7	**275.97****	**275.97**	0	175.7	*277.72**	*277.72*	-0.63%	152.4
a3-30	**426.40**	**422.57**	0.56%	1171.6	**424.93***	**424.93**	0	3401.9	NA	417.06	1.85%	7200.0
a3-36	500.57	**491.80**	0.45%	2191.6	**494.04**	**493.71**	0.07%	7200.0	**494.04**	485.91	1.65%	7200.0
a4-16	**223.13***	**223.13**	0	8.4	**223.13***	**223.13**	0	8.4	**223.13***	**223.13**	0	67.2
a4-24	318.24	316.24	0.13%	84.3	**316.65****	**316.65**	0	129.7	*318.21**	*318.19*	-0.49%	1834.8
a4-32	**397.87**	**396.84**	0.26%	734.7	**397.87**	**397.78**	0.02%	7200.0	430.07	387.99	2.48%	7200.0
a4-40	**467.72**	**466.96**	0.16%	3252.4	**467.72***	**467.72**	0	5238.4	NA	443.62	5.15%	7200.0
a4-48	NA	*476.54*	NA	7200.0	NA	*476.54*	NA	7200.0	NA	524.92	NA	7200.0
a5-40	**418.75**	**417.25**	0.36%	2182.2	**418.75***	**418.75**	0	6808.1	447.63	405.99	4.78%	7200.0
a5-50	NA	*176.58*	NA	7200.0	NA	*176.58*	NA	7200.0	NA	522.37	NA	7200.0
Avg			**0.32%**	1890.1			**0.01%**	3389.2			1.70%	4333.5
Summary	$\#opt$	$\#bestlb$	$\overline{LB\%}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB_1\%}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB_2\%}$	$\#bestub$
	28	34	0.13%	36	37	40	0.003%	40	24	26	1.06%	28

a: Due to incorrect big M values, some of the reported results of Bongiovanni et al. (2019) are higher than the actual optimal values. Those results are highlighted in italics;

Table 4.2 Root node results of CG with labeling algorithm on type-u instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	CG results				CG with cutting planes results				Bongiovanni et al., (2019) results [a]			
Instance	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	**57.61**	57.08	0.92%	43.6	**57.61***	**57.61**	0	72.1	**57.61***	**57.61**	0	21.0
u2-20	**55.59***	**55.59**	0	248.2	**55.59***	**55.59**	0	248.2	**55.59***	**55.59**	0	9.6
u2-24	91.36	90.55	0.12%	851.7	**90.66****	**90.66**	0	7001.9	*91.27**	*91.27*	-0.67%	432.0
u3-18	**50.74***	**50.74**	0	36.6	**50.74***	**50.74**	0	36.6	**50.74***	**50.74**	0	10.8
u3-24	**67.56***	**67.56**	0	107.0	**67.56***	**67.56**	0	107.0	**67.56***	**67.56**	0	130.2
u3-30	**76.75***	**76.75**	0	998.1	**76.75***	**76.75**	0	998.1	**76.75***	**76.75**	0	438.0
u3-36	104.39	103.94	0.10%	1894.6	**104.04***	**104.04**	0	3293.4	**104.04***	**104.04**	0	1084.8
u4-16	**53.58***	**53.58**	0	12.9	**53.58***	**53.58**	0	12.9	**53.58***	**53.58**	0	48.0
u4-24	**89.83***	**89.83**	0	51.3	**89.83***	**89.83**	0	51.3	**89.83***	**89.83**	0	13.2
u4-32	**99.29***	**99.29**	0	434.3	**99.29***	**99.29**	0	434.3	**99.29***	**99.29**	0	1158.6
u4-40	**133.11***	**133.11**	0	3385.9	**133.11***	**133.11**	0	3385.9	**133.11***	**133.11**	0	185.4
u4-48	**148.08**	**147.02**	0.72%	7200.0	**148.08**	**147.02**	0.72%	7200.0	148.30	134.48	9.18%	7200.0
u5-40	**121.86***	**121.86**	0	1632.0	**121.86***	**121.86**	0	1632.0	**121.86**	114.12	6.35%	7200.0
u5-50	**142.82**	**142.75**	0.05%	7100.9	**142.82**	**142.75**	0.05%	7100.9	143.10	132.69	7.09%	7200.0
Avg			**0.14%**	1714.1			**0.06%**	2255.3			1.66%	1795.1
$\gamma = 0.4$	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	**57.65***	**57.65**	0	53.2	**57.65***	**57.65**	0	53.2	**57.65***	**57.65**	0	25.8
u2-20	56.61	56.06	0.50%	407.3	**56.34***	**56.34**	0	665.2	**56.34***	**56.34**	0	12.0
u2-24	91.62	90.80	0.51%	1140.2	**91.27****	90.95	0.33%	4266.0	*91.63**	*91.63*	-0.39%	757.2
u3-18	**50.74***	**50.74**	0	45.2	**50.74***	**50.74**	0	45.2	**50.74***	**50.74**	0	13.8
u3-24	**67.56***	**67.56**	0	109.5	**67.56***	**67.56**	0	109.5	**67.56***	**67.56**	0	220.8
u3-30	**76.75***	**76.75**	0	912.8	**76.75***	**76.75**	0	912.8	**76.75***	**76.75**	0	336.6
u3-36	**104.06***	**104.06**	0	4922.7	**104.06***	**104.06**	0	4922.7	**104.06***	**104.06**	0	2010.0
u4-16	**53.58***	**53.58**	0	13.6	**53.58***	**53.58**	0	13.6	**53.58***	**53.58**	0	44.4
u4-24	**89.83***	**89.83**	0	56.8	**89.83***	**89.83**	0	56.8	**89.83***	**89.83**	0	28.2
u4-32	**99.29***	**99.29**	0	696.8	**99.29***	**99.29**	0	696.8	**99.29***	**99.29**	0	2667.6
u4-40	**133.78****	133.37	0.31%	2186.1	**133.78****	**133.46**	0.24%	3958.2	*133.91**	*133.91*	-0.10%	2653.2
u4-48	**147.63**	***146.37***	0.85%	7200.0	**147.63**	***146.37***	0.85%	7200.0	NA	133.86	9.33%	7200.0
u5-40	**121.96***	**121.96**	0	1249.1	**121.96***	**121.96**	0	1249.1	122.23	112.58	7.69%	7200.0
u5-50	**142.84**	**142.75**	0.06%	7200.0	**142.84**	**142.75**	0.06%	7200.0	143.14	134.09	6.13%	7200.0
Avg			**0.16%**	1871.0			**0.11%**	2239.2			1.69%	2169.3
$\gamma = 0.7$	Obj	LB	$LB\%$	CPU(s)	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	60.01	58.48	1.20%	89.6	**59.19***	**59.19**	0	2597.2	**59.19***	**59.19**	0	338.4
u2-20	**56.86***	**56.86**	0	2247.6	**56.86***	**56.86**	0	2247.6	**56.86***	**56.86**	0	72.0
u2-24	**92.17**	**91.48**	0.75%	2056.3	**92.17**	**91.71**	0.50%	7200.0	NA	90.83	1.45%	7200.0
u3-18	**50.99**	50.94	0.10%	65.4	**50.99***	**50.99**	0	116.4	**50.99***	**50.99**	0	24.0
u3-24	68.44	68.00	0.57%	119.9	68.44	68.12	0.47%	176.2	**68.39***	**68.39**	0	400.2
u3-30	**77.41****	77.33	0.10%	955.3	**77.41****	**77.41**	0	1778.3	*78.14**	*78.14*	-0.94%	3401.4
u3-36	106.45	**105.46**	0.31%	7200.0	106.45	**105.46**	0.31%	7200.0	**105.79**	104.37	1.34%	7200.0
u4-16	**53.87***	**53.87**	0	33.8	**53.87***	**53.87**	0	33.8	**53.87***	**53.87**	0	88.8
u4-24	**89.96***	**89.96**	0	82.3	**89.96***	**89.96**	0	82.3	**89.96***	**89.96**	0	22.8
u4-32	**99.50***	**99.50**	0	4559.3	**99.50***	**99.50**	0	4559.3	**99.50***	**99.50**	0	2827.2
u4-40	135.33	**134.56**	0.54%	1959.5	**135.29**	**134.65**	0.47%	5555.5	NA	133.01	2.46%	7200.0
u4-48	**185.16**	*-216.07*	NA	7200.0	**185.16**	*-216.07*	NA	7200.0	NA	**132.49**	NA	7200.0
u5-40	124.01	**122.82**	0.81%	4774.6	**123.82**	**122.87**	0.77%	7200.0	NA	109.28	11.88%	7200.0
u5-50	216.07	*132.79*	NA	7200.0	216.07	*132.79*	NA	7200.0	**144.36**	**133.33**	7.64%	7200.0
Avg			**0.36%**	2753.1			**0.21%**	3796.2			1.98%	3598.2
Summary	$\#opt$	$\#bestlb$	$\overline{LB}\%$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB}\%$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB'}\%$	$\#bestub$
	22	30	0.22%	35	29	38	0.13%	39	26	28	1.78%	29

a: Due to incorrect big M values, some of the reported results of Bongiovanni et al. (2019) are higher than the actual optimal values. Those results are highlighted in italics;

at the root node, while all other instances have quite small gaps (0.22%) to the best-found objective values. The reported B&C results in Bongiovanni et al. (2019) have an average gap of 1.78% between the obtained lower bounds and the best-found objective values. Additionally, we obtain 21 equal lower bounds, and we improve 9 lower bounds by employing the CG algorithm, with an improvement of up to 11.88%. Moreover, we report 12 new best solutions and 23 equal solutions on previously solved and unsolved instances. After adding cuts to the continuous MP, 7 more instances are solved optimally, and 8 lower bounds are further improved. We also observe that we obtain better integer solutions than the reportedly optimal solutions in Bongiovanni et al. (2019) as they used incorrect "big M" values (see analysis in Appendix A). The associated instances are u2-24-0.1, u2-24-0.4, u4-40-0.4, u3-30-0.7.

CG Results on type-r instances under different minimum battery level restrictions.

In Table 4.3, we present the CG results on type-r instances under different γ values. Since no optimal solutions are available for these instances from the literature, we compare the CG results to the heuristic results of Su et al. (2023) (presented in the last column). BKS denotes the best-known solutions obtained by the DA algorithm of Su et al. (2023). To tackle large-sized instances, we extend the time limit of our CG algorithm to 5 hours.

Table 4.3 Root node results of CG with labeling algorithm on type-r instances under $\gamma = 0.1, 0.4$

$\gamma = 0.1$	CG results			CG with cutting planes results			DA in Section 3.3
Instance	Obj	LB	CPU(s)	Obj_1	LB_1	CPU(s)	BKS
r5-60	**687.80**	**682.27**	11263.2	**683.39***	**683.39**	18000.0	691.83
r6-48	**506.45***	**506.45**	1109.1	**506.45***	**506.45**	1109.1	506.72
r6-60	**692.25**	**689.31**	2389.2	**689.45***	**689.45**	16387.4	692.00
r6-72	**761.34**	***748.90***	18000.0	**761.34**	***748.90***	18000.0	777.44
r7-56	**613.19**	**611.97**	1402.5	**612.02***	**612.02**	4975.3	613.10
r7-70	**760.23**	**753.56**	7275.6	**754.28**	**753.56**	18000.0	760.90
r7-84	975.26	***638.59***	18000.0	975.26	***638.59***	18000.0	**889.38**
r8-64	**632.22***	**632.22**	3246.5	**632.22***	**632.22**	3246.5	641.99
r8-80	**788.99***	**788.99**	14563.8	**788.99***	**788.99**	14563.8	803.52
r8-96	1329.75	***651.32***	18000.0	1329.75	***651.32***	18000.0	**1053.11**
Avg			13028.2				
$\gamma = 0.4$	Obj	LB	CPU(s)	Obj_1	LB_1	CPU(s)	BKS
r5-60	**686.85**	**682.76**	15555.3	**686.85**	**683.56**	18000.0	697.97
r6-48	**506.45***	**506.45**	1195.6	**506.45***	**506.45**	1195.6	506.91
r6-60	695.69	**688.81**	4109.1	**689.48**	**689.33**	18000.0	694.78
r6-72	NA	*-303.92*	18000.0	NA	*-303.92*	18000.0	**799.60**
r7-56	**612.17**	**611.97**	1496.3	**612.02***	**612.02**	3991.7	613.66
r7-70	**759.27**	**753.56**	10380.5	**754.28***	**754.28**	18000.0	766.05
r7-84	1081.76	*-404.31*	18000.0	1081.76	*-404.31*	18000.0	**932.12**
r8-64	**632.22***	**632.22**	3030.0	**632.22***	**632.22**	3030.0	638.36
r8-80	**788.99**	***780.81***	18000.0	**788.99**	***780.81***	18000.0	811.19
Avg			12913.0				
Summary	$\#opt$	$\#bestub$	$\#bestlb$	$\#opt$	$\#bestub$	$\#bestlb$	$\#bestub$
	5	14	17	10	15	17	4

As shown in Table 4.3, we obtain 15 better solutions than the heuristic solutions reported in Su et al. (2023), and

10 of these solutions are proven optimal. We report the computed lower bound values in the third column, which are the first lower bound values for the type-r instances. For setting $\gamma = 0.7$, neither the CG algorithm nor the DA of Su et al. (2023) is able to provide feasible solutions. We improve a majority of the integer solutions, while several DA heuristic solutions are still better than CG-obtained solutions.

B&P results on type-a, -u, and -r instances under different minimum battery level restrictions.

From previous experiments, adding cuts is shown to be efficient in enhancing the quality of lower bounds and can close the integrality gaps for some instances. In this part, we explore the performance of the B&P algorithm by applying it to solve instances that the CG algorithm obtains fractional solutions. The maximum time limit for executing the B&P algorithm is set to two hours, and the maximum number of nodes in the B&P tree is 400. The obtained B&P results on type-a and -u instances are shown in Table 4.4, 4.5 and are compared with the results of the CG algorithm with cuts and the best-reported B&C results of Bongiovanni et al. (2019). For type-r instances, we extend the maximum time limit to 5 hours and compare the obtained B&P results to the CG results with adding cuts. The B&P results on type-r instances are summarized in Table 4.6.

Compared with the best-reported B&C results in Bongiovanni et al. (2019), we finally solve 71 out of 84 instances optimally within the two-hour time limit, while the B&C algorithm can only solve 49 instances optimally. In addition, we obtain 26 new best solutions and 54 equal solutions and enhanced 30 lower bounds. As for computational efficiency, we observe a decrease of 43% in the average CPU time with the proposed B&P algorithm in solving type-a instances, compared to the reported CPU times of the B&C algorithm. On type-u instances, the average CPU time of the B&P algorithm is similar to that of the B&C algorithm. The reason might be that type-a instances include identical vehicles and depot locations, which actually defines identical subproblems. Therefore, it is enough to solve one subproblem to find minimum-reduced-cost columns. In contrast, type-u instances define heterogeneous subproblems and require more computational efforts to solve. Then, we compared our B&P algorithm results with our results of the CG algorithm with cuts. On type-a and -u instances, the proposed B&P algorithm solves 5 additional instances optimally, obtains 6 tighter lower bounds, and generates 3 additional new best solutions. In addition, benefiting from the good quality of lower bounds obtained at the root node, only a few nodes of the search tree suffice to close the gap when applying the B&P. As a result, branching seems to be more computationally attractive than adding valid cuts, as subset-row inequalities introduce additional complexity to solve the subproblems while branching will not. We have the same conclusion from the obtained B&P results on type-r instances. Compared with the results of the CG algorithm with cutting planes, the proposed B&P algorithm solves 3 more instances into optimality, further tightens 4 previously reported lower bounds, and improves 3 previously generated integer solutions.

Table 4.4 B&P results on type-a instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	CG with cutting planes results				B&P results					Bongiovanni et al., (2019) results [a]			
Instance	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**237.38***	**237.38**	0	23.1	**237.38***	**237.38**	0	1	23.1	**237.38***	**237.38**	0	1.2
a2-20	**279.08***	**279.08**	0	140.6	**279.08***	**279.08**	0	1	140.6	**279.08***	**279.08**	0	4.2
a2-24	**346.21***	**346.21**	0	274.2	**346.21***	**346.21**	0	1	274.2	**346.21***	**346.21**	0	9.0
a3-18	**236.82***	**236.82**	0	14.2	**236.82***	**236.82**	0	1	14.2	**236.82***	**236.82**	0	4.8
a3-24	**274.80***	**274.80**	0	116.4	**274.80***	**274.80**	0	1	116.4	**274.80***	**274.80**	0	13.8
a3-30	**413.27***	**413.27**	0	351.5	**413.27***	**413.27**	0	1	351.5	**413.27***	**413.27**	0	102.0
a3-36	**481.17***	**481.17**	0	884.6	**481.17***	**481.17**	0	1	884.6	**481.17***	**481.17**	0	106.8
a4-16	**222.49***	**222.49**	0	11.4	**222.49***	**222.49**	0	5	8.8	**222.49***	**222.49**	0	3.6
a4-24	**310.84***	**310.84**	0	28.8	**310.84***	**310.84**	0	1	28.8	**310.84***	**310.84**	0	31.2
a4-32	**393.96***	**393.96**	0	311.3	**393.96***	**393.96**	0	1	311.3	**393.96***	**393.96**	0	612.0
a4-40	**453.84***	**453.84**	0	764.0	**453.84***	**453.84**	0	1	764.0	**453.84***	**453.84**	0	517.2
a4-48	**554.54***	**554.54**	0	2148.5	**554.54***	**554.54**	0	1	2148.5	**554.54**	526.96	5.04%	7200.0
a5-40	**414.51***	**414.51**	0	805.2	**414.51***	**414.51**	0	7	446.7	**414.51***	**414.51**	0	1141.8
a5-50	**559.17***	**559.17**	0	1521.6	**559.17***	**559.17**	0	1	1521.6	**559.17**	531.15	5.01%	7200.0
Avg			**0**	528.2			**0**	1.7	502.5			0.72%	1210.5
$\gamma = 0.4$	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**237.38***	**237.38**	0	40.4	**237.38***	**237.38**	0	1	40.4	**237.38***	**237.38**	0	1.8
a2-20	**280.70***	**280.70**	0	134.2	**280.70***	**280.70**	0	1	134.2	**280.70***	**280.70**	0	49.8
a2-24	**347.04****	**347.04**	0	535.5	**347.04****	**347.04**	0	1	535.5	*348.04**	*348.04*	-0.29%	25.2
a3-18	**236.82***	**236.82**	0	22.8	**236.82***	**236.82**	0	1	22.8	**236.82***	**236.82**	0	4.2
a3-24	**274.80***	**274.80**	0	81.0	**274.80***	**274.80**	0	1	81.0	**274.80***	**274.80**	0	16.8
a3-30	**413.34****	**413.34**	0	484.6	**413.34****	**413.34**	0	1	484.6	*413.37**	*413.37*	-0.01%	99.0
a3-36	**482.75****	482.64	0.02%	7200.0	**482.75****	**482.75**	0	13	4576.6	*484.14**	*484.14*	-0.06%	306.6
a4-16	**222.49***	**222.49**	0	10.1	**222.49***	**222.49**	0	3	10.5	**222.49***	**222.49**	0	5.4
a4-24	**311.03***	**311.03**	0	43.2	**311.03***	**311.03**	0	1	43.2	**311.03***	**311.03**	0	39.6
a4-32	**394.26***	**394.26**	0	276.9	**394.26***	**394.26**	0	1	276.9	**394.26***	**394.26**	0	681.6
a4-40	**453.84***	**453.84**	0	681.8	**453.84***	**453.84**	0	1	681.8	**453.84***	**453.84**	0	417.6
a4-48	**554.60***	**554.60**	0	6648.4	**554.60***	**554.60**	0	1	6648.4	**554.60**	529.22	4.58%	7200.0
a5-40	**414.51***	**414.51**	0	481.6	**414.51***	**414.51**	0	11	542.4	**414.51***	**414.51**	0	1221.0
a5-50	**559.51***	**559.51**	0	1984.5	**559.51***	**559.51**	0	1	1984.5	560.50	528.91	5.47%	7200.0
Avg			**0**	1330.3			**0**	2.7	1147.3			0.74%	1233.5
$\gamma = 0.7$	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
a2-16	**240.66***	**240.66**	0	56.1	**240.66***	**240.66**	0	1	56.1	**240.66***	**240.66**	0	5.4
a2-20	**293.27***	**293.27**	0	1733.8	**293.27***	**293.27**	0	3	1331.8	NA	287.17	2.08%	7200.0
a2-24	**353.18****	**353.18**	0	1057.4	**353.18****	**353.18**	0	1	1057.4	*358.21**	*358.21*	-1.42%	961.2
a3-18	**240.58***	**240.58**	0	38.6	**240.58***	**240.58**	0	1	38.6	**240.58***	**240.58**	0	48.0
a3-24	**275.97****	**275.97**	0	175.7	**275.97****	**275.97**	0	1	175.7	*277.72**	*277.72*	-0.63%	152.4
a3-30	**424.93***	**424.93**	0	3401.9	**424.93***	**424.93**	0	3	2360.8	NA	417.06	1.85%	7200.0
a3-36	**494.04**	493.71	0.07%	7200.0	**494.04**	**494.01**	0.01%	12	7200.0	**494.04**	485.91	1.65%	7200.0
a4-16	**223.13***	**223.13**	0	8.4	**223.13***	**223.13**	0	1	8.4	**223.13***	**223.13**	0	67.2
a4-24	**316.65****	**316.65**	0	129.7	**316.65***	**316.65**	0	5	130.0	*318.21**	*318.19*	-0.49%	1834.8
a4-32	**397.87**	**397.78**	0.02%	7200.0	**397.87***	**397.87**	0	9	1164.8	430.07	387.99	2.48%	7200.0
a4-40	**467.72***	**467.72**	0	5238.4	**467.72***	**467.72**	0	5	2161.3	NA	443.62	5.15%	7200.0
a4-48	NA	*476.54*	NA	7200.0	NA	*476.54*	NA	1	7200.0	NA	524.92	NA	7200.0
a5-40	**418.75***	**418.75**	0	6808.1	**418.75***	**418.75**	0	7	1086.7	447.63	405.99	4.78%	7200.0
a5-50	NA	*176.58*	NA	7200.0	NA	*176.58*	NA	1	7200.0	NA	522.37	NA	7200.0
Avg			0.01%	3389.2			**0**	3.6	2226.5			1.70%	4333.5
Summary	$\#opt$	$\#bestlb$	$\overline{LB_1\%}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB'\%}$	$\overline{N_{node}}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB_2\%}$	$\#bestub$
	37	38	0.003%	40	39	40	0	2.7	40	24	26	1.06%	28

a: Due to incorrect big M values, some of the reported results of Bongiovanni et al. (2019) are higher than the actual optimal values. Those results are highlighted in italics;

Table 4.5 B&P results on type-u instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	CG with cutting planes results				B&P results					Bongiovanni et al., (2019) results [a]			
Instance	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	**57.61***	**57.61**	0	72.1	**57.61***	**57.61**	0	5	47.2	**57.61***	**57.61**	0	21.0
u2-20	**55.59***	**55.59**	0	248.2	**55.59***	**55.59**	0	1	248.2	**55.59***	**55.59**	0	9.6
u2-24	**90.66****	**90.66**	0	7001.9	**90.66****	**90.66**	0	17	6064.3	*91.27**	*91.27*	-0.67%	432.0
u3-18	**50.74***	**50.74**	0	36.6	**50.74***	**50.74**	0	1	36.6	**50.74***	**50.74**	0	10.8
u3-24	**67.56***	**67.56**	0	107.0	**67.56***	**67.56**	0	1	107.0	**67.56***	**67.56**	0	130.2
u3-30	**76.75***	**76.75**	0	998.1	**76.75***	**76.75**	0	1	998.1	**76.75***	**76.75**	0	438.0
u3-36	**104.04***	**104.04**	0	3293.4	**104.04***	**104.04**	0	19	7200.0	**104.04***	**104.04**	0	1084.8
u4-16	**53.58***	**53.58**	0	12.9	**53.58***	**53.58**	0	1	12.9	**53.58***	**53.58**	0	48.0
u4-24	**89.83***	**89.83**	0	51.3	**89.83***	**89.83**	0	1	51.3	**89.83***	**89.83**	0	13.2
u4-32	**99.29***	**99.29**	0	434.3	**99.29***	**99.29**	0	1	434.3	**99.29***	**99.29**	0	1158.6
u4-40	**133.11***	**133.11**	0	3385.9	**133.11***	**133.11**	0	1	3385.9	**133.11***	**133.11**	0	185.4
u4-48	**148.08**	**147.02**	0.72%	7200.0	**148.08**	**147.02**	0.72%	1	7200.0	148.30	134.48	9.18%	7200.0
u5-40	**121.86***	**121.86**	0	1632.0	**121.86***	**121.86**	0	1	1632.0	**121.86**	114.12	6.35%	7200.0
u5-50	**142.82**	**142.75**	0.05%	7100.9	**142.82**	**142.75**	0.05%	1	7100.9	143.10	132.69	7.09%	7200.0
Avg			**0.06%**	2255.3			**0.06%**	3.7	2465.6			1.66%	1795.1
$\gamma = 0.4$	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	**57.65***	**57.65**	0	53.2	**57.65***	**57.65**	0	1	53.2	**57.65***	**57.65**	0	25.8
u2-20	**56.34***	**56.34**	0	665.2	**56.34***	**56.34**	0	7	343.0	**56.34***	**56.34**	0	12.0
u2-24	91.27	90.95	0.33%	4266.0	**91.16****	**91.16**	0	15	6191.6	*91.63**	*91.63*	-0.52%	757.2
u3-18	**50.74***	**50.74**	0	45.2	**50.74***	**50.74**	0	1	45.2	**50.74***	**50.74**	0	13.8
u3-24	**67.56***	**67.56**	0	109.5	**67.56***	**67.56**	0	1	109.5	**67.56***	**67.56**	0	220.8
u3-30	**76.75***	**76.75**	0	912.8	**76.75***	**76.75**	0	1	912.8	**76.75***	**76.75**	0	336.6
u3-36	**104.06***	**104.06**	0	4922.7	**104.06***	**104.06**	0	1	4922.7	**104.06***	**104.06**	0	2010.0
u4-16	**53.58***	**53.58**	0	13.6	**53.58***	**53.58**	0	1	13.6	**53.58***	**53.58**	0	44.4
u4-24	**89.83***	**89.83**	0	56.8	**89.83***	**89.83**	0	1	56.8	**89.83***	**89.83**	0	28.2
u4-32	**99.29***	**99.29**	0	696.8	**99.29***	**99.29**	0	1	696.8	**99.29***	**99.29**	0	2667.6
u4-40	**133.78****	133.46	0.24%	3958.2	**133.78****	**133.70**	0.06%	24	7200.0	*133.91**	*133.91*	-0.10%	2653.2
u4-48	**147.63**	***146.37***	0.85%	7200.0	**147.63**	***146.37***	0.85%	1	7200.0	NA	133.86	9.33%	7200.0
u5-40	**121.96***	**121.96**	0	1249.1	**121.96***	**121.96**	0	1	1249.1	122.23	112.58	7.69%	7200.0
u5-50	**142.84**	**142.75**	0.06%	7200.0	**142.84**	**142.75**	0.06%	1	7200.0	143.14	134.09	6.13%	7200.0
Avg			0.11%	2239.2			**0.07%**	4.1	2143.5			1.70%	2169.3
$\gamma = 0.7$	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	Obj_2	LB_2	$LB_2\%$	CPU(s)
u2-16	**59.19***	**59.19**	0	2597.2	**59.19***	**59.19**	0	49	1307.7	**59.19***	**59.19**	0	338.4
u2-20	**56.86***	**56.86**	0	2247.6	**56.86***	**56.86**	0	1	2247.6	**56.86***	**56.86**	0	72.0
u2-24	92.17	**91.71**	0.40%	7200.0	**92.08**	91.60	0.52%	4	7200.0	NA	90.83	1.36%	7200.0
u3-18	**50.99***	**50.99**	0	116.4	**50.99***	**50.99**	0	9	92.2	**50.99***	**50.99**	0	24.0
u3-24	68.44	68.12	0.32%	176.2	**68.34****	**68.34**	0	47	1127.1	*68.39**	*68.39*	-0.07%	400.2
u3-30	**77.41****	**77.41**	0	1778.3	**77.41****	**77.41**	0	9	2086.5	*78.14**	*78.14*	-0.94%	3401.4
u3-36	106.45	**105.46**	0.31%	7200.0	106.45	**105.46**	0.31%	1	7200.0	**105.79**	104.37	1.34%	7200.0
u4-16	**53.87***	**53.87**	0	33.8	**53.87***	**53.87**	0	1	33.8	**53.87***	**53.87**	0	88.8
u4-24	**89.96***	**89.96**	0	82.3	**89.96***	**89.96**	0	1	82.3	**89.96***	**89.96**	0	22.8
u4-32	**99.50***	**99.50**	0	4559.3	**99.50***	**99.50**	0	1	4559.3	**99.50***	**99.50**	0	2827.2
u4-40	**135.29**	134.65	0.47%	5555.5	135.80	**134.99**	0.22%	44	7200.0	NA	133.01	2.46%	7200.0
u4-48	**185.16**	*-216.07*	NA	7200.0	**185.16**	*-216.07*	NA	1	7200.0	NA	**132.49**	NA	7200.0
u5-40	123.82	122.87	0.05%	7200.0	**122.93***	**122.93**	0	5	2394.4	NA	109.28	11.88%	7200.0
u5-50	216.07	*132.79*	NA	7200.0	216.07	*132.79*	NA	1	7200.0	**144.36**	**133.33**	7.64%	7200.0
Avg			0.13%	3796.2			**0.09%**	12.4	3566.5			1.97%	3598.2
Summary	$\#opt$	$\#bestlb$	$\overline{LB_1\%}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB\%}$	$\overline{N_{node}}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB'\%}$	$\#bestub$
	29	35	0.10%	36	32	39	0.07%	6.7	39	25	27	1.78%	28

a: Due to incorrect big M values, some of the reported results of Bongiovanni et al. (2019) are higher than the actual optimal values. Those results are highlighted in italics;

Table 4.6 B&P results on type-r instances under $\gamma = 0.1, 0.4$

$\gamma = 0.1$	CG with cutting planes results				B&P results					DA
Instance	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	BKS
r5-60	**683.39***	**683.39**	0	18000.0	**683.39***	**683.39**	0	7	7817.0	691.83
r6-48	**506.45***	**506.45**	0	1109.1	**506.45***	**506.45**	0	1	1109.1	506.72
r6-60	**689.45***	**689.45**	0	16387.4	**689.45***	**689.45**	0	15	3695.7	692.00
r6-72	**761.34**	***748.90***	NA	18000.0	**761.34**	***748.90***	NA	1	18000.0	777.44
r7-56	**612.02***	**612.02**	0	4975.3	**612.02***	**612.02**	0	3	810.8	613.10
r7-70	**754.28**	753.56	0.01%	18000.0	**754.28***	**754.28**	0	31	18000.0	760.90
r7-84	975.26	***638.59***	NA	18000.0	975.26	***638.59***	NA	1	18000.0	**889.38**
r8-64	**632.22***	**632.22**	0	3246.5	**632.22***	**632.22**	0	1	3246.5	641.99
r8-80	**788.99***	**788.99**	0	14563.8	**788.99***	**788.99**	0	1	14563.8	803.52
r8-96	1329.75	***651.32***	NA	18000.0	1329.75	***651.32***	NA	1	18000.0	**1053.11**
Avg			0	13028.2			0	6.2	10324.3	
$\gamma = 0.4$	Obj_1	LB_1	$LB_1\%$	CPU(s)	Obj'	LB'	$LB'\%$	N_{node}	CPU′	BKS
r5-60	686.85	683.56	0.54%	18000.0	**684.49***	**684.49**	0	9	18000.0	697.97
r6-48	**506.45***	**506.45**	0	1195.6	**506.45***	**506.45**	0	1	1195.6	506.91
r6-60	689.48	689.33	0.02%	18000.0	**689.46***	**689.46**	0	3	2521.1	694.78
r6-72	NA	*-303.92*	NA	18000.0	NA	*-303.92*	NA	1	18000.0	**799.60**
r7-56	**612.02***	**612.02**	0	3991.7	**612.02***	**612.02**	0	3	803.0	613.66
r7-70	**754.28***	**754.28**	0	18000.0	**754.28***	**754.28**	0	11	13664.6	766.05
r7-84	1081.76	*-404.31*	NA	18000.0	1081.76	*-404.31*	NA	1	18000.0	**932.12**
r8-64	**632.22***	**632.22**	0	3030.0	**632.22***	**632.22**	0	1	3030.0	638.36
r8-80	**788.99**	***780.81***	NA	18000.0	**788.99**	***780.81***	NA	1	18000.0	811.19
Avg			0.09%	12913.0				0	10357.1	
Summary	$\#opt$	$\#bestlb$	$\overline{LB\%}$	$\#bestub$	$\#opt$	$\#bestlb$	$\overline{LB\%}$	$\overline{N_{node}}$	$\#bestub$	$\#bestub$
	10	14	0.05%	13	13	17	0	3.4	16	4

Allowing unlimited visits to recharging stations.

In Bongiovanni et al. (2019), each recharging station $s \in S$ can be visited at most once. The authors allow multiple visits to each recharging station by replicating the set of recharging stations S. Therefore, the maximum number of visits per recharging station (denoted as $N^s_{max}, s \in S$) must be predefined. Only three different values of N^s_{max}, $N^s_{max} = \{1, 2, 3\}$ are tested on the B&C algorithm for type-u instances. From their results, the computational time increases substantially with a higher value of N^s_{max}, which prevents them from applying the B&C algorithm to allow unlimited visits to each recharging station ($N^s_{max} = \infty$).

In this part, we investigate the case of $N^s_{max} = \infty$. To allow unlimited visits per recharging station, the constraints (4.3) in MP are changed to only restrict the visits to destination depots. In the labeling algorithm, constraints (4.24) are deleted to allow visits to the same recharging station. The B&P algorithm is executed on type-a, -u, and -r instances. If the adapted B&P algorithm solves instances optimally at the root node, the corresponding results are marked with an asterisk in the column named "Obj". The time limit of the B&P algorithm to solve type-a and -u instances is set to two hours. For type-r instances, we allow a maximum time of 5 hours.

Table 4.7, Table 4.8, and Table 4.9 show the results of allowing unlimited visits to each recharging station under

different settings of γ for type-a, -u, and -r instances, respectively. For $\gamma = 0.1$, as the optimal solutions are obtained without visiting recharging stations multiple times for type-a instances, we only conduct experiments for $\gamma = 0.4, 0.7$ for type-a instances. For small-to-medium-sized instances (type-a and -u instances), our B&P algorithm obtains optimal solutions for 62 out of 70 instances, among which 60 are solved optimally at the root node without branching. For large-scale instances (type-r instances), the proposed B&P algorithm solves 18 instances optimally (12 instances are solved optimally at the root node). Feasible solutions are yielded for all instances. Hence, we demonstrate in all cases that the B&P algorithm can be well adapted to allow unlimited visits to recharging stations.

To analyze the effect of allowing unlimited visits, we compare our obtained results with the best-reported results of allowing at most one recharging visit among the B&P and B&C algorithms (presented in the column named "BKS′"). The column of "BKS′%" represents the deviation between BKS and the best-obtained results allowing unlimited charging visits per station. If a solution yielded under $N^s_{max} = \infty$ has a lower cost than under $N^s_{max} = 1$, the corresponding BKS′% is positive. We also report the maximum number of recharging visits on a recharging station for the obtained solutions in the column named "N^s_{max}". The major observations include: (1) we find 52 out of 100 obtained solutions contain multiple visits to the same recharging station, especially under $\gamma = 0.7$, and we observe considerable improvements in solution feasibility and quality with setting $N^s_{max} = \infty$; (2) there are several solutions for type-a instances that contain multiple recharging visits but have the same cost as the obtained solutions with $N^s_{max} = 1$ (e.g., a3-24-0.4). This situation is possible as all recharging stations for a type-a instance are at the same location and one can have an equal-cost solution by replacing one recharging station with another; (3) setting $N^s_{max} = 3$ seems enough for solving type-u instances allowing unlimited recharging visits, while one needs to set N^s_{max} to 4 and 8 for type-a and -r instances, respectively.

4.4.3 Practical insights from CG, B&P and DA algorithm results

In this part, we compare our obtained CG and B&P algorithm results with the best-reported DA algorithm results presented in Chapter 3. We present aggregated results of different algorithms on type-a and -u instances in Table 4.10. The reference results are best-reported results of Bongiovanni et al. (2019). As the results of different algorithms on type-r instances have been presented in Table 4.3 and 4.6, we will not repeat them in this part.

Table 4.7 Allowing unlimited visits to each recharging station: B&P algorithm results on Type-a instances under $\gamma = 0.4, 0.7$

$\gamma = 0.4$			Unlimited charging visits				At-most-one visit	
Instance	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
a2-16	**237.38***	237.38	0	1	1	34.6	**237.38***	0
a2-20	**280.70***	280.70	0	1	1	255.4	**280.70***	0
a2-24	**346.28***	346.28	0	1	2	964.6	347.04*	0.21%
a3-18	**236.82***	236.82	0	1	1	30.1	**236.82***	0
a3-24	**274.80***	274.80	0	1	2	86.7	**274.80***	0
a3-30	**413.28***	413.28	0	1	3	742.9	413.34*	0.02%
a3-36	**481.17***	481.17	0	1	3	1522.8	482.75	0.33%
a4-16	**222.49***	222.49	0	1	0	9.6	**222.49***	0
a4-24	**311.03***	311.03	0	1	0	34.5	**311.03***	0
a4-32	**394.26***	394.26	0	1	2	205.0	**394.26***	0
a4-40	**453.84***	453.84	0	1	1	1027.1	**453.84***	0
a4-48	**554.54***	554.54	0	1	3	4630.9	554.60*	0.01%
a5-40	**414.51***	414.51	0	1	2	539.0	**414.51***	0
a5-50	**559.48***	559.48	0	1	2	2661.4	559.51*	0.004%
Avg			0	1.0	1.6	909.6		0.04%
$\gamma = 0.7$	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
a2-16	**240.66***	240.66	0	1	1	225.4	**240.66***	0
a2-20	**285.86***	285.86	0	1	2	469.6	293.27*	2.53%
a2-24	**350.32***	350.32	0	1	3	3513.6	353.18*	0.81%
a3-18	**238.82***	238.82	0	1	3	62.1	240.58*	0.73%
a3-24	**275.20***	275.20	0	1	2	244.7	275.97*	0.28%
a3-30	**413.35***	413.35	0	1	4	1556.4	424.93*	2.70%
a3-36	**483.08***	483.08	0	1	3	1254.4	494.04	2.22%
a4-16	**222.49***	222.49	0	1	2	11.0	223.13*	0.29%
a4-24	**315.40***	315.40	0	7	2	193.63	316.65*	0.39%
a4-32	**394.94***	394.94	0	1	4	427.7	397.87	0.74%
a4-40	**457.76***	457.76	0	1	4	1604.2	467.72*	2.13%
a4-48	570.31	556.99	1.95%	1	4	7200.0	NA	NA
a5-40	**415.88***	415.88	0	1	4	1171.2	418.75*	0.69%
a5-50	580.00	565.89	2.30%	1	4	7200.0	NA	NA
Avg			0.30%	1.4	3	1795.3		1.13%
Summary	$\#opt$	$\#bestub$	$\overline{LB\%}$	$\overline{N_{node}}$	$\overline{N^s_{max}}$	$\overline{CPU}$(s)	$\#bestub$	$\overline{BKS'\%}$
	26	26	0.15%	1.2	2.3	1352.4	10	0.59%

Table 4.8 Allowing unlimited visits to each recharging station: B&P algorithm results on Type-u instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$			Unlimited charging visits				At-most-one visit	
Instance	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
u2-16	**57.61***	57.61	0	1	1	69.5	**57.61***	0
u2-20	**55.59***	55.59	0	1	1	215.3	**55.59***	0
u2-24	**90.66***	90.66	0	1	1	7200.0	90.66*	0
u3-18	**50.74***	50.74	0	1	0	46.0	**50.74***	0
u3-24	**67.56***	67.56	0	1	1	109.6	**67.56***	0
u3-30	**76.75***	76.75	0	1	0	750.8	**76.75***	0
u3-36	**103.93***	103.93	0	1	2	4326.5	104.04*	0.11%
u4-16	**53.58***	53.58	0	1	0	6.5	**53.58***	0
u4-24	**89.83***	89.83	0	1	1	63.2	**89.83***	0
u4-32	**99.29***	99.29	0	1	1	416.4	**99.29***	0
u4-40	**133.11***	133.11	0	1	1	2586.1	**133.11***	0
u4-48	**147.33**	146.74	0.40%	1	2	7200.0	148.08	0.51%
u5-40	**121.86***	121.86	0	1	1	1591.2	**121.86***	0
u5-50	**142.82**	142.75	0.05%	1	1	7200.0	142.82	0
Avg			0.03%	1.0	0.93	2270.1		0.04%
$\gamma = 0.4$	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
u2-16	**57.65***	57.65	0	1	1	35.7	**57.65***	0
u2-20	**56.34***	56.34	0	1	1	329.9	**56.34***	0
u2-24	**90.84***	90.84	0	1	2	2408.5	91.27	0.47%
u3-18	**50.74***	50.74	0	1	1	62.6	**50.74***	0
u3-24	**67.56***	67.56	0	1	1	141.0	**67.56***	0
u3-30	**76.75***	76.75	0	1	1	1457.8	**76.75***	0
u3-36	**104.06***	104.06	0	1	1	3729.6	**104.06***	0
u4-16	**53.58***	53.58	0	1	0	7.1	**53.58***	0
u4-24	**89.83***	89.83	0	1	1	39.0	**89.83***	0
u4-32	**99.29***	99.29	0	1	1	438.3	**99.29***	0
u4-40	**133.36***	133.36	0	1	2	1581.5	133.78	0.31%
u4-48	**147.56**	146.97	0.40%	1	2	7200.0	147.63	0.05%
u5-40	**121.96***	121.96	0	1	1	1073.8	**121.96***	0
u5-50	**142.83***	142.83	0	1	1	6587.9	142.84	0.007%
Avg			0.03%	1.0	1.14	1927.5		0.06%
$\gamma = 0.7$	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
u2-16	**58.17***	58.17	0	1	2	172.2	59.19*	1.72%
u2-20	**56.86***	56.86	0	1	1	397.0	**56.86***	0
u2-24	**91.33***	91.33	0	1	2	5250.1	92.17	0.91%
u3-18	**50.99***	50.99	0	1	1	47.4	**50.99***	0
u3-24	**68.06***	68.06	0	1	2	558.9	68.44	0.48%
u3-30	**77.29***	77.29	0	1	2	807.9	77.41*	0.16%
u3-36	106.72	104.85	1.07%	1	2	7200.0	**106.50**	-0.88%
u4-16	**53.87***	53.87	0	1	1	16.7	**53.87***	0
u4-24	**89.83***	89.83	0	1	2	74.6	**89.96***	0.14%
u4-32	**99.50***	99.50	0	1	1	1600.3	**99.50***	0
u4-40	**134.38**	134.16	0.16%	45	3	7200.0	135.29	0.67%
u4-48	152.72	145.99	2.35%	1	2	7200.0	185.16	17.52%
u5-40	**123.00**	122.72	0.23%	1	1	7200.0	123.82	0.66%
u5-50	**142.89***	142.89	0	1	2	5628.9	144.36	1.02%
Avg			0.27%	4.1	1.71	3096.7		1.60%
Summary	$\#opt$	$\#bestub$	$\overline{LB\%}$	$\overline{N_{node}}$	$\overline{N^s_{max}}$	$\overline{CPU}$(s)	$\#bestub$	$\overline{BKS\%}$
	35	40	0.11%	2.0	1.3	2431.4	26	0.57%

Table 4.9 Allowing unlimited visits to each recharging station: B&P algorithm results on Type-r instances under $\gamma = 0.1, 0.4, 0.7$

$\gamma = 0.1$	Unlimited charging visits						At-most-one visit	
Instance	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
r5-60	**683.39***	683.39	0	1	0	13941.5	**683.39***	0
r6-48	**506.45***	506.45	0	1	0	1675.1	**506.45***	0
r6-60	**689.45***	689.45	0	9	3	3305.3	**689.45***	0
r6-72	761.91	740.87	2.76%	1	1	18000.0	**761.34**	-0.07%
r7-56	**612.02***	612.02	0	1	1	2452.9	**612.02***	0
r7-70	**754.28***	754.28	0	21	1	18000.0	**754.28**	0
r7-84	**870.21**	861.39	1.01%	1	1	18000.0	889.38	2.16%
r8-64	**632.22***	632.22	0	1	0	4005.4	**632.22***	0
r8-80	**788.99***	788.99	0	1	1	4883.0	**788.99**	0
r8-96	1098.12	*807.04*	NA	1	1	18000.0	**1053.11**	-4.27%
Avg			0.42%	3.8	0.9	10226.3		-0.22%
$\gamma = 0.4$	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
r5-60	**683.58***	683.58	0	1	3	14205.2	686.85	0.48%
r6-48	**506.45***	506.45	0	1	0	1225.8	**506.45***	0
r6-60	**689.46***	689.46	0	1	2	5133.0	689.48	0.003%
r6-72	**762.65**	756.44	0.81%	1	3	18000.0	NA	NA
r7-56	**612.02***	612.02	0	3	1	846.1	**612.02**	0
r7-70	**754.28***	754.28	0	19	1	18000.0	**754.28**	0
r7-84	**891.44**	864.14	3.06%	1	3	18000.0	1081.76	17.59%
r8-64	**632.22***	632.22	0	1	1	5757.1	**632.22***	0
r8-80	**788.99***	788.99	0	1	4	11757.7	**788.99**	0
r8-96	**1144.15**	*881.91*	NA	1	3	18000.0	NA	NA
Avg			0.39%	3.0	2.1	11853.9		1.81%
$\gamma = 0.7$	Obj	LB	$LB\%$	N_{node}	N^s_{max}	CPU(s)	BKS′	BKS′%
r5-60	**688.84**	684.03	0.70%	1	5	18000.0	NA	NA
r6-48	**508.10***	508.10	0	1	4	2027.7	NA	NA
r6-60	**689.55***	689.55	0	1	7	6604.7	NA	NA
r6-72	**788.83**	749.70	4.96%	1	4	18000.0	NA	NA
r7-56	**616.16***	616.16	0	59	7	14247.4	NA	NA
r7-70	**765.49**	754.14	1.48%	1	6	18000.0	NA	NA
r7-84	**1004.81**	*787.22*	NA	1	4	18000.0	NA	NA
r8-64	**632.61***	632.61	0	1	6	18000.0	NA	NA
r8-80	**794.41**	794.41	0	1	8	15051.3	NA	NA
r8-96	**1174.49**	*755.46*	NA	1	4	18000.0	NA	NA
Avg			0.81%	6.8	5.5	14593.1		NA
Summary	$\#opt$	$\#bestub$	$\overline{LB\%}$	$\overline{N_{node}}$	$\overline{N^s_{max}}$	$\overline{CPU}$(s)	$\#bestub$	$\overline{BKS\%}$
	18	28	0.54%	4.5	2.8	12224.4	14	0.79%

Table 4.10 Aggregated results of CG, B&P, and DA algorithm on type-a and -u instances

	CG results			B&P results			DA results	
γ values	$\#opt$	$\#bestub$	$\overline{CPU}$(s)	$\#opt$	$\#bestub$	$\overline{CPU}$(s)	$\#bestub$	$\overline{CPU}$(s)
$\gamma = 0.1$	21	25	1103.7	26	28	1484.1	22	164.9
$\gamma = 0.4$	20	25	1382.7	25	28	1645.4	22	249.0
$\gamma = 0.7$	9	18	2281.1	20	23	2896.5	26	445.5
Summary	50	68	1589.2	71	79	2008.7	70	286.5

Comparing our obtained results of CG, B&P, and DA algorithms on different instance sets, we have the following conclusions: (1) for small-to-medium-sized instances (i.e., type-a and -u instances), the B&P algorithm can obtain optimal solutions for most instances in a reasonable computational time, and it yields the highest number of best solutions. The DA and CG algorithms perform comparably well in terms of the number of best solutions found. However, the DA algorithm stands out with its notably short average computational time. As we tackle the static E-ADARP, all requests are pre-booked. When there is no urgency, the B&P algorithm can be applied to obtain minimum-cost solutions. In the case of urgency, the DA algorithm is more applicable to help decision-makers make prompt decisions while still obtaining high-quality solutions. (2) for large-scale instances, applying the DA algorithm is a more appropriate choice, as it provides high-quality solutions in a far less average computational time, compared to that of the CG and B&P algorithms.

4.5 Conclusion

This chapter has presented an effective CG algorithm relying on a problem-tailored labeling algorithm to solve the E-ADARP. The proposed CG algorithm is then integrated into the B&P scheme to yield optimal solutions for the E-ADARP. The E-ADARP is differentiated from the classical DARP and E-VRP regarding the following features: weighted-sum objectives, EAVs, partial recharging, multiple depots, final battery restriction, and limited visits to recharging stations. The weighted-sum objective function minimizes both total travel time and total excess user ride time. To solve the E-ADARP efficiently, the design of the labeling algorithm plays a key role in the overall performance. To guarantee the efficiency and accurateness of the labeling algorithm, we first introduce (1) a fragment-based representation of paths that replaces a sequence of REFs with a single one, then propose (2) a novel approach that abstracts fragments to arcs with excess user ride time being minimized to build a new sparser graph that preserves all feasible routes of the original graph and apply (3) strong dominance rules and constant time feasibility checks on the new graph.

In the computational experiments, we first demonstrate the performance of the CG algorithm on a large set of

benchmark instances and compare the obtained results with the best reported exact results in Bongiovanni et al. (2019). With our proposed CG algorithm, 66 out of 84 instances are solved optimally at the root node. We generate high-quality lower bounds with a 0.07% average deviation to the best-known solutions. We obtain 49 equal lower bounds and 29 better lower bounds than those reported in Bongiovanni et al. (2019). The overall quality of the lower bound is improved by 1.35% on average. In addition, we provide 25 new best solutions on previously solved and unsolved instances and identify 17 new optimal solutions. On type-r instances, we compare the CG results to the best-reported heuristic results of Su et al. (2023). We report 15 better solutions (10 of them are optimal) as well as 17 new lower bounds.

Then, the CG algorithm is integrated into the B&P framework, and we further solve 5 additional instances optimally, obtain 6 tighter lower bounds and generate 3 additional new best solutions. Benefiting from the good quality of lower bounds obtained at the root node, our B&P algorithm searches only a few nodes of the search tree to close the gaps. Compared with the best-reported B&C results in Bongiovanni et al. (2019), we finally solve 71 out of 84 instances optimally within the two-hour time limit, while the B&C algorithm can only solve 49 instances optimally. In addition, we obtain 26 new best solutions and 54 equal solutions and enhanced 30 lower bounds. The average computational time of our B&P algorithm decreases by 16% compared with that of the B&C algorithm. On larger-sized instances (i.e., type-r instances), we obtain 16 new best solutions, compared with the existing results of Su et al. (2023). Also, the proposed B&P algorithm can be easily adapted to handle other problem variants, such as the one with unlimited recharging station visits, whereas the B&C cannot. In this case, the adapted B&P algorithm improves 49 solutions compared to the best-obtained results of allowing at most one recharging visit per station. Therefore, the superiority of our B&P algorithm upon the existing exact method in the literature to solve the E-ADARP has been proved.

The success of the proposed CG algorithm can be attributed to several aspects. First, the CG-based subproblem formulation with all intra-route constraints enhances the quality of lower bounds. Second, the key to such stronger lower bounds is that we can compute them efficiently and exactly with the proposed labeling algorithm on the constructed new graph, where excess-user-ride-time optimality is guaranteed on each arc, and strong dominance rules are applied. Last, the application of cutting planes further improves the lower bounds and reduces the remaining integrality gaps to a large extent. As a result, our proposed CG algorithm solves the majority of instances optimally at the root node without branching.

Our CG and B&P algorithms also offer new insights into designing an exact algorithm for solving a practical version of the electric DARP, considering multiple objectives. One important “by-product” of our labeling algorithm is the first exact scheduling procedure that can efficiently determine the excess-user-ride-time optimal schedule for a given E-ADARP route. This scheduling procedure can also be applied to optimize excess user ride time in the classical DARP or the DARP with multiple objectives, in which total excess user ride time is minimized in a separate objective. The proposed efficient approaches can be adapted to tackle the dynamic DARP/E-ADARP, where new requests arrive in real time.

Chapter 5

Solving the Bi-objective Electric Autonomous Dial-A-Ride Problem

The development of ride-sharing services presents a fundamental trade-off between the operational costs and the users' convenience. While ride-sharing operations reduce operational costs, users may experience certain inconveniences such as longer ride times when sharing their rides with others. This trade-off is critical for service providers to consider when designing and implementing their services. On the one hand, ride-sharing can lead to significant cost savings through the sharing of resources such as drivers and vehicles. On the other hand, users may be less likely to use ride-sharing services if they perceive them as inconvenient or time-consuming. Therefore, ride-sharing companies must try to strike a balance between minimizing operational costs and providing a convenient and efficient service to their users.

This chapter is based on our conference paper "The bi-objective electric autonomous dial-a-ride problem" presented at Roadef conference, where some preliminary results were reported. These results compose the basis of this chapter and our forthcoming journal paper. In this chapter, we extend the E-ADARP to the Bi-objective E-ADARP (BO-EADARP), where the total travel time and the total excess user ride time are treated as separate objectives. We strive to identify the set of Pareto optimal solutions (also called "efficient solution") with regard to the two objectives. To solve the BO-EADARP, we consider both criterion space search algorithms (i.e., ϵ-constraint method and balanced box method) and decision space search algorithm (i.e., the generalized B&P). The structure of this chapter is outlined as follows. In Section 5.1, we present the basic notations and definitions that are used in the context of bi-objective optimization. Then, we present the criterion space search algorithms that are applied to solve the BO-EADARP in Section 5.2. In Section 5.3, we investigate the decision space search algorithm. Section 5.4 presents the computational results of solving the BO-EADARP with the developed algorithms and several practical implications obtained from efficient solutions. This chapter closes with conclusions and discussions in the last part.

5.1 Preliminaries

We replace the objective function (2.1) presented in Chapter 2 with Equation (5.1) and (5.2):

- Objective 1: minimize the total travel time

$$\min \sum_{i,j \in V} t_{i,j} x_{i,j}^k \tag{5.1}$$

- Objective 2: minimize the total excess user ride time

$$\min \sum_{i \in P} R_i \tag{5.2}$$

Without loss of generality, we present the objective of the BO-EADARP as:

$$\underset{x \in \mathcal{X}}{\text{minimize}}\, z(x) := \{z_1(x), \cdots, z_p(x)\} \tag{5.3}$$

where $\mathcal{X} \subset \mathbb{R}^n$ denotes the set of all feasible solutions in the decision space (also called *feasible set in the decision space*) and p is the number of objectives. $z(x)$ is the vector of objective functions, and we denote $\mathcal{Y}$ as the *feasible set in the criterion space*, defined by $\mathcal{Y} := \{z(x) : x \in \mathcal{X}\}$. As we solve the bi-objective case, we have $p = 2$ in Equation (5.3). To facilitate the presentation, we use $\mathbb{R}^p_{\geq} := \{y \in \mathbb{R}^p : y \geq 0\}$ to denote the non-negative orthant of $\mathbb{R}^p$. Similarly, we use $\mathbb{R}^p_{>} := \{y \in \mathbb{R}^p : y > 0\}$ to denote the positive orthant of $\mathbb{R}^p$. Finally, we use $conv(\mathcal{Y})$ to denote the convex hull of feasible set $\mathcal{Y}$ in the objective space. For the convenience of the organization, we introduce the following definitions from Przybylski et al. (2008).

Definition 6 (weakly efficient solution). A feasible solution $x' \in \mathcal{X}$ is called *weakly efficient* if there is no other $x \in \mathcal{X}$ such that $z_p(x) < z_p(x'), p = 1, 2$. If x' is weakly efficient, then $z(x')$ is called a *weakly non-dominated point*.

Definition 7 (efficient solution). A feasible solution $x^* \in \mathcal{X}$ is called *efficient* if there does not exist any other feasible solution $x \in X$ such that $z_p(x) \leqslant z_p(x^*), p = 1, 2$, with at least one strict inequality. $z(x^*)$ is called a *non-dominated* points. The set of efficient solutions is denoted as $\mathcal{X}_E$ and $\mathcal{Y}_N := \{z(x) : x \in \mathcal{X}_E\}$ is called the non-dominated frontier or efficient frontier.

After defining efficient solutions $\mathcal{X}_E$, we partition $\mathcal{X}_E$ into the following two sets:

Definition 8 (supported efficient solutions). A feasible solution x is called a *supported efficient solution* if x is an optimal solution of a weighted-sum objective problem:

$$\underset{x \in \mathcal{X}}{\text{minimize}}\, \lambda z_1(x) + (1 - \lambda) z_2(x) \tag{5.4}$$

where $1 > \lambda > 0$.

Definition 9 (non-supported efficient solutions). A feasible solution x is called a *non-supported efficient solution* if x is not optimal solution of a weighted-sum objective under any weight combinations.

We use $\mathcal{X}_{SE}$ to denoted the set of supported efficient solutions, and the image of $\mathcal{X}_{SE}$ in the objective space is called the set of *supported nondominated points*, denoted as $\mathcal{Y}_{SN}$. The set of non-supported efficient solutions is denoted as $\mathcal{X}_{NE}$ and its image in the objective space is called the set of *non-supported non-dominated points* and is denoted as $\mathcal{Y}_{NN}$. For the sake of readability, we show in Figure 5.1 an example for y_{SN} and y_{NN}, where y_{SN} is presented by solid red points and y_{NN} is presented by solid black points. Other hollow points are dominated points. Also, we use the red line to show that a supported efficient solution can be obtained by solving a weighted-sum objective problem.

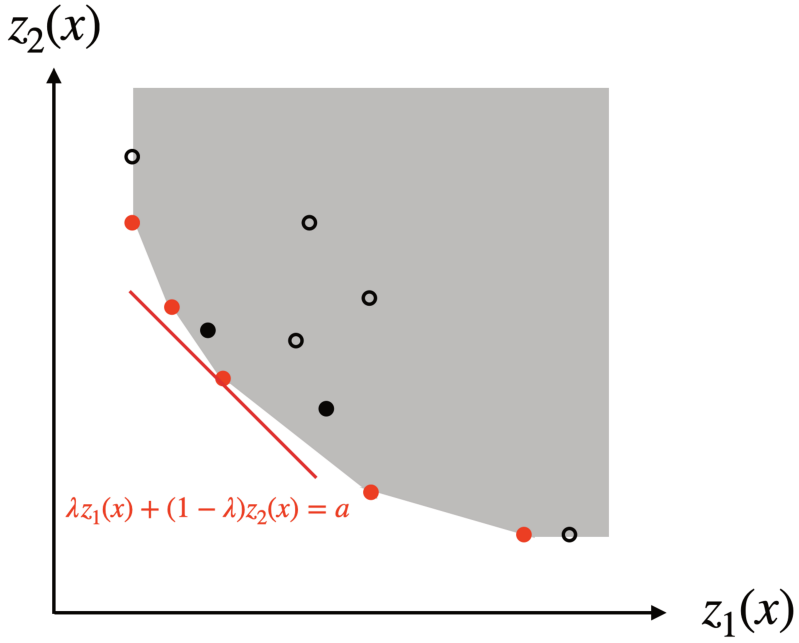


Figure 5.1 Illustration for y_{SN} and y_{NN}

Observation 1. All non-supported non-dominated points are located in the interior of $conv(Y + \mathbb{R}^p_{\leq})$. All supported non-dominated points are points of $conv(Y + \mathbb{R}^p_{\leq})$.

Set $\mathcal{Y}_{SN}$ can be further partitioned into set of *extreme supported non-dominated points*, denoted as $\mathcal{Y}_{SN1}$, corresponding to the set of extreme points of $conv(Y + \mathbb{R}^p_{\leq})$. Other points located in the relative interior are called the set of *non-extreme supported non-dominated points*, denoted as $\mathcal{Y}_{SN2}$.

Definition 10. A feasible solution $x \in \mathcal{X}$ is called *ideal* if $x \in arg\min_{x \in \mathcal{X}} z_i(x), \forall i = 1, 2$, i.e., it minimizes all objectives simultaneously. The corresponding point $z(x)$ in the objective space is called ideal point.

Definition 11. A feasible solution $x \in \mathcal{X}$ is called *nadir* if $x \in arg\max_{x \in \mathcal{X}} z_i(x), \forall i = 1, 2$, i.e., it maximizes all objectives simultaneously. The corresponding point $z(x)$ in the objective space is called nadir point.

5.2 Criterion Space Search Algorithms

In this section, we investigate two criterion space search algorithms that are proven efficient in obtaining non-dominated points and efficient solutions. One is the widely-used ϵ-constraint method (Haimes (1971)), which explores the criterion space by keeping one objective in the objective function and bounding the other with an ϵ value in the constraint. Taking advantage of the obtained efficient solutions, the ϵ values are continuously lowered until the problem become infeasible to solve. The other criterion space search method is the balanced box method, which was first introduced in Boland et al. (2015). The balanced box method searches the criterion space by iteratively exploring rectangle search areas defined by newly-found efficient points in the criterion space.

5.2.1 Epsilon-constraint Method

The mechanism of the ϵ-constraint method is illustrated in Figure 5.2, where the solid black points are obtained non-dominated points, the solid red point is the next non-dominated point, and hollow black points are non-dominated points that are not obtained.

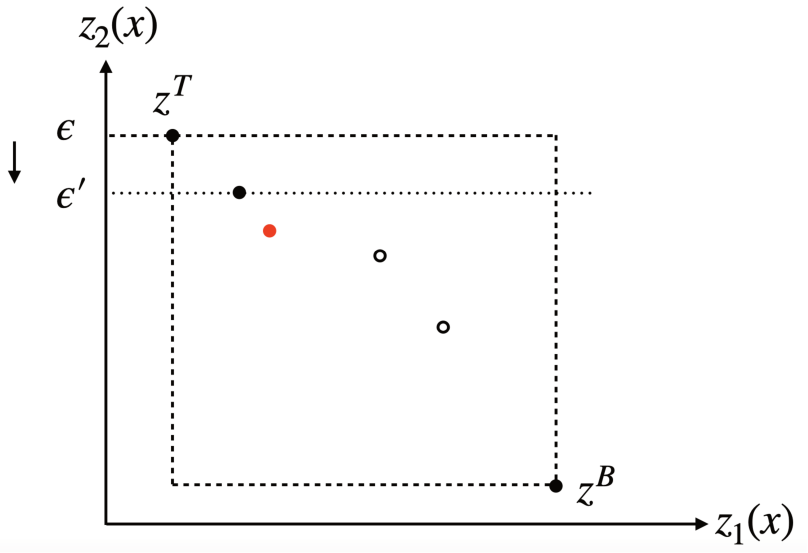


Figure 5.2 Applying the ϵ-constraint method to find non-dominated points

We start the program by solving two objectives in lexicographical order (i,e.,$lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$ and $lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$). The term $lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$ describes the process in which we find solutions with the smallest values for $z_2(x)$ among all feasible solutions with the smallest values for $z_1(x)$ and similar for $lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$. The obtained non-dominated points z^T and z^B define the search area where other non-dominated points are included. The ϵ-constraint method always optimizes one objective (e.g., $z_1(x)$) while the other is bounded by an ϵ value (i.e., $z_2(x) < \epsilon$). In each iteration, the ϵ value is updated with the $z_2(x')$, where x' is the newly-found non-dominated solution, as shown in Figure 5.2. By using the value of the other objective function to

restrict the search in an iterative way, all the non-dominated points are obtained. The ϵ-constraint method finishes when z^B is reached.

In the implementation of the ϵ-constraint method to solve the BO-EADARP, we set the total travel time as the objective and add the total excess user ride time to the MP formulation (as presented in Section 4.1). The MP is modified as follows and we denote the adapted MP as MP$^\epsilon$:

$$\min \sum_{\omega \in \Omega'} T_\omega y_\omega + \sum_{i \in P} P_i a_i \tag{5.5}$$

subject to:

$$\sum_{\omega \in \Omega} \theta_{i\omega} y_\omega \geqslant 1 - a_i, \forall i \in P \tag{5.6}$$

$$\sum_{\omega \in \Omega} \sum_{i \in P} R_{i\omega} y_\omega \leqslant \mathcal{E}_1 - \delta \tag{5.7}$$

$$\sum_{\omega \in \Omega} \phi_{f\omega} y_\omega \leqslant 1, \forall f \in S \cup F \tag{5.8}$$

$$\sum_{\omega \in \Omega} \epsilon_{o\omega} y_\omega \leqslant 1, \quad \forall o \in O \tag{5.9}$$

$$\sum_{\omega \in \Omega} y_\omega \leqslant |K| \tag{5.10}$$

$$y_\omega \in \{0, 1\}, \forall \omega \in \Omega \tag{5.11}$$

$$a_i \in \{0, 1\}, \forall i \in P \tag{5.12}$$

In the objective function, T_ω is the total travel time for route ω. The Constraint (5.7) is the constraint for excess user ride time. $R_{i,\omega}$ denotes the excess user ride time for customer i on route ω. The objective function of the pricing subproblem is:

$$c_\omega - \sum_{i \in P} \theta_{i,\omega} \pi_i - \sum_{f \in S \cup F} \phi_{f\omega} \tau_f - \sum_{o \in O} \epsilon_{o\omega} \zeta_o - \sigma - \sum_{i \in P} R_{i\omega} \kappa \tag{5.13}$$

where $\pi_i, i \in P$, $\tau_f, f \in S \cup F$, and $\zeta_o, o \in O$ are the dual variable of Constraints (5.6), (5.8), (5.9), respectively. Another dual variable associated with Constraint (5.10) is σ. The newly introduced dual variable by Constraint (5.7) is denoted as κ. For each single-objective problem (with different ϵ values), the B&P is applied. It should be mentioned that the ϵ-constraint method can also be implemented in the other direction. That is, considering the total excess

user ride time as the objective and bounding the total travel time with ϵ value. However, our preliminary experiments showed that the ϵ-constraint method in the other direction yields much poorer performance compared to considering the total travel time as objective. The reason is that solutions with different total travel times may have the same objective value (i.e., the same value of total excess user ride time). As a result, we must conduct several computations before reaching the one that has the minimum total travel time among the solutions that have the same value of total excess user ride time (i.e., the real non-dominated point), which introduces extensive computations. In contrast, using total travel time as the objective allows for more efficient and effective identification of non-dominated points, leading to better performance. Hence, we will only present the ϵ-constraint method that considers the total travel time as the objective in this chapter.

The framework of the ϵ-constraint method is described in Algorithm 3. We first solve $lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$, $lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$ to obtain z^T, z^B and ϵ value is initialized to z_2^T. The parameter δ is set to a sufficiently small value (e.g., 0.01) to obtain a full Pareto frontier. We apply an adapted version of the single-objective B&P algorithm presented in Chapter 4 to solve each single-objective problem with respect to minimizing the total travel time. Each time we obtain the optimal solution x^* with an ϵ value, we add it to the set of efficient solutions $\mathcal{X}_E$ and similarly for its corresponding point y^* in the criterion space.

Algorithm 3 ϵ-constraint method framework

Input: $\mathcal{X}_E \leftarrow \emptyset$, $\mathcal{Y}_N \leftarrow \emptyset$; $z^T \leftarrow lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$, $z^B \leftarrow lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$;
$\mathcal{X}_E \leftarrow \mathcal{X}_E \cup \{x^T, x^B\}$, $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{z^T, z^B\}$; $\epsilon \leftarrow z_2^T$.
Output: Set of non-dominated points $\mathcal{Y}_N$ and set of efficient solutions $\mathcal{X}_E$.
1: **while** the feasible region is not empty **do**
2: Solve MP$^\epsilon$ with the adapted B&P algorithm and obtain optimal solution x^*.
3: Compute the total excess ride time $z_2(x^*)$ and update $\epsilon \leftarrow z_2(x^*) - \delta$.
4: $\mathcal{X}_E \leftarrow \mathcal{X}_E \cup \{x^*\}$; $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{y^*\}$.
5: **end while**
6: **Return** $\mathcal{X}_E$ and $\mathcal{Y}_N$.

5.2.2 Balanced Box Method

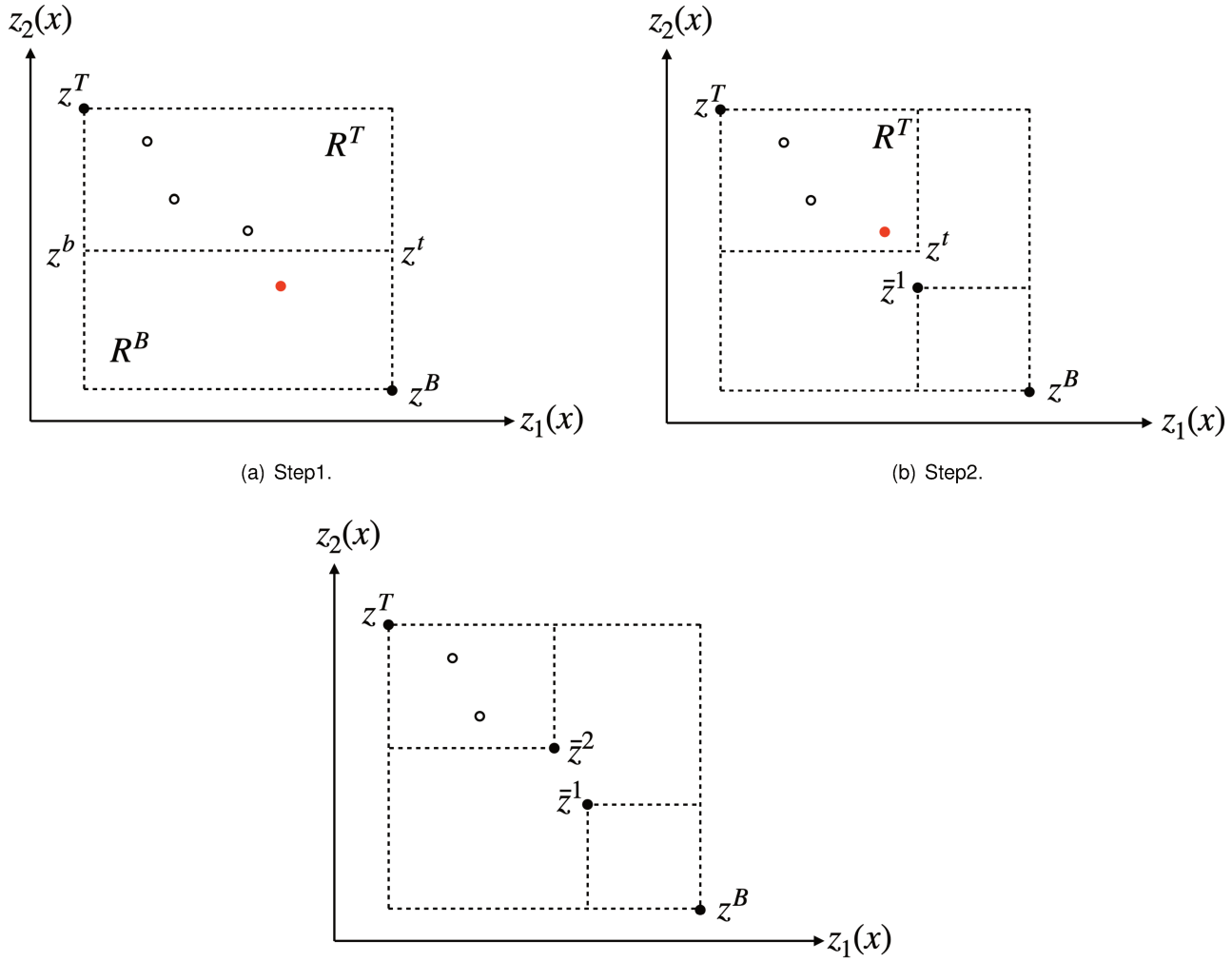
Similar to the ϵ-constraint method, the balanced box method also starts by solving two objectives in lexicographical order (i,e., $lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$ and $lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$) to define an initial rectangle. We keep the notations z^T and z^B to denote the obtained non-dominated points through the above processes. These two points define a rectangle, denoted as $R(z^T, z^B)$. The balanced box method computes non-dominated points recursively:

1. First, the rectangle $R(z^T, z^B)$ is split in the horizontal direction along $z_2(x)$ axis into two equal small rectangles R^T and R^B. These two small rectangles are defined by z^T and z^t, z^b and z^B, respectively, with $z^t = (z_1^B, (z_2^T + z_2^B)/2)$ and $z^b = (z_1^T, (z_2^T + z_2^B)/2)$. As shown in Figure 5.3(a), we define two rectangles R^T and R^B. The red solid point is the next non-dominated point that we will obtain in the next step, and the hollow black points are those that we have not obtained yet.

2. Then, rectangle R^B is searched to find the next non-dominated point by solving lexicographically $z_1(x)$ and $z_2(x)$, i.e., $lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x) : z(x) \in R^B\}$. The obtained solution $\bar{x}^1$ is an efficient solution, and its corresponding point $\bar{z}^1$ is a non-dominated point, as proved in Boland et al. (2015). This point is used to further partition the rectangle R^T, as some parts in R^T have been dominated by $\bar{z}^1$ (shown in Figure 5.3(b)). Then, we focus on finding the next non-dominated point (marked in red) in rectangle R^T.

3. Next, the rectangle R^T is explored by solving lexicographically $z_2(x)$ and $z_1(x)$, i.e., $lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x) : z(x) \in R^T\}$. Supposing the obtained solution is $\bar{x}^2$ and its corresponding point is $\bar{z}^2$ defines a new rectangle $R(z^T, \bar{z}^2)$ (as shown in Figure 5.3(c)).

The above process is repeated until non-unexplored rectangles remain, and we depict in Algorithm 4 the search process.



(a) Step1.

(b) Step2.

(c) Step3.

Figure 5.3 Illustration for balanced box method

Algorithm 4 Balanced box method framework

Input: $\mathcal{X}_E \leftarrow \emptyset$, $\mathcal{Y}_N \leftarrow \emptyset$, $Rec \leftarrow \emptyset$, $z^T \leftarrow lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x)\}$, $z^B \leftarrow lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x)\}$;
$\mathcal{X}_E \leftarrow \mathcal{X}_E \cup \{x^T, x^B\}$, $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{z^T, z^B\}$, $Rec \leftarrow Rec \cup R(z^T, z^B)$.
Output: Set of non-dominated points $\mathcal{Y}_N$ and set of efficient solutions $\mathcal{X}_E$.

while $Rec \neq \emptyset$ **do**
  Extract the first rectangle $R(z^1, z^2)$ from Rec, $Rec \leftarrow Rec \setminus R(z^1, z^2)$.
  $R^B \leftarrow ((z_1^1, (z_2^1 + z_2^2)/2), z^2)$.
  $\bar{z}^1 \leftarrow lex\min_{x\in\mathcal{X}}\{z_1(x), z_2(x) : z(x) \in R^B\}$.
  if $\bar{z}^1 \neq z^2$ **then**
    $\mathcal{X}_E \leftarrow \mathcal{X}_E \cup \{\bar{x}^1\}$, $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{\bar{z}^1\}$.
    $Rec \leftarrow R(\bar{z}^1, z^2)$.
  end if
  $R^T \leftarrow R(z^1, (\bar{z}_1^1 - \delta, (z_2^1 + z_2^2)/2))$.
  $\bar{z}^2 \leftarrow lex\min_{x\in\mathcal{X}}\{z_2(x), z_1(x) : z(x) \in R^T\}$.
  if $\bar{z}^2 \neq z^1$ **then**
    $\mathcal{X}_E \leftarrow \mathcal{X}_E \cup \{\bar{x}^2\}$, $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{\bar{z}^2\}$.
    $Rec \leftarrow R(z^1, \bar{z}^2)$.
  end if
end while
Return $\mathcal{X}_E$ and $\mathcal{Y}_N$.

5.3 Decision Space Search Algorithm

We extend the B&P algorithm developed in Chapter 4 and the BIOBAB algorithm developed in Parragh & Tricoire (2019) to handle the BO-EADARP. Most ingredients of the proposed algorithm are introduced from Parragh & Tricoire (2019), while several enhancements are considered:

1. We apply a problem-tailored and effective CG algorithm (as presented in Chapter 4) to compute the lower bound set at each node of the B&P tree;

2. We consider a new case (case 2 presented in Section 5.3.2) to further enhance the lower bound set filtering process.

3. We propose a new branching strategy (as presented in Section 5.3.3) to enhance the subproblem formulation at some nodes of the B&P tree. This branching strategy is applied jointly with state-of-the-art branching strategies.

The principle of the bi-objective B&P is similar to the single-objective one, which aims to divide the original problem into easier subproblems and store them in the form of *"nodes"*. We denote each subproblem of the BO-EADARP as $P(\eta)$, where η represents the associated node. However, bi-objective B&P is different from the single-objective case as lower bound and upper bound sets (instead of single numerical values) are used to decide whether to fathom a node. The main ingredients of the bi-objective B&P are presented as follows:

- **Calculate lower bound set and update upper bound set:** On each branch-and-bound node, we calculate the lower bound set with the dichotomic method proposed in Aneja & Nair (1979). To solve each weighted-sum objective problem, our CG algorithm is applied. Once the lower bound set of the analyzed node η (denoted as

$\mathcal{L}(\eta)$) is calculated, we first check if new non-dominated points are obtained. If it is the case, the upper bound set $\mathcal{U}$ is updated.

- **Lower bound filtering and node fathoming:** Then, the lower bounds in the set are filtered with the current upper bound set $\mathcal{U}$, which stores each candidate point that corresponds to the integer solution that is not dominated by other points in the set. The *filtering* process compares the current $\mathcal{L}(\eta)$ with $\mathcal{U}$ and returns a set of non-dominated portions. If no portion is generated after the filtering process, then the analyzed node η can be fathomed, as it is fully dominated by the current upper bound set $\mathcal{U}$.

- **Branching procedure:** If the analyzed node cannot be fathomed, branching is applied to generate child nodes. We considered three kinds of branching strategies and apply them to each disjoint non-dominated portion. After branching, a set of child nodes are added to the unprocessed node set $\mathcal{T}$.

The tree search terminates when there is no unprocessed node remaining in $\mathcal{T}$, and we have the set of non-dominated points $\mathcal{Y}_N$ equals to $\mathcal{U}$. The outline of the above process is shown in Algorithm 5.

Algorithm 5 Bi-objective Branch-and-Price (BOBP) Algorithm

Input: Calculate the lower bound set of the root node $\mathcal{L}(\eta_0)$; Set $\mathcal{T} \leftarrow \{\eta_0\}$ and update $\mathcal{U}$.
Output: Set of non-dominated points $\mathcal{Y}_N$.

while $\mathcal{T} \neq \emptyset$ **do**
  Select a node η from $\mathcal{T}$ and update $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\eta\}$.
  Apply dichotomic method calculate $\mathcal{L}(\eta)$ for $P(\eta)$.
  Check if new non-dominated points are obtained and update $\mathcal{U}$.
  Filter $\mathcal{L}(\eta)$ with $\mathcal{U}$.
  if η cannot be fathomed **then**
    Split $P(\eta)$ into disjoint subproblems $P(\eta^1), \cdots, P(\eta^h)$.
    Store each subproblem in a unique child node of η and add them to $\mathcal{T}$.
  end if
end while
Return $\mathcal{U}$

The following parts of this section are organized as follows. We first present the update process of the upper bound set and then introduce the notion of lower bound segments, as in Parragh & Tricoire (2019). By defining lower bound segments (abbreviated as LB segments), each LB segment included in $\mathcal{L}(\eta)$ at node η is compared to all the points in $\mathcal{U}$ and dominated LB segments are removed. Then, the filtered $\mathcal{L}(\eta)$ is served as input to the branching procedure, which includes the objective branching concept proposed in Parragh & Tricoire (2019) and other branching rules.

5.3.1 Updating upper bound set

The first step after calculating lower bound set $\mathcal{L}(\eta)$ at a node η is to check if new integer solutions are obtained. Each time an integer solution is obtained, the upper bound set $\mathcal{U}$ is updated. This process must be conducted before

filtering and branching, as the newly identified non-dominated points can help to further restrict the search area. One of the three cases can happen when we obtain an integer solution x.

1. Solution x is dominated by solutions corresponding to points in the current upper bound set $\mathcal{U}$. In this case, x is discarded.

2. Solution x is not dominated by any solutions corresponding to points in the current upper bound set $\mathcal{U}$. In this case, x is added to the set of efficient solutions, and its corresponding point is added to $\mathcal{U}$. Moreover, if solution x dominates some solutions corresponding to points in $\mathcal{U}$, then dominated points are removed from $\mathcal{U}$.

5.3.2 Lower bound set filtering and node fathoming

One of the main challenges in designing an effective bi-objective B&B algorithm is to evaluate the obtained lower bound set in comparison with the current upper bound set and eliminate search areas that will not contain any non-dominated solutions. In our BOBP algorithm, we obtain the lower bound set $\mathcal{L}(\eta)$ for a given node η (associated with problem $P(\eta)$) by solving weighted-sum problems in a dichotomic scheme (Aneja & Nair (1979)). To solve each weighted-sum problem, we employ the CG algorithm, which is presented in Chapter 4. Once we have obtained the lower bound set $\mathcal{L}(\eta)$, we sort the points in $\mathcal{L}(\eta)$ in increasing order of the $z_1(x)$ axis. Any two consecutive points in the sorted $\mathcal{L}(\eta)$ accompanying a valid local nadir point define a search area where new non-dominated points may exist (as shown in Figure 5.4). A valid local nadir point can be obtained by simply taking large enough values for both objectives as coordinates. In the implementation, we take the maximum values of both objectives among all the extreme supported points that we obtain when applying the dichotomic method at the root node. The sorted $\mathcal{L}(\eta)$ is then filtered by the current upper bound set $\mathcal{U}$. Assuming Ξ be the set of all points in the objective space that correspond to feasible solutions of $P(\eta)$. The filtering process is based on the concept of the LB segment, which was first introduced in Parragh & Tricoire (2019).

Definition 12 (LB segments defined in Parragh & Tricoire (2019)). Assuming there are two points $p = (p_1, p_2)$, $q = (q_1, q_2)$, and a local nadir $c = (c_1, c_2)$, such that $p_1 < q_1$, $p_2 > q_2$ and $c_1 > q_1$, $c_2 > p_2$. These three points define a segment s if and only if $\{(z_1, z_2) \in \Xi | z_2 < az_1 + b\} = \emptyset$, where $a = (q_2 - p_2)/(q_1 - p_1)$ presents the slope of the line connecting p and q and $b = p_2 - a * p_1$ is the intercept of this line on z_2-axis.

An example of a LB segment is shown in Figure 5.4. The Definition 12 is extended to Definition 13 to include the case where p and q are the same points in the objective space.

Definition 13. Two points $p = (p_1, p_2)$ and $c = (c_1, c_2)$ define a *lower bound segment* s if and only if the following conditions hold: (1) there does not exist a feasible point z that dominates p, (2) c is a valid local nadir point, such that $c_1 \geqslant p_1$ and $c_2 \geqslant p_2$.

With the notion of LB segment, the dominance between a given lower bound set $\mathcal{L}(\eta)$ and the current upper bound set $\mathcal{U}$ can be evaluated by splitting $\mathcal{L}(\eta)$ into LB segments and filter each LB segment with each point of $\mathcal{U}$.

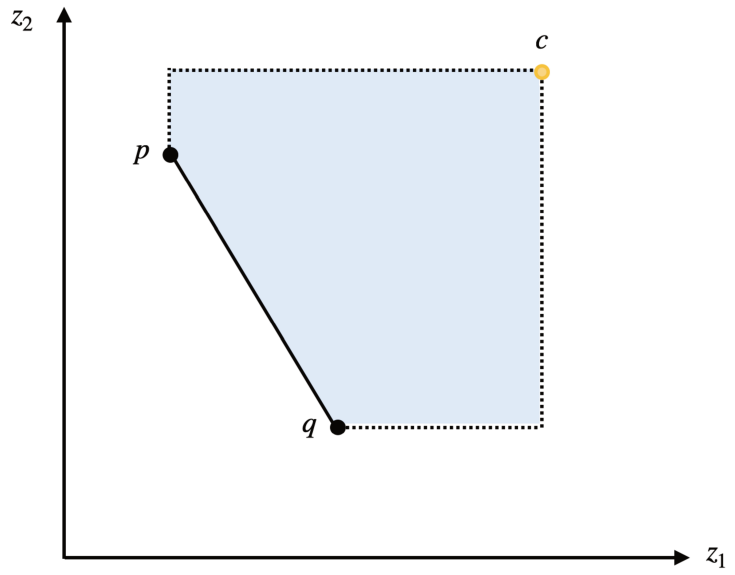


Figure 5.4 Example of a LB segment that defines a search area where non-dominated points may exist

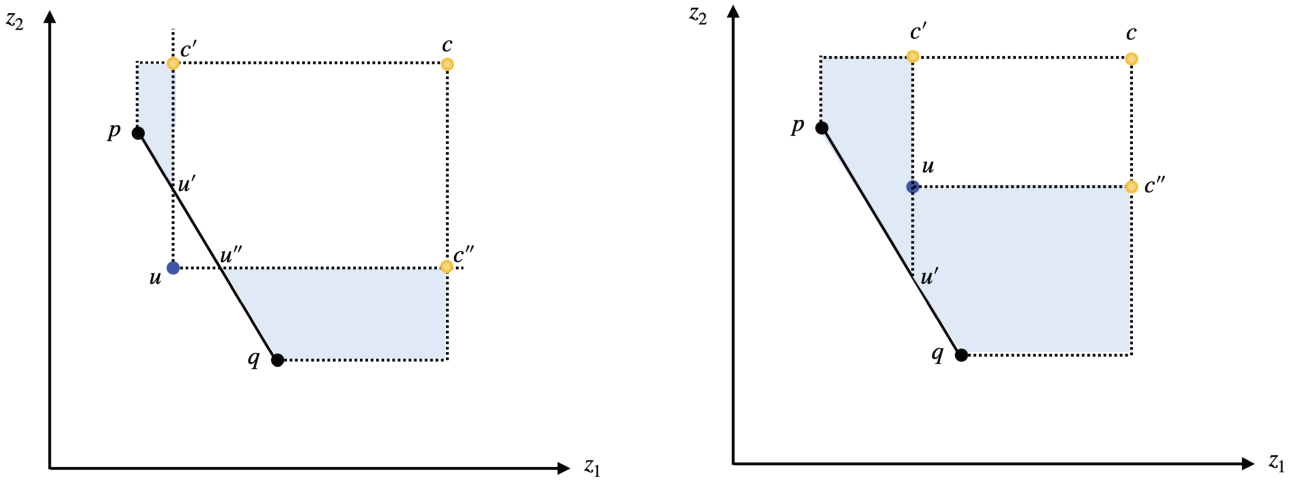
If each LB segment in $\mathcal{L}(\eta)$ is dominated by $\mathcal{U}$, then node η is fathomed. The dominance between a LB segment s with a point $u \in \mathcal{U}$ is checked by subtracting the space that is dominated by point u from the defined space by s. Assuming that the slope and z_2-intercept of segment s is denoted as a and b, there exist three cases:

1. point u locates under segment s, i.e., $u_2 < a \cdot u_1 + b$, as shown in Figure 5.5(a);
2. point u locates upon segment s, i.e., $u_2 \geqslant a \cdot u_1 + b$, as shown in Figure 5.5(b);
3. point u locates in other areas, i.e., $(u_2 \geqslant p_2) \wedge (u_1 \leqslant q_2)$ or $(u_1 \geqslant q_1) \wedge (u_2 \leqslant p_1)$, as shown in Figure 5.6(a) and 5.6(b), respectively.

In the first case, u locates under the segment defined by point p, q, and c. As point u dominates some portions in the search area defined by the segment, the dominated portion must be subtracted. As a result, two new segments are generated. The first segment is defined by point p, u', and c', where $u' = (u_1, a \cdot u_1 + b)$ and $c' = (u_1, c_2)$ and the second segment is defined by points u'', q, and c'', where $u'' = ((u_2 - b)/a, u_2)$, $c'' = (c_1, u_2)$. This case is also discussed in Parragh & Tricoire (2019), and the authors conduct the same operation to generate new segments.

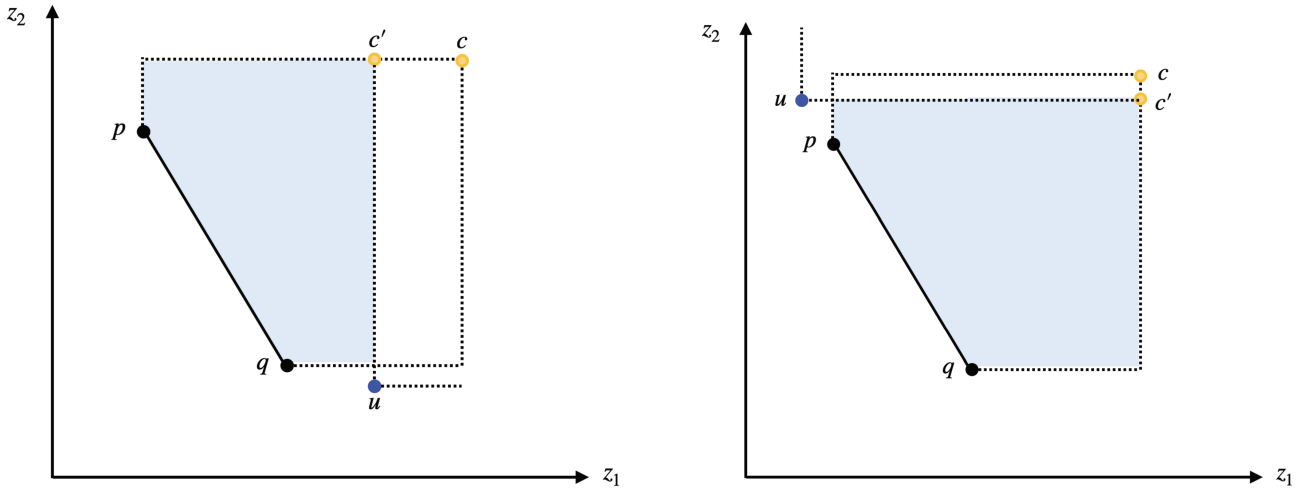
In the second case, u does not dominate any point on the line connecting point p and q, while it still dominates some portions in the defined search area. Similarly, two new segments are generated, which are defined by points p, u', c', and points u', q, and c'', respectively. It should be noted that this case has not been discussed yet in the literature.

The third case follows the idea in Parragh & Tricoire (2019), and we further improve the local nadir point by taking advantage of point u. In the case of point u locating in the right-downside of point q, the local nadir point c is moved leftward to point c'. Similarly, if point u locates in the left-upside of point p, the local nadir point can also be enhanced.



(a) Case 1. (b) Case 2.

Figure 5.5 Checking the dominance between segment s (defined by p, q, c) and a point $u \in \mathcal{U}$ for case 1 and 2



(a) Case 2.1. (b) Case 2.2.

Figure 5.6 Checking the dominance between segment s (defined by p, q, c) and a point $u \in \mathcal{U}$ for case 3

It should be noted that if there are still non-dominated portions generated after filtering the lower bound set, the node cannot be fathomed, and branching is called. In the case that no portion can be generated after filtering the lower bound set, the node is fathomed.

5.3.3 Branching procedure

After filtering the obtained lower bound set $\mathcal{L}(\eta)$ at a node η, if this node cannot be fathomed, we branch on η to generate new child nodes. We consider three kinds of branching strategies for a given LB segment: (1) decision space branching, (2) integer branching, and (3) objective space branching. The first and the third branching strategies are adapted from literature (Parragh & Tricoire (2019); Forget & Parragh (2022)), and the second one is a newly-

introduced branching strategy. In the following part, we first explain the first and the second branching strategy, as they both involve analyzing the obtained solutions. Then, we explain the third branching strategy, which is applied to each non-dominated portion. To explain, we assume that s is a LB segment resulting from the filtering procedure and p, q are two endpoints of s.

1. The first strategy is applied when at least one point between point p and q corresponds to a fractional solution in the decision space. Here we discuss two cases:

   - if only one point between p and q corresponds to a fractional solution, then we consider to branch on (a) the total number of vehicles, (b) the total flow of an arc.

   - if both p and q correspond to fractional solutions, then we choose to branch on the arc with the "most often fractional" flow among two fractional solutions. To select such an arc to branch on, we record the total flows of each arc and the number of times the arc has fractional flows. Then, among those who have the highest number of times being traversed by fractional flow, we select the arc with a fractional flow closest to 0.5. The advantage of branching on the "most often fractional" flow variable is that we could avoid repeated computations resulting from branching on a flow variable that is fractional in one solution and integer in another. Therefore, it helps us to move towards integer solutions more quickly.

2. If p and q both correspond to integer solutions, the second branching strategy is called. This case may happen when we obtain new non-dominated points in calculating lower bounds in the previous iterations. In this case, the first branching strategy will not work. Rather than randomly selecting an arc to branch, we select an arc that has different values of total flow to branch, i.e., we branch on an arc (i, j) that is contained in one solution (denoted as sol_1) but is not contained in the other (denoted as sol_2). After branching, we create two child nodes: one with constraint $x_{i,j} = 0$ and the other with constraint $x_{i,j} = 1$. This idea ensures that we can always restrict the search area and move toward other integer solutions under any situation. The drawback of this branching strategy is that when we calculate the lower bound set for the first child node (i.e., the one with $x_{i,j} = 0$), it is unavoidable to obtain sol_2, as it already satisfies $x_{i,j} = 0$, and similarly when calculating the lower bound set for the second child node.

3. The third branching strategy is adopted from Parragh & Tricoire (2019) and is called "objective space branching". When applying objective space branching, we consider the local nadir point (denoted as $c = (c_1, c_2)$) associated with the analyzed LB segment and add two cutting planes to restrict the two objectives of the generated solutions being better than c_1 and c_2. In other words, two constraints are added to the continuous MP formulation: $z_1(x) \leqslant c_1$ and $z_2(x) \leqslant c_2$.

5.4 Computational Results

In this section, we conduct experiments on small-to-medium-sized instances under different minimum battery level restrictions (i.e., $\gamma = 0.1, 0.4, 0.7$). In Section 5.4.1, we present the analyzed instances in the experiments and summarize the abbreviations and their meanings in Tables. Section 5.4.2 illustrates the implementation details of the ϵ-constraint method, the balanced box method, and the BOBP algorithm. Then, we compare the performance of different algorithms with regard to their computational efficiency and the number of obtained efficient solutions on each considered instance in Section 5.4.3. Finally, we analyze the impacts of battery restrictions on obtained efficient solutions and summarize the managerial insights for different service providers in Section 5.4.4. All algorithms are coded in Julia 1.7.2 and are performed on a standard PC with an Intel Xeon Gold 6230 20C at 2.1GHz.

5.4.1 Analyzed benchmark instances and abbreviations in tables

Due to the complexity of the BO-EADARP, we only consider small-to-medium-sized type-a instances in the numerical studies. The instances we considered in this section are instances with pattern "$ak - n - \gamma$", where $k = 2, 3, 4$, $n = 16, 18, 20, 24$, $\gamma = 0.1, 0.4, 0.7$. Overall, 18 instances are analyzed.

The meaning of the abbreviations in the tables is as follows:

1. $c^1_\gamma(x)$ and $c^2_\gamma(x)$: $z_1(x)$ and $z_2(x)$ (i.e., the total travel time and total excess user ride time) values for the obtained solution x under the analyzed γ value, where $\gamma \in \{0.1, 0.4, 0.7\}$.
2. CPU$_\gamma$: the CPU time in seconds of solving the corresponding problem under the analyzed setting of γ, $\gamma \in \{0.1, 0.4, 0.7\}$.
3. $\mathcal{X}_E^\gamma$: the set of efficient solutions under the analyzed setting of γ.
4. $|\mathcal{X}_E^\gamma|$: the number of efficient solutions in $\mathcal{X}_E$ on an instance under the analyzed setting of γ, $\gamma \in \{0.1, 0.4, 0.7\}$.
5. Avg^γ_{sol}: The average CPU time in seconds for obtaining an efficient solution under the analyzed setting of γ, $\gamma \in \{0.1, 0.4, 0.7\}$.

5.4.2 Implementation details

For the sake of clarity, we present the implementation details of each algorithm in this part. The maximum run time for each algorithm to solve each instance is set to 5 hours. As for the ϵ-constraint and balanced box methods (presented in Section 5.2.1 and 5.2.2), we call the single-objective B&P algorithm (presented in Chapter 4) to solve each single-objective problem. The time limit for applying the B&P algorithm is set to 2 hours. The B&P algorithm terminates when the maximum time limit is reached. As for the BOBP algorithm (presented in Section 5.3), the maximum number of processed nodes allowed in the bi-objective B&P tree is set to 200. In case of the maximum

number of processed nodes or a five-hours time limit is reached, the tree search is finished. It should be noted that the final reported computational time may exceed five hours, as the computational time is updated after solving single-objective problems with the B&P algorithm or a bi-objective B&P tree node in each iteration. Since we allow the single-objective B&P and CG algorithms to spend two hours to solve single-objective problems or calculate lower bound sets, the total computational time may exceed the maximum time limit for some instances.

5.4.3 Compare the performances of three algorithms

In this section, we compare the performance of three different algorithms. Table 5.1 summarizes the computational time CPU$_\gamma$, the average CPU time to obtain each efficient solution Avg^{γ}_{sol}, and the total number of obtained efficient solutions $|\mathcal{X}^{\gamma}_{E}|$ by applying three different methods (i.e., the ϵ-constraint, balanced box, BOBP algorithms) on each considered instance under three different settings of γ.

Table 5.1 Aggregated results of three considered algorithms

	ϵ-constraint			Balanced box			BOBP algorithm		
Instances	$CPU_{0.1}$	$Avg^{0.1}_{sol}$	$\lvert\mathcal{X}^{0.1}_{E}\rvert$	$CPU_{0.1}$	$Avg^{0.1}_{sol}$	$\lvert\mathcal{X}^{0.1}_{E}\rvert$	$CPU_{0.1}$	$Avg^{0.1}_{sol}$	$\lvert\mathcal{X}^{0.1}_{E}\rvert$
a2-16-0.1	18423.8	1674.9	11	19988.5	1537.6	13	2724.5	194.61	14
a2-20-0.1	23865.5	3977.6	6	26640.7	4440.1	6	18034.0	1502.8	12
a3-18-0.1	12521.3	834.8	14	19788.7	1522.2	13	4860.4	347.2	14
a3-24-0.1	25215.6	5043.1	5	25151.4	4191.9	6	18978.0	1725.3	11
a4-16-0.1	20524.6	6841.5	3	22296.3	3716.1	6	1276.4	141.8	9
a4-24-0.1	23686.7	4737.4	5	25084.3	4180.7	6	18016.5	948.2	19
Avg	20706.3	3851.5	7.3	23158.3	3264.8	8.3	10648.3	810.0	13.1
Instances	$CPU_{0.4}$	$Avg^{0.4}_{sol}$	$\lvert\mathcal{X}^{0.4}_{E}\rvert$	$CPU_{0.4}$	$Avg^{0.4}_{sol}$	$\lvert\mathcal{X}^{0.4}_{E}\rvert$	$CPU_{0.4}$	$Avg^{0.4}_{sol}$	$\lvert\mathcal{X}^{0.4}_{E}\rvert$
a2-16-0.4	21650.0	2405.6	9	22703.7	2064.0	11	22279.4	1591.4	14
a2-20-0.4	29573.6	5914.7	5	19991.6	4997.9	4	18092.8	1809.3	10
a3-18-0.4	9848.4	656.6	14	20052.7	2005.3	10	6935.5	495.4	14
a3-24-0.4	28504.2	4750.7	6	28447.9	4741.3	6	18094.8	2010.5	9
a4-16-0.4	24124.8	4020.8	6	29201.2	4866.9	6	1035.4	115.1	9
a4-24-0.4	30366.3	6073.3	5	26241.6	4373.6	6	18154.3	1067.9	17
Avg	24011.2	3970.3	7.5	24439.8	3841.5	7.2	14098.7	1181.6	12.1
Instances	$CPU_{0.7}$	$Avg^{0.7}_{sol}$	$\lvert\mathcal{X}^{0.7}_{E}\rvert$	$CPU_{0.7}$	$Avg^{0.7}_{sol}$	$\lvert\mathcal{X}^{0.7}_{E}\rvert$	$CPU_{0.7}$	$Avg^{0.7}_{sol}$	$\lvert\mathcal{X}^{0.7}_{E}\rvert$
a2-16-0.7	21000.6	2625.1	8	18220.1	3644.0	5	6293.8	699.3	9
a2-20-0.7	47612.6	9522.5	5	18782.4	9391.2	2	28112.5	14056.3	2
a3-18-0.7	23090.7	1649.3	14	18138.1	1648.9	11	4952.4	412.7	12
a3-24-0.7	24416.7	4883.3	5	30476.1	5079.4	6	20016.4	2859.5	7
a4-16-0.7	23532.4	5883.1	4	32173.6	4021.7	8	1315.1	164.4	8
a4-24-0.7	29257.9	5851.6	5	18218.6	4554.7	4	18362.0	2623.2	7
Avg	28151.8	5069.2	6.8	22668.2	4723.3	6.0	13175.4	3469.2	7.5

From Table 5.1, we have the following conclusions: (1) with an increasing value of γ, the average computational time of three algorithms to solve each instance grows significantly. We also observe a decrease in the average number of generated efficient solutions. (2) among the three analyzed algorithms, the BOBP algorithm generates the highest number of efficient solutions, while the balanced box method generates the least. (3) The BOBP algorithm

has the shortest average computational time for solving each instance and spends the least time in obtaining each efficient solution. (4) The balanced box method has the worst performance among the three algorithms on considered instances, as it takes a similar computational time as the ϵ-constraint method, but it generates the least efficient solutions on average.

5.4.4 Analyze the obtained efficient solutions under different battery restrictions

One important benefit of using the bi-objective approach is that the obtained efficient solutions can provide a clear picture of the compromises that need to be made when optimizing one objective at the expense of another. Also, efficient solutions can help decision-makers to make more informed and reliable choices. In this part, we analyze the obtained efficient solutions under different settings of γ for each instance, majorly from a management perspective. For the sake of explanation, we take an example with the results of instance a2-16 under $\gamma = 0.1, 0.4, 0.7$, as we generated complete sets of efficient solutions on this instance (results are shown in Table 5.2). We also visualize the efficient solutions we obtained from the ϵ-constraint method on instance a2-16 under $\gamma = 0.1, 0.4, 0.7$, as shown in Figure 5.7.

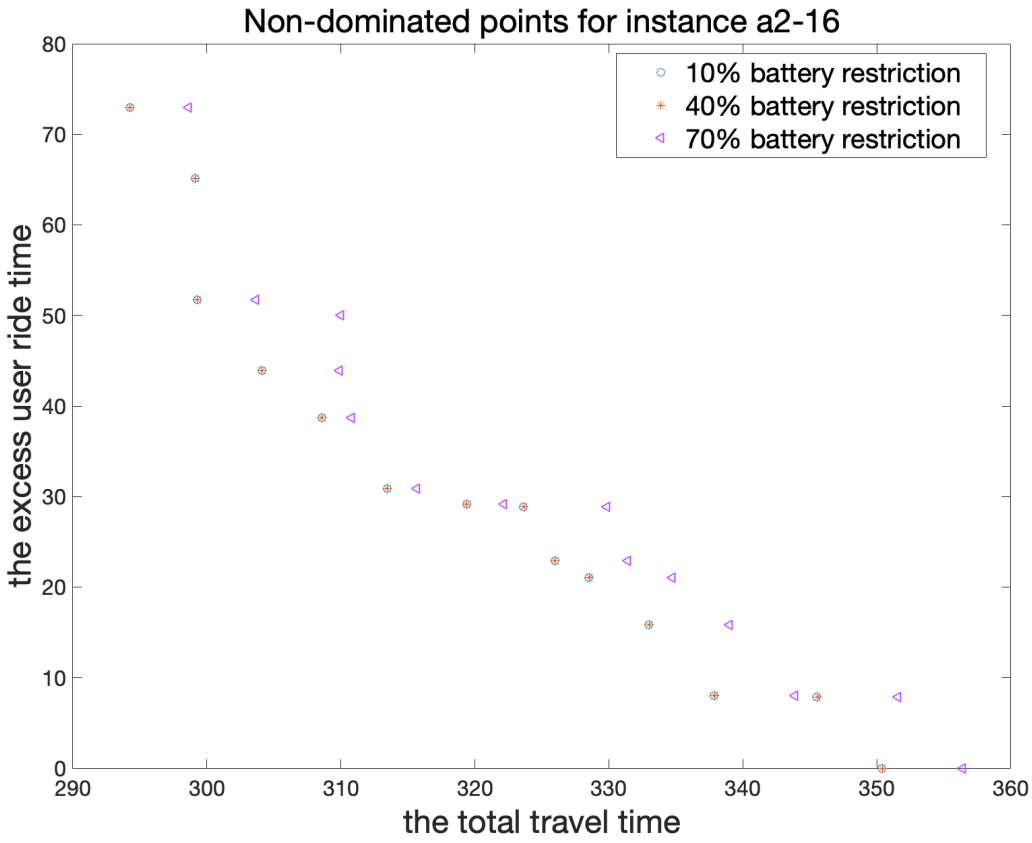


Figure 5.7 Non-dominated points obtained for instance a2-16 under $\gamma = 0.1, 0.4, 0.7$

From the efficient solutions we obtained under different γ settings, we have the following observations:

1. When increasing γ from 0.4 to 0.7, at the same level of total excess user ride time, we find an obvious increase in the total travel time of the obtained efficient solutions. This is consistent with the fact that vehicles need to make detours to recharging stations more frequently in order to satisfy minimum-battery-level constraints at the destination depot. As a vehicle performs recharging only when there is no passenger on board, then it will not introduce extra excess user ride time. Consequently, the efficient solutions obtained under $\gamma = 0.7$ has increased total travel times, while the total excess user ride times remain almost the same as in the case of $\gamma = 0.1, 0.4$.

2. In the case of $\gamma = 0.1, 0.4$, most of the obtained non-dominated points are the same. Therefore, we may potentially improve the computational efficiency of the considered algorithm (e.g., the ϵ-constraint algorithm) by feeding the set of efficient solutions $\mathcal{X}_E^\gamma$ obtained under lower values of γ (e.g., $\gamma = 0.1$) into the computations of efficient solutions under higher γ values ($\gamma = 0.4$).

Table 5.2 Non-dominated points for instance a2-16 under $\gamma = 0.1, 0.4, 0.7$

a2-16	$c_{0.1}^1(x)$	$c_{0.1}^2(x)$	CPU$_{0.1}$(s)	$c_{0.4}^1(x)$	$c_{0.4}^2(x)$	CPU$_{0.4}$(s)	$c_{0.7}^1(x)$	$c_{0.7}^2(x)$	CPU$_{0.7}$(s)
	294.25	72.98	8.1	294.25	72.98	9.2	298.63	72.98	31.9
	299.11	65.15	54.3	299.11	65.15	308.2	303.64	51.73	1504.0
	299.26	51.73	32.9	299.26	51.73	61.8	309.97	50.02	1830.5
	304.12	43.90	62.7	304.12	43.90	222.0	309.90	43.90	1673.1
	308.60	38.72	271.6	308.60	38.72	189.9	310.80	38.72	712.9
	313.46	30.89	1047.2	313.46	30.89	260.0	315.66	30.89	1317.9
	319.38	29.19	616.5	319.38	29.19	825.9	322.15	29.19	847.8
	323.64	28.85	424.9	323.67	28.85	389.7	329.87	28.85	3528.3
	333.33	21.18	1270.2	325.99	22.88	807.8	331.39	22.88	1073.6
	328.51	21.02	374.2	328.54	21.02	1065.8	334.74	21.02	675.6
	332.98	15.84	353.0	333.05	15.84	548.6	339.02	15.84	1338.6
	337.84	8.01	432.6	337.91	8.01	992.0	343.89	8.01	1325.7
	345.51	7.83	157.4	345.58	7.83	106.4	351.56	7.83	295.4
	350.38	0.00	414.7	350.44	0.00	197.0	356.42	0.00	482.7

From the efficient solutions we obtained at each γ setting, several managerial insights are offered:

1. from the non-dominated points located in the left-upside of the Pareto front, we find that the total excess user ride time can be improved obviously with a slight increase in the total travel time. For example, the total excess user ride time decreased from 65.15 minutes to 51.73 minutes with only 0.15 minute's increase in the total travel time (from 299.11 to 299.26 minutes). These non-dominated points correspond to efficient solutions of low operational costs and high user inconvenience. These efficient solutions have practical interests for profitable service providers (e.g., Uber, Didi) that it is possible to improve the service quality significantly while keeping the operational cost nearly optimal.

2. from the non-dominated points located in the right-downside, we have an inverse observation: the total travel time can be improved obviously with a slight increase in the total excess user ride time. For example, the total travel time can be reduced from 337.84 minutes to 332.98 minutes with only 0.18 minute's increase on the total excess user ride time (from 7.83 to 8.01 minutes). These efficient solutions are interesting for non-profitable associations (e.g., Red Cross) as they allow to reduce operational costs significantly while maintaining high service quality.

5.5 Conclusion

This chapter has introduced a new problem variant of the E-ADARP, namely the BO-EADARP, by considering the total travel time and the total excess user ride time as separate objectives. The BO-EADARP is much more complicated to be solved than the E-ADARP, as one must fully explore the bi-dimension search area in order to demonstrate the completeness of the Pareto front. To tackle the BO-EADARP, we introduce two criterion space search algorithms (i.e., the ϵ-constraint and balanced box methods) and a decision space search algorithm (i.e., the BOBP algorithm). The BOBP algorithm is based on the generalized B&B algorithm proposed in Parragh & Tricoire (2019), where the lower bound set is calculated by the problem-tailored CG algorithm (as presented in Chapter 4). In the computational experiments, we solve the BO-EADARP with three different algorithms on small-to-medium-sized instances under different minimum battery restrictions (i.e., $\gamma = 0.1, 0.4, 0.7$). Due to the complexity of the problem, it is very hard to obtain a complete set of efficient solutions, even for small-sized instances. Therefore, we compare the three algorithms by the number of generated efficient solutions and their average computational time. Among the three algorithms, the BOBP algorithm seems to be the most efficient algorithm, which generates the highest number of efficient solutions in the least average computational time. Then, we analyze the efficient solutions under different γ settings. We observe an obvious increase in the total travel times for the obtained efficient solutions with increasing γ values while the corresponding total excess user ride times remain stable. On each level of γ, the obtained efficient solutions offer managerial insights for different service providers: (1) for profitable service providers, it is possible to significantly improve service quality while keeping near-optimal operational costs; (2) for non-profitable service providers, there exist efficient solutions of high service quality while at lower operational costs. These efficient solutions are much interesting for this kind of service provider. To sum up, the obtained efficient solutions can help decision-makers to select Pareto-optimal transportation plans according to their priorities and preferences. Our next step will focus on enhancing the computational efficiency of the BOBP algorithm. One possible improvement is that we can store columns generated under $\gamma = 0.1, 0.4$ and feed them to solve the considered instance under $\gamma = 0.7$. Another idea is to develop methods that can efficiently check the existence of non-dominated points in a given search area. If we can prove there do not exist any non-dominated points in a given search area, then we do not need to branch. The last idea is to add cutting planes to enhance the lower bound set so that the number of nodes in the

B&P tree is reduced.

Chapter 6

Conclusion and Extensions

In this chapter, we first revisit the objectives, key findings, and contributions of this thesis in Section 6.1. Then, Section 6.2 closes this thesis by discussing the interesting research directions for future works.

6.1 Contributions and Key Findings

In this thesis, we develop highly-efficient heuristic and exact methods (i.e., the DA and B&P algorithms) to solve the E-ADARP, whose objective is to design minimum-cost routes for EAVs to accommodate all customer requests that specify their origins and destinations. The E-ADARP includes two important features: (i) the employment of EAVs and partial recharging allowed at recharging stations; (ii) the weighted-sum objective function that minimizes the total travel time and the total excess user ride time. These two features impact the design of heuristic and exact methods in different ways. For a heuristic method, the first aspect (i) requires evaluating the battery feasibility for a given route, while the second aspect (ii) requires calculating the minimal excess user ride time for a feasible solution. These aspects involve feasibility checking and cost calculation when evaluating a generated E-ADARP route. As route evaluations are performed voluminously when running a heuristic, the efficiency of the route evaluation process plays a crucial role in the overall heuristic performance. How to design an efficient route evaluation process is the key challenge in the design of a heuristic. When it comes to designing an effective labeling algorithm in the B&P scheme, one must decide the excess-user-ride-time optimal schedules from battery feasible schedules along the extension of the partial path. As the path is not fixed until the destination depot is reached, the key challenge is to determine all excess-user-ride-time optimal schedules in the extension of the partial path to ensure finding the minimum-negative-reduced-cost route. After designing effective heuristic and exact methods to solve the single-objective E-ADARP, we extend the model by considering the total travel time and the total excess user ride time in two separate objectives. Our objective is to find a set of efficient solutions and analyze the fundamental trade-off between operational cost and service quality.

The contributions of this thesis are summarized according to the following perspectives:

1. **Model and theoretical level:**

   (a) We introduce a fragment-based representation of paths, which extend the one proposed in Rist & Forbes (2021) by additionally considering battery constraints for ensuring overall route feasibility in terms of energy consumption. Based on this representation of paths, each E-ADARP route can be represented by a series of battery-restricted fragments.

   (b) In the process of designing an effective heuristic method to solve the E-ADARP, we propose a new approach that efficiently computes minimum excess user ride time by introducing a fragment-based representation of paths. Then, we apply an exact route evaluation scheme that executes feasibility checking in linear time. Combining these two methods, we propose an exact and efficient optimization of excess user ride time for an E-ADARP route.

   (c) In the process of designing an effective labeling algorithm to solve the E-ADARP subproblems, we handle the excess-user-ride-time optimality in the extension of an E-ADARP partial path by taking advantage of the fragment-based representation defined in (a). On each fragment, we apply a new approach to determine the minimum excess user ride time and abstract the fragment to an arc while guaranteeing excess-user-ride-time optimality. Then, we construct a new graph that preserves all feasible routes of the original one, and we ensure excess-user-ride-time optimality on each arc of the new graph. We define strong dominance rules on the new graph to allow fast shortest-path computations in solving the E-ADARP subproblems.

   (d) We extend the E-ADARP to consider unlimited recharging visits to each recharging station and conduct sensitivity analysis to investigate the effect of this relaxation.

   (e) We extend the single-objective E-ADARP to the BO-EADARP, which considers the total travel time and the total excess user ride time as two separate objectives.

2. **Algorithm level:**

   - We adapt the DA algorithm to tackle the E-ADARP by integrating the proposed excess user ride time optimization method. To the best of our knowledge, this is the first time an exact excess user ride time optimization has been developed for computing locally optimal solutions within an algorithm for solving the E-ADARP. This method allows computing the minimum excess user ride time for a feasible E-ADARP route in linear time after preprocessing.
   - We develop a highly-efficient labeling algorithm and then integrate it into the CG algorithm and the B&P algorithm to solve the E-ADARP. One important “by-product” of our labeling algorithm is the first exact

scheduling procedure that can efficiently determine the excess-user-ride-time optimal schedule for a given E-ADARP route.

- We adapt the state-of-the-art criterion space search algorithms (i.e., the ϵ-constraint method and balanced box method) and decision space search algorithm (i.e., the BOBP algorithm) to solve the BO-EADARP.

3. **Experimental level:**

   - We demonstrate the performance of the proposed DA algorithm through extensive numerical experiments. On the previously solved instances, the DA algorithm improves the solution quality by 0.16% on average. We provide the best solutions for 70 out of 84 instances, among which 25 are new best solutions. To further test our algorithm in solving large-scale instances, we construct new benchmark instances with up to 8 vehicles and 96 requests, and we provide 19 new solutions on newly-introduced instances. We also extend the E-ADARP model to investigate the effects of allowing unlimited visits to recharging stations. The major difficulties for local search introduced by highly-constrained instances are lessened considering this more realistic situation.
   - The numerical results demonstrate the superiority of our CG algorithm over the state-of-the-art methods, with our algorithm being able to provide 40 new best solutions. We improve 29 previously-reported lower bounds by 1.35% on average and provide 17 new lower bounds for large-scale instances with up to 8 vehicles and 96 requests. We prove optimality for 66 out of 84 instances without branching. For other instances, very small average gaps of 0.07% between lower bounds and the best-known upper bounds are observed. In addition, our algorithm can easily be adapted to tackle another problem variant where unlimited visits to each recharging station are allowed. The proposed labeling algorithm can also serve as the first exact scheduling procedure that generates excess-user-ride-time optimal schedules for a feasible E-ADARP route.
   - We integrate the CG algorithm into the B&P framework, and we further solve 5 additional instances optimally, obtain 6 tighter lower bounds and generate 3 additional new best solutions. Benefiting from the good quality of lower bounds obtained at the root node, our B&P algorithm searches only a few nodes of the search tree to close the gaps. Compared with the best-reported B&C results in Bongiovanni et al. (2019), we finally solve 71 out of 84 instances optimally within the two-hour time limit, while the B&C algorithm can only solve 49 instances optimally. In addition, we obtain 26 new best solutions and 54 equal solutions and enhanced 30 lower bounds. The average computational time of our B&P algorithm decreases by 16% compared with that of the B&C algorithm. On larger-sized instances (i.e., type-r instances), we obtain 16 new best solutions, compared with the existing results of Su et al. (2023). Therefore, the superiority of our B&P algorithm upon the existing exact method in the literature to solve the E-ADARP has been proved.

- We solve the BO-EADARP with three different algorithms (i.e., the ϵ-constraint, balanced box, and the BOBP algorithms) on small-to-medium-sized instances under different minimum battery restrictions (i.e., $\gamma = 0.1, 0.4, 0.7$). Among the three algorithms, the BOBP algorithm seems to be the most efficient algorithm, which generates the highest number of efficient solutions in the least average computational time.

The key findings of this thesis are summarized as follows:

1. In Chapter 3 and 4, we develop highly-efficient heuristic and exact methods (i.e., DA and B&P algorithms) to solve the static version of the E-ADARP. The key challenges mentioned at the beginning of this chapter are handled appropriately by (1) constructing an exact route evaluation scheme for the heuristic algorithm and (2) constructing a new graph that preserves all feasible routes of the original graph, and we ensure excess-user-ride-time optimality on each arc of the new graph. We develop an efficient labeling algorithm with strong dominance rules on the new graph to allow fast shortest-path computations in solving the E-ADARP subproblems.

2. In the numerical experiments, our proposed DA and B&P algorithms provide 70 and 79 equal or improved solutions on 84 existing instances, respectively. In total 25 new best solutions are found compared to the best-reported literature results. Thirty previously reported lower bounds are further enhanced with our B&P algorithm. On newly-introduced large-scale instances, we report 19 new solutions and 17 new lower bounds.

3. We identify in Chapter 3 and 4 that the model of Bongiovanni et al. (2019) relies on incorrect "big M" values (detailed analysis is presented in Appendix A). Due to these incorrect "big M" values, in several instances, the actual optimal solutions are cut off as they are considered infeasible. In this thesis, with our B&P algorithm, we provide correct optimal solutions for concerned instances, which can serve as new benchmark results for future studies.

4. From our experiments on allowing multiple visits per recharging station (recalling n_{as} is the maximum number of charging visits per station), we have the following conclusions:

   - significant increases in the average numbers of visited recharging stations in best-obtained solutions are observed on all instances with increasing γ value, especially on type-r instances. Also, allowing multiple visits to each recharging station improves the solution quality as we found lower-cost solutions.
   - for type-a and -r instances, relaxing to $n_{as} = \infty$ seems to be more computationally attractive as it does not introduce additional computational time, compared to the results obtained by replicating recharging stations. For type-u instances, having a pre-calculated n_{as} would be more computationally favorable;
   - on average, allowing at-most-two and -three visits per station slightly increases the computational time. Allowing at-most-three visits per station seems to strike a good balance between solution quality and computational time;

- $n_{as} = 3$ seems to be a good upper bound for solving type-u instances allowing multiple recharging visits, while one needs to set n_{as} to 4 and 7 for type-a and -r instances, respectively.

5. From analyzing the obtained efficient solutions under different battery restriction levels (i.e., $\gamma = 0.1, 0.4, 0.7$), we have the following observations:

   - With increasing minimum battery level restrictions, we find an obvious increase in the total travel time of the obtained efficient solutions while the total excess user ride time remains stable.
   - On each level of γ, the obtained efficient solutions offer managerial insights for service providers:
     (a) We find efficient solutions at near-optimal operational costs while providing much better service quality than the one at a slightly-lower operational cost. Hence, it is possible to improve the service quality significantly while keeping the operational cost nearly optimal. These efficient solutions are interesting for profitable service providers (e.g., Uber, Didi) to largely improve their service quality only with a slight increase in their operational costs.
     (b) We also find efficient solutions of high service quality while at lower operational costs. Hence, it is possible to reduce operational costs significantly while maintaining high service quality. These efficient solutions are interesting for non-profitable associations (e.g., Red Cross) as they can make savings without degrading their service quality.

6.2 Future Research Directions

This thesis proposes highly-efficient heuristic and exact methods to solve the E-ADARP, a complex combinatorial optimization problem that integrates autonomous mobility, the management of electric vehicles, and ride-sharing services. After developing solid and efficient algorithms to solve the single-objective E-ADARP, we investigate a new problem variant of the E-ADARP (i.e., the BO-EADARP) from algorithmic and operational aspects. From our proposed methods and results, we find several interesting directions for future studies, which are summarized as follows:

1. **Problem level:**

   - The E-ADARP model may be improved by taking into account more real-life characteristics. For example, time-dependent travel times occur with traffic jams in peak hours, non-linear recharging and discharging functions.
   - Relatedly, the static E-ADARP can be extended to dynamic E-ADARP, taking into account updates of requests during the day (e.g., new requests, cancellations, modifications).

2. **Algorithm level:**

- Some instances remain unsolvable even after 50 independent runs of the DA algorithm. One reason may be that no feasible solution exists for these instances, which remains challenging for future studies using heuristic and exact methods.
- An interesting investigation would be examining the effects of more randomness in the algorithm, for example, considering a sequence of randomly ordered operators.
- The computational efficiency of our DA algorithm could be further improved by applying a more intelligent insertion strategy of recharging stations, adapting a parallel version of the DA algorithm, and designing stopping criteria to terminate the algorithm before completing all iterations, as in Ropke & Pisinger (2006).
- The proposed DA and CG algorithm can be adapted to tackle the dynamic DARP/E-ADARP, where new requests arrive in real time. Quick and efficient routing and scheduling heuristics for the dynamic E-ADARP also seem promising.

3. **Theoretical level:**

- Our proposed labeling algorithm is the first exact scheduling procedure that can efficiently determine the excess-user-ride-time optimal schedule for a given E-ADARP route. This scheduling procedure can also be applied to optimize excess user ride time in the classical DARP or the DARP with multiple objectives, in which total excess user ride time is minimized in a separate objective.
- From the first-hand results of the BO-EADARP, it would be interesting to develop efficient acceleration strategies to speed up the BOBP algorithm. One possible direction is that we can store columns generated under $\gamma = 0.1, 0.4$ and feed them to solve the considered instance under $\gamma = 0.7$. Another direction is to develop methods that can efficiently check the existence of non-dominated points in a given search area. Finally, adding valid cuts to further enhance the lower bound set would also be worth investigating.

Appendix A

Appendix for Chapter 2

Several solutions are found by our DA and B&P algorithm are strictly better than the optimal solutions reported in Bongiovanni et al. (2019). In this section, we analyze why the model proposed in the original paper leads to incorrect results. To facilitate illustration, we take our obtained solution of instance a2-24-0.4 as an example, where we get a solution of cost **347.04** while the reportedly optimal solution is **348.03**. The solution gap is higher than the 0.01% default tolerance gap of Gurobi. To find the conflicting constraints in the MIP model, we set the binary variables $x_{i,j}^k$ to the solution we found as constraints and use "compute conflict" in Julia and print the conflicting constraints. The optimal solution we obtained is:

route1: $[51, 7, 31, 11, 35, 10, 34, 56, 5, 29, 4, 21, 20, 28, 45, 44, 1, 25, 12, 8, 36, 32, 54]$

route2: $[52, 17, 41, 19, 43, 22, 46, 18, 2, 42, 26, 15, 39, 6, 30, 16, 13, 40, 37, 23, 47, 14, 38, 57, 3, 27, 24, 9, 48, 33, 55, 53]$

A.1 Error: Incorrect Value for Big "M"

The problematic constraints in Bongiovanni et al. (2019) are:

$$E_s^k \leqslant T_s^k - t_{i,s} - T_i^k + M_{i,s}^k \left(1 - x_{i,s}^k\right), \quad \forall s \in S, i \in D \cup S \cup O_k, k \in K, i \neq s \tag{A.1}$$

$$E_s^k \geqslant T_s^k - t_{i,s} - T_i^k - M_{i,s}^k \left(1 - x_{i,s}^k\right), \quad \forall s \in S, i \in D \cup S \cup O_k, k \in K, i \neq s \tag{A.2}$$

where E_s^k are the decision variables indicating the recharging time at recharging station s for vehicle k. T_i^k and T_s^k are the decision variables indicating the time at which vehicle k starts its service at location i and s, respectively.

In their real implementation, constraints (A.1) and (A.2) are implemented as constraints (A.3) and (A.4). Indeed, as the time windows on the recharging visits are not binding, we need to consider the path from s to i to restrict the

recharging time at station s. The recharging time constraints are formulated as follows:

$$E_s^k \leqslant T_i^k - t_{s,i} - T_s^k + M_{s,i}^k \left(1 - x_{s,i}^k\right), \quad \forall s \in S, i \in P \cup S \cup F, k \in K, i \neq s \tag{A.3}$$

$$E_s^k \geqslant T_i^k - t_{s,i} - T_s^k - M_{s,i}^k \left(1 - x_{s,i}^k\right), \quad \forall s \in S, i \in P \cup S \cup F, k \in K, i \neq s \tag{A.4}$$

That is, we have the following constraints always hold:

$$T_i^k - t_{s,i} - M_{s,i}^k(1 - x_{s,i}^k) \leqslant T_s^k + E_s^k \leqslant T_j^k - t_{s,j} + M_{s,j}^k(1 - x_{s,j}^k), \tag{A.5}$$
$$\forall s \in S, i \in P \cup S \cup F, k \in K, i \neq s$$

However, the value of big "M" used in Bongiovanni et al. (2019) is: $M_{i,j}^k = max\{0, l_i + s_i + t_{i,j} - e_j\}$. With this value of big "M", it will not hold for some cases. We take an example with our obtained solution of a2-24-0.4, where $x_{56,24}^1 = 0$, $x_{56,5}^1 = 1$, $M_{56,24}^1 = 127.20$, $M_{56,5}^1 = 445.32$, $t_{56,24} = 15.86$, $t_{56,5} = 4.98$. As the earliest time window at node 24 is 603.0, the latest time window at node 5 is 302.85, the time window constraints on node 24 and node 5 are:

$$T_{24}^1 \geqslant 603.0, \quad T_5^1 \leqslant 302.85 \tag{A.6}$$

Recharging time constraints are:

$$T_{24}^1 - t_{56,24} - M_{56,24}^1(1 - x_{56,24}^1) \leqslant T_{56}^1 + E_{56}^1 \tag{A.7}$$

$$T_{56}^1 + E_{56}^1 \leqslant T_5^1 - t_{56,5} + M_{56,5}^1(1 - x_{56,5}^1) \tag{A.8}$$

Introducing constraints (A.6) and $x_{56,24}^1 = 0$, $x_{56,5}^1 = 1$, $M_{56,24}^1 = 127.20$, $M_{56,5}^1 = 445.32$, $t_{56,24} = 15.86$, $t_{56,5} = 4.98$ to constraint (A.7) and (A.8), we have:

$$459.94 \leqslant T_{56}^1 + E_{56}^1 \leqslant 297.87 \tag{A.9}$$

which is a contradiction!

To obtain the correct big "M" value, we calculate a lower bound for big "M" parameters from constraints (A.3) as follows:

$$E_s^k - T_i^k + t_{s,i} + T_s^k \leqslant M_{s,i}^k \tag{A.10}$$

The maximum value of the left-hand side is obtained when $E_s^k = \frac{Q}{\alpha}$, $T_i^k = e_i$, $T_s^k = T_p$, where T_p is the planning horizon. We obtain:

$$\frac{Q}{\alpha} - e_i + t_{s,i} + T_p \leqslant M_{s,i}^k \longrightarrow M_{s,i}^k = \min\{\frac{Q}{\alpha} - e_i + t_{s,i} + T_p, T_p\} \tag{A.11}$$

We take T_p directly as the value of big "M" and the contradiction is solved.

A.2 Typo: Incorrect Value of Big "G"

Furthermore, there is also a typo related to the value of big "G" in Bongiovanni et al. (2019) with the following constraints:

$$L_i^k + l_j + G_{i,j}^k \left(1 - x_{i,j}^k\right) \geqslant L_j^k, \quad \forall i \in V \setminus F, j \in V \setminus O_k, i \neq j, k \in K \tag{A.12}$$

The big "G" values used in Bongiovanni et al. (2019) is $G_{i,j}^k = \min\{C^k, C^k + l_i\}$. With this value, the solutions that have continuously visited C^k times drop-off nodes are cut off. For example, our obtained solution has three continuous visits of drop-off nodes: [28, 45, 44], and we have $x_{28,45}^1 = x_{45,44}^1 = 1.0$ and $x_{44,28,1} = 0.0$. As these nodes are drop-off nodes, the loads on these nodes are negative: $l_{45} = l_{44} = l_{28} = -1.0$. The maximal vehicle capacity $C^k(k = 1)$ equals three. Based on constraints (A.12), we have:

$$L_{28}^1 + l_{45} \geqslant L_{45}^1 \tag{A.13}$$

$$L_{45}^1 + l_{44} \geqslant L_{44}^1 \tag{A.14}$$

$$L_{44}^1 + l_{28} + G_{44,28}^1 \geqslant L_{28}^1 \tag{A.15}$$

However, with the defined value in Bongiovanni et al. (2019), $G_{44,28}^1$ must satisfy:

$$G_{44,28}^1 \leqslant C^1 + l_{44} \tag{A.16}$$

Introducing constraint (A.14), (A.15), and (A.16) into constraints (A.13) leads to contradiction:

$$-1.0 = C^1 + l_{44} + l_{45} + l_{28} \geqslant 0 \tag{A.17}$$

The correct value is $G_{i,j}^k = \max\{C^k, C^k + l_i\}$. After revising, the contradiction is eliminated.

Appendix B

Appendix for Chapter 3

We first present the detailed results of fragment enumeration, as shown below.

Table B.1 Details of fragments enumeration for all the instances

	N_{frag}	Leg_{avg}	Leg_{max}	N_{LP}	CPU(s)
a2-16	32	3.06	6	0	0.94
a2-20	51	3.41	6	1	0.23
a2-24	64	3.72	8	1	0.09
a3-18	71	4.25	8	4	0.04
a3-24	110	4.71	12	0	0.06
a3-30	89	3.66	8	0	0.12
a3-36	114	4.12	12	1	0.27
a4-16	78	4.51	8	4	0.04
a4-24	91	4.07	8	2	0.07
a4-32	206	5.58	12	3	0.20
a4-40	242	5.45	12	6	0.37
a4-48	355	5.33	12	15	0.61
a5-40	337	5.65	12	3	0.38
a5-50	659	8.25	24	33	0.99
Avg	178.5	4.70	10.57	5.21	0.32
u2-16	61	3.80	6	0	1.05
u2-20	180	5.26	12	7	0.32
u2-24	66	3.27	4	0	0.06
u3-18	78	3.95	8	0	0.04
u3-24	129	4.25	8	0	0.08
u3-30	255	5.06	8	19	0.29
u3-36	276	5.14	12	12	0.30
u4-16	75	4.03	8	1	0.04
u4-24	57	3.19	6	0	0.05
u4-32	177	4.14	10	3	0.21
u4-40	149	4.01	8	2	0.26
u4-48	1177	9.01	18	7	1.69
u5-40	335	5.28	14	1	0.49
u5-50	584	6.13	14	6	0.96
Avg	257.07	4.75	9.71	4.14	0.42
r5-60	632	6.44	16	44	2.61
r6-48	4082	14.20	36	414	6.89
r6-60	809	6.58	18	40	1.65
r6-72	1080	7.12	22	36	2.51
r7-56	1089	7.92	18	83	1.70
r7-70	2340	8.32	18	183	4.14
r7-84	2892	11.66	30	405	7.77
r8-64	11694	18.23	42	3517	40.52
r8-80	5822	14.89	30	260	14.07
r8-96	3155	9.30	26	312	9.65
Avg	3359.50	10.47	25.6	526.4	9.15

We present the parameter tuning results for θ_{max} in Table B.2 and B.3, where we have considered seven different values of θ_{max}. It should be noted that the symbol "-" indicates we obtain new best solutions on previously unsolved instances, and the gap cannot be calculated.

Table B.2 DA algorithm results on type-a instances with different settings of θ_{max}

	$\theta_{max}=0.6$		$\theta_{max}=0.9$		$\theta_{max}=1.2$		$\theta_{max}=1.5$	
$\gamma=0.1$	$BC_{0.6}\%$	$AC_{0.6}\%$	$BC_{0.9}\%$	$AC_{0.9}\%$	$BC_{1.2}\%$	$AC_{1.2}\%$	$BC_{1.5}\%$	$AC_{1.5}\%$
a2-16	0	0	0	0	0	0	0	0
a2-20	0	0	0	0	0	0	0	0
a2-24	0	0	0	0	0	0	0	0
a3-18	0	0	0	0	0	0	0	0
a3-24	0	0	0	0	0	0.74%	0	0.37%
a3-30	0	0	0	0	0	0	0	0
a3-36	0	0	0	0	0	0.12%	0	0
a4-16	0	0	0	0	0	0	0	0
a4-24	0	0	0	0	0	0	0	0.70%
a4-32	0	0.13%	0	1.04%	0.02%	0.79%	0.08%	1.31%
a4-40	0	0.62%	0	1.24%	0	1.23%	0.89%	1.38%
a4-48	0.34%	0.71%	0.30%	0.94%	0.67%	1.78%	0.90%	1.99%
a5-40	0.44%	1.36%	0.29%	1.91%	0.42%	1.34%	0.78%	1.75%
a5-50	0.71%	1.68%	0.74%	2.14%	1.61%	2.99%	1.83%	2.56%
Avg	0.11%	0.49%	0.10%	0.53%	0.19%	0.74%	0.32%	0.83%
$\gamma=0.4$	$BC_{0.6}\%$	$AC_{0.6}\%$	$BC_{0.9}\%$	$AC_{0.9}\%$	$BC_{1.2}\%$	$AC_{1.2}\%$	$BC_{1.5}\%$	$AC_{1.5}\%$
a2-16	0	0	0	0	0	0	0	0
a2-20	0	0	0	0	0	0	0	0
a2-24	-0.29%	-0.29%	-0.29%	-0.29%	-0.29%	-0.29%	-0.29%	-0.21%
a3-18	0	0	0	0	0	0	0	0
a3-24	0	0	0	0.27%	0	0	0	0.23%
a3-30	-0.01%	NC	-0.01%	-0.01%	-0.01%	-0.01%	-0.01%	-0.01%
a3-36	-0.22%	-0.06%	-0.16%	0.01%	-0.06%	0.05%	-0.03%	0.30%
a4-16	0	0	0	0	0	0	0	0
a4-24	0	0.45%	0	0.09%	0.08%	0.54%	0.08%	0.70%
a4-32	0	0.20%	0.22%	1.26%	0.22%	0.75%	0	1.0%
a4-40	0.27%	0.98%	0	1.28%	0	1.41%	0.27%	1.70%
a4-48	1.04%	1.50%	1.46%	2.37%	1.8%	2.53%	1.85%	2.88%
a5-40	0.35%	1.56%	0.55%	1.19%	0.86%	1.40%	0.55%	1.52%
a5-50	1.46%	2.03%	2.07%	2.84%	1.19%	3.45%	3.14%	4.30%
Avg	0.19%	NC	0.27%	0.68%	0.27%	0.79%	0.40%	0.95%
$\gamma=0.7$	$BC_{0.6}\%$	$F_{0.6}$	$BC_{0.9}\%$	$F_{0.9}$	$BC_{1.2}\%$	$F_{1.2}$	$BC_{1.5}\%$	$F_{1.5}$
a2-16	0	9	0	10	0	10	0	10
a2-20	-	7	-	8	-	10	-	7
a2-24	0.85%	1	0.85%	8	0.85%	8	0.85%	8
a3-18	0	10	0	10	0	10	0	10
a3-24	-0.63%	10	-0.63%	10	-0.33%	10	-0.63%	10
a3-30	-	5	-	9	-	6	-	9
a3-36	0.51%	4	0	10	0.02%	10	0.36%	10
a4-16	0	10	0	10	0	10	0	10
a4-24	0	10	-0.49%	10	-0.49%	10	0	10
a4-32	-7.31%	10	-6.40%	10	-7.49%	10	-6.80%	10
a4-40	-	2	-	1	-	2	-	4
a4-48	NA	0	NA	0	NA	0	NA	0
a5-40	-4.43%	10	-3.81%	10	-5.23%	10	-2.44%	10
a5-50	NA	0	NA	0	NA	0	NA	0

Table B.3 DA algorithm results on type-a instances with different settings of θ_{max} (continue)

	$\theta_{max} = 1.8$		$\theta_{max} = 2.1$		$\theta_{max} = 2.4$	
$\gamma = 0.1$	$BC_{1.8}\%$	$AC_{1.8}\%$	$BC_{2.1}\%$	$AC_{2.1}\%$	$BC_{2.4}\%$	$AC_{2.4}\%$
a2-16	0	0	0	0	0	0
a2-20	0	0	0	0	0	0
a2-24	0	0	0	0	0	0
a3-18	0	0	0	0	0	0
a3-24	0	0	0	0.15%	0	0.52%
a3-30	0	0	0	0	0	0
a3-36	0	0.11%	0	0.10%	0	0.62%
a4-16	0	0	0	0	0	0
a4-24	0	0.67%	0	0.35%	0	0.70%
a4-32	0	1.02%	0.06%	1.11%	0.09%	1.22%
a4-40	0	1.6%	0	1.81%	1.24%	1.85%
a4-48	0.82%	2.16%	1.17%	2.38%	1.55%	3.3%
a5-40	0.96%	2.46%	1.16%	2.75%	1.16%	2.55%
a5-50	2.33%	3.29%	1.57%	3.80%	3.13%	3.98%
Avg	0.29%	0.82%	0.28%	0.94%	0.51%	1.07%
$\gamma = 0.4$	$BC_{1.8}\%$	$AC_{1.8}\%$	$BC_{2.1}\%$	$AC_{2.1}\%$	$BC_{2.4}\%$	$AC_{2.4}\%$
a2-16	0	0	0	0	0	0
a2-20	0	0	0	0	0	0
a2-24	-0.29%	0.05%	-0.29%	-0.12%	-0.29%	0.01%
a3-18	0	0	0	0	0	0
a3-24	0	0.74%	0	0.74%	0	0.53%
a3-30	-0.01%	0.06%	-0.01%	0.08%	-0.01%	-0.01%
a3-36	-0.13%	0.94%	-0.22%	0.64%	-0.06%	1.14%
a4-16	0	0	0	0.12%	0	0.12%
a4-24	0	0.50%	0.08%	0.55%	0	0.70%
a4-32	0	1.64%	1.18%	1.65%	0.17%	1.55%
a4-40	0.99%	1.61%	1.18%	2.58%	1.01%	3.41%
a4-48	2.36%	4.44%	2.87%	4.02%	3.17%	4.54%
a5-40	1.21%	2.24%	1.42%	2.35%	1.86%	3.36%
a5-50	2.71%	3.94%	3.56%	5.25%	2.92%	5.63%
Avg	0.49%	1.18%	0.7%	1.36%	0.63%	1.54%
$\gamma = 0.7$	$BC_{1.8}\%$	$F_{1.8}$	$BC_{2.1}\%$	$F_{2.1}$	$BC_{2.4}\%$	$F_{2.4}$
a2-16	0	10	0	10	0	10
a2-20	-	10	-	7	-	8
a2-24	0.82%	10	0.82%	10	0.82%	10
a3-18	0	10	0	10	0	10
a3-24	-0.63%	10	-0.33%	10	-0.38%	10
a3-30	-	10	-	8	-	8
a3-36	0.56%	10	0.98%	10	2.34%	10
a4-16	0	10	0	10	0	10
a4-24	-0.49%	10	0.03%	10	0.03%	10
a4-32	-5.20%	10	-5.06%	10	-6.07%	10
a4-40	-	2	NA	0	NA	0
a4-48	NA	0	NA	0	NA	0
a5-40	-1.25%	10	-3.00%	10	-1.47%	10
a5-50	NA	0	NA	0	NA	0

Bibliography

Alyasiry, A. M., Forbes, M., & Bulmer, M. (2019). An exact algorithm for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*, *53*, 1695–1705.

Aneja, Y. P., & Nair, K. P. (1979). Bicriteria transportation problem. *Management Science*, *25*, 73–78.

Boland, N., Charkhgard, H., & Savelsbergh, M. (2015). A criterion space search algorithm for biobjective integer programming: The balanced box method. *INFORMS Journal on Computing*, *27*, 735–754.

Bongiovanni, C., Geroliminis, N., & Kaspi, M. (2022a). A ride time-oriented scheduling algorithm for dial-a-ride problems. *arXiv preprint arXiv:2211.07347*, .

Bongiovanni, C., Kaspi, M., Cordeau, J.-F., & Geroliminis, N. (2022b). A machine learning-driven two-phase metaheuristic for autonomous ridesharing operations. *Transportation Research Part E: Logistics and Transportation Review*, *165*, 102835.

Bongiovanni, C., Kaspi, M., & Geroliminis, N. (2019). The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, *122*, 436–456.

Braekers, K., Caris, A., & Janssens, G. K. (2014). Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B: Methodological*, *67*, 166–186.

Braekers, K., & Kovacs, A. A. (2016). A multi-period dial-a-ride problem with driver consistency. *Transportation Research Part B: Methodological*, *94*, 355–377.

Bräysy, O., Dullaert, W., Hasle, G., Mester, D., & Gendreau, M. (2008). An effective multirestart deterministic annealing metaheuristic for the fleet size and mix vehicle-routing problem with time windows. *Transportation Science*, *42*, 371–386.

Bruglieri, M., Pezzella, F., Pisacane, O., & Suraci, S. (2015). A variable neighborhood search branching for the electric vehicle routing problem with time windows. *Electronic Notes in Discrete Mathematics*, *47*, 221–228.

Ceselli, A., Felipe, Á., Ortuño, M. T., Righini, G., & Tirado, G. (2021). A branch-and-cut-and-price algorithm for the electric vehicle routing problem with multiple technologies. In *Operations Research Forum* (pp. 1–33). Springer volume 2.

Chalmet, L., Lemonidis, L., & Elzinga, D. (1986). An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research*, *25*, 292–300.

Chevrier, R., Liefooghe, A., Jourdan, L., & Dhaenens, C. (2012). Solving a dial-a-ride problem with a hybrid evolutionary multi-objective approach: Application to demand responsive transport. *Applied Soft Computing*, *12*, 1247–1258.

Conrad, R. G., & Figliozzi, M. A. (2011). The recharging vehicle routing problem. In *Proceedings of the 2011 industrial engineering research conference* (p. 8). IISE Norcross, GA.

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, *54*, 573–586.

Cordeau, J.-F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, *37*, 579–594.

Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of operations research*, *153*, 29–46.

Desaulniers, G., Desrosiers, J., & Spoorendonk, S. (2011). Cutting planes for branch-and-price algorithms. *Networks*, *58*, 301–310.

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

Desaulniers, G., Lessard, F., & Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*, 387–404.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, *40*, 342–354.

Detti, P., Papalini, F., & de Lara, G. Z. M. (2017). A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega*, *70*, 1–14.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, *90*, 161–175.

Duman, E. N., Taş, D., & Çatay, B. (2021). Branch-and-price-and-cut methods for the electric vehicle routing problem with time windows. *International Journal of Production Research*, (pp. 1–22).

Erdoğan, S., & Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation research part E: logistics and transportation review*, *48*, 100–114.

Felipe, Á., Ortuño, M. T., Righini, G., & Tirado, G. (2014). A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research Part E: Logistics and Transportation Review*, *71*, 111–128.

Firat, M., & Woeginger, G. J. (2011). Analysis of the dial-a-ride problem of hunsaker and savelsbergh. *Operations Research Letters*, *39*, 32–35.

Forget, N., & Parragh, S. N. (2022). Enhancing branch-and-bound for multi-objective 0-1 programming. *arXiv preprint arXiv:2210.05385*, .

Froger, A., Mendoza, J. E., Jabali, O., & Laporte, G. (2017). *A matheuristic for the electric vehicle routing problem with capacitated charging stations*. Ph.D. thesis Centre interuniversitaire de recherche sur les reseaux d'entreprise, la ....

Garaix, T., Artigues, C., Feillet, D., & Josselin, D. (2011). Optimization of occupancy rate in dial-a-ride problems via linear fractional column generation. *Computers & Operations Research*, *38*, 1435–1442.

Glize, E., Jozefowiez, N., & Ngueveu, S. U. (2022). An ε-constraint column generation-and-enumeration algorithm for bi-objective vehicle routing problems. *Computers & Operations Research*, *138*, 105570.

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

Gschwind, T. (2015). A comparison of column-generation approaches to the synchronized pickup and delivery problem. *European Journal of Operational Research*, *247*, 60–71.

Gschwind, T. (2019). Route feasibility testing and forward time slack for the synchronized pickup and delivery problem. *OR Spectrum*, *41*, 491–512.

Gschwind, T., & Drexl, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, *53*, 480–491.

Gschwind, T., & Irnich, S. (2015). Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, *49*, 335–354.

Guerriero, F., Bruni, M. E., & Greco, F. (2013). A hybrid greedy randomized adaptive search heuristic to solve the dial-a-ride problem. *Asia-Pacific Journal of Operational Research*, *30*, 1250046.

Haimes, Y. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, (pp. 296–297).

Haugland, D., & Ho, S. C. (2010). Feasibility testing for dial-a-ride problems. In *Algorithmic Aspects in Information and Management: 6th International Conference, AAIM 2010, Weihai, China, July 19-21, 2010. Proceedings 6* (pp. 170–179). Springer.

Hiermann, G., Hartl, R. F., Puchinger, J., & Vidal, T. (2019). Routing a mix of conventional, plug-in hybrid, and electric vehicles. *European Journal of Operational Research*, *272*, 235–248.

Hiermann, G., Puchinger, J., Ropke, S., & Hartl, R. F. (2016). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, *252*, 995–1018.

Ho, S. C., & Haugland, D. (2011). Local search heuristics for the probabilistic dial-a-ride problem. *Or Spectrum*, *33*, 961–988.

Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421.

Hunsaker, B., & Savelsbergh, M. (2002). Efficient feasibility testing for dial-a-ride problems. *Operations research letters*, *30*, 169–173.

Jepsen, M., Petersen, B., Spoorendonk, S., & Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, *56*, 497–511.

Jin, S. T., Kong, H., Wu, R., & Sui, D. Z. (2018). Ridesourcing, the sharing economy, and the future of cities. *Cities*, *76*, 96–104.

Keskin, M., & Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation research part C: emerging technologies*, *65*, 111–127.

Kirchler, D., & Calvo, R. W. (2013). A granular tabu search algorithm for the dial-a-ride problem. *Transportation Research Part B: Methodological*, *56*, 120–135.

Kirlik, G., & Sayın, S. (2014). A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, *232*, 479–488.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., & Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, *33*, 101–116.

Lam, E., Desaulniers, G., & Stuckey, P. J. (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*, (p. 105870).

Laumanns, M., Thiele, L., & Zitzler, E. (2006). An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, *169*, 932–942.

Lehuédé, F., Masson, R., Parragh, S. N., Péton, O., & Tricoire, F. (2014). A multi-criteria large neighbourhood search for the transportation of disabled people. *Journal of the Operational Research Society*, *65*, 983–1000.

Liu, M., Luo, Z., & Lim, A. (2015). A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B: Methodological*, *81*, 267–288.

Lokman, B., & Köksalan, M. (2013). Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, *57*, 347–365.

Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations research*, *53*, 1007–1023.

Madsen, O. B., Ravn, H. F., & Rygaard, J. M. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of operations Research*, *60*, 193–208.

Masin, M., & Bukchin, Y. (2008). Diversity maximization approach for multiobjective optimization. *Operations Research*, *56*, 411–424.

Masmoudi, M. A., Braekers, K., Masmoudi, M., & Dammak, A. (2017). A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & operations research*, *81*, 1–13.

Masmoudi, M. A., Hosny, M., Braekers, K., & Dammak, A. (2016). Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research Part E: Logistics and Transportation Review*, *96*, 60–80.

Masmoudi, M. A., Hosny, M., Demir, E., Genikomsakis, K. N., & Cheikhrouhou, N. (2018). The dial-a-ride problem with electric vehicles and battery swapping stations. *Transportation research part E: logistics and transportation review*, *118*, 392–420.

Masson, R., Lehuédé, F., & Péton, O. (2014). The dial-a-ride problem with transfers. *Computers & Operations Research*, *41*, 12–23.

Mavrotas, G., & Diakoulaki, D. (1998). A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, *107*, 530–541.

Mavrotas, G., & Diakoulaki, D. (2005). Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied mathematics and computation*, *171*, 53–71.

Molenbruch, Y., Braekers, K., Caris, A., & Berghe, G. V. (2017). Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation. *Computers & Operations Research*, *77*, 58–71.

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

Ozlen, M., Burton, B. A., & MacRae, C. A. (2014). Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, *160*, 470–482.

Özpeynirci, Ö., & Köksalan, M. (2010). An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, *56*, 2302–2315.

Paquette, J., Cordeau, J.-F., Laporte, G., & Pascoal, M. M. (2013). Combining multicriteria analysis and tabu search for dial-a-ride problems. *Transportation Research Part B: Methodological*, *52*, 1–16.

Parragh, S. N. (2011). Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C: Emerging Technologies*, *19*, 912–930.

Parragh, S. N., Cordeau, J.-F., Doerner, K. F., & Hartl, R. F. (2012). Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints. *OR spectrum*, *34*, 593–633.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, *37*, 1129–1138.

Parragh, S. N., Doerner, K. F., Hartl, R. F., & Gandibleux, X. (2009). A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks: An International Journal*, *54*, 227–242.

Parragh, S. N., Pinho de Sousa, J., & Almada-Lobo, B. (2015). The dial-a-ride problem with split requests and profits. *Transportation Science*, *49*, 311–334.

Parragh, S. N., & Tricoire, F. (2019). Branch-and-bound for bi-objective integer programming. *INFORMS Journal on Computing*, *31*, 805–822.

Pedersen, C. R., Nielsen, L. R., & Andersen, K. A. (2008). The bicriterion multimodal assignment problem: Introduction, analysis, and experimental results. *INFORMS Journal on Computing*, *20*, 400–411.

Przybylski, A., Gandibleux, X., & Ehrgott, M. (2008). Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, *185*, 509–533.

Przybylski, A., Gandibleux, X., & Ehrgott, M. (2010). A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, *7*, 149–165.

Qu, Y., & Bard, J. F. (2013). The heterogeneous pickup and delivery problem with configurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, *32*, 1–20.

Qu, Y., & Bard, J. F. (2015). A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity. *Transportation Science*, *49*, 254–270.

Ralphs, T. K., Saltzman, M. J., & Wiecek, M. M. (2006). An improved algorithm for solving biobjective integer programs. *Annals of Operations Research*, *147*, 43–70.

Reinhardt, L. B., Clausen, T., & Pisinger, D. (2013). Synchronized dial-a-ride transportation of disabled passengers at airports. *European Journal of Operational Research*, *225*, 106–117.

Rist, Y., & Forbes, M. A. (2021). A new formulation for the dial-a-ride problem. *Transportation Science*, *55*, 1113–1135.

Ropke, S., & Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, *43*, 267–286.

Ropke, S., Cordeau, J.-F., & Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks: An International Journal*, *49*, 258–272.

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

Savelsbergh, M. W. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, *4*, 146–154.

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

Sourd, F., & Spanjaard, O. (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, *20*, 472–484.

Stidsen, T., Andersen, K. A., & Dammann, B. (2014). A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, *60*, 1009–1032.

Su, Y., Dupin, N., & Puchinger, J. (2023). A deterministic annealing local search for the electric autonomous dial-a-ride problem. *European Journal of Operational Research*, .

Tang, J., Kong, Y., Lau, H., & Ip, A. W. (2010). A note on “efficient feasibility testing for dial-a-ride problems”. *Operations Research Letters*, *38*, 405–407.

Toth, P., & Vigo, D. (1996). Fast local search algorithms for the handicapped persons transportation problem. In *Meta-Heuristics* (pp. 677–690). Springer.

Ulungu, E. L. (1993). Optimisation combinatoire multicritére: Détermination de l’ensemble des solutions efficaces et méthodes interactives. *Unpublished Ph. D. dissertation. Faculté Polytechnique de Mons, Belgium*, .

Vincent, T., Seipp, F., Ruzika, S., Przybylski, A., & Gandibleux, X. (2013). Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers & Operations Research*, *40*, 498–509.

Visée, M., Teghem, J., Pirlot, M., & Ulungu, E. L. (1998). Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, *12*, 139–155.