UNIVERSITÉ DE TOURS

ÉCOLE DOCTORALE : MIPTIS

Laboratoire d'Informatique Fondamentale et Appliquée de Tours

THÈSE présentée par :

Nicholas D. KULLMAN

soutenue le : **04 mai 2020**

pour obtenir le grade de : **Docteur de l'université de Tours**

Discipline/ Spécialité : INFORMATIQUE

Dynamic Decision Making Under Uncertainty in Vehicle Routing and Logistics

THÈSE dirigée par :

BILLAUT Jean-Charles	Professeur, Université de Tours, France
MENDOZA Jorge E.	Professeur Visiteur, HEC Montréal, Canada

Co-encadrement :

GOODSON Justin C.	Associate Professor, Saint Louis University, Etats-Unis

RAPPORTEURS :

JABALI Ola	Associate Professor, Politecnico di Milano, Italie
VAN HENTENRYCK Pascal	Professeur, Georgia Tech, Etats-Unis

JURY :

BILLAUT Jean-Charles	Professeur, Université de Tours, France
COUSINEAU Martin	Professeur Adjoint, HEC Montréal, Canada
GOODSON Justin C.	Associate Professor, Saint Louis University, Etats-Unis
JABALI Ola	Associate Professor, Politecnico di Milano, Italie
MENDOZA Jorge E.	Professeur Visiteur, HEC Montréal, Canada
PUCHINGER Jakob	Professeur, Université Paris-Saclay, France
SOUKHAL Ameur	Professeur, Université de Tours, France
VAN HENTENRYCK Pascal	Professeur, Georgia Tech, Etats-Unis

Acknowledgment

My earning this degree would have been impossible without the help of many others. I want to start off by thanking my primary advisors, Jorge Mendoza and Justin Goodson, who have brought significant joy (and some pain) to my life and have made me a better researcher. Their wisdom, flexibility, confidence, dedication, and, especially, their friendship will be forever appreciated. I also want to thank my thesis director, Jean-Charles Billaut, whose vigilant eye overseeing this research has made it possible and more enjoyable. In addition, I want to extend my appreciation to Martin Cousineau. Much of the work here would have been beyond our reach were it not for his tenacity and knowledge of machine learning. I also want to thank the outstanding group of researchers that have agreed to serve on my jury: Jakob Puchinger, Ameur Soukhal, and additional thanks to my reporters, Ola Jabali and Pascal Van Hentenryck. It has been a pleasure to get to know each of them, and I am grateful for their expertise.

My greatest thanks go to my wife, Rachel (affectionately, "The Kullman Foundation"), for her unlimited support during my degree, having been my sounding board, counselor, editor, bank, and the best travel companion I could ask for. No amount of thanks could be enough. I also cannot give sufficient thanks to my family, especially my parents and grandparents. Their encouragement throughout my life and the emphasis they have always placed on education has made me who I am, and I am eternally grateful.

I want to thank everyone in Tours as well – faculty, staff, and fellow students – who has welcomed me and made my time there special. In particular, my thanks to Christelle Grange for her infinite patience with me and my terrible French, and for turning painful processes into easy ones. I also want to give special thanks to Aurelien Froger. His selfless offering of time and code greatly improved the research here. My thanks also to Clement Grodecoeur for accelerating the work on our Atari project and being a great temporary addition to our team. Finally, I want to acknowledge the Société des Professeurs Français et Francophones d'Amérique (SPFFA) for their generous funding, as well as the high performance computing team at Saint Louis University for allowing me access to computational resources that were invaluable during this work.

Résumé

Cette thèse présente trois études menées sur des problèmes de tournées dynamiques. En particulière, elle se concentre sur les challenges résultants de l'utilisation de véhicules électriques dans les systèmes logistiques et de transports. Dans la première étude, nous introduisons le problème de tournées de véhicules électriques avec des bornes de recharge publiques et privées. Dans ce contexte, les véhicules peuvent recharger leurs batteries en route, dans des bornes publiques, ainsi qu'au dépôt (bornes privées). Pour se protéger contre l'incertitude de la disponibilité des bornes publiques, nous présentons des politiques de routage qui anticipent la dynamique des files d'attente des bornes. Nos politiques se basent sur une décomposition du problème en deux phases : routage et planification des opérations de recharge. Grâce à cette décomposition, nous obtenons la politique statique optimale, ainsi qu'un certain nombre de politiques dites « anticipatoires » et une borne inférieure. Des tests numériques effectués sur des instances réelles fournies par une entreprise, monter que nos politiques sont capables de livrer des solutions avec un gap d'optimalité de moins de 5%. Nos tests montrent aussi que permettre aux véhicules de charger en dehors du dépôt (même en présence d'incertitude sur la disponibilité des bornes) se traduit par des économies considérables dans la durée des routes.

Dans la deuxième étude, nous considérons le problème d'un opérateur contrôlant une flotte de véhicules de tourisme avec chauffeur (VTCs) électriques. L'opérateur, qui cherche à maximiser ses revenus, doit affecter les véhicules aux demandes au fur et à mesure de leur apparition ainsi que charger et repositionner les véhicules en prévision des demandes futures. Pour attaquer ce problème, nous utilisons des approches basées sur l'apprentissage par renforcement profond. Pour mesurer la qualité de nos approches, nous avons développé aussi une heuristique proche de celle typiquement utilisée dans l'affectation de taxis, ainsi que des bornes supérieures. Nous testons nos approches dans des instances construites à partir de données réelles de l'île de Manhattan. Nos tests montrent que notre meilleure politique basée sur l'apprentissage profond livre des résultats supérieurs à ceux livrés par l'heuristique. Les tests montrent aussi que cette stratégie passe facilement à l'échelle et peut être déployée sur de plus grandes instances sans entrainement supplémentaire.

La dernière étude introduit une nouvelle approche générique pour modéliser des problèmes d'optimisation dynamique sous la forme de jeux vidéo de type Atari. L'objectif est de les rendre abordables à travers de méthodes de solution issus de communauté d'apprentissage par renforcement profond. L'approche est flexible et applicable à un large éventail de problèmes. Pour illustrer son application, nous nous attaquons à un problème bien établie dans la littérature : le problème de tournées de véhicules avec des requêtes de service stochastiques. Nos résultats préliminaires sur ce problème sont très

encourageants et montrent que « l'Atari-fication » peut être la voie pour résoudre des problèmes d'optimisation dynamique qui s'avèrent difficiles pour les approches basées sur les outils classiques de la recherche opérationnelle.

Les derniers chapitres présentent deux logiciels développées pour supporter nos recherches. Le premier, nommé frvcpy, permet de déterminer l'insertion optimal des opérations de recharge dans une tournée prédéterminée. Ce logiciel et son code source, présenté comme une bibliothèque Python, a été mis à disposition de la communauté scientifique. Le deuxième outil, VRP-REP Mapper, est un outil web pour visualiser et analyser des solutions pour les problèmes de tournées de véhicules. Cette outil a été intégré a `www.vrp-rep.org`, la plateforme de référence pour le partage de données scientifiques dans le domaine.

Abstract

This thesis details three problems and two software tools related to dynamic decision making under uncertainty in vehicle routing and logistics, with an emphasis on the challenges encountered when adopting electric vehicles. We first introduce the electric vehicle routing problem with public-private recharging strategy in which vehicles may recharge en-route at public charging infrastructure as well as at a privately-owned depot. To hedge against uncertain demand at public charging stations, we design routing policies that anticipate station queue dynamics. We leverage a decomposition to identify good routing policies, including the optimal static policy and fixed-route-based rollout policies that dynamically respond to observed queues. The decomposition also enables us to establish dual bounds, providing a measure of goodness for our routing policies. In computational experiments using real instances from industry, we show the value of our policies to be within five percent of the value of an optimal policy in the majority of instances and within eleven percent on average. Further, we demonstrate that our policies significantly outperform the industry-standard routing strategy in which vehicle recharging generally occurs at a central depot. Our proposed methods for this problem stand to reduce the operating costs associated with electric vehicles, facilitating the transition from internal-combustion engine vehicles.

We then consider the problem of an operator controlling a fleet of electric vehicles for use in a ridehailing service. The operator, seeking to maximize revenue, must assign vehicles to requests as they arise and recharge and reposition vehicles in anticipation of future requests. To solve this problem, we employ deep reinforcement learning, developing policies whose decision making uses Q-value approximations learned by deep neural networks. We compare these policies against a common taxi dispatching heuristic and against dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information which we establish using a Benders-based decomposition. We assess performance on instances derived from real data for the island of Manhattan in New York City. We find that, across instances of varying size, our best policy trained with deep reinforcement learning outperforms the taxi dispatching heuristic. We also provide evidence that this policy may be effectively scaled and deployed on larger instances without retraining.

We then present a new general approach to modeling research problems as Atari-like videogames to make them amenable to recent solution methods from the deep reinforcement learning community. The approach is flexible, applicable to a wide range of problems. Here, we demonstrate its application on the well-studied vehicle routing problem with stochastic service requests. Our preliminary results on this problem, though not transformative, show

signs of success and suggest that Atari-fication may be a useful modeling approach for researchers studying problems involving sequential decision making under uncertainty.

We then introduce *frvcpy*, the first of our two proposed software tools. In the routing of electric vehicles, one of the most challenging tasks is determining how to make good charging decisions for an electric vehicle traveling a given route. This is known as the fixed route vehicle charging problem. An exact and efficient algorithm for this task exists, but its implementation is sufficiently complex to deter researchers from adopting it. Our proposed tool, *frvcpy*, is an open-source Python package implementing this algorithm. Our aim with the package is to make it easier for researchers to solve electric vehicle routing problems, facilitating the development of optimization tools that may ultimately enable the mass adoption of electric vehicles.

Finally, we introduce the second software tool, *Mapper*. *Mapper* is a simple web-based visualizer of problem instances and solutions for vehicle routing problems. It is designed to accompany the suite of tools already available to users of the vehicle routing community's website, *The Vehicle Routing Problem Repository*.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Transportation decisions have important economic impacts. Studies suggest, for example, that transportation costs account for up to 25% of the price of goods (Avella et al. 2004) and between one and two thirds of logistics costs (Tseng et al. 2005). Transportation decisions also have meaningful social and environmental consequences. As climate change becomes an increasingly important global issue, Ritchie and Roser (2020) state that transportation is responsible for approximately 20% of carbon emissions globally, and Office of Transportation and Air Quality (2019) reports that it comprises one third of carbon emissions in the United States. Making sound transportation decisions is thus of broad concern.

The pursuit of sound decision-making in transportation is the focus of many researchers in the Operations Research (OR) domain. Of specific interest here are those developing optimization tools to solve Vehicle Routing Problems (VRPs). VRPs generally consist of determining the most efficient means by which one or more vehicles delivers goods or services to or from a set of customers, subject to a set of business constraints. The first VRP in the literature is due to Dantzig and Ramser with their study of the capacitated VRP (CVRP) in 1959. In the CVRP, the objective is to find routes minimizing the cost of servicing customer demands with a fleet of capacitated vehicles. This problem has effectively served as the basis for many VRP variants since, each of which considers a unique set of additional constraints. These additional constraints accommodate, for example, heterogeneous fleets (e.g., Golden et al. (1984), Gendreau et al. (1999b), Markov et al. (2016)), routes whose customers require both pickups and deliveries (e.g., Savelsbergh and Sol (1995), Berbeglia et al. (2007)), time-dependent travel times (e.g., Jabali et al. (2012)), customers whose demand can only be serviced by a particular vehicle or on a particular day (e.g., Nag et al. (1988), Vidal et al. (2012)), customers with time-windows (e.g., Savelsbergh (1985), Hiermann et al. (2016)), customers with mobile delivery locations (e.g., Reyes et al. (2017)), delivery via unmanned aerial vehicles (drones; e.g., Zhen et al. (2019)), and many other features that ultimately make the addressed problem more closely mirror actual business cases. These variants have been nicely summarized in reviews such as Laporte (2009), and in books such as Toth and Vigo (2014).

It is often the case that the business applications motivating VRPs do not perfectly align with the corresponding academic research. As a common example of this discrepancy, consider that in practice, decision makers are typically not privy to perfect information, as

is assumed to be the case in the majority of VRP research. That is, most VRP studies do not specifically model uncertainties in problem parameters despite their presence in the underlying business applications. This poses challenges, since evidence suggests that what are good routing solutions to deterministic VRPs may ultimately yield poor solutions in the context of uncertain problem parameters (Sörensen and Sevaux 2009). In the worst-case scenario, deterministic solutions may lead to *route failures*, in which a realized uncertainty renders a planned route infeasible. As an example, in the CVRP with uncertain customer demands, a vehicle may arrive to a customer with insufficient capacity to serve the customer's realized demand. For this reason, dedicated solution methods are required to solve these uncertain, i.e. *stochastic*, VRPs (SVRPs).

A common approach to solving SVRPs is to provide a *recourse* strategy to be implemented in the event of a route failure. A simple recourse strategy for the previous example would be for the vehicle to return to its central depot to replenish its capacity, then return to the customer and continue its route. Such solutions have historically garnered much attention in the vehicle routing community (see, e.g., examples given in the reviews by Gendreau et al. (2014) and by Ritzinger et al. (2016)). While they may perform well on average, additional improvements may be obtained by dynamically modifying the routing plan as new information becomes known. Thanks to recent advances in communication technologies, new information is becoming known increasingly frequently. This information includes, for example, real-time access to vehicle locations and capacities, customer demands, and traffic data. Having access to more abundant and more frequently updated information has invigorated research efforts to address SVRPs more dynamically.

Dynamically solved SVRPs are often termed dynamic VRPs (DVRPs). In general, the solution to a DVRP is a routing *policy*, i.e., a plan that determines what routing action to take given the current situation (the current *state*). A number of solution methods exist to produce routing policies. A common method is *reoptimization* in which a static VRP is resolved occasionally – either at fixed intervals or as new information is revealed – to produce a route to be followed until the time of the next solution. Reoptimization is an especially common method among early DVRPs (e.g., Gendreau et al. (1999a)) but remains a popular approach (e.g., Bertsimas et al. (2019)) due to the extensive knowledge base available for solving static VRPs. Another common method is the use of so-called *lookahead algorithms*, which explicitly consider possible future states or scenarios. Especially popular lookahead algorithms include rollouts (e.g. Goodson et al. (2013) and Ulmer et al. (2018)), as well as the Multiple Scenario Approach proposed by Bent and Van Hentenryck (2004). *Value function approximations* (VFAs) are also commonly employed to solve DVRPs (e.g., Secomandi (2000) and Toriello et al. (2014)). VFAs leverage the recursive relationship of dynamic programs, a suitable modeling construct for DVRPs, to estimate the immediate and future impact of the actions available to the decision maker from the current state.

While recent reviews (Ulmer et al., Psaraftis et al. (2016), Braekers et al. (2016), Ritzinger et al. (2016)) note that the number of studies considering DVRPs is growing, they remain a minority, and the DVRP literature still lags its static counterpart. The work in this thesis seeks to contribute to the DVRP literature by (i) studying real-world vehicle routing problems in uncertain environments that have not yet been addressed; (ii) proposing dynamic solution methods for these problems combining reoptimization, lookahead algorithms, and VFAs; and (iii) producing tools that facilitate the study of

VRPs by future researchers.

Within these goals, this thesis also gives special attention to routing problems for electric vehicles (EVs). EVs are of interest to operators seeking to reduce their carbon emissions, but they entail additional business constraints not encountered by their conventional vehicle (CV) counterparts. The commercial adoption of EVs requires routing optimization tools that incorporate these additional constraints, but the availability of such tools is currently lacking compared to those available for CVs (Pelletier et al. 2016). Three of the five subsequent chapters are devoted to further developing these tools for EV routing optimization.

Contributions

The remainder of this chapter summarizes the contents and contributions of the chapters that comprise this thesis. We describe three research projects in Chapters 2-4, and two software tools in Chapters 5 and 6.

Chapter 2: EV Routing with Public Charging Stations

In Chapter 2, we introduce the EV routing problem (E-VRP) with public-private recharging strategy (E-VRP-PP) in which vehicles may recharge en-route at public charging infrastructure as well as at a privately-owned depot. We assume that demand is uncertain at the public charging infrastructure, meaning the vehicle may have to wait to recharge after it arrives. To hedge against this uncertain demand, we design routing policies that anticipate station queue dynamics. These policies are constructed using both a reoptimization approach and lookahead algorithms. While the E-VRP-PP is inherently a DVRP, we identify a decomposition of the problem that facilitates the solution of its static counterpart. We leverage this decomposition to identify good routing policies, including the optimal static policy and fixed-route-based rollout policies that dynamically respond to observed queues. The decomposition also enables us to establish dual bounds, providing a measure of goodness for our routing policies. In computational experiments using real instances from industry, we show the value of our policies to be within five percent of the value of an optimal policy in the majority of instances and within eleven percent on average. Further, we demonstrate that our policies significantly outperform the industry-standard routing strategy in which vehicle recharging occurs only at the privately-owned depot. Our proposed methods for this problem stand to reduce the operating costs associated with EVs, facilitating the transition from CVs.

This work is currently in the second round of reviews at *Transportation Science*, but a pre-print is available on HAL.[1] Partial results of the work were also presented at the following international conferences and workshops:

- 2016 INFORMS Annual Meeting in Nashville, Tennessee, USA
- 2017 conference of the INFORMS Transportation Science and Logistics (TSL) Society in Chicago, Illinois, USA

[1] https://hal.archives-ouvertes.fr/hal-01928730

- 2018 ODYSSEUS workshop on Freight Transportation and Logistics in Cagliari, Italy

Chapter 3: Dynamic Ridehailing with Electric Vehicles

In Chapter 3, we consider the problem of an operator controlling a fleet of electric vehicles for use in a ridehailing service. The operator, seeking to maximize revenue, must assign vehicles to requests as they arise and recharge and reposition vehicles in anticipation of future requests. To solve this problem, we employ deep reinforcement learning (RL), a type of VFA. Specifically, we use deep RL to develop policies that make decisions in real time using Q-value approximations learned by deep neural networks. The developed policies are *model-free*, meaning they learn to anticipate future demand without any prior knowledge of its shape. We compare these deep RL policies against a common taxi dispatching heuristic and against dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information which we establish using a Benders-based decomposition. We assess performance on instances derived from real data reflecting 2018 ridehailing operations on the island of Manhattan in New York City. We find that, across instances of varying size, our best policy trained with deep reinforcement learning outperforms the taxi dispatching heuristic. We also provide evidence that this policy may be effectively scaled and deployed on larger problem instances without additional training. This is encouraging for operators of ridehail companies, as it suggests robustness to changes in the scale of operations: in the event of atypical demand or a change in the number of vehicles (e.g., due to fleet maintenance), the policy should still provide reliable service.

This work was conducted in collaboration with Martin Cousineau at HEC Montréal. It is under review at *Transportation Science* (submitted January 2020), but a pre-print is available on HAL.[2] It was also presented at the following international conferences and workshops:

- 2019 Optimization Days conference in Montréal, Quebec, Canada
- 2019 workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog) in Seville, Spain
- 2019 VeRoLog workshop on the Logistics of Autonomous Vessels in Bergen, Norway
- 2019 workshop of the INFORMS Transportation Science and Logistics (TSL) Society in Vienna, Austria
- 2019 INFORMS Annual Meeting in Seattle, Washington, USA

Chapter 4: Atari-fying the VRP with Stochastic Service Requests

In Chapter 4, we discuss a novel solution method for DVRPs that also extends more broadly to many research problems involving dynamic decision making under uncertainty. The ideas in this chapter were born as a potential solution method for the research problem considered in Chapter 3, but eventually became a standalone research project. The proposed

[2] https://hal.archives-ouvertes.fr/hal-02463422

solution method involves modeling research problems as Atari-like videogames to make them amenable to recent solution methods from the deep RL community. We refer to this modeling process as *Atari-fication*. We demonstrate its application on the well-studied VRP with stochastic service requests (VRPSSR). Our primary contributions in this work include 1) the introduction of the concept of Atari-fying research problems, 2) demonstrating the Atari-fication of the VRPSSR, and 3) (by way of (2)), offering the first application of deep RL to this problem. Our preliminary results for the VRPSSR, though not transformative, show signs of success and suggest that this process may be a useful modeling approach for researchers studying problems involving dynamic decision making under uncertainty.

This work was conducted in collaboration with Martin Cousineau at HEC Montréal. A preprint of this work is available on arXiv.[3] It was also presented at the

- 2019 INFORMS Annual Meeting in Seattle, Washington, USA

and is scheduled for presentation at the upcoming

- 2020 conference of the INFORMS TSL Society in Washington, D.C., USA.

Chapter 5: frvcpy

In Chapter 5, we introduce the software tool *frvcpy*. In the routing of electric vehicles, one of the most challenging tasks is determining how to make good charging decisions for an EV traveling a given route. This is known as the fixed route vehicle charging problem (FRVCP). The FRVCP naturally arises as a subproblem in many E-VRP variants, since its solution is required in order to determine the true duration or cost of a given route. Indeed, we encounter the FRVCP in the E-VRPs studied in both Chapters 2 and 3. Having a capable solution method for the FRVCP is thus crucial to the advancement of E-VRP research. An exact and efficient algorithm solving the FRVCP exists, but its implementation is sufficiently complex to deter researchers from adopting it. Our proposed tool, *frvcpy*, is an open-source Python package implementing this algorithm. Our aim with the package is to make the solution of E-VRPs easier, facilitating the development of EV routing optimization tools that may help enable their mass adoption.

frvcpy is currently under review for publication in the recently established "Software Tools" area of the *INFORMS Journal on Computing*; a pre-print is available on HAL.[4] It has also been published as open-access software (Kullman et al. 2020b) available on the Python Package Index (PyPI)[5]. Its source code is available on GitHub.[6]

Chapter 6: Mapper

Finally, in Chapter 6, we describe the *Mapper* software tool. *Mapper* is a simple web-based visualizer of problem instances and solutions for vehicle routing problems. It is designed to

[3] https://arxiv.org/abs/1911.05922
[4] https://hal.archives-ouvertes.fr/hal-02496381
[5] https://pypi.org/project/frvcpy/
[6] https://github.com/e-VRO/frvcpy

accompany the suite of tools already available to users of the vehicle routing community's website, *The Vehicle Routing Problem Repository* (VRP-REP; `vrp-rep.org`).

Mapper is published on the VRP-REP website.[7] Its source code and instruction manual are available in a GitHub repository owned by VRP-REP.[8]

[7] `http://www.vrp-rep.org/resources.html`
[8] `https://github.com/VRP-REP/mapper/`

Chapter 2

Electric Vehicle Routing with Public Charging Stations[1]

2.1 Introduction

Electric vehicles (EVs) are beginning to replace internal-combustion engine vehicles (CVs) in supply chain distribution and in service routing. Logistics firms such as FedEx (2017), UPS (2018), Anheuser-Busch (2017) and La Poste (2011) are increasingly incorporating EVs into their commercial fleets, which have historically been comprised of CVs. EVs are also being adopted in home healthcare (Ferrándiz et al. 2016), utilities service (Orlando Utilities Commission 2018), and vehicle repair (Tesla 2018). Despite their increase in popularity, EVs pose operational challenges to which their CV counterparts are immune. For instance, EVs' driving ranges are often much less than that of CVs, charging infrastructure is still relatively sparse compared to the network of refueling stations for CVs, and the time required to charge an EV can range from 30 minutes to several hours – orders of magnitude longer than the time needed to refuel a CV (Pelletier et al. 2016). Companies choosing to adopt electric vehicles require fleet management tools that address these additional challenges.

The operations research community has responded with a body of work on electric vehicle routing problems (E-VRPs), an extension to the existing literature on (conventional) vehicle routing problems (VRPs). E-VRPs address many of the same variants that exist in the VRP domain, such as time-windows, restrictions on freight capacity, mixed fleet, and technician routing; for examples, see Schneider et al. (2014) and Villegas et al. (2018). Nearly all existing E-VRP research makes the assumption that the charging infrastructure utilized by the EVs is privately owned, i.e., that the EV has priority access to the charging infrastructure and may begin charging immediately when it arrives to a charging station (CS). While companies may have a depot at which this assumption holds, most do not have the means to acquire charging infrastructure outside the depot. If companies wish to

[1] The research described in this chapter has been submitted for publication at *Transportation Science* and is currently in the second round of review. For a preprint, see

N. D. Kullman, J. C. Goodson, and J. E. Mendoza. Electric Vehicle Routing with Public Charging Stations. Working paper, Sept. 2019. URL https://hal.archives-ouvertes.fr/hal-01928730

use only the charging infrastructure that is privately-owned, then the EVs are restricted to charging only at the depot. We refer to this recharging strategy as *private-only* or *depot-only*.

Alternatively, we can relax the assumption of using only privately-owned CSs and consider the case where the vehicle may utilize public *extradepot* CSs – those available at locations such as municipal buildings, parking facilities, car dealerships, and grocery stores. We refer to this recharging strategy as *public-private*. At public CSs, all EVs share access to the charging terminals. If a vehicle arrives to charge and finds all terminals in use, it must wait for one to become available or seek another CS. While providing additional flexibility, the public-private strategy introduces uncertainty, which firms often wish to avoid.

Villegas et al. (2018) compares the private-only and public-private strategies in the case of French electricity provider ENEDIS who is replacing a portion of their CV fleet with EVs. Under the assumption of zero waiting times at public charging stations, they find that for the routes which cannot be serviced in a single charge, solutions using the public-private strategy offered savings up to 16% over those using private-only. Despite the suggested savings, ENEDIS chose not to implement the public-private recharging strategy, citing the uncertainty in availability at public charging infrastructure. This reduces the utility of EVs as members of the fleet, potentially impeding their broader adoption.

In an attempt to recapture this lost utility and encourage the use of the public-private recharging strategy, we provide in this work dynamic routing solutions that specifically address the uncertainty at public charging infrastructure. We demonstrate these routing solutions on real instances, using customer data from the ENEDIS instances of Villegas et al. (2018) and charging station data from the French national government. We claim the following contributions in this work:

- We introduce a new variant of the E-VRP: the E-VRP with public-private recharging strategy (E-VRP-PP), where demand at public charging stations is unknown and follows a realistic queuing process. We model the E-VRP-PP as a Markov decision process (MDP) and propose an approximate dynamic programming solution that allows the route planner to adapt to realized demand at public CSs.

- We offer a decomposition of the E-VRP-PP that facilitates the search of good policies. The decomposition allows the use of machinery from static and deterministic routing in solution methods for our stochastic and dynamic routing problem.

- We propose static and dynamic routing policies, including the optimal static policy. To construct static policies, we employ a Benders-based branch-and-cut algorithm to solve the decomposition of the E-VRP-PP. We then incorporate these static policies into dynamic lookaheads (rollouts), in which they serve as base policies.

- Using the same decomposition and Benders-based algorithm in conjunction with an information relaxation, we establish the value of an optimal policy with perfect information, which serves as a bound on the value of the optimal policy.

- In solving the subproblem of the Benders-based algorithm, we address a new variant of the fixed-route vehicle charging problem (FRVCP): the FRVCP with time-dependent

waiting times and discrete charging decisions. In general, FRVCPs deal with the problem of ensuring energy feasibility for electric vehicle routes. We modify the labeling algorithm of Froger et al. (2019) to solve this new variant exactly. Additionally, to help bridge the gap between VRP and E-VRP research, we provide an open-source implementation of Froger et al.'s algorithm for the FRVCP.

- We demonstrate the application of our routing policies and the establishment of bounds on real world instances, which we make publicly available via the Vehicle Routing Problem Repository (VRP-REP) (Mendoza et al. 2014). Further, we show that our routing policies are competitive with the optimal policy, within 11% on average and within 5% for the majority of instances.
- We show that all of our policies under the public-private recharging strategy soundly outperform the optimal solution under the industry-standard private-only recharging strategy, with our best policies offering savings of over 25% on average. These results lend further motivation for companies to adopt the public-private recharging strategy, which may extend EVs' utility in commercial applications and facilitate the transition from internal-combustion engine vehicles.

In addition, we also improve on the perfect information dual bound by developing nonlinear information penalties that punish the decision maker for using information about the future to which they would not naturally have access. While our application of these penalties was limited to a small artificial instance, this success marks a first in vehicle routing, serving as a proof of concept for future research.

The remainder of the paper is organized as follows. We define the problem and formulate the dynamic routing model in §2.2. In §2.3 we review relevant EV routing literature. In §2.4 we explain the role of fixed routes in solving the E-VRP-PP, especially in the context of a decomposition, which we describe in the same section. We then outline our routing policies in §2.5 and detail the derivation of dual bounds for these policies in §2.6. Finally, we discuss computational experiments in §2.7 and provide concluding remarks in §2.8.

2.2 Problem Definition and Model

We address the *electric vehicle routing problem with public-private recharging strategy* (E-VRP-PP). The problem is characterized by making routing and charging decisions for an electric vehicle which visits a set of customers and returns to a depot from which it started. These decisions are subject to energy feasibility constraints. To ensure energy feasibility, the EV may need to stop and charge at CSs at which it may encounter a queue. The objective is to minimize the expected time to visit all customers and return to the depot, including any time spent detouring to, queuing at, and charging at CSs. We define the problem then formulate the MDP model.

2.2.1 Problem Definition

We have a set of known customers $\mathcal{N} = \{1, \ldots, N\}$ and CSs $\mathcal{C} = \{0, N+1, \ldots, N+C\}$ and a single EV. At time 0, the EV begins at the depot, which we denote by node $0 \in \mathcal{C}$.

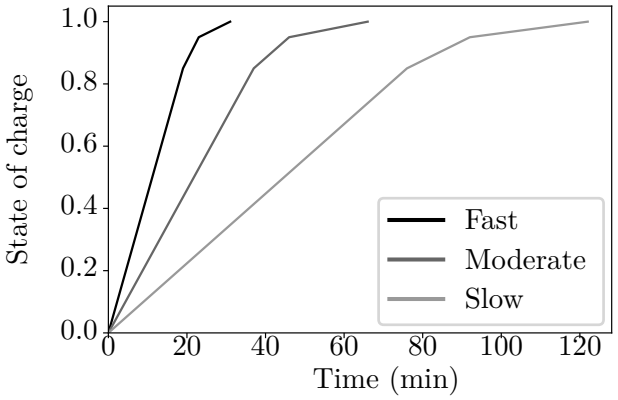

Figure 2.1: The vehicle's charging function for different charging technologies. We assume a concave piecewise-linear charging function as in Montoya et al. (2017).

It then traverses arcs in the complete graph $\mathcal{G}$ with vertices $V = \mathcal{N} \cup \mathcal{C}$. The vehicle must visit each customer and return to the depot. We assume the time and energy required to travel between $i, j \in V$ is deterministic and known to be t_{ij} and e_{ij}, respectively. We also assume the triangle inequality holds, so for any $i, j, k \in V$, we have $t_{ik} \leq t_{ij} + t_{jk}$ and $e_{ik} \leq e_{ij} + e_{jk}$.

To make its journey energy-feasible, the EV may restore its energy at a CS $c \in \mathcal{C}$ before or after customer visits. The depot is private, meaning the EV can always access the charging terminals (or *chargers*) at the depot and may therefore begin charging immediately. In contrast, we assume extradepot CSs $\mathcal{C}' = \mathcal{C} \setminus \{0\}$ are public, so the chargers may be occupied by other EVs. We assume the EV is unaware of the demand at extradepot CSs prior to its arrival. (This represents the worst-case scenario for EV operators, as routing solutions can only improve as more information on CS demand becomes available. Access to real-time data on CS demand, while improving, is also still an exception to the norm.)

If all chargers are occupied when the EV arrives, it must either queue or leave. We model queuing dynamics at extradepot CSs $c \in \mathcal{C}'$ as pooled first-come-first-served systems with ψ_c identical chargers, infinite capacity, and exponential inter-arrival and service times with known rate parameters ($p_{c,\text{arrive}}$ and $p_{c,\text{depart}}$, respectively): $\text{M/M}/\psi_c/\infty$. If a vehicle queues at a CS it must remain in queue until a charger becomes available, after which it must charge. When the EV charges, it may restore its energy to a capacity $q \in \mathcal{Q}$, where $\mathcal{Q}$ is a set of discrete energy levels, such as every 10% (in which case $\mathcal{Q} = \{0, 0.1Q, \ldots, 0.9Q, Q\}$). We assume a concave piecewise-linear charging function where the EV accumulates charge faster at lower energies than at higher energies (see Figure 2.1). These piecewise-linear charging functions were shown in Montoya et al. (2017) to be a good approximation of actual performance. In the same study, the authors also demonstrate that the use of a simple linear approximation leads to solutions that may be either overly expensive or infeasible. We assume that the energy levels of the breakpoints in the piecewise-linear charging functions are also elements in $\mathcal{Q}$.

2.2.2 Model

We formulate the E-VRP-PP as an MDP whose components are defined as follows.

States.

An epoch $k \in \{0, \ldots, K\}$ is triggered when a vehicle arrives to a new location, reaches the front of the queue at a CS, or completes charging. At each epoch we describe the state of the system by the vector $s_k = \left(t_k, i_k, i_{k-1}, q_k, q_{k-1}, \bar{\mathcal{N}}_k, z_k\right)$, which contains all information required for making routing and charging decisions: the current time $t_k \in \mathbb{R}_{\geq 0}$; the vehicle's current location and its location in the previous epoch $i_k, i_{k-1} \in V$; the energy currently in the vehicle's battery and at the beginning of the previous epoch $q_k, q_{k-1} \in [0, Q]$; the set of customers that have not yet been visited $\bar{\mathcal{N}}_k \subseteq \mathcal{N}$; and the vehicle's position in queue at its current location $z_k \in \mathbb{N}$. (We require the previous-epoch location and charge to filter out certain illegal actions; see equations (2.2)-(2.5).) We define $z_k = 1$ when $i_k \in \{0\} \cup \mathcal{N}$. This definition of the system state yields the state space $\mathcal{S} = \mathbb{R}_{\geq 0} \times V \times V \times [0, Q] \times [0, Q] \times \mathcal{N} \times \mathbb{N}$. The system is initialized in epoch 0 at time 0 with the vehicle at the depot, the battery at maximum capacity, and all customers yet to be visited:

$$s_0 = (0, 0, 0, Q, 0, \mathcal{N}, 1)\,. \tag{2.1}$$

The problem ends at some epoch K when all customers have been visited and the EV returns to the depot: $s_K \in \{(t_K, 0, i_{K-1}, q_K, q_{K-1}, \emptyset, 1) | t_K \in \mathbb{R}_{\geq 0}; i_{K-1} \in V; q_K, q_{K-1} \in [0, Q]\}$.

Actions.

Given a pre-decision state s_k in some epoch k, the action space $\mathcal{A}(s_k)$ defines the possible actions that may be taken from that state. Informally, $\mathcal{A}(s_k)$ consists of energy-feasible routing and charging decisions. We define actions $a \in \mathcal{A}(s_k)$ to be location-charge pairs

$a = (a^i, a^q)$ and formally define the action space as

$$\begin{aligned}
\mathcal{A}(s_k) = \Big\{ & (a^i, a^q) \in \{\bar{\mathcal{N}}_k \cup \mathcal{C}\} \times [0, Q] : \\
& a^i = i_k, a^q = q_k, \\
& \quad i_k \in \mathcal{C}' \wedge \psi_{i_k} < z_k && (2.2) \\
& a^i = i_k, a^q \in \Big\{ \tilde{q} \in \mathcal{Q} \Big| \, \tilde{q} > q_k \wedge \\
& \Big(\Big(\exists c \in \mathcal{C}, \exists j \in \bar{\mathcal{N}}_k : \, \tilde{q} \geq e_{i_k j} + e_{jc} \Big) \vee \Big(\bar{\mathcal{N}}_k = \emptyset \wedge \tilde{q} \geq e_{i_k 0} \Big) \Big) \Big\}, \\
& \quad i_k \in \mathcal{C} \wedge z_k \leq \psi_{i_k} \wedge q_k \leq q_{k-1} && (2.3) \\
& a^i \in \bar{\mathcal{N}}_k, a^q = q_k - e_{i_k a^i}, \\
& \quad (\exists c \in \mathcal{C} : a^q \geq e_{a^i c}) \wedge (i_k \neq i_{k-1} \vee q_k \neq q_{k-1}) && (2.4) \\
& a^i \in \mathcal{C} \setminus \{i_k\}, a^q = q_k - e_{i_k a^i}, \\
& \quad (i_k \neq i_{k-1} \vee q_k \neq q_{k-1}) \wedge q_k \geq e_{i_k a^i} \\
& \quad \wedge (q_k > q_{k-1} \Rightarrow (k = 0 \vee (\bar{\mathcal{N}}_k = \emptyset \wedge a^i = 0))) \Big\}. && (2.5)
\end{aligned}$$

Equation (2.2) defines the queuing action, in which the vehicle waits in queue until a charger becomes available. In this case, its location and charge remain unchanged. Queuing actions are feasible when the EV is at an extradepot charging station without available chargers. Equation (2.3) defines the charging actions. The allowable charge levels are those which are greater than the EV's current charge and which allow the EV to reach a customer and a subsequent CS (unless $\bar{\mathcal{N}}_k = \emptyset$, in which case it must charge enough to be able to reach the depot). Charging actions are present in the action space when the vehicle resides at a charging station with an available charger and did not charge in the previous epoch. Equation (2.4) defines routing decisions to unvisited customers. These actions are permitted so long as the vehicle has sufficient charge to reach the customer and a subsequent CS. In addition, we require that the vehicle must not have queued in the previous epoch, because we require that an EV always charge after queuing. Finally, equation (2.5) defines routing decisions to charging station nodes. Again, we require that the vehicle not have queued in the previous epoch and that it have sufficient charge to reach the charging station. We also disallow visits to CSs after the vehicle has just charged except when the EV is initially departing the depot and when it has served all customers and is en route to terminate at the depot.

Pre-to-Post-decision Transition.

Following the selection of an action $a = (a^i, a^q) \in \mathcal{A}(s_k)$ from the pre-decision state s_k, we undergo a deterministic transition to the post-decision state $s_k^a = \left(t_k^a, i_k^a, i_{k-1}^a, q_k^a, q_{k-1}^a, \bar{\mathcal{N}}_k^a, z_k^a\right)$. In s_k^a we have updated the vehicle's previous location and charge to the location and charge in epoch k: $i_{k-1}^a = i_k$, $q_{k-1}^a = q_k$. The vehicle's new current location and charge are inherited from the action: $i_k^a = a^i$, $q_k^a = a^q$. Finally, we update the set of unvisited

customers: $\bar{\mathcal{N}}_k^a = \bar{\mathcal{N}}_k \setminus \{a^i\}$. The time and position in queue remain unchanged from the pre-decision state.

Information and Post-to-Pre-decision Transition.

The system transitions from a post-decision state s_k^a to a pre-decision state s_{k+1} when one of the following events occurs to trigger the next decision epoch: the vehicle reaches a new location, the vehicle reaches the front of a queue, or the vehicle completes a charging operation. At this time, we update position in queue and the time, which were unchanged in the pre-to-post-decision transition. In the first two epoch-triggering events, our transition to s_{k+1} may be stochastic and depend on the observation of exogenous information. For instance, if in the first case we arrive to an extradepot CS, then we observe exogenous information in the form of the queue length. In the second case, when we have waited at an extradepot CS, we observe the time the vehicle waits before a charger becomes available.

We define the exogenous information observed in epoch k to be W_{k+1}, a pair consisting of a time and position in queue: $W_{k+1} = (w^t, w^z)$. The set of all exogenous information that may be observed given a post-decision state is called the information space $\mathcal{I}(s_k^a)$ and is defined as

$$\begin{aligned}
\mathcal{I}(s_k^a) = \Big\{ & (w^t, w^z) \in \mathbb{R}_{\geq 0} \times \mathbb{N} : \\
& w^t = t_k^a + t_{i_{k-1}^a i_k^a}, w^z = 1, \\
& \quad i_k^a \in \mathcal{N} \cup \{0\} \wedge i_k^a \neq i_{k-1}^a && (2.6) \\
& w^t = t_k^a + \bar{u}(q_{k-1}^a, q_k^a), w^z = z_k^a, \\
& \quad q_k^a > q_{k-1}^a && (2.7) \\
& w^t = t_k^a + t_{i_{k-1}^a i_k^a}, w^z \in \mathbb{N}, \\
& \quad i_k^a \neq i_{k-1}^a \wedge i_k^a \in \mathcal{C}' && (2.8) \\
& w^t \in (t_k^a, \infty), w^z = \psi_{i_k^a}, \\
& \quad i_k^a = i_{k-1}^a \wedge q_k^a = q_{k-1}^a \Big\}, && (2.9)
\end{aligned}$$

where $\bar{u} : [0, Q]^2 \to \mathbb{R}_{\geq 0}$ is a function defining the time required to charge from some energy level q_{initial} to another charge level q_{final} according to the vehicle's charging function. We assume a concave piecewise-linear charging function as shown in Figure 2.1.

Equations (2.6) and (2.7), respectively, define the (deterministic) information observed when the vehicle arrives to the depot or to a customer and when the vehicle completes a charging operation. In equation (2.6), the observed time is simply the previous time plus the travel time to reach the new node, and the vehicle's position is one by definition. In equation (2.7), we update the time to account for the time that the vehicle spent charging, and there is no update to the vehicle's position in queue, which we assume to be the same as when it began charging. The information defined by equations (2.8) and (2.9) involves uncertainty in queue dynamics at extradepot CSs $c \in \mathcal{C}'$. In equation (2.8), the EV has just arrived to an extradepot CS, so the time is deterministic, but we observe an unknown

queue length. In equation (2.9), the EV has finished queuing. We assume the vehicle now occupies the last ($\psi_{i_k^a}$-th) charger, but the time of the next epoch is unknown.

Given exogenous information $W_{k+1} = (w^t, w^z)$ and post-decision state s_k^a, we transition to the pre-decision state s_{k+1} where $t_{k+1} = w^t$ and $z_{k+1} = w^z$. The other state components, all of which were updated in the transition to the post-decision state, remain the same.

Contribution Function.

When we select an action $a = (a^i, a^q)$ from a pre-decision state s_k, we incur cost

$$C(s_k, a) = \begin{cases} t_{i_k a^i} & a^i \neq i_k \\ \bar{u}(q_k, a^q) & a^q > q_k \quad x \\ (z_k - \psi_{i_k})/(\psi_{i_k} p_{i_k,\text{depart}}) & \text{otherwise.} \end{cases} \tag{2.10}$$

In equation (2.10), the first case corresponds to traveling to a new node, for which we incur cost equal to the travel time to reach the node. In the second case, the action is charging, and we incur cost equal to the charging time. Finally, in the third case, we have chosen to wait in queue, for which we incur cost equal to the expected waiting time conditional on the queue length.

Objective Function.

Let Π denote the set of Markovian deterministic policies, where a policy $\pi \in \Pi$ is a sequence of decision rules $(X_0^\pi, X_1^\pi, \ldots, X_K^\pi)$ where each $X_k^\pi : s_k \to \mathcal{A}(s_k)$ is a function mapping a state to an action. We seek an optimal policy $\pi^\star \in \Pi$ that minimizes the expected total cost of the tour conditional on the initial state:

$$\tau(\pi^\star) = \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0 \right]. \tag{2.11}$$

In our solution methods, it is often convenient to think of a policy beginning from a given pre-decision state $s_{k'}$. In this case, a policy is defined as a set of decision rules from epoch k' onwards (e.g., $\left(X_{k'}^\pi, X_{k'+1}^\pi, \ldots, X_K^\pi\right)$), and its objective value is equivalent to Equation (2.11) but with the summation beginning at epoch k' and the expectation conditional on initial state $s_{k'}$.

2.3 Literature Review

The body of literature on electric vehicle routing is growing quickly. Our review first considers some of the seminal works in E-VRPs before concentrating specifically on those that consider public charging stations and dynamic solution methods. For a more in-depth review of the E-VRP literature, we refer the reader to Pelletier et al. (2016).

The Green VRP introduced by Erdoğan and Miller-Hooks (2012) is often cited as the origin of E-VRPs. The authors use mixed-integer-linear programming to assign routing

and refueling decisions for a homogeneous fleet of alternative fuel vehicles. In the work, a number of simplifying assumptions are made that are difficult to justify for electric vehicles, such as that vehicles always fully restore their energy when they refuel and that refueling operations require constant time. The latter assumption was addressed in Schneider et al. (2014) who focus specifically on electric vehicle routing. They propose an E-VRP with time windows and capacity constraints (E-VRP-TW) for which they offer a heuristic solution. While still requiring full recharges, they relax the constant-time assumption for charging operations, instead assuming that the time required to perform these recharging operations is linear with the amount of energy to be restored. Desaulniers et al. (2016) offer exact solution methods for four variants of the E-VRP-TW and additionally relax the assumption on full recharging: two of the E-VRP-TW variants they address allow partial recharging operations. These operations are again assumed to take linear time with respect to the restored energy. In their work on the E-VRP with nonlinear charging functions, Montoya et al. (2017) demonstrate that the assumption of linear-time recharging operations can lead to infeasible or overly-expensive solutions. The aforementioned studies assume a homogeneous fleet of vehicles, but heterogeneous fleets consisting (at least in part) of EVs have also been considered in a number of studies, including Goeke and Schneider (2015); Hiermann et al. (2016); Hiermann et al. (2019a); and Villegas et al. (2018). A number of additional E-VRP variants, such as those considering location-routing (Schiffer et al. 2018), congestion (Omidvar and Tavakkoli-Moghaddam 2012), and public transportation (Barco et al. 2017) have also been studied.

Despite the breadth of variants addressed, a common shortcoming in existing E-VRP studies is the lack of consideration of access to public charging infrastructure. Instead, studies generally make one of the two following assumptions: that the vehicles charge only at the depot (they adopt the private-only recharging strategy); or they allow extradepot (public-private) recharging, but the extradepot stations behave as if they were private, allowing EVs to begin charging immediately upon their arrival. Operating under the latter assumption promises solutions that are no worse than those under the former, as it simply enlarges the set of CSs at which EVs may charge. However, in reality, the adoption of the public-private recharging strategy introduces uncertainty and risk, and current E-VRP solution methods do not address this. As evidenced in Villegas et al. (2018), this leads companies to prefer the private-only approach despite results suggesting that the public-private approach offers better solutions. Having access to a dynamic routing solution capable of responding to uncertainty may encourage companies to utilize public CSs. However, such solutions are lacking, as research on dynamic routing of EVs is limited.

In a recent review of the dynamic routing literature by Psaraftis et al. (2016), the authors note the current dearth of dynamic EV routing research, citing only one study (Adler and Mirchandani 2014) and acknowledging that it would be more properly classified as a path-finding problem than a VRP. In that study, Adler and Mirchandani (2014) consider EVs randomly requesting routing guidance and access to a network of battery-swap stations (BSSs). The work addresses the problem from the perspective of the owner of the BSSs, aiming to minimize average total delay for all vehicles requesting guidance. Because reservations are made for EVs as they request and receive routing guidance, waiting times for the EVs at the BSSs are known in advance, eliminating uncertainty in their total travel time. A more recent study by Sweda et al. (2017) considers a path-finding problem

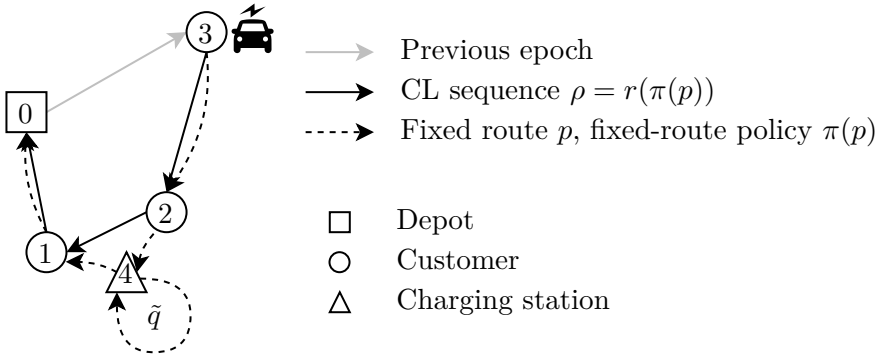

Figure 2.2: Shown is an EV that relocated from the depot to customer 3 in epoch 0. The CL sequence ρ (solid black arrows) considered by the vehicle from its current state is $(3, 2, 1, 0)$. The fixed route p (dashed black arrows) includes a detour to CS 4 where it charges to $\tilde{q}$, as indicated by the self-directed arc at 4 (p is given by equation (2.12)).

in which a vehicle travels from an origin to a destination on a network with CSs at every node, and where each CS has a probability of being available and some expected waiting time (known a priori to the planner) if it is not. The decision maker dynamically decides the vehicle's path and recharging decisions so as to arrive at the destination as quickly as possible. The authors provide analytical results, including the optimal a priori routing policy. However, similar to Adler and Mirchandani (2014), the problem addressed more closely aligns with the family of path-finding problems rather than VRPs. Thus, a review of the literature reveals little existing work on dynamic E-VRPs. We seek to contribute to this domain with our research here.

2.4 Fixed Routes in the E-VRP-PP

We call a *fixed route* a complete set of routing and charging instructions from some origin node to a destination node, through some number of CSs and customer locations, that is prescribed to a vehicle prior to its departure. We often think of fixed routes in the context of static routing (e.g. Campbell and Thomas (2008)), but we can map them to dynamic routing as well, where a fixed route represents a predetermined sequence of actions from some state s_k to a terminal state s_K. The expected cost of a fixed route is the expected sum of the costs of these actions, which we can use as an estimate of the expected cost-to-go from s_k, the route's starting state. This makes fixed routes a useful tool in solving dynamic routing problems, such as the E-VRP-PP. In the coming sections, we show how fixed routes can be used to develop both static and dynamic policies, as well as establish dual bounds. In this section, we first formalize the concept of fixed routes for the E-VRP-PP in §2.4.1, then introduce a decomposition that facilitates the search for good fixed routes in §2.4.2. The decomposition is conducive to solving via classical methods from static and deterministic routing, which we detail in §2.4.3 and §2.4.4.

2.4.1 Definitions and AC Policies

Fixed Routes. In the E-VRP-PP, a fixed route consists of a set of instructions specifying the order in which to visit nodes $v \in V$ and to which $\tilde{q} \in \mathcal{Q}$ to charge when visiting CS nodes. Formally, we define a fixed route p to be a sequence of directions: $p = (p_1, p_2, \ldots, p_{|p|})$, where each direction $p_j = (p_j^i, p_j^q)$ is a location-charge pair. Let us consider a vehicle in the state $s_1 = (t_{0,3}, 3, 0, Q - e_{0,3}, Q, \{1, 2\}, 1)$, as in Figure 2.2. We might consider the fixed route

$$p = ((3, Q - e_{0,3}), (2, Q - e_{0,3} - e_{3,2}), (4, Q - e_{0,3} - e_{3,2} - e_{2,4}), (4, \tilde{q}), (1, \tilde{q} - e_{4,1}), (0, \tilde{q} - e_{4,1} - e_{1,0}))\,, \tag{2.12}$$

which consists of routing instructions to the remaining unvisited customers $\bar{\mathcal{N}}_1$, as well as a detour to CS 4 at which the vehicle charges to some energy $\tilde{q} \in \mathcal{Q}$.

Fixed-Route Policies. The sequence of directions comprising a fixed route p constitutes a *fixed-route policy*, equivalently, a *static policy* $\pi(p) \in \Pi$, which is defined by decision rules

$$X_k^{\pi(p)}(s_k) = \begin{cases} p_{j^\star - 1} & i_k \in \mathcal{C}' \wedge \left(p_{j^\star}^q > p_{j^\star - 1}^q \wedge z_k > \psi_{i_k}\right) \\ p_{j^\star} & \text{otherwise,} \end{cases} \tag{2.13}$$

where $j^\star$ is the index of the next direction in p to be followed by the vehicle. Specifically, for state s_k, $j^\star$ is the index in p such that $\left(i_k = p_{j^\star - 1}^i \wedge q_k = p_{j^\star - 1}^q \wedge \bar{\mathcal{N}}_k = \left(\bigcup_{l=j^\star}^{|p|} p_l^i\right) \setminus \mathcal{C}'\right)$. Equation (2.13) simply directs the vehicle to follow the fixed route p. The first case addresses waiting actions which are not explicitly outlined in the routing instructions. If the vehicle encounters a queue at a CS at which it is instructed to charge, fixed-route policies dictate that it simply wait until a charger is available. The second case handles all other decision making as instructed by the fixed route. If we again consider the example in Figure 2.2 with fixed route p given by equation (2.12), then the corresponding fixed-route policy $\pi(p)$ would consist of the following sequence of decision rules and resulting actions:

$$\pi(p) = \Big(X_1^{\pi(p)}(s_1) = (2, q_1 - e_{3,2}), X_2^{\pi(p)}(s_2) = (4, q_2 - e_{2,4}), X_3^{\pi(p)}(s_3) = (4, \tilde{q})^*,$$
$$X_4^{\pi(p)}(s_4) = (1, \tilde{q} - e_{4,1}), X_5^{\pi(p)}(s_5) = (0, q_5 - e_{1,0})\Big).$$

The asterisk on action $(4, \tilde{q})$ in the third epoch indicates the potential presence of an additional prior epoch: if the vehicle arrives to CS 4 and there is a queue, then the vehicle must first wait before it can charge; in this case, an epoch $X_3^{\pi(p)}(s_3) = (4, q_2 - e_{2,4})$ is inserted, and the subsequent decision rules are shifted back (e.g., $X_5^{\pi(p)}$ becomes $X_6^{\pi(p)}$). Note that if we know waiting times in advance (see §2.6.2), then the existence of a waiting epoch would be known a priori.

From a state s_k, the set of all fixed-route or static policies is $\Pi^{\mathrm{S}} \subseteq \Pi$, defined as the set $\Pi^{\mathrm{S}} = \{\pi(p) \in \Pi |\, p \in P\}$ where P is the set of all feasible fixed routes (for a formal definition of P, see §2.4.1.1). We refer to such policies as static, because they offer no meaningful way in which to respond to uncertainty.

Paths and Compulsory-Location Sequences. Given a fixed-route policy $\pi(p) \in \Pi^{\mathrm{S}}$, let us denote by $R(\pi(p))$ the sequence of locations visited, which we call a *path*: $R(\pi(p)) = (p^i_j)_{j\in\{1,\dots,|p|\}}$. In the above example, the path is $R(\pi(p)) = (3, 2, 4, 4, 1, 0)$. Notice that some of the locations in the path $R(\pi(p))$ must be visited by all valid fixed-route policies initialized from state s_1. Namely, all fixed-route policies have to include the vehicle's starting point ($p^i_1 = 3$), its ending point (the depot, $p^i_{|p|} = 0$), and the unvisited customers (1 and 2). We denote by $r(\pi(p))$ the subsequence of $R(\pi(p))$ consisting of only these locations: $r(\pi(p)) = (3, 2, 1, 0)$. In general,

$$r(\pi(p)) = p^i_1 \oplus (p^i_j : p^i_j \in \mathcal{N})_{1<j<|p|} \oplus p^i_{|p|} \tag{2.14}$$

("$\oplus$" is the concatenation operator). We call sequences like $r(\pi(p))$ *compulsory-location* (CL) *sequences*. Paths $R(\pi(p))$ contain additional elements for charging operations that occur while traversing a CL sequence. If it were energy-feasible for a vehicle to traverse a CL sequence $r(\pi(p))$ directly, then there would be no need to perform charging operations along the way, so $r(\pi(p))$ would be equivalent to $R(\pi(p))$ (under a good policy). That is, we can think of fixed routes and paths as being the energy-feasible analogs to CL sequences.

Our current definition of CL sequences (2.14) requires a fixed route p. However, we may also consider CL sequences from the perspective of a vehicle that does not currently have a prescribed fixed route. Recall that CL sequences begin with the vehicle's current location, end with the depot, and must contain in between the unvisited customers. We define the set of all CL sequences from a state s_k as those in the set $\mathcal{R}(s_k) = \{i_k \frown S_i \frown 0 : S_i \in \mathrm{Sym}(\bar{\mathcal{N}}_k)\}$, where $\mathrm{Sym}(\bar{\mathcal{N}}_k)$ is the set of all permutations of unvisited customers $\bar{\mathcal{N}}_k$. For our example with a vehicle occupying state s_1 in Figure 2.2, the set of CL sequences is $\mathcal{R}(s_1) = \{(3, 1, 2, 0), (3, 2, 1, 0)\}$. In general, we refer to CL sequences in $\mathcal{R}(s_k)$ by $\rho = (\rho_1, \dots, \rho_{|\rho|})$, where $\rho_1 = i_k$ and $\rho_{|\rho|} = 0$.

AC Policies. Without loss of optimality, we can restrict our search of fixed-route polices (and fixed routes) to those that always charge when visiting a charging station. We refer to policies meeting this criterion as *AC policies* $\Pi^{\mathrm{AC}} \subseteq \Pi^{\mathrm{S}}$ (AC for "always charge"). The set of AC policies is defined as $\Pi^{\mathrm{AC}} = \Pi^{\mathrm{S}} \setminus \Pi^{\mathrm{B}}$, where $\Pi^{\mathrm{B}} = \left\{\pi(p) \in \Pi^{\mathrm{S}} \,\middle|\, p \in P \wedge \left(\exists j \in \{2, \dots, |p| - 1\} : p^i_j \in \mathcal{C} \wedge p^i_{j-1} \neq p^i_j \wedge p^i_j \neq p^i_{j+1}\right)\right\}$ is the set of static policies that include CS visits at which no charging is performed. Note that all static policies belong to either Π^{B} or Π^{AC} (but not both). We justify the restriction of static policies to AC policies in the proof of Theorem 1. Going forward, all mention of static or fixed-route policies, unless explicitly stated otherwise, refers to those that are AC.

Theorem 1 (Good fixed-route policies are AC Policies)**.** *For all static, non-AC-policies $\pi \in \Pi^S \setminus \Pi^{AC}$, there exists an AC policy $\pi^{AC} \in \Pi^{AC}$ whose objective value is no worse: $\tau(\pi^{AC}) \leq \tau(\pi)$.*

Proof. Proof. In order for a policy $\pi \in \Pi^{\mathrm{B}}$ to be non-AC, it must visit CSs without charging at them. We refer to this as "balking" a CS. Consider a vehicle operating under the static non-AC policy π which balks CSs. We wish to show that there exists a static AC-policy π^{AC} such that $\tau(\pi^{\mathrm{AC}}) \leq \tau(\pi)$. We can trivially construct such a policy by simply mimicking π, except when π balks a CS. In that case, the constructed policy π^{AC} would skip visiting the balked CS and proceed directly to the subsequent location. For

instance, if the static policy π dictates the relocation from some node j to a charging station c and then immediately relocate to j', policy π^{AC} would proceed directly from j to j'. In so doing, the objective value of policy π^{AC} will differ from that of π by an amount $t_{jc} + t_{cj'} - t_{jj'}$. Because the triangle inequality holds for travel times and queues are served first-in-first-out (FIFO), this policy will have expected cost no larger than that of π. □

The intuition is that because static policies follow a predetermined set of actions, visiting a charging station without the intent to charge serves no purpose except to increase the time required to complete the route. In the case of dynamic policies, they may visit a charging station and ultimately balk, but this would be in response to the observation of the queue length at the charging station, rather than a premeditated immediate departure. The construction strategy for π^{AC} in the proof requires knowledge of these immediate departures a priori, so it is therefore only valid in the context of static policies.

2.4.1.1 Defining the set of fixed routes

To define the set P of all possible fixed routes from a state $s_{k'}$, we first define the modified action space $\mathcal{A}^-(s_k)$, which allows the EV to start charging immediately regardless of queue length:

$$
\begin{aligned}
\mathcal{A}^-(s_k) = \Bigg\{ & (a^i, a^q) \in \{\bar{\mathcal{N}}_k \cup \mathcal{C}\} \times [0, Q] : \\
& a^i = i_k, a^q \in \Bigg\{ \tilde{q} \in \mathcal{Q} \Bigg| \tilde{q} > q_k \wedge \\
& \Big(\Big(\exists c \in \mathcal{C}, \exists j \in \bar{\mathcal{N}}_k : \ \tilde{q} \geq e_{i_k j} + e_{jc} \Big) \vee \Big(\bar{\mathcal{N}}_k = \emptyset \wedge \exists c \in \mathcal{C} : \ \tilde{q} \geq e_{i_k c} \wedge e_{c0} \leq Q \Big) \Big) \Bigg\}, \\
& \quad i_k \in \mathcal{C} \wedge q_k \leq q_{k-1} \qquad (2.15) \\
& a^i \in \bar{\mathcal{N}}_k, a^q = q_k - e_{i_k a^i}, \\
& \quad (\exists c \in \mathcal{C} : a^q \geq e_{a^i c}) \qquad (2.16) \\
& a^i \in \mathcal{C} \setminus \{i_k\}, a^q = q_k - e_{i_k a^i}, \\
& \quad q_k \geq e_{i_k a^i} \\
& \quad \wedge (q_k > q_{k-1} \Rightarrow (k = 0 \vee (\bar{\mathcal{N}}_k = \emptyset \wedge a^i = 0))) \Bigg\}. \qquad (2.17)
\end{aligned}
$$

In contrast to the definition of $\mathcal{A}(s_k)$, there is no condition on position in queue in order to charge in (2.15). Further, we remove waiting actions from the action space, so we also remove the conditions on not relocating if having just waited in equations (2.16) and (2.17). In addition, we define $\mathcal{S}^-(s_k, a)$ to be the set of reachable states in epoch $k+1$ when choosing action a from state s_k and when the exogenous information observed is $W_{k+1} \in \{(w^t, w^z) | (w^t, w^z) \in \mathcal{I}(s_k^a) \wedge w^z = 1\}$ (effectively, we ignore any information regarding position in queue, assuming it is 1 everywhere we go). Then we may define the set P of all fixed routes from a state $s_{k'}$ recursively as follows:

$$P = \{(p_1, p_2, \ldots, p_D) | p_j \in P_j, \ 1 \leq j \leq D\},$$

where D is the (variable) index of the terminal direction and

$$\begin{aligned} P_1 &= \{(i_{k'}, q_{k'})\} \\ P_2 &= \mathcal{A}^-(s_{k'}) \\ &\vdots \\ P_j &= \bigcup_{s' \in \mathcal{S}^-(s_{(k'+j-3)}, p_{j-1})} \mathcal{A}^-(s') \\ &\vdots \\ P_D &= \{(0, q) | q \in [0, Q]\} \end{aligned}$$

2.4.2 Decomposition of the E-VRP-PP

Because fixed routes are central to our development of solution methods for the E-VRP-PP, we seek ways to establish good fixed routes. To do so, we leverage a decomposition of the problem into routing and charging decisions. Let us assume a vehicle occupies some state s_k. For a given CL sequence $\rho \in \mathcal{R}(s_k)$, we may search over the corresponding set of fixed-route policies, $\Pi_\rho \subseteq \Pi^{\text{AC}}$, where $\Pi_\rho = \{\pi(p) \in \Pi^{\text{AC}} : r(\pi(p)) = \rho\}$. Note that it is possible that for some ρ, the set Π_ρ will be empty. That is, there may exist CL sequences such that there does not exist an energy-feasible way in which to traverse the sequence. We offer the following decomposition:

Theorem 2. *For AC policies beginning in a state s_k, the E-VRP-PP can be decomposed into routing and charging decisions with objective*

$$\min_{\pi(p) \in \Pi^{AC}} \mathbb{E}\left[\sum_{k'=k}^{K} C(s_{k'}, X_{k'}^{\pi(p)}(s_{k'}))\right] = \min_{\rho \in \mathcal{R}(s_k)} \left\{ \min_{\pi \in \Pi_\rho} \mathbb{E}\left[\sum_{k'=k}^{K} C(s_{k'}, X_{k'}^{\pi}(s_{k'}))\right] \right\}. \quad (2.18)$$

Proof. Proof. Because each AC policy $\pi(p) \in \Pi^{\text{AC}}$ maps to a CL sequence $r(\pi(p))$ given by equation (2.14), we may equivalently write the set of AC policies as $\Pi^{\text{AC}} = \bigcup_{\rho \in \mathcal{R}(s_k)} \Pi_\rho$, where $\Pi_\rho = \{\pi(p) \in \Pi^{\text{AC}} : r(\pi(p)) = \rho\}$. This partitioning of the policy set allows us to write the objective function as a nested minimization over CL sequences and their corresponding fixed-route policies:

$$\min_{\pi(p) \in \Pi^{\text{AC}}} \mathbb{E}\left[\sum_{k'=k}^{K} C(s_{k'}, X_{k'}^{\pi(p)}(s_{k'}))\right] = \min_{\rho \in \mathcal{R}(s_k)} \left\{ \min_{\pi \in \Pi_\rho} \mathbb{E}\left[\sum_{k'=k}^{K} C(s_{k'}, X_{k'}^{\pi}(s_{k'}))\right] \right\}.$$

□

A solution to Equation (2.18) is an optimal fixed route – equivalently, an optimal fixed-route policy – whose cost provides an estimate of the cost-to-go from the route's starting state s_k. We exploit this in the construction of routing policies as well as in the establishment of dual bounds, where it aids in the computation of the value of an optimal policy with perfect information.

2.4.3 Solving the Decomposed E-VRP-PP

To solve Equation (2.18) we employ a Benders-like decomposition, taking the outer minimization over CL sequences as the master problem and the inner minimization over charging decisions as the sub-problem. Specifically, we use a *Benders-based branch-and-cut* algorithm in which at each integer node of the branch-and-bound tree of the master problem, the solution is sent to the subproblem for the generation of Benders cuts. We discuss the master problem in §2.4.3.1 and the subproblem and the generation of cuts in §2.4.3.2.

2.4.3.1 Master Problem: Routing.

The master problem corresponds to the outer minimization of Equation (2.18) in which we search over CL sequences. CL sequences are comprised of elements in the set $\mathcal{M}_k = i_k \cup \bar{\mathcal{N}}_k \cup \{0\}$. The master problem approximates the cost of traversing a CL sequence $\rho \in \mathcal{R}(s_k)$ by its *direct-travel cost* $T_D(\rho) = \sum_{j=1}^{|\rho|-1} t_{\rho_j,\rho_{j+1}}$. This approximation gives the cost of traversing ρ without charging.

To search CL sequences, we use a subtour-elimination formulation of the TSP (Dantzig et al. 1954) over the nodes in the subgraph of $\mathcal{G}$ with vertex set $\mathcal{M}_k$. This yields the following master problem:

$$\text{minimize} \quad \sum_{i\in\mathcal{M}_k}\sum_{j\in\mathcal{M}_k} t_{ij}x_{ij} + \theta \tag{2.19}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{M}_k} x_{ij} = 1, \quad \forall i \in \mathcal{M}_k \tag{2.20}$$

$$\sum_{i\in\mathcal{M}_k} x_{ij} = 1, \quad \forall j \in \mathcal{M}_k \tag{2.21}$$

$$x_{ii} = 0, \quad \forall i \in \mathcal{M}_k \tag{2.22}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1, \quad \forall S \subset \mathcal{M}_k, |S| \geq 2 \tag{2.23}$$

$$x_{ij} \in \{0,1\}, \theta \geq 0 \tag{2.24}$$

If the vehicle is not initially at the depot (if $i_k \neq 0$), we add the constraint

$$x_{0i_k} = 1, \tag{2.25}$$

and set $t_{0i_k} = 0$, ensuring that the CL sequence ends at the depot and begins at i_k. Constraints (2.20) and (2.21), respectively, ensure that the vehicle departs from and arrives to each node exactly once; and constraints (2.22) prohibit self-directed arcs. Constraints (2.23) are the subtour elimination constraints, and (2.24) defines variables' scopes.

The binary variables x_{ij} take value 1 if node i immediately precedes node j in the CL sequence. A solution to the master problem is denoted by $\mathbf{x}$, and we call the subset of variables that take nonzero value $\mathbf{x}_\rho = \{x_{ij} | x_{ij} = 1\}$. The variables $\mathbf{x}_\rho$ define a CL sequence ρ, with $\rho_1 = i_k$ and all other ρ_i equal to the element in the singleton $\{j | x_{\rho_{i-1}j} \in \mathbf{x}_\rho\}$.

In addition to the direct travel cost of ρ, the objective function (2.19) contains the variable θ whose value reflects the additional cost of making the traversal of ρ energy

feasible. To improve the master problem's estimation of this cost, we add valid inequalities for the minimum time that must be spent detouring to and recharging at charging stations:

$$\sum_{i\in\mathcal{M}_k}\sum_{j\in\mathcal{M}_k}\left(e_{ij}x_{ij}+m^e_{ij}y_{ij}\right)-q_k\leq e_R \tag{2.26}$$

$$\frac{1}{r^\star}e_R\leq t_R \tag{2.27}$$

$$\frac{1}{Q}e_R\leq n_e \tag{2.28}$$

$$\sum_{i\in\mathcal{M}_k}\sum_{j\in\mathcal{M}_k}y_{ij}\geq n_e \tag{2.29}$$

$$y_{ij}\leq x_{ij}\qquad \forall i,j\in\mathcal{M}_k \tag{2.30}$$

$$\sum_{i\in\mathcal{M}_k}\sum_{j\in\mathcal{M}_k}m^t_{ij}y_{ij}\leq t_D \tag{2.31}$$

$$\theta\geq t_D+t_R \tag{2.32}$$

$$y_{ij}\in[0,1];e_R,n_e,t_D,t_R\geq 0 \tag{2.33}$$

We have introduced new variables: t_D and t_R are the minimum time spent detouring to and recharging at charging stations, e_R is the minimum energy that must be recharged, n_e is the minimum number of recharging events that must occur, and y_{ij} are variables indicating whether a recharging event should occur between nodes i and j. We have also introduced parameters m^e_{ij} and m^t_{ij} equal to the minimum energy and time to detour to a charging station between nodes i and j, as well as $r^\star$ which is the fastest recharging rate across all charging stations.

Equation (2.26) sets a lower bound for the amount of energy that must be recharged, and Equation (2.27) uses this to set a lower bound for the amount of time that the vehicle must spend recharging. Equation (2.28) sets a lower bound for the number of recharging events n_e that must occur, and Equation (2.29) requires the sum of insertion variables y_{ij} to be at least that amount. Equation (2.30) ensures that we only consider insertions along selected arcs. Equation (2.31) sets a lower bound for the time spent detouring to charging stations, and finally, Equation (2.32) uses the established bounds on the detouring and recharging times to increase the lower bound for θ. Note that in Equation (2.33), where we define scopes for the new variables, we have defined n_e and the y_{ij}s to be continuous. Although this is a less natural definition that results in a looser bound on θ, we find that this reduction in the number of integer (branching) variables leads to better performance.

2.4.3.2 Subproblem: Charging.

The master problem (2.19)-(2.33) produces a solution $\mathbf{x}_\rho$ that is passed to the subproblem. The subproblem is responsible for determining the optimal charging decisions along the sequence ρ and correcting, through the variable θ, the objective function value of the master problem associated with the solution $\mathbf{x}_\rho$. Call $Y^\star(\rho)$ the set of optimal charging decisions for sequence ρ. The decisions $Y^\star(\rho)$ include to which CSs to detour between which stops in the sequence and to what energy level to charge during these CS visits. Together, ρ and $Y^\star(\rho)$ constitute a fixed-route policy $\pi\in\Pi_\rho$. This problem of finding the optimal charging

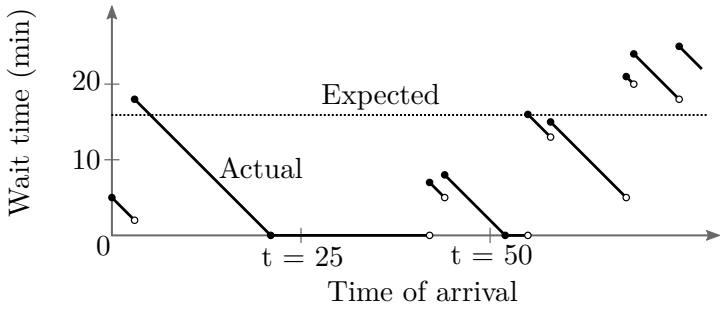

Figure 2.3: Waiting times at an extradepot CS under natural and perfect information filtrations. Under the natural filtration, the operator only has access to the expected waiting time at the CS (dashed line), whereas under the perfect information filtration, they are aware of the actual waiting time (solid line), which depends on time of arrival.

decisions given a CL sequence (the inner minimization of Equation (2.18)) is referred to as a *fixed-route vehicle charging problem*, or FRVCP (Montoya et al. 2017).

The subproblem will be one of two variants of the FRVCP, depending on the amount of information available to the decision maker. The amount of available information is known as the information filtration and is discussed in more detail in §2.6.2. If we assume the decision maker is operating under the natural filtration in which they can access all information that would naturally be available according to the problem definition in §2.2, then we solve the *FRVCP-N*. In the FRVCP-N, when we consider visiting a charging station $c \in \mathcal{C}$, in addition to the detouring and charging costs, we incur a cost equal to the *expected* waiting time at c. Alternatively, if we assume the decision maker is operating under the perfect information filtration, then we solve the *FRVCP-P*. With perfect information, the decision maker knows how long the vehicle must wait at every CS at every point in time. Hence, in the FRVCP-P, when we consider visiting a charging station c, we incur a cost equal to the *actual* waiting time as determined by realizations of queue dynamics. For a depiction of waiting times under natural and perfect information filtrations, see Figure 2.3.

In general, we can model FRVCPs using dynamic programming. The formulation of this dynamic program (DP) for the subproblem is identical to the primary formulation for the E-VRP-PP outlined in §2.2, except we now operate under a more restricted action space $\mathcal{A}^{\text{AC}}(s_k, \rho)$. This action space disallows non-AC policies, and it ensures that the vehicle follows the CL sequence ρ. Let $\bar{\mathcal{N}}'_k = \bar{\mathcal{N}}_k \cup \{0\}$, and define the function $n : (\mathcal{R} \times \mathcal{S}) \to \bar{\mathcal{N}}'_k$ which maps a CL sequence ρ and state s_k to the next element in ρ to be visited. For

simplicity, we call this element $n^\star = n(\rho, s_k)$. Then we define $\mathcal{A}^{\mathrm{AC}}(s_k, \rho)$ by the following:

$$\mathcal{A}^{\mathrm{AC}}(s_k, \rho) = \Big\{ (a^i, a^q) \in \{n^\star \cup \mathcal{C}\} \times [0, Q] :$$

$$a^i = i_k, a^q = q_k,$$
$$i_k \in \mathcal{C}' \wedge \psi_{i_k} < z_k \tag{2.34}$$

$$a^i = i_k, a^q \in \{\tilde{q} \in \mathcal{Q} | \tilde{q} > q_k \wedge (\exists c \in \mathcal{C} : \tilde{q} \geq e_{i_k n^\star} + e_{n^\star c})\},$$
$$i_k \in \mathcal{C} \wedge z_k \leq \psi_{i_k} \wedge q_k \leq q_{k-1} \tag{2.35}$$

$$a^i = n^\star, a^q = q_k - e_{i_k a^i},$$
$$(\exists c \in \mathcal{C} : a^q \geq e_{a^i c}) \wedge (i_k \neq i_{k-1} \vee q_k \neq q_{k-1}) \wedge (i_k \in C \Rightarrow q_k > q_{k-1}) \tag{2.36}$$

$$a^i \in \mathcal{C} \setminus \{i_k\}, a^q = q_k - e_{i_k a^i},$$
$$(i_k \neq i_{k-1} \vee q_k \neq q_{k-1}) \wedge q_k \geq e_{i_k a^i} \wedge (i_k \in C \Rightarrow q_k > q_{k-1})$$
$$\wedge (q_k > q_{k-1} \Rightarrow (k = 0 \vee (\bar{\mathcal{N}}_k = \emptyset \wedge a^i = 0)))\Big\}. \tag{2.37}$$

The action space $\mathcal{A}^{\mathrm{AC}}(s_k, \rho)$ is identical to $\mathcal{A}(s_k)$ with the following exceptions. First, it contains the additional condition $i_k \in C \Rightarrow q_k > q_{k-1}$ in equations (2.36) and (2.37). This condition specifies that the vehicle may only depart a CS if it charged in the previous epoch. Second, we require $a^i = n^\star$ in equation (2.36). This ensures that, when deciding to visit a customer, it is the next one in the CL sequence ρ. Finally, we modify the set of charging decisions in equation (2.35) such that the vehicle always charges to an energy level sufficient to reach the next location $n^\star$.

To solve the subproblem DP, we use the exact labeling algorithm for the FRVCP proposed by Froger et al. (2019). However, the FRVCP under consideration here requires discrete charging decisions and, for the FRVCP-P, the inclusion of time-dependent waiting times. We modify the labeling algorithm to account for these two additional features, which were not present in Froger et al.'s original formulation. The algorithm and our modifications to it are discussed in more detail in §2.4.4 and §2.4.4.1.

Optimality cuts. An optimal solution to an FRVCP is an optimal fixed route with CL sequence ρ. Call $T(\rho, Y^\star(\rho))$ the cost of the fixed route with CL sequence ρ and optimal charging decisions $Y^\star(\rho)$. If the direct-travel costs for ρ are $T_D(\rho)$, then the subproblem objective is $\theta_\rho := T(\rho, Y^\star(\rho)) - T_D(\rho)$, and we add to the master problem the following Benders optimality cuts:

$$\theta \geq \theta_\rho \left(\left(\sum_{x \in \mathbf{x}_\rho} x \right) - (|\mathbf{x}_\rho| - 1) \right) \tag{2.38}$$

The constraint works by ensuring that if the master problem selects sequence ρ by setting all $x \in \mathbf{x}_\rho$ to 1, then $\theta \geq \theta_\rho$. Otherwise, the right-hand side of (2.38) is at most 0, which is redundant given the non-negativity constraint on θ (2.24).

The optimality cuts in Equation (2.38) apply only to the complete CL sequence ρ. Cuts that apply to multiple sequences would be stronger, having the potential to eliminate

more nodes from the branch-and-bound tree of the master problem. To build more general cuts, we consider *substrings* (consecutive subsequences) of ρ of length at least two. For example, for customer set $\mathcal{N} = (1, 2, 3)$ and CL sequence $\rho = (0, 2, 3, 1, 0)$, we would consider substrings $(0, 2)$, $(0, 2, 3)$, $(0, 2, 3, 1)$, $(2, 3)$, $(2, 3, 1)$, $(2, 3, 1, 0)$, $(3, 1)$, $(3, 1, 0)$, and $(1, 0)$. Denote the set of substrings of ρ by $\mathcal{P}_\rho$. We define the set $\bar{\mathcal{P}}_\rho \subseteq \mathcal{P}_\rho$ consisting of those substrings which cannot be traversed without charging: $\bar{\mathcal{P}}_\rho = \left\{\sigma \in \mathcal{P}_\rho \middle| e^\star_{\sigma_1} < e_{\sigma_1\sigma_2} + e_{\sigma_2\sigma_3} + \cdots + e_{\sigma_{|\sigma|-1}\sigma_{|\sigma|}}\right\}$, where $e^\star_j = \max_{c\in\mathcal{C}}(Q - e_{cj})$ is the max charge an EV can have when departing location j. For each $\sigma \in \bar{\mathcal{P}}_\rho$, as we did for the complete sequence ρ, we compute $\theta_\sigma = T(\sigma, Y^\star(\sigma)) - T_D(\sigma)$, the difference between the minimum cost of an energy-feasible route through σ and its direct-travel costs. We then add cuts

$$\theta \geq \theta_\sigma \left(\left(\sum_{x\in\mathbf{x}_\sigma} x \right) - (|\mathbf{x}_\sigma| - 1) \right) \qquad \forall \sigma \in \bar{\mathcal{P}}_\rho,$$

where $\mathbf{x}_\sigma$ are the nonzero variables from the master problem solution $\mathbf{x}$ that define the substring σ.

To compute the values $T(\sigma, Y^\star(\sigma))$ for substrings $\sigma \in \bar{\mathcal{P}}_\rho$, we follow a process similar to the one used to compute $T(\rho, Y^\star(\rho))$ for the full sequence ρ. That is, $T(\sigma, Y^\star(\sigma))$ is the cost of the fixed route resulting from solving an FRVCP on the substring σ. However, we need to modify the FRVCP from the original model solved for ρ. First, of course, the CL sequence for which we solve for charging decisions is now σ instead of ρ. Next, for any substring $\sigma' \in \bar{\mathcal{P}}'_\rho = \{\sigma \in \bar{\mathcal{P}}_\rho | \sigma_1 \neq \rho_1\}$ that begins from a different location than ρ does, the time and charge at which the route begins are unknown. This is because prior to visiting σ'_1 along the sequence ρ, the EV may have stopped to charge. Having an unknown initial time means we can no longer solve an FRVCP with time-dependent waiting times (such as for the FRVCP-P), because when considering the insertion of a charging station into the route, we cannot say at what time the EV would arrive. In this case, in order to produce a conservative bound on the time required to travel the substring σ', we assume that all waiting times at charging stations are zero. Analogously, to account for unknown initial charge, we assume that we begin with the maximum possible charge ($e^\star_{\sigma'_1}$).

Feasibility cuts. If no feasible solution exists to the FRVCP for the CL sequence ρ, then it is impossible to traverse the sequence in an energy feasible manner, so the objective of the subproblem is infinite ($\theta_\rho = \infty$). This corresponds to the case where no fixed-route policy with CL sequence ρ exists ($\Pi_\rho = \emptyset$). In this case we add a feasibility cut eliminating the sequence ρ from the master problem: $\sum_{x\in\mathbf{x}_\rho} x \leq |\mathbf{x}_\rho| - 1$.

As we did for optimality cuts, we look to introduce stronger, more general feasibility cuts that may eliminate additional solutions in the master problem. We consider the substrings obtained by successively removing the first element in the sequence ρ. For each substring, we resolve an FRVCP, and if no feasible solution exists, add an optimality cut of the form $\sum_{x\in\mathbf{x}_{\rho'}} x \leq |\mathbf{x}_{\rho'}| - 1$, where ρ' is the substring $(\rho_j, \rho_{j+1}, \ldots, \rho_{|\rho|})$ formed by removing the first $j-1$ elements of ρ, and $\mathbf{x}_{\rho'}$ is the set of corresponding nonzero variables from the solution to the master problem. We continue this process until the sequence ρ' is reduced to length one or until we find a feasible solution for the FRVCP for ρ'. In the

latter case we may stop, because a feasible solution will also exist for any substring of ρ'. As for the optimality cuts, we again assume that the initial charge when solving the FRVCP for a sequence ρ' is $e^{\star}_{\rho'_1}$. However, unlike for the optimality cuts, time-dependence is irrelevant, because we are simply searching for energy-feasibility of traversing ρ'. We may ignore waiting times completely and assume they are all zero.

2.4.4 Solving the FRVCP

The FRVCP entails the prescription of charging decisions for an electric vehicle following a fixed CL sequence such that traveling the sequence is energy feasible. The objective is to minimize the time required to reach the last node in the sequence. Froger et al. (2019) propose an exact algorithm to solve the FRVCP when the charging functions are concave and piecewise-linear and the charging decisions are continuous. In their implementation, waiting times at charging stations are not considered. We modify the algorithm to accommodate discrete charging decisions and time-dependent waiting times at the charging stations. These modifications are described in §2.4.4.1. We first provide here a brief overview of the algorithm.

To find the optimal charging decisions for a given CL sequence ρ, the FRVCP is reformulated as a resource-constrained shortest path problem. The algorithm then works by setting labels at nodes on a graph $\mathcal{G}'$ which reflects the vehicle's possible movements along ρ (see Figure 2.4). Labels are defined by *state-of-charge* (SoC) *functions*. (To maintain consistency with Froger et al. (2019), we continue to use the term "state-of-charge" here, which refers to the relative amount of charge remaining in a vehicle's battery, such as 25%; however, in general we measure the state of the battery in terms of its actual energy, such as 4 kWh.) SoC functions are piecewise-linear functions comprised of *supporting points* $z = (z^t, z^q)$ that describe a state of arrival to a node in terms of time z^t and battery level z^q. See Figure 2.5 for an example.

During the algorithm's execution, labels are extended along nodes in the graph $\mathcal{G}'$. Whenever a label is extended to a charging station node, we create new supporting points for each possible charging decision. Consider Figure 2.5, which depicts this process when extending a label along the edge from node 0 to node $4a$ in Figure 2.4. Initially there is only one supporting point, corresponding to the EV's arrival to CS 4 directly from the depot. That supporting point $z_1 = (t_{0,4}, Q - e_{0,4})$ is depicted by the black diamond in the left graph of Figure 2.5. We then consider the set of possible charging decisions at that CS. The right graph of Figure 2.5 shows the charging function at CS 4 with circles for the set of charging decisions $\mathcal{Q}$ for this example. Only the black circles q'_1 and q'_2 are valid charging decisions, however, since the others are less than z^q_1 the vehicle's charge upon arrival to CS 4. For each valid charging decision, we add a supporting point to the SoC function (left), whose time and charge reflect the decision to engage in the charging operation. The figure shows this explicitly for the new supporting point z_3 corresponding to charging decision q'_2.

We continue to extend labels along nodes in $\mathcal{G}'$ until the destination node $\rho_{|\rho|} = 0$ is reached, whereat the algorithm returns the earliest arrival time of the label's SoC function. Bounds on energy and time are established in pre-processing and are used alongside dominance rules during the algorithm's execution in order to improve its efficiency. For

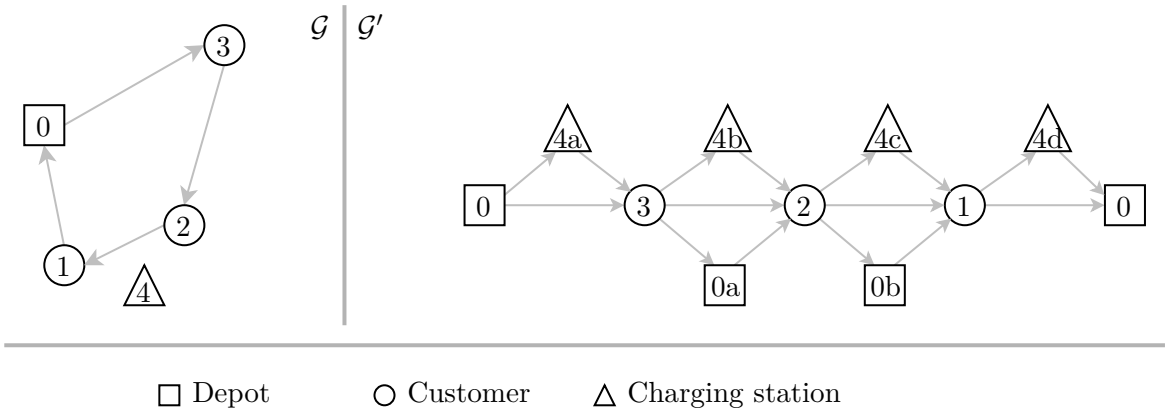

Figure 2.4: Left is an example of an original graph $\mathcal{G}$ for the E-VRP-PP. The gray path in the figure shows a CL sequence ρ. Right shows the corresponding modified graph $\mathcal{G}'$ used to model and solve the FRVCP, which includes a node for each possible CS visit.

complete details on the algorithm, we refer the reader to Froger et al. (2019).

With our modifications (§2.4.4.1), we can use the labeling algorithm to solve FRVCPs and create energy-feasible fixed routes for the E-VRP-PP. In the coming sections, we demonstrate the application of fixed routes in the construction of static and dynamic policies and in the establishment of dual bounds.

As we demonstrate in this work, the labeling algorithm from Froger et al. (2019) serves as a strong foundation upon which other researchers may build in order to solve their own variants of electric vehicle recharging problems. However, the implementation of this algorithm is non-trivial and may stand as a barrier for researchers interested in E-VRPs. In an attempt to remove this barrier, we provide at the following link an implementation of the labeling algorithm from Froger et al. (2019) that is open source and freely available to the community: `https://github.com/e-VRO/frvcp-py`.

2.4.4.1 Modifications to Froger et al. (2019) algorithm for the FRVCP

Froger et al. (2019) propose an exact algorithm to solve the FRVCP when the charging functions are concave and piecewise-linear and the charging decisions are continuous. In their implementation, waiting times at charging stations are not considered. We modify the algorithm to accommodate discrete charging decisions and time-dependent waiting times at the charging stations. For this discussion, additional information about the algorithm beyond the overview in §2.4.4 is necessary. We refer the reader to the description of Algorithm 3 in Froger et al. (2019), which is primarily located in their §5.3 and Appendix E.

To handle discrete charging decisions, we first modify the set of breakpoints that define the charging functions. Namely, we include a "breakpoint" in the charging function at each $q' \in \mathcal{Q}$ (even if the slope of the charging function does not change at q'). Next, we modify the process of extending a label. Consider the example of the edge connecting nodes 1

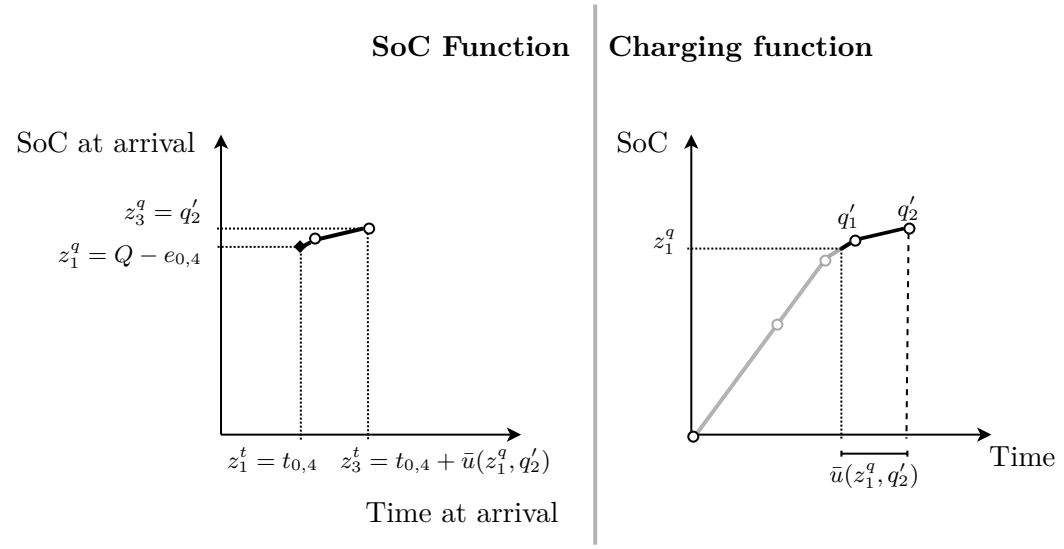

Figure 2.5: Depicting the creation of new supporting points at CS nodes for the case of node 4a in Figure 2.4. Left shows the SoC function at node 4a. The initial supporting point is the black diamond ($z_1 = (t_{0,4}, Q - e_{0,4})$). We create additional supporting points (z_2 and z_3, circles) for each possible charging decision. Possible charging decisions q_1' and q_2' are the black circles in the charging function (right graph). Axis labels on the SoC function for the new supporting point z_3 show how it is created from the charging decision q_2'.

and nodes 4d in the graph $\mathcal{G}'$ in Figure 2.4. During the translation of the SoC function by $(t_{14}, -e_{14})$, as in the original implementation, we remove all resulting supporting points with negative SoC. However, in the original implementation in which charging decisions were continuous, a new supporting point was added at the translated SoC function's intersection with the x-axis. This allowed the vehicle to charge just enough at the previous CS to be able to reach the new node with zero energy. With discrete charging decisions, we no longer create this point, so the SoC function for the label at node 4d has only three supporting points: $\{\tilde{z}_1, \tilde{z}_2, \tilde{z}_3\}$. See Figure 2.6.

To accommodate time-dependent waiting times, we make additional adjustments to the SoC function when extending a label to a CS node, such as to node 4d. We want the supporting points in the SoC function to reflect the time at which the vehicle can enter service at the CS. To do so, after the initial translation (depicted in Figure 2.6), we shift the SoC function supporting points again according to the underlying wait time. Define the function $w : (\mathbb{R}_{\geq 0} \times \mathcal{C}) \to \mathbb{R}_{\geq 0}$ that specifies how long the EV must wait if it arrives to some CS c at some time t. Because functions $w(t, c')$ are not generally continuous for a given CS c', we cannot represent the resulting SoC function as continuous. We group the supporting points based on discontinuities in $w(t, c')$ and create a new label for each group.

For example, consider again extending the label from customer node 1 to CS node 4d in Figure 2.4. After the initial shift of the SoC function, we are left with the supporting points $\{\tilde{z}_1, \tilde{z}_2, \tilde{z}_3\}$ shown in black in Figure 2.6. Now, in Figure 2.7 we consider time-dependent waiting times. The underlying wait-time function $w(t, 4)$ (top graph, in gray)

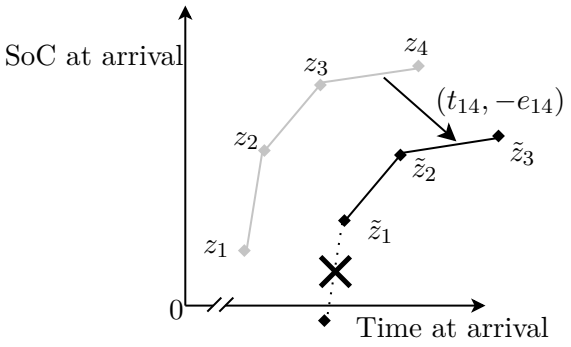

Figure 2.6: An example of shifting the SoC function as we extend the label along the edge from customer node 1 to CS node 4d in Figure 2.4. The SoC function for node 1 (in gray) is translated by $(t_{14}, -e_{14})$. The resulting SoC function for the label at node 4d (in black) contains one fewer supporting point, since the translation of z_1 yields an infeasible point with negative SoC.

has a discontinuity at the time $t = \phi$ between supporting points $\tilde{z}_2$ and $\tilde{z}_3$. As a result, the supporting points are split into two groups ($\{\tilde{z}_1, \tilde{z}_2\}$ and $\{\tilde{z}_3\}$, shown in bottom graphs) each of which comprises a new label. All supporting points $\tilde{z}_j$ are then shifted by the amount $w(\tilde{z}_j^t, 4)$ to produce the final SoC functions for these labels.

Figure 2.7 depicts an example for the FRVCP-P, in which waiting times are time-dependent. For the FRVCP-N, waiting times are constant, so there are no discontinuities in $w(t, c')$, and there is no need to divide the supporting points and create multiple labels. Instead, all supporting points are simply shifted by the constant waiting time value. We note that for both time-dependent and constant waiting times, labels' resulting supporting points are still guaranteed to produce a concave SoC function, because the queues obey the first-come-first-served property.

2.5 Policies

In this section, we describe routing policies to solve the E-VRP-PP. We divide our discussion into static policies and dynamic policies. These classes of policies differ in when they make decisions and their use of exogenous information. We begin by describing static policies, whose decisions are made in advance and do not change in response to exogenous information. We then describe dynamic policies, which may use exogenous information to inform their decision making at each epoch.

2.5.1 Static Policies

The decomposition in Theorem 2 provides a convenient way to find the optimal static policy. This is the first policy we propose to solve the E-VRP-PP. Then, because solving for an optimal static policy is computationally expensive, we also consider an approximation which we call the TSP Static policy. For both, following from Theorem 1, we restrict our

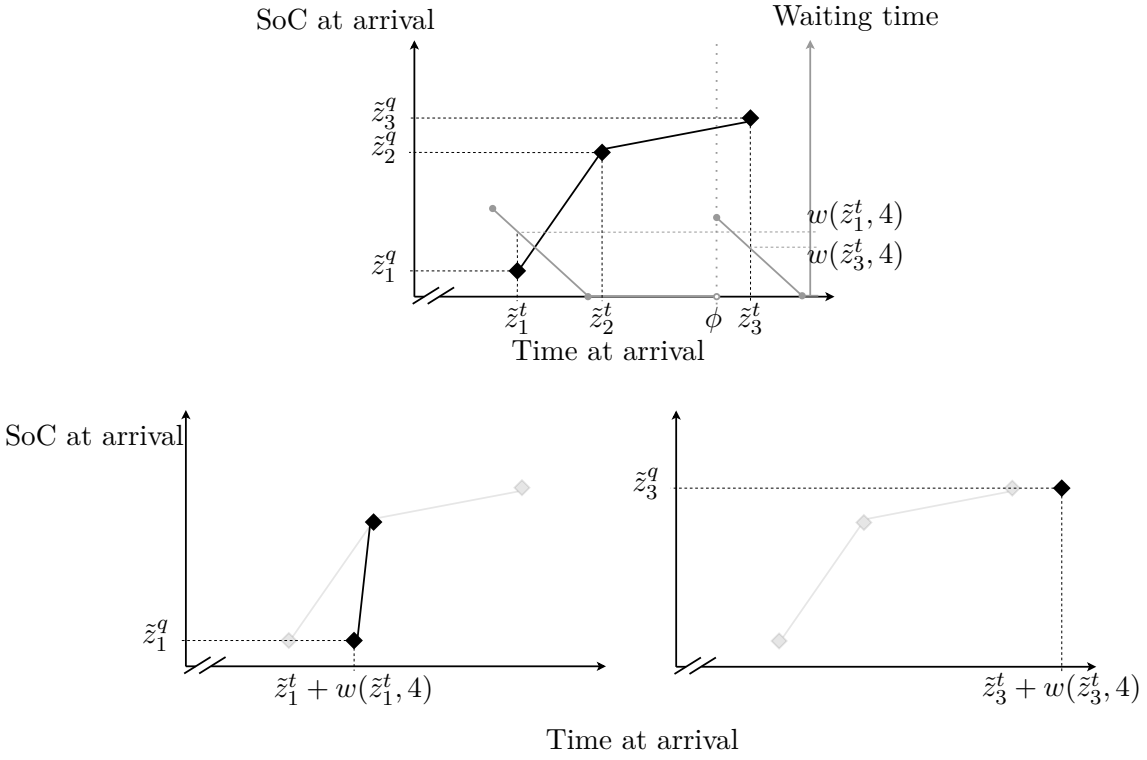

Figure 2.7: Depiction of handling time-dependent waiting times. In the top graph, we have the resulting SoC function after the initial translation from node 1 to node 4d depicted in Figure 2.6. This is superimposed over the wait-time function $w(t, 4)$, plotted in gray. The supporting points for the SoC function are divided into groups on either side of the discontinuity at $t = \phi$, resulting in two new labels shown in the bottom two graphs. After this division, the SoC functions' supporting points are shifted by their wait times. The final SoC functions are shown in black, superimposed over the pre-divided, pre-shifted SoC function.

search to only those static policies that are AC.

2.5.1.1 Optimal Static Policy.

An optimal static policy represents the best performance a decision maker can achieve when unable to respond dynamically to uncertainty. This serves as an upper bound on the optimal policy, since $\Pi^{\text{AC}} \subseteq \Pi$. For the E-VRP-PP, we can find such a policy by solving the nested minimization of equation (2.18); this solution produces an optimal fixed route from which an optimal fixed-route policy can be constructed. To solve equation (2.18), we use the Benders-based branch-and-cut algorithm described in §2.4.3.

2.5.1.2 TSP Static Policy.

Because solving Equation (2.18) to get an optimal static policy is computationally expensive, we introduce an approximation of the optimal static policy, the *TSP Static policy* π^{TSP}, that is easier to compute. The procedure to construct π^{TSP} is motivated by the decomposition in §2.4.2; however, we abbreviate our search over CL sequences, performing only a single iteration of the master and subproblems. The solution to a single iteration of the master problem is a CL sequence ρ^{TSP} representing the shortest Hamiltonian path over the unvisited customers and the depot. (We refer to this policy as the TSP Static policy, because when solving from the depot in the initial state, the shortest Hamiltonian path corresponds to the optimal TSP tour over $\mathcal{N} \cup \{0\}$.) We then optimally solve the FRVCP for ρ^{TSP} to generate an energy-feasible fixed route whose corresponding fixed-route policy we denote π^{TSP}.

2.5.2 Dynamic Policies

By definition, static policies do not use exogenous information to inform their decision making. The vehicle's instructions are prescribed in advance, and it simply follows them. Assuming exogenous information has value, these policies will be suboptimal. In this vein, we develop two dynamic policies leveraging rollout algorithms. As a benchmark, we also offer a myopic policy.

Rollout algorithms are lookahead techniques used in approximate dynamic programming to guide action selection. They may be classified by the extent of their lookahead, i.e., how far into the future they anticipate. Commonly implemented rollouts include one-step, post-decision (half-step), and pre-decision (zero-step). An m-step rollout requires the enumeration of the set of reachable states m steps into the future, constructing and evaluating a *base policy* at each future state to provide an estimate of the cost-to-go. This results in a trade-off: in general, deeper lookaheads and better base policies offer better estimations of the cost-to-go, but they require additional computation. Thus, as we consider deeper lookaheads, we are forced to consider simpler base policies. Here, we implement a pre-decision rollout with an Optimal Static base policy and a post-decision rollout with a TSP Static base policy.

2.5.2.1 Pre-decision Rollout of the Optimal Static Policy.

A pre-decision (or zero-step) rollout implements a base policy $\pi(s_k)$ from the pre-decision state s_k to select an action. The decision rule for pre-decision rollouts is simply to perform the action dictated by the base policy: $a^\star = X_k^{\pi(s_k)}(s_k)$. This strategy is also referred to as reoptimization, because the base policy is often determined by the solution to a math program that is repeatedly solved at each decision epoch. Following suit, we use the optimal static policy as our base policy, in each epoch following the procedure in §2.4.3 to determine the optimal fixed route from pre-decision state s_k and executing the first action prescribed by the fixed route. We call our pre-decision rollout of the optimal static policy *PreOpt*.

2.5.2.2 Post-decision Rollout of the TSP Static Policy.

Post-decision rollouts evaluate expected costs-to-go from post-decision states half of an epoch into the future. This is more computationally intensive than the procedure for pre-decision rollouts, because it requires the construction of a base policy from each post-decision state – of which there are $|\mathcal{A}(s_k)|$ – instead of only once from the pre-decision state s_k. Consider, for instance, action selection from some state s_k in which the vehicle just served a customer $i_k \in \mathcal{N}$. With $\bar{N} = |\bar{\mathcal{N}}_k|$ unvisited customers and C charging stations, there are up to $\bar{N} + C$ possible actions, corresponding to the relocation of the vehicle to each of these nodes. Finding the optimal static policy from each such post-decision state in each epoch is intractable. For this reason, we use the TSP static policy π^{TSP} as the base policy in our first post-decision rollout. We call the post-decision rollout with the TSP Static base policy *PostTSP*.

Let $\mathcal{S}_{\text{post}}(s_k) = \{s_k^a | a \in \mathcal{A}(s_k)\}$ be the set of reachable post-decision states. From each $s_k^a \in \mathcal{S}_{\text{post}}(s_k)$, we solve for the shortest Hamiltonian path over the set $i_k^a \cup \bar{\mathcal{N}}_k^a \cup \{0\}$ to produce a CL sequence ρ^a, then solve an FRVCP-N on ρ^a to produce a fixed-route policy $\pi^{\text{TSP}}(s_k^a)$ that serves as the base policy $\pi_{\text{b}} = \pi^{\text{TSP}}(s_k^a)$. The expected cost of this policy is the expected cost of the fixed route, given by $T(\rho^a, Y^\star(\rho^a))$. The post-decision rollout decision rule is then to select an action $a^\star$ solving

$$\min_{a \in \mathcal{A}(s_k)} \left\{ C(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} C(s_i, X_i^{\pi_{\text{b}}}(s_i)) \middle| s_k \right] \right\} = \min_{a \in \mathcal{A}(s_k)} \left\{ C(s_k, a) + T(\rho^a, Y^\star(\rho^a)) \right\}. \tag{2.39}$$

2.5.2.3 Myopic Policy.

As a benchmark for our other static and dynamic routing policies, we implement a myopic policy. Myopic policies ignore future costs in action selection, simply preferring actions with the cheapest immediate cost. More formally, myopic policies choose an action minimizing $\min_{a \in \mathcal{A}(s_k)} C(s_k, a)$. In practice, a myopic policy following this decision rule will result in exceptionally poor performance. For this reason, we bolster our myopic policy with the following rules: if all customers have been visited and the vehicle can reach the depot, we disallow all other actions; if the vehicle is at an available charging station and can charge,

we require it to charge to full battery capacity; if the vehicle has arrived to a charging station where the current queue length is less than the expected queue length, the vehicle must queue; we disallow relocations to nodes other than customers, provided a customer can be reached; and the vehicle may not visit more than two extradepot charging stations between nodes in $\mathcal{N}_k \cup \{0\}$.

2.6 Dual Bounds

While we seek to produce policies that perform favorably relative to industry methods, gauging policy quality is hampered by the lack of a strong bound on the value of an optimal policy, a *dual bound*. Without an absolute performance benchmark, it is difficult to know if a policy's performance is "good enough" for practice or if additional research is required to improve the routing scheme. In §2.6.1 we first discuss a technology-based dual bound where we assume that the vehicle is powered by an internal-combustion engine. This bound ignores the need to detour, wait, and recharge at CSs. Assuming that these actions have non-negligible cost, this bound will likely be loose. In §2.6.2 we describe our efforts to establish a tighter dual bound using the expected value of an optimal policy with perfect information, i.e., the performance achieved via a clairvoyant decision maker.

With the aim of further tightening the dual bound, we develop nonlinear information penalties that punish the decision maker for using information about the future to which they would not naturally have access. These penalties are constructed using the fixed-route machinery from §2.4. We apply the penalties on action selection in a modified version of the decomposed problem (Equation (2.18)). To the best of our knowledge, our successful implementation of these penalties marks a first in the field of vehicle routing. However, this success is limited, because we could only apply the penalties to small instances; the computational costs to apply them on larger instances is prohibitive. As a result, the penalties did not provide practical value in tightening the dual bound on our real problem instances described in §2.7. To limit the length of this text, we present the detailed discussion of our information penalties in a supplementary section after the conclusion, §2.9.

2.6.1 Conventional Vehicle Bound

To compute the optimal value with a conventional vehicle, we assume that the vehicle has infinite energy autonomy and no longer needs to recharge in order to visit all customers. We refer to this bound as the CV bound. The CV bound is a valid dual bound because it is a relaxation of the action space. Specifically, we relax the condition $(\exists c \in \mathcal{C} : a^q \geq e_{a^i c})$ in equation (2.4) and the condition $(q_k \geq e_{i_k a^i})$ in equation (2.5). These conditions are responsible for ensuring that the vehicle has sufficient charge to relocate. By relaxing these conditions, the vehicle can always relocate to an unvisited customer or a CS. Under a relaxation, the set of feasible policies increases: $\Pi \subseteq \Pi_{\text{CV}}$, where Π is the set of feasible policies under the original action space and Π_{CV} is the set of feasible policies under the relaxed action space. Additionally, we know that there is an optimal policy $\pi^\star \in \Pi_{\text{CV}}$ that does not visit any charging stations; see Theorem 3. The CV bound is the value of this

policy.

Because the optimization can be restricted to policies that do not visit CSs, uncertainty in CS queues can be ignored. Consequently, we can further restrict the search to static policies and proceed as in §2.4.3. Without the need to perform charging operations, the subproblem objective (inner minimization of equation (2.18)) over charging decisions is zero, so an optimal solution is simply a CL sequence that minimizes direct-travel costs (the outer minimization). The resulting problem of finding this CL sequence is simply a classical traveling salesman problem (TSP) over the set of customers and the depot.

Theorem 3. *Let $\mathcal{A}_{CV}(s_k)$ be a relaxation of action space $\mathcal{A}(s_k)$ defined by the removal of conditions $(\exists c \in \mathcal{C} : a^q \geq e_{a^i c})$ in equation (2.4) and $(q_k \geq e_{i_k a^i})$ in equation (2.5). Further, let Π_{CV} be the set of feasible policies under $\mathcal{A}_{CV}$. Then there exists an optimal policy $\pi^\star \in \Pi_{CV}$ that does not visit any charging stations.*

Proof. Proof. First, we note that the feasibility of $\pi^\star$ is guaranteed by the construction of $\mathcal{A}_{\text{CV}}$, since the relaxed conditions ensure that the vehicle can always relocate to an unvisited customer or a CS.

We proceed by contradiction. If $\pi^\star$ is not optimal, then there exists a policy π that does visit CSs and has a lower objective value. However, it is easy to construct a policy π' with better performance by following policy π, except when it chooses to visit CSs. In those cases, π' advances directly to the next customer visited by π (or the depot, if terminating). In so doing, the objective value of π' will be no greater than that of π. But this contradicts our assumption that π has a strictly lower objective value, so it must be that an optimal policy exists that does not visit any CSs. □

2.6.2 Perfect Information Relaxation

Let $\mathscr{F}$ be the σ-algebra defining the set of all realizations of uncertainty. As in Brown et al. (2010), we define a *filtration* $\mathbb{F} = (\mathscr{F}_0, \ldots, \mathscr{F}_K)$ where each $\mathscr{F}_k \subseteq \mathscr{F}$ is a σ-algebra describing the information known to the decision maker from pre-decision state s_k. Intuitively, a filtration defines the information available to make decisions.

We will denote by $\mathbb{F}$ the *natural filtration*, i.e., the information that is naturally available to a decision maker. We describe any policy operating under the natural filtration as being *non-anticipative*. Given another filtration $\mathbb{G} = (\mathscr{G}_0, \ldots, \mathscr{G}_K)$, we say it is a *relaxation* of $\mathbb{F}$ if for each epoch k, $\mathscr{F}_k \subseteq \mathscr{G}_k$, meaning that in each epoch the decision maker has access to no less information under $\mathbb{G}$ than they do under $\mathbb{F}$. If $\mathbb{G}$ is a relaxation of $\mathbb{F}$, we will write $\mathbb{F} \subseteq \mathbb{G}$. In the current problem, for example, we could define a relaxation $\mathbb{G}$ wherein from a state s_k, the decision maker knows the current queue length at each CS.

In Brown et al. (2010), the authors prove that the value of the optimal policy under a relaxation of the natural filtration provides a dual bound on the value of the optimal non-anticipative policy. We use this result to formulate a bound on the optimal policy using what is known as the *perfect information* (PI) *relaxation*.

The perfect information relaxation is defined by the relaxation $\mathbb{I} = (\mathscr{I}_0, \ldots, \mathscr{I}_K)$ where each $\mathscr{I}_k = \mathscr{F}$. That is, the decision maker is always aware of the exogenous information that would be observed from any state; they are effectively clairvoyant, and there is no

uncertainty. With all uncertainty removed, we can rewrite the objective function as

$$\min_{\pi\in\Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^{\pi}(s_k))\middle| s_0\right] = \mathbb{E}\left[\min_{\pi\in\Pi} \sum_{k=0}^{K} C(s_k, X_k^{\pi}(s_k))\middle| s_0\right]. \tag{2.40}$$

Notice that the *perfect information problem* (2.40) can be solved with the aid of simulation. We may rely on the law of large numbers – drawing random realizations of uncertainty, solving the inner minimization for each, and computing a sample average – to achieve an unbiased and consistent estimate of the true objective value. Per Brown et al. (2010), this value serves as a dual bound on the optimal non-anticipative policy, a bound we refer to as the *perfect information bound*.

In the context of the E-VRP-PP, a clairvoyant decision maker would know in advance the queue dynamics at each extradepot CS at all points in time. This information is summarized in the solid line in Figure 2.3, which shows the time an EV must wait before entering service at an extradepot CS as a function of its arrival time. Then a realization of uncertainty, which we will call ω, contains the information describing such queue dynamics at all extradepot CSs across the operating horizon. Let us call the set of all possible realizations of queue dynamics Ω. Then to estimate the objective value of (2.40), we sample queue dynamics ω from Ω, grant the decision maker access to this information, solve for the optimal policy for each ω, and compute the sample average.

In the absence of uncertainty that results from having access to the information ω, the inner minimization can be solved deterministically. That is, all information is known upfront, so no information is revealed to the decision maker during the execution of a policy. As a result, there is no advantage in making decisions dynamically (epoch by epoch) rather than statically (making all decisions at time 0). This permits the use of static policies to solve the PI problem. Following from Theorem 1, which applies to static policies regardless of information filtration, we may restrict our search to AC policies. Further, as demonstrated in Theorem 2, we can decompose the search over AC policies into routing and charging decisions. As a result, we can rewrite the objective of the PI problem as

$$\mathbb{E}\left[\min_{\pi(p)\in\Pi^{\mathrm{AC}}} \sum_{k=0}^{K} C(s_k, X_k^{\pi(p)}(s_k))\middle| s_0\right] = \mathbb{E}\left[\min_{\rho\in\mathcal{R}(s_0)} \left\{\min_{\pi\in\Pi_\rho} \sum_{k=0}^{K} C\left(s_k, X_k^{\pi}(s_k)\right)\right\}\middle| s_0\right]. \tag{2.41}$$

To solve the nested minimization for a given ω, we use the same decomposition and Benders-based branch-and-cut algorithm described in §2.4.2 and §2.4.3, respectively. Because we are operating under the perfect information filtration, the subproblem now corresponds to the FRVCP-P.

2.7 Computational Experiments

To evaluate the performance of our routing policies, we assemble a testbed comprised of 102 real world instances. These instances are derived from the study by Villegas et al. (2018), in which French electricity giant ENEDIS rejected the public-private recharging strategy, citing concerns about uncertainty and risk at public CSs. We describe the generation of these instances in §2.7.1, then explore the results of our computational experiments in §2.7.2.

with special emphasis on the comparison of private-only and public-private recharging strategies in §2.7.3.

2.7.1 Instance Generation

In the study by Villegas et al. (2018), the authors explain that ENEDIS divides its maintenance and service operations into geographical zones. On the days of operation considered in their study, these zones contained between 54 and 167 customers each. For each zone, there is a set of technicians that serves the associated customers. The authors were responsible for assigning customers to and providing routing instructions for the technicians, subject to a number of constraints. In total across all zones, the solution by Villegas et al. included customer assignments for 81 technicians. It is from these 81 assignments that we create our instances. Specifically, our instances are derived from the subset of 34 of these 81 assignments whose shortest Hamiltonian cycle (TSP) cannot be traveled in a single charge by the EV proposed in their study. We assume worst-case energy consumption rates in order to provide the largest possible set of instances.

The charging stations included in our instances were taken from a database provided by the French national government (Etalab 2014). The database provides information on the number of chargers available at each charging station, as well as their maximum power output. We divide the charging stations into two types – moderate (power output less than 20 kW) and fast (greater than 20 kW) – that roughly correspond to the common Level 2 and Level 3 charging types. We assume the depot locations in the ENEDIS instances also contain fast charging terminals. Based on data from Morrissey et al. (2016), we set the mean service time μ_c of a CS c to be 26.62 minutes for fast CSs and 128.78 minutes for moderate CSs. The probability of departure from an occupied charger at the CS in a given minute is then $p_{c,\text{depart}} = 1/\mu_c$. For each of the 34 assignments, we consider a low, moderate, and high demand scenario, corresponding, respectively, to an average utilization u of 40%, 65%, and 90%. As an example, this means that under the high demand scenario the probability of all chargers being occupied when a vehicle arrives is 90%. Given a utilization u, the number of chargers at a CS ψ_c, and the probability of departure $p_{c,\text{depart}}$, we can compute the arrival probability according to $p_{c,\text{arrive}} = u \cdot \psi_c \cdot p_{c,\text{depart}}$. We assume the CSs have an infinite buffer so that a vehicle will never be stranded – it can always choose to wait. In practice, however, we use a finite value for the system buffer ℓ_c, chosen such that it is practically infinite. That is, the limiting probability of observing more than $\ell_c + 1$ vehicles in the queue is less than some $0 < \epsilon \ll 1$. The charging functions for our CSs are those given in Montoya et al. (2017), which are piecewise linear and have breakpoints (changes in charge rate) at 85% and 95% of the vehicle's maximum battery capacity, which is $Q = 16$ kWh. We assume that the vehicle travels at a speed of 40 km/hr and consumes energy at a rate of 0.25 kWh/km. The set of energy levels to which the vehicle can charge $\mathcal{Q}$ consists of the charge function breakpoints as well as increments of 10%.

With 34 technician assignments and three demand scenarios for each, we have a primary testbed of 102 instances (assignment-demand pairs). These instances have between 8 and 26 customers (with an average of 16) and 6 and 79 extradepot charging stations (with an average of 49). The instances are publicly available at VRP-REP (Mendoza et al. 2014)

under VRP-REP-ID: 2019-0004[2]. Results over this set of instances are described in §2.7.2. To compare with the industry-standard private-only recharging strategy, we also consider a "private-only" scenario of each technician assignment in which we remove all extradepot CSs. These are not included in the set on VRP-REP, since they can be easily reproduced from the primary instances. Discussion of this comparison to the private-only recharging strategy is in §2.7.2.

[2]The instances currently have private visibility. We will update visibility to public after completion of the review process.

Table 2.1: Detailed results comparing route-based policies to the value of the optimal policy with perfect information. *Note: Values for each demand scenario are averages over 34 technician assignments (50 realizations of uncertainty for each), excluding those for which a PI bound could not be established.*

	low					moderate					high				
	PI	PreOpt	Opt Static	PostTSP	TSP Static	PI	PreOpt	Opt Static	PostTSP	TSP Static	PI	PreOpt	Opt Static	PostTSP	TSP Static
Objective	157.7	160.4	162.2	162.4	162.8	158	169.8	171	171.1	169.6	158.4	194	194	195.6	200.7
Objective (% diff from PI)	0.0	1.7	2.8	3.0	3.2	0.0	7.5	8.3	8.3	7.3	0.0	22.5	22.5	23.5	26.7
Detouring Time	1.8	2.3	1.9	1.6	2.6	1.9	7.5	7.3	2.7	3.7	1.9	19.9	19.9	11.6	28.5
Charging Time	12.3	12.7	12.5	12.7	13.1	12.4	13.7	13.6	13.8	13.8	12.6	18.2	18.2	19.1	20.7
Waiting Time	0.0	1.3	3.3	3.4	3.2	0.1	3.3	4.2	4.1	6.5	0.1	5.1	5.1	5.1	5.5
Charge Rate (kW)	41.3	41.2	41.2	40.9	40.3	41.2	42.6	42.6	42.2	39.5	41.2	41.6	41.6	40.6	39.8

Table 2.2: Computational effort for the routing policies (upper) and the establishment of dual bounds (lower). Left column is the aggregate over all computational experiments. Right column indicates the time for policies to make a decision in each epoch. *Note: (1) CV bound is an aggregation over a single solution for each technician assignment; PI bound and all policies are aggregations over 50 realizations of uncertainty for each instances. (2) PreOpt values include the time to solve for the optimal static policy for the first epoch. If the optimal static fixed route was available in advance of running the PreOpt policy, then the computational effort would be the value shown for PreOpt minus the value for the optimal static policy.*

	Total computational effort (dd-hh:mm:ss)	Per-epoch computational effort (s)
PreOpt	08-21:25:00.5	7.0
Optimal Static	08-02:35:10.7	6.4
PostTSP	01-01:55:08.1	0.8
TSP Static	00-00:05:53.0	3.8E-03
Myopic	00-00:00:00.4	2.9E-06
PI Bound	43-21:29:01.7	-
CV Bound	00-00:00:03.7	-

Table 2.3: Comparing routing policies to solutions using private-only recharging strategy. *Note: Values are averages over the 14 technician assignments for which the private-only solution was feasible. Policies' reported performances are expectations over 50 realizations of uncertainty for each demand scenario.*

		low				moderate				high			
	Private-only	PreOpt	Opt Static	PostTSP	TSP Static	PreOpt	Opt Static	PostTSP	TSP Static	PreOpt	Opt Static	PostTSP	TSP Static
Objective	201.2	128.5	131.3	131.7	132.5	143.3	143.9	144.0	140.9	175.8	175.8	176.3	179.2
Direct-travel time (CL seq)	144.4	114.8	114.8	115.4	114.2	114.9	114.9	124.1	114.2	120.8	120.8	143.5	114.2
Detouring Time	34.8	5.2	4.1	3.5	5.2	16.8	16.6	7.4	9.0	36.8	36.8	14.4	44.1
Charging Time	21.9	7.4	7.2	7.4	7.8	9.2	9.1	9.2	9.0	16.5	16.5	16.6	18.4
Waiting Time	0.0	0.9	4.7	4.8	4.7	1.9	2.4	2.3	6.3	1.1	1.1	1.1	1.8

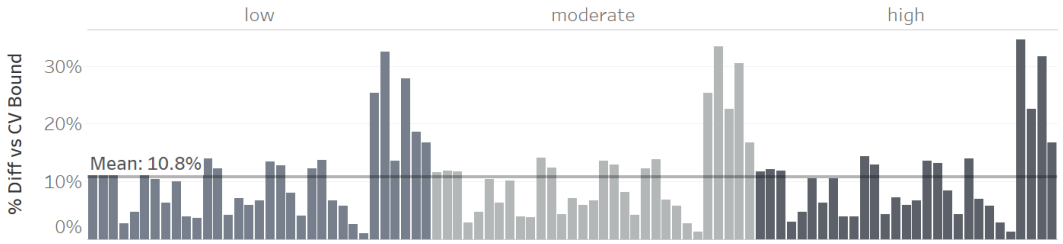

Figure 2.8: Comparing dual bounds. The figure contains a bar for each instance showing the relative performance difference between the optimal value with a CV and the value of an optimal policy with PI. Across all instances, the PI bound is on average 10.8% higher than the CV bound, offering a tighter dual bound and a better measure of goodness of our policies. *Note: We omit instances for which we could not solve sufficiently many realizations of uncertainty to establish the PI bound (at least 38/50).*

2.7.2 Results on Primary Instances

We divide the discussion of the results on our primary instances as follows. First, we compare the two dual bounds proposed in §2.6. We then investigate policies' performances, first giving a brief overview, then comparing static and dynamic policies, and lastly comparing the policies to the dual bound. Finally, we comment on the computational effort to perform these experiments.

Comparing dual bounds.

We begin our analysis by comparing the two dual bounds: the optimal value of performing service with a CV, the *CV bound*, and the value of an optimal policy with perfect information, the *PI bound*. To establish the PI bound for each instance we take the average over 50 realizations of uncertainty. We omit results for those instances for which we could not optimally solve at least 75% of the realizations (at least 38/50). In total, we were able to compute the PI bound for 93/102 instances: 33/34 in the low demand scenario, 31/34 with moderate demand, and 29/34 with high demand. For the CV bound, since there is no uncertainty, we need solve it only once for each technician assignment. We were able to solve for the optimal CV bound for 34/34 assignments, yielding a CV bound for each instance.

Figure 2.8 offers a comparison of the PI bound to the CV bound for the 93 instances for which the PI bound was available. We find that the PI bound is a significantly better dual bound than the CV bound, offering an improvement of 10.8% on average. This tighter dual bound allows us to make stronger statements about the goodness of our routing policies, and, in general, indicates that there is value in the effort to establish a tighter bound. Going forward, the reported performance gaps for our policies are stated relative to the PI bound. More broadly, these results lend support to the notion that E-VRPs should indeed be considered a distinct family of problems from conventional VRPs.

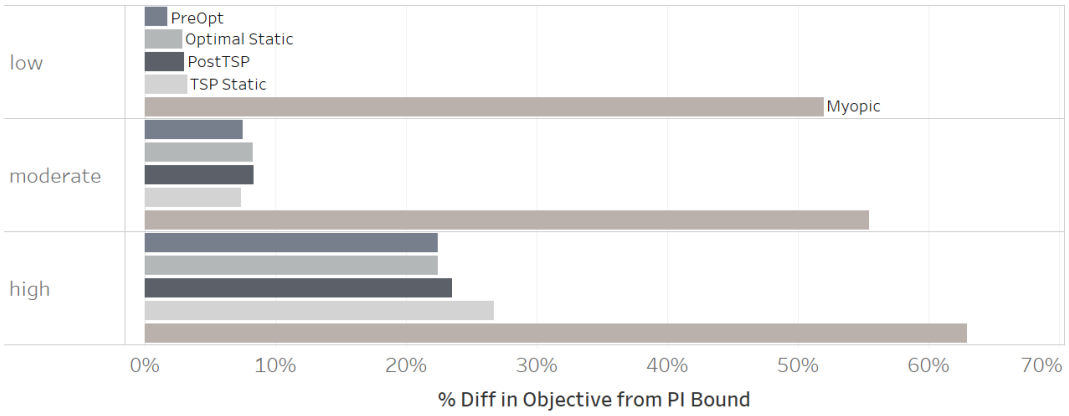

Figure 2.9: Visual summary of policy performance relative to the PI bound. Each bar is an average over the 34 technician assignments, with 50 realizations of uncertainty for each. *Note: We exclude instances for which the PI bound could not be calculated.*

Summary of Policy Performance.

To assess policies' performance on an instance, we average over 50 realizations of uncertainty, as we did to establish the PI bound. In computing the optimal static policy, we are able to solve equation (2.18) exactly in 40/102 instances. For the remainder, we use the best known solution (BKS) after three hours. To execute the PreOpt policy, in the first epoch we use the solution found by the optimal static policy, then allow two minutes to resolve the optimal static policy at all subsequent epochs, taking the best solution after two minutes if the optimal solution is not found in that time. Note that the optimal static policy need only be recomputed in epochs following the observation of (non-deterministic) exogenous information. For example, if in the first epoch the optimal static policy dictates relocating from the depot to a customer, then in the subsequent epoch the vehicle can continue to follow the optimal static policy without recomputing it, as no additional information was observed when it arrived to the customer. In the tables and figures that follow, unless noted otherwise, units are minutes.

As seen in Figure 2.9, we find that our route-based policies are competitive with one another, while the myopic policy serves as a distant upper bound. This contrast between the performance of our route-based policies and the myopic policy demonstrates the value in route-planning and the anticipation of charging station queues. Further, we find that the route-based policies are competitive with the PI dual bound, especially in the low and moderate demand scenarios (a more detailed discussion of policies' performance relative to the PI bound is below). As expected from queuing theory, the objective values of our policies increase with the demand for extradepot CSs. Of our policies, PreOpt performs the best on average, followed by the optimal static policy, PostTSP, then TSP Static. Comparing static policies, across demand scenarios, the optimal static policy offers on average a 0.7% improvement over the TSP static policy. The difference between our dynamic policies is similar, with PreOpt offering a 0.8% improvement over PostTSP across demand scenarios.

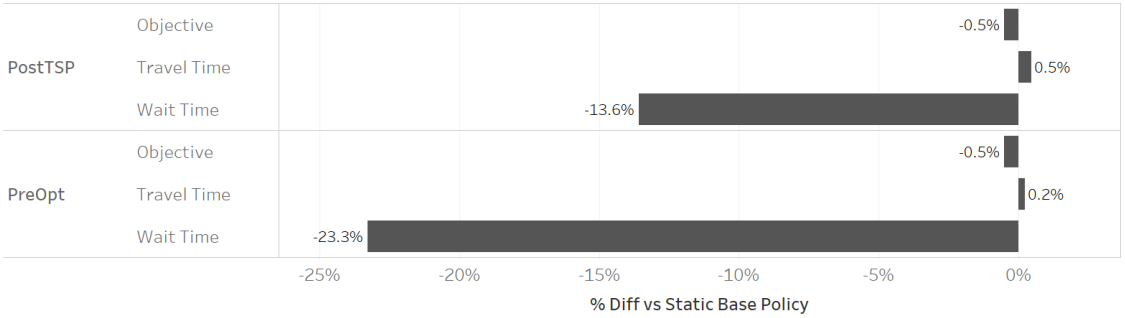

Figure 2.10: Comparing dynamic policies to the static base policies from which they are built. Top panel shows the percent difference in various metrics of PostTSP from the TSP static policy; bottom shows the same for PreOpt relative to the optimal static policy. Values reflect averages over all instances.

Performance of static vs. dynamic policies.

Figure 2.10 depicts the advantage that dynamic policies stand to offer over static policies – namely, that their additional flexibility in making routing decisions should yield improvements in objective values. We find this to be true here, with dynamic policies exhibiting a small edge over static policies, outperforming them by 0.5%. We see that this edge is largely attained through a reduction in waiting times, which outweighs an observed increase in travel times. These observations align with intuition. One would expect that static policies, which must wait at extradepot CSs regardless of observed queue length, would wait longer on average than dynamic policies, which can choose to balk CSs if queues are long. Consequently, relative to static policies, which wait in queue, dynamic policies should spend more time traveling as they explore additional CSs.

Policy performance relative to PI bound.

Table 2.1 compares policies' performance to the value of the optimal policy with perfect information. We find that on average our best policy is within 5% in the low and moderate demand scenarios, and within 11% overall. As seen in Figure 2.9, the gap between our routing policies and the PI bound widens as demand for extradepot CSs increases, from an average of 2.5% under the low demand scenario to 25% under the high demand scenario. The results in Table 2.1 show that this widening gap is due in large part to additional detouring. The non-anticipative policies have an estimate of the expected waiting time at extradepot CSs which increases with increasing demand. When seeking to avoid long expected queues, the routing policies perform longer detours, often back to the depot at which there is no queue. This also results in increased charging times for the non-anticipative policies. A particularly good example of lengthy detours is the high demand case for the TSP Static policy: it spends on average 14.2% of its time detouring, compared to an average of 8.8% for the other routing policies (and has the longest recharge times and worst objective performance as a result). We also note that in the moderate and high demand scenarios, the policy with PI does not achieve the fastest average charge

rate. Instead, to avoid waiting at CSs or performing lengthier detours, it will sacrifice fast charging, either by charging on slower segments of the charging function or by choosing a CS with slower charging technology. Lastly, to achieve a near-constant objective value with increasing demand, the optimal policy with PI is consistently able to find convenient extradepot CSs at which it incurs near-zero waiting times. The large gap between our policies and the PI bound emphasizes the value of this information.

Computational effort.

In Table 2.2, we report the computational effort for our policies and dual bounds. For the policies, we also include the average time required to make a decision in each epoch. For the PI bound and the routing policies, in general the better (lower) the objective value, the more computation time is required. As these results show, the 0.7% improvement of the optimal static policy over the TSP static policy and the 0.8% improvement of PreOpt over PostTSP come at a significant computational cost: more than eight days for the former and seven days for the latter. TSP Static's competitive objective achievements and relatively short computation time make it a good candidate for inclusion in more complex lookahead procedures, such as PostTSP. Here, we find that embedding TSP Static into a post-decision rollout improves performance by 0.5% while maintaining an average per-epoch computational effort of less than one second.

Interestingly, the time to compute the PI bound decreases with an increasing ratio of charging stations per customer (see Figure 2.11). This is likely due to the structure of the objective function in Equation (2.18). Recall that while the master problem (specifically, inequalities (2.26)-(2.32)) has approximations for the detouring and recharging time required to feasibly traverse a CL sequence, the exact amount – and any waiting time – is unknown and only revealed by solving the subproblem. As CSs become more abundant, more opportunities are available for low-cost detours and short waiting times, so the required amount of detouring and recharging time decreases. This improves the master problem's approximations of these values, ultimately leading to faster solution times.

Disaggregated results over the testbed of instances are available in §2.7.4.

2.7.3 Public-Private vs. Private-Only Recharging Strategies

Perhaps most importantly, we wish to demonstrate that even in the face of uncertainty at public charging stations our policies perform favorably relative to the private-only strategy. This is true by default a majority of the time, as the private-only strategy is energy-infeasible for 20/34 technician assignments. For the remaining instances, averaging across demand scenarios, we find that all proposed policies soundly outperform the best private-only solutions, with our best-performing policies outperforming the private-only solutions by 25.8%. See Table 2.3 and Figure 2.12. Together with the large number of infeasiblilities, our results suggest that committing to a private-only recharging strategy may lead to higher costs associated with the use of EVs, potentially hampering their adoption in commercial applications.

Of the 14 technician assignments for which the private-only strategy is feasible, we solve six to optimality and use the best solution found after three hours of computing time for

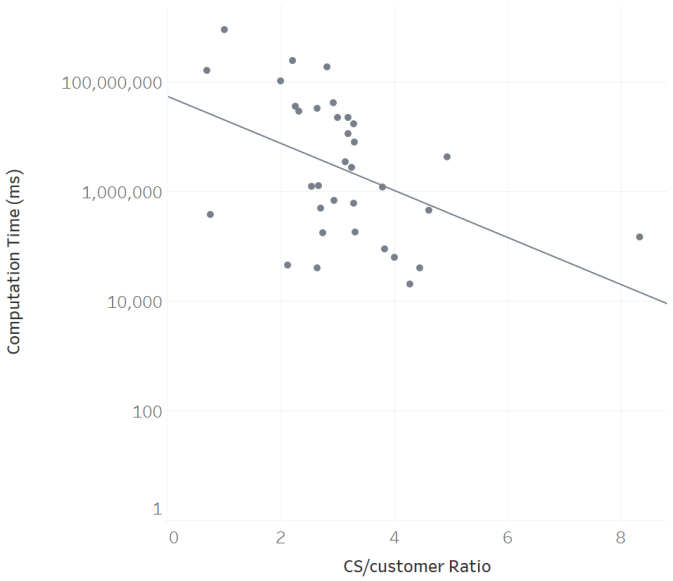

Figure 2.11: Computation times for instances' PI bounds versus their ratio of CSs to customers. *Note: The figure contains a point for each technician assignment, representing the sum over 50 realizations of uncertainty for each demand scenario.*

the remaining eight. In general, we find that our route-based policies tend to outperform the best private-only solution by wide margins: 34.9% in low-demand scenarios, 28.9% in moderate-demand scenarios, and 12.1% in high-demand scenarios. This decrease in the gap with increasing demand is not due to longer waiting times at CSs, as intuition might suggest. Rather, as in the comparison to the PI bound, it is primarily due to detouring. The policies increasingly revert to routing behavior that more often relies on the depot for recharging, as expected waiting times at extradepot CSs exceed the additional time required to detour back to the depot. That is, the private-only recharging approach is used as a fallback for public-private routing policies in scenarios with high expected demand.

As EVs continue to increase in popularity and related technologies develop, it is likely that the performance gap between the private-only and public-private recharging strategies will widen further. Assuming charging infrastructure increases at a rate similar to its demand, the average performance of policies should improve. Intuitively, as more charging stations become available at which to charge, the detouring and waiting an EV must do prior to charging will reduce.

Further, as more real-time information becomes available regarding demand at extradepot CSs, more informed routing decisions can be made as there will be less uncertainty, which should lead to better policy performance. In fact, access to this real-time information may be modeled as a relaxation for the current problem. That is, we could grant the decision maker in the E-VRP-PP access to the current state of the queue at each CS and assess policies' performance under this filtration (call it $\mathbb{Q}$). The performance of the decision maker under $\mathbb{Q}$ would be bounded below by the dual bounds and above by our current best-performing policy, since $\mathbb{F} \subseteq \mathbb{Q} \subseteq \mathbb{I}$. Our observation of the policy with perfect

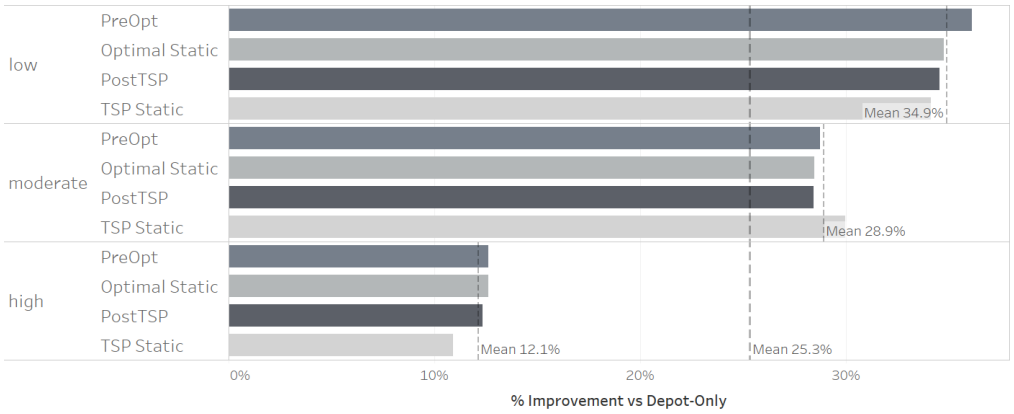

Figure 2.12: Depicting objective improvement of routing policies compared to the private-only solution. Dashed lines indicate mean values over demand levels. *Note: Values are averages over the instances corresponding to the 14 technician assignments for which the private-only solution was feasible. Policies were evaluated on 50 realizations of uncertainty for each.*

information consistently achieving near-zero waiting times under even the high-demand scenario suggests that the bound under filtration $\mathbb{Q}$ may be very close to that under $\mathbb{I}$.

2.7.4 Disaggregated Results of Computational Experiments

Table 2.5 contains disaggregated results for the computational experiments described in §2.7.2 and §2.7.3. The naming convention for the instances (or, more accurately, the technician assignments) is "*geography_zoneID-technicianID*." Optimal static, PreOpt, and Private-only entries marked with asterisks denote instances that we were able to solve to optimality; the rest are the best solutions found after three hours of computation. Empty cells for the PI bound denote instances for which we could not solve at least 38/50 realizations of uncertainty to optimality (all PI values shown are optimal). Entries marked "inf" are infeasible under the private-only recharging strategy.

Interestingly, in the high demand scenario for technician assignments rural_20-4 and rural_21-7 the myopic policy is the best-performing non-anticipative policy. In these instances, there is a cluster of customers far from the depot that cannot all be served without recharging. While the route-based policies prefer private-only-style recharging schemes for these instances (to avoid the long expected queues), the myopic policy thoughtlessly explores extradepot CSs in search of one at which to recharge. This behavior appears to have worked in its favor for these two instances. See Table 2.4 and Figure 2.13 for the example of instance rural_20-4.

Table 2.4: Data explaining the myopic policy's outperforming of the more sophisticated route-based policies in the case of high demand for instance rural_20-4. Route-based policies, to avoid long queues at extradepot CSs, return to the depot to charge and thereby incur a large amount of detouring time. Meanwhile, the myopic policy explores more nearby extradepot CSs, waiting more but detouring much less.

	PI	PreOpt	Optimal Static	PostTSP	TSP Static	Myopic
Objective	116.2	219.6	219.6	219.6	221.0	184.8
Waiting Time	0.0	0.0	0.0	0.0	0.0	8.8
Detouring Time	0.2	75.6	75.6	48.7	78.3	17.2
Visited Extradepot CSs	1.1	1.0	1.0	1.0	1.0	3.0

Values are in minutes, averaged over 50 realizations of uncertainty.

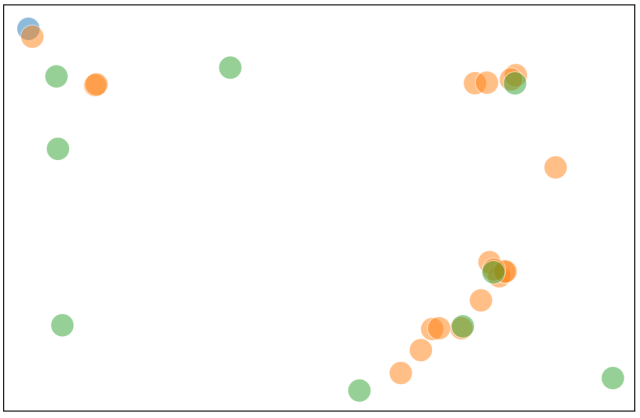

Figure 2.13: Simple depiction of instance rural_20-4. The grouping of customers (orange circles) on the right side of the image are far from the depot (blue circle) near the upper left of the image. Route-based policies tend to go back to the depot to recharge, even when in the middle of serving customers in the far-right grouping. Conversely, the myopic policy takes a chance with the extradepot CSs (green circles) near that grouping of customers.

Table 2.5: Disaggregated objective values over testbed of instances, including CV bound and the objective value under the private-only recharging strategy. *Note: Values are in minutes. "inf" entries (private-only) were infeasible. For PI, all shown results are optimal; we leave cells blank when we could not solve at least 38/50 realizations of uncertainty to optimality. For the optimal static policy, PreOpt, and the private-only solutions, asterisks indicate optimality; the rest are the best solution found after three hours of computation time.*

			low						moderate						high					
Instance	CV	Private-only	PI	PreOpt	Opt Static	PostTSP	TSP Static	Myopic	PI	PreOpt	Opt Static	PostTSP	TSP Static	Myopic	PI	PreOpt	Opt Static	PostTSP	TSP Static	Myopic
rural_18-1	180.1	inf	200.6	204.6*	206.0*	204.9	207.2	335.8	200.9	210.3	210.6	208.0	210.7	356.8	201.1	226.0	226.0	227.6	222.2	343.9
rural_18-3	176.9	inf	197.8	203.1*	204.0*	204.0	201.5	263.3	197.9	210.7*	210.4*	210.4	207.8	270.9	198.4	276.6	276.6	285.9	287.2	284.8
rural_18-4	174.5	inf	194.4	197.1*	197.1*	197.1	199.3	361.2	195.0	204.5*	204.5*	204.5	205.7	366.8	195.2	206.9*	206.9*	206.9	207.8	371.5
rural_18-5	107.3	172.8	110.3	112.7*	114.1*	114.5	114.5	192.5	110.4	118.2	118.2	118.2	121.8	195.2	110.5	172.8	172.8	172.9	127.8	206.4
rural_18-6	118.2	239.0	123.8	127.7	127.7	127.7	128.1	218.4	123.8	130.2	130.2	130.2	131.6	220.5	123.9	135.6	135.6	135.6	134.0	227.4
rural_19-0	211.9	inf	239.9	244.5	246.2	246.2	244.8	333.4		251.7	251.5	251.5	254.4	343.7		276.5	276.5	276.0	292.7	359.9
rural_19-1	170.1	inf	187.8	191.7	192.6	192.6	192.6	292.6	187.9	197.4	197.4	199.8	198.1	298.7	188.0	203.0	203.0	206.2	218.4	302.7
rural_19-3	126.7	inf	134.7	135.1	138.7	138.7	138.4	241.6	134.8	137.8	142.4	142.4	143.0	243.0	134.8	176.0	176.0	196.8	199.2	254.3
rural_20-0	155.5	inf	171.0	174.8	174.8*	174.8	177.7	285.5	171.2	178.0	178.0	178.0	183.6	293.3	171.8	182.2	182.2	182.2	192.1	293.5
rural_20-3	111.1	133.5*	115.5	115.7*	118.4*	119.5	119.0	176.3	115.5	125.2*	124.9*	124.9	126.0	174.1	115.5	133.5*	133.5*	133.5	134.0	181.9
rural_20-4	111.7	219.6	115.9	116.5*	119.5*	119.5	119.5	172.8	116.1	121.9	126.2	126.2	126.2	180.1	116.2	219.6	219.6	219.6	221.0	184.8
rural_20-5	211.7	inf	241.3	250.1	250.3*	250.3	246.8	299.2	241.5	256.2	256.2	255.9	258.0	302.7	242.0	265.2	265.2	265.2	266.0	318.7
rural_20-6	194.8	inf	218.7	224.2*	223.8*	223.8	223.8	356.2	219.0	230.8	231.4	231.6	230.7	364.3	220.1	240.9	240.9	240.9	239.4	379.9
rural_20-7	113.6	inf	118.4	118.7*	121.3*	121.3	121.5	158.8	118.4	121.4*	128.2*	128.2	125.8	161.6	118.5	128.0	128.0	128.0	133.0	175.5
rural_21-0	135.3	149.1*	145.0	146.1*	146.6*	147.1	147.3	271.6	145.0	149.1*	149.1*	150.3	150.4	273.2	145.1	149.1*	149.1*	150.3	150.4	328.2
rural_21-1	125.2	313.4	132.7	133.0	133.3	133.3	133.6	205.7	132.7	136.2	135.7	135.7	136.7	217.3	132.7	140.6	140.6	140.6	139.9	231.5
rural_21-3	132.5	inf	141.3	141.8	141.8	141.8	144.0	215.7	141.3	144.6	145.0	145.0	146.0	220.8	141.5	148.3	148.3	148.3	149.5	221.3
rural_21-4	214.3	inf	243.2	244.6*	248.7*	250.1	248.7	296.5	243.4	252.3	259.9	259.9	259.9	303.6	243.6	306.2	306.2	306.1	306.1	311.2
rural_21-6	203.1	inf	229.2	232.6*	233.5*	233.3	233.3	276.5	229.5	240.0*	239.7*	240.2	240.0	282.8	230.0	251.1	251.1	250.5	251.5	301.0
rural_21-7	145.7	277.9	157.5	159.1*	159.5*	159.5	159.5	237.8	157.6	164.2*	164.3*	164.3	164.3	240.9	158.0	278.1	278.1	280.0	292.4	247.3
rural_21-8	112.0	155.8*	116.5	117.9*	121.5*	121.5	121.5	192.8	116.6	120.7*	124.2*	124.2	124.2	192.8	116.9	155.8*	155.8*	155.8	159.9	192.8
rural_22-0	196.0	inf	220.0	225.4	225.9	225.9	225.9	261.6	220.0	233.4	234.0	234.0	234.0	272.1		268.8	268.8	268.8	268.9	278.4
rural_22-1	220.1	inf	250.2	257.3	257.9	257.9	256.2	318.0	250.7	267.6	267.2	267.2	266.5	329.9	250.8	277.4	277.4	277.4	274.5	336.6
rural_22-2	135.2	inf	144.3	144.3	148.3	148.3	147.5	196.0	144.4	146.7	150.4	150.4	152.3	200.3	144.5	162.0	162.0	167.2	173.1	211.3
rural_22-3	125.3	inf	132.5	132.6	136.8	136.8	136.8	194.2	132.5	134.7	138.6	138.6	138.6	196.6	132.6	183.9	183.9	183.9	186.6	203.3
rural_22-4	105.4	163.0	108.2	110.9*	110.9*	110.9	111.9	246.1	108.2	112.6*	112.6*	112.6	123.1	264.2	108.4	163.0	163.0	166.1	167.9	291.2
rural_22-5	98.0	192.5*	99.1	100.6*	103.6*	103.6	103.6	138.0	99.3	106.5*	106.5*	106.5	106.9	138.7	99.4	112.6*	112.6*	112.6	111.7	143.1
semi_urbain_18-8	120.9	263.0	151.5	160.1	171.5	170.8	172.0	264.6	151.5	185.0	186.3	185.2	191.0	281.3		263.5	263.5	264.4	194.7	308.4
semi_urbain_20-1	100.2	207.7	132.8	140.2*	140.2*	144.8	149.5	213.7	133.7	207.9	207.9	208.5	172.7	216.9	134.9	207.9	207.9	208.5	266.8	266.7
semi_urbain_21-7	104.8	inf		129.7*	129.7*	129.7	130.7	186.9	128.3	133.1	133.1	133.1	131.6	197.9	128.3	137.9	137.9	137.9	135.9	196.6
semi_urbain_21-11	112.0	inf	127.2	142.0	138.7	138.7	142.0	227.5		175.5	175.5	175.5	177.7	236.5		183.1	183.1	183.1	184.8	267.4
semi_urbain_22-0	97.2	200.0*	124.3	129.4*	142.0*	142.0	142.3	180.9	126.9	200.0*	200.0*	200.0	165.5	187.9	128.0	200.0*	200.0*	200.0	275.9	238.4
semi_urbain_22-1	131.7	inf	156.2	164.7	163.7	163.7	165.8	229.3		191.6	189.4	189.4	191.6	247.4		271.0	271.0	267.0	286.0	273.1
semi_urbain_22-2	110.3	128.8*	128.8	128.8*	128.8*	128.8	132.4	174.0	128.8	128.8*	128.8*	128.8	132.4	183.1	128.8	128.8*	128.8*	128.8	132.4	177.9

2.8 Concluding Remarks

We have introduced the E-VRP with public-private recharging strategy and proposed an approximate dynamic programming solution. Through a decomposition of the E-VRP-PP, we bridge static and deterministic routing methods with dynamic and stochastic routing problems. Using these methods, we construct static and dynamic routing policies, including rollout algorithms and the optimal static policy. To better measure the goodness of these policies, we provide dual bounds. First, we provide a bound equal to the value of using a conventional vehicle. We then establish a tighter dual bound on the value of the optimal policy through the use of a perfect information relaxation. Using that dual bound, we have demonstrated that our routing policies are competitive with the optimal policy, coming within 11% on average and within 5% in the majority of instances.

Our work was motivated by an example from industry in which an EV operator rejected the public-private recharging strategy to avoid uncertainty at public charging stations. We sought to answer whether with a good dynamic routing policy such companies could adopt a public-private recharging strategy that would be cheaper than the private-only strategy. In computational experiments using real instances from industry, we found this to be true, demonstrating that all of our policies under the public-private recharging strategy soundly outperform the solution under a private-only strategy, with our best policies offering savings of approximately 26% on average. Ultimately, we hope this work encourages companies to adopt a public-private recharging strategy, increasing the utility of EVs in commercial applications and accelerating the transition to sustainable transportation.

2.9 Information Penalties

The dual bound achieved with perfect information (see §2.6.2) is often loose, because no decision maker is clairvoyant and advanced knowledge of the future is often valuable. To tighten the bound, we can penalize the decision maker and attempt to eliminate any benefit of using advanced information. These information penalties manifest as additional costs $z(s_k, a)$ incurred during action selection in the perfect information problem. We write the objective function of the penalized perfect information problem as

$$\mathbb{E}\left[\min_{\pi\in\Pi}\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) + z\left(s_k, X_k^\pi(s_k)\right)\middle| s_0\right]. \tag{2.42}$$

The form of the information penalty we use is $z(s_k, a) = \mathbb{E}\left[V_{k+1}(s_k, a)|\mathscr{F}_k\right] - \mathbb{E}\left[V_{k+1}(s_k, a)|\mathscr{I}_k\right]$, where $V_{k+1}(s_k, a)$ is the value of being in the pre-decision state s_{k+1} reached by choosing action a from state s_k. The penalty captures the difference in the expected cost-to-go under the natural and perfect information filtrations. The form of this penalty aligns with that of Theorem 2.3 (and Proposition 2.2) of Brown et al. (2010), which promises strong duality. Strong duality guarantees that the optimal objective value of the penalized perfect information problem (2.42) will be equal to the objective value of the optimal non-anticipative policy. In practice, however, the values $\mathbb{E}\left[V_{k+1}(s_k, a)|\mathscr{F}_k\right]$ and $\mathbb{E}\left[V_{k+1}(s_k, a)|\mathscr{I}_k\right]$ are unknown. To approximate them, we follow an approach suggested in Brown et al. (2010), employing value function approximations for $V_{k+1}(s_k, a)$.

Let $v^{\mathbb{G}}_{k+1}(s_k, a)$ be the approximation of $\mathbb{E}\left[V_{k+1}(s_k, a)|\mathscr{G}_k\right]$ under a filtration $\mathbb{G}$. Then we can write our approximated penalty as $\hat{z}(s_k, a) = v^{\mathbb{F}}_{k+1}(s_k, a) - v^{\mathbb{I}}_{k+1}(s_k, a)$. To compute $v^{\mathbb{G}}_{k+1}(s_k, a)$ we utilize an *estimating policy* $\pi(s_{k+1}, \mathbb{G})$ to approximate the cost-to-go from a future state s_{k+1} under the filtration $\mathbb{G}$: $v^{\mathbb{G}}_{k+1}(s_k, a) = \mathbb{E}\left[\sum_{i=k+1}^{K} C\left(s_i, X_i^{\pi(s_{k+1},\mathbb{G})}(s_i)\right)\middle| s_k, a\right]$. For our estimating policy, we use the TSP static policy (see §2.5.1.2). Then we may write our penalty explicitly as

$$\hat{z}(s_k, a) = \mathbb{E}\left[\sum_{i=k+1}^{K} C\left(s_i, X_i^{\pi^{\text{TSP}}(s_{k+1},\mathbb{F})}(s_i)\right)\middle| s_k, a\right] - \mathbb{E}\left[\sum_{i=k+1}^{K} C\left(s_i, X_i^{\pi^{\text{TSP}}(s_{k+1},\mathbb{I})}(s_i)\right)\middle| s_k, a\right] \tag{2.43}$$

The objective for the penalized PI problem with our approximation is

$$\begin{aligned}&\mathbb{E}\left[\min_{\pi\in\Pi}\sum_{k=0}^{K} C\left(s_k, X_k^{\pi}(s_k)\right) + \hat{z}\left(s_k, X_k^{\pi}(s_k)\right)\middle| s_0\right] \\ &= \mathbb{E}\left[\min_{\pi\in\Pi}\sum_{k=0}^{K} C\left(s_k, X_k^{\pi}(s_k)\right) + v^{\mathbb{F}}_{k+1}\left(s_k, X_k^{\pi}(s_k)\right) - v^{\mathbb{I}}_{k+1}\left(s_k, X_k^{\pi}(s_k)\right)\middle| s_0\right].\end{aligned} \tag{2.44}$$

As in the unpenalized perfect information problem, without loss of optimality, we may restrict our search of policies to those that are AC. We justify this restriction in Theorem 4.

Theorem 4 (Optimal policies for penalized PI problem are AC)**.** *Let $\tau_{\hat{z}}(\pi)$ be the value of a policy $\pi \in \Pi$ for the penalized perfect information problem (2.42), where the penalty is $\hat{z}$ as defined in equation (2.43). Then for any non-AC policy $\pi \in \Pi^B$, there exists an AC policy $\pi^{AC} \in \Pi^{AC}$ such that $\tau_{\hat{z}}(\pi^{AC}) \leq \tau_{\hat{z}}(\pi)$.*

Proof. Proof. See §2.9.2. □

Following from Theorems 2 and 4, we may decompose the penalized perfect information problem into routing and charging decisions as before, so the objective function becomes

$$\mathbb{E}\left[\min_{\rho\in\mathcal{R}(s_0)}\left\{\min_{\pi\in\Pi_\rho}\sum_{k=0}^{K} C\left(s_k, X_k^{\pi}(s_k)\right) + v^{\mathbb{F}}_{k+1}\left(s_k, X_k^{\pi}(s_k)\right) - v^{\mathbb{I}}_{k+1}\left(s_k, X_k^{\pi}(s_k)\right)\right\}\middle| s_0\right]. \tag{2.45}$$

We can again estimate the objective value of (2.45) using simulation, as we did to estimate the unpenalized objective value with perfect information in (2.41). The inner minimization of (2.45) is still an FRVCP which can be modeled as a modified version of our original dynamic program, as in §2.4.3.2. To solve the penalized FRVCP, we use the classical reaching algorithm (Denardo 2003) that enumerates in forward-DP fashion all states that can be realized along a fixed CL sequence ρ. The restriction to AC policies in Theorem 4 is crucial, as it significantly reduces the number of realizable states that must be enumerated in the reaching algorithm. While time-consuming, the reaching algorithm allows for the consideration of nonlinear penalties, which can no longer be accommodated by the labeling algorithm nor by more classical solution methods, such as mixed integer-linear programs.

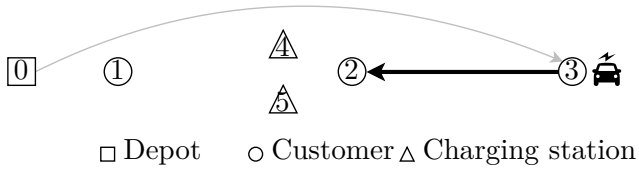

Figure 2.14: A vehicle at customer 3 in the beginning of epoch one. The vehicle must visit customers 2 and 1 before returning to the depot, but before it can visit customer 1, it must first charge. We illustrate the construction of information penalties using the action associated with the bolded arrow as an example.

For an example of the construction of information penalties, let us consider Figure 2.14 with the vehicle in state $s_1 = (t_{0,3}, 3, 0, Q - e_{0,3}, Q, \{2,1\}, 1)$. We assume CSs 4 and 5 are identical, meaning they have the same charging technology and number of chargers. Further, we assume that $t_{2,4} = t_{2,5}$ and $t_{4,1} = t_{5,1}$ (likewise for the energy to traverse these arcs). We compute a penalty for each action in the action space $\mathcal{A}(s_1)$, which consists of relocation actions to customer 2 and charging stations 4 and 5 (relocating to nodes 0 and 1 is energy infeasible). Abusing notation slightly, we have $\mathcal{A}(s_1) = \{a_2 \equiv (2, q_1 - e_{3,2}); a_4 \equiv (4, q_1 - 3_{3,4}); a_5 \equiv (5, q_1 - e_{3,5})\}$. In this example, we will illustrate the computation of the penalty $\hat{z}(s_1, a_2)$ corresponding to the action a_2 in which the EV relocates to customer 2.

First, from the post-decision state $s_1^{a_2}$, we sample realizations of queue dynamics at CSs 4 and 5. For simplicity, let us assume we are conducting a single sample denoted by $\omega \in \Omega$. We realize the (deterministic) exogenous information $W_2 = (t_1 + t_{3,2}, 1) \in \mathcal{I}(s_1^{a_2})$ and transition to state $s_2 = (t_{0,3} + t_{3,2}, 2, 3, Q - e_{0,3} - e_{3,2}, Q - e_{0,3}, \{1\}, 1)$. From this state, we wish to construct TSP Static policies $\pi(s_2, \mathbb{F})$ and $\pi(s_2, \mathbb{I})$ for use in $v_2^{\mathbb{F}}(s_1, a_2)$ and $v_2^{\mathbb{I}}(s_1, a_2)$, respectively. Per §2.5.1.2, the CL sequence followed by the vehicle will be the same under both filtrations, so we determine it first. To do so, we solve a single iteration of the outer minimization of equation (2.18). This finds the shortest Hamiltonian path from customer 2, through the remaining customers, terminating at the depot, which is the sequence $\rho = (2, 1, 0)$. Then, given ρ, we solve a single iteration of the inner minimization to establish the fixed route for the TSP static policies: we solve the FRVCP-N on ρ to construct the fixed route we call $p^{\mathbb{F}}$ and its corresponding policy $\pi(s_2, \mathbb{F}) = \pi(p^{\mathbb{F}})$, and we solve the FRVCP-P on ρ to construct the fixed route we call $p^{\mathbb{I}}$ with corresponding policy $\pi(s_2, \mathbb{I}) = \pi(p^{\mathbb{I}})$. For the former, let us assume that the expected waiting time at CS 4 is 40 min, and the expected waiting time at CS 5 is 45 min. This leads to the fixed-route solution $p^{\mathbb{F}} = ((2, q_2), (4, q_2 - e_{2,4}), (4, \tilde{q}), (1, \tilde{q} - e_{4,1}), (0, \tilde{q} - e_{4,1} - e_{1,0}))$, which includes a stop to charge at CS 4 to charge level $\tilde{q} = \min\{q \in \mathcal{Q}'\}$ where $\mathcal{Q}' = \{q \in \mathcal{Q} : q \geq e_{4,1} + e_{1,0}\}$. The cost of $p^{\mathbb{F}}$ we denote $\tau(\pi(p^{\mathbb{F}})) = t_{2,4} + 40 + \bar{u}(q_2 - e_{2,4}, \tilde{q}) + t_{4,1} + t_{1,0}$. For the FRVCP-P we proceed similarly, except now we have access to ω, which grants us knowledge of the queue dynamics at CS 4 and 5 at all points in time. Say we know the wait time at CS 4 will actually be 20 min, and the wait time at CS 5 will be 5 min. Then the solution to the FRVCP-P is the fixed route $p^{\mathbb{I}} = ((2, q_2), (5, q_2 - e_{2,5}), (5, \tilde{q}), (1, \tilde{q} - e_{5,1}), (0, \tilde{q} - e_{5,1} - e_{1,0}))$ with corresponding cost $\tau(\pi(p^{\mathbb{I}})) = t_{2,5} + 5 + \bar{u}(q_2 - e_{2,5}, \tilde{q}) + t_{5,1} + t_{1,0}$. The values $v_2^{\mathbb{F}}(s_1, a_2)$ and $v_2^{\mathbb{I}}(s_1, a_2)$ are then equal to the average of the route costs associated with $\pi(p^{\mathbb{F}})$ and $\pi(p^{\mathbb{I}})$, respectively, over samples from Ω (of which there is only one in this example). Thus, we

have $\hat{z}(s_1, a_2) = v_2^{\mathbb{F}}(s_1, a_2) - v_2^{\mathbb{I}}(s_1, a_2) = \mathbb{E}\left[\tau(\pi(p^{\mathbb{F}}))\right] - \mathbb{E}\left[\tau(\pi(p^{\mathbb{I}}))\right] = 40 - 5 = 35$, so the penalized cost of choosing action a_2 from state s_1 is $C(s_1, a_2) + \hat{z}(s_1, a_2) = t_{3,2} + 35$. The value of the penalty represents the benefit of using advanced information in decision making, capturing the difference in expected costs-to-go $\mathbb{E}\left[V_2(s_1, a_2)|\mathscr{F}_1\right]$ and $\mathbb{E}\left[V_2(s_1, a_2)|\mathscr{I}_1\right]$.

While the CL sequence $\rho = (2, 1, 0)$ will be the same for each sample from Ω, the same is not generally true of $p^{\mathbb{I}}$ and $p^{\mathbb{F}}$, which must be resolved for each sample of queue dynamics. This process is repeated for each action in the action space and at each decision epoch.

As the example illustrates, the application of information penalties increases computation significantly, which restricts the size of instances in which we can apply them. This exercise may not be not in vain, however, as methods that yield near-optimal policies for smaller instances may portend toward good methods for larger instances.

2.9.1 Experiments with Information Penalties

To demonstrate the utility of information penalties we seek instances for which access to perfect information is exceptionally valuable. These instances should result in a large gap between the performance of a non-anticipative policy and one with perfect information, making for a weak dual bound. Good information penalties should then tighten the dual bound, demonstrating that our policies are closer to the optimal policy than originally suggested by the PI bound. We attempt to construct such an instance here by 1) including "competing" charging stations between which the EV must choose, and 2) increasing the amount of stochastic costs (waiting costs) relative to deterministic costs (traveling and charging costs). The former produces more uncertainty and a larger action space, both of which stand to increase the value of perfect information. The latter aims to simply highlight this value.

Because the reaching algorithm used to solve the penalized FRVCP (the inner minimization of (2.45)) enumerates all reachable states along a fixed CL sequence, we must be mindful of instance size in these experiments. To ensure tractability, we construct an instance with four customers and two extradepot CSs. Further, we limit the set of chargeable battery states $\mathcal{Q}$ to the charging function breakpoints and multiples of 25% ($\mathcal{Q} = \{0, 0.25Q, 0.5Q, 0.75Q, 0.85Q, 0.95Q, Q\}$). Despite these restrictions, the computational effort required to solve just one realization of uncertainty with information penalties is almost ten minutes. This is in contrast to the negligible computation time (milliseconds) required to establish the perfect information bound for this instance, as well as execute all other routing policies.

The experimental results for this instance over 250 samples of uncertainty are shown in Figure 2.15. The figure shows the performance of the optimal policy with perfect information ("PI"), the optimal policy with penalized access to perfect information ("PI + Penalty"), and our best dynamic and static policies (PostTSP and the optimal static policy, respectively). The size of the gap between our best policy and the PI bound (15.3%) suggests that we were successful in creating an instance in which information was valuable. The penalties' potential is evident in these results, as they yield a dual bound that is more than twice as strong: the gap between our best non-anticipative policy and the dual bound is 7.6% with penalties, compared to 15.3% with the PI bound alone.

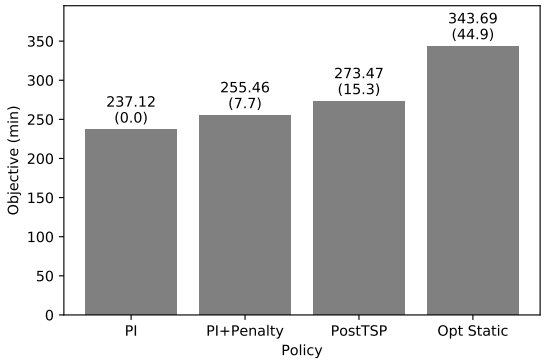

Figure 2.15: Comparing our best dynamic and static non-anticipative policies to the dual bounds afforded by the value of the optimal policy with perfect information and the value of the optimal policy with penalized access to perfect information. *Note: Bar labels are average objective achievement over 250 samples of uncertainty with percent difference from the PI bound in parentheses.*

To the best of our knowledge, these experiments represent the first successful demonstration of information penalties in vehicle routing and the first successful application of information penalties in general to a combinatorial perfect information problem lacking any special structure making the problem easier to solve. While scalability remains an issue, we hope that this serves as a proof-of-concept for future endeavors from other researchers.

2.9.2 Proof of Theorem 4

We begin by repeating the statement of Theorem 4:

Let $\tau_{\hat{z}}(\pi)$ be the value of a policy $\pi \in \Pi$ for the penalized perfect information problem (2.42), where the penalty is $\hat{z}$ as defined in equation (2.43). Then for any non-AC policy $\pi \in \Pi^B$, there exists an AC policy $\pi^{AC} \in \Pi^{AC}$ such that $\tau_{\hat{z}}(\pi^{AC}) \leq \tau_{\hat{z}}(\pi)$.

Proof. Proof. We proceed similarly as in the proof of Theorem 1. Consider a vehicle operating under the non-AC policy π which balks CSs. We wish to show that there exists an AC policy π^{AC} such that $\tau_{\hat{z}}(\pi^{\text{AC}}) \leq \tau_{\hat{z}}(\pi)$. We can construct such a policy by mimicking π, except when π balks a CS. In that case, the constructed policy π^{AC} would skip visiting the balked CS and proceed directly to the subsequent location. For instance, if the policy π dictates the relocation from some node j to a charging station c and then immediately relocate to j', policy π^{AC} would proceed directly from j to j'.

In the proof of Theorem 1, we relied on the triangle inequality and the fact that our queues are served first-in-first-out to reason that the constructed policy π^{AC} would outperform π. Now in the presence of penalties, while the FIFO principle still holds, it is less obvious that the triangle inequality holds. We prove here that it does by comparing the costs and penalties incurred between j and j' under policies π^{AC} and π. More specifically,

we want to show that

$$t_{j,j'} + \hat{z}(s_j, a_{j,j'}) \leq t_{j,c} + \hat{z}(s_j, a_{j,c}) + t_{c,j'} + \hat{z}(s_c, a_{c,j'}),$$

where s_j is the initial state of the vehicle at j; $a_{j,j'}$ is the action of traveling directly from j to j'; $a_{j,c}$ is the action of traveling from j to c; s_c is the state of the vehicle after taking action $a_{j,c}$ from state s_j; and $a_{c,j'}$ is the action of traveling from c to j'. The left-hand side of the equation represents the costs associated with traveling directly from j to j' (π^{AC}) and the right-hand side represents the costs associated with traveling from j to c, balking at c, then traveling to j' (π).

By the unpenalized triangle inequality, $t_{j,j'} \leq t_{j,c} + t_{c,j'}$, so it is sufficient to show that

$$\hat{z}(s_j, a_{j,j'}) \leq \hat{z}(s_j, a_{j,c}) + \hat{z}(s_c, a_{c,j'}). \tag{2.46}$$

Further, each penalty term $\hat{z}(s_k, a)$ is non-negative, because the terms are defined as $\hat{z}(s_k, a) = v^{\mathbb{F}}_{k+1}(s_k, a) - v^{\mathbb{I}}_{k+1}(s_k, a)$ and

$$v^{\mathbb{I}}_{k+1}(s_k, a) = \mathbb{E}\left[\min_{\pi \in \Pi_{\rho\text{TSP}}} \sum_{k'=k}^{K} C(s_{k'}, X^{\pi}_{k'}(s'_k))\right] \leq \min_{\pi \in \Pi_{\rho\text{TSP}}} \mathbb{E}\left[\sum_{k'=k}^{K} C(s_{k'}, X^{\pi}_{k'}(s'_k))\right] = v^{\mathbb{F}}_{k+1}(s_k, a).$$

The reversal of expectation and minimization that produces the middle inequality is a result of the use of perfect information in the construction of $v^{\mathbb{I}}_{k+1}(s_k, a)$. As a result, $\hat{z}(s_c, a_{c,j'}) \leq \hat{z}(s_j, a_{j,c}) + \hat{z}(s_c, a_{c,j'})$, so if we can show that

$$\hat{z}(s_j, a_{j,j'}) \leq \hat{z}(s_c, a_{c,j'}), \tag{2.47}$$

then we are done.

Writing the penalties explicitly and somewhat abusing notation for epoch indices, inequality (2.47) is equivalent to

$$v^{\mathbb{F}}_{j+1}(s_j, a_{j,j'}) - v^{\mathbb{I}}_{j+1}(s_j, a_{j,j'}) \leq v^{\mathbb{F}}_{c+1}(s_c, a_{c,j'}) - v^{\mathbb{I}}_{c+1}(s_c, a_{c,j'}). \tag{2.48}$$

Notice, however, that each term represents an expected cost-to-go from node j'. The terms on the left-hand side represent costs-to-go from node j' after traveling directly from j, while terms on the right-hand side represent costs-to-go after first balking CS c. Notice also that, for a given filtration, the cost-to-go from node j' cannot be better after balking at CS c than if having traveled directly. To prove this is the case, we refer the reader to Lemma 1.

Thus, $v^{\mathbb{F}}_{j+1}(s_j, a_{j,j'}) \leq v^{\mathbb{F}}_{c+1}(s_c, a_{c,j'})$ and $v^{\mathbb{I}}_{j+1}(s_j, a_{j,j'}) \leq v^{\mathbb{I}}_{c+1}(s_c, a_{c,j'})$, so equation (2.48) holds, meaning the triangle inequality also does in the presence of penalties. □

By Theorem 4, because the optimal policy for the penalized perfect information problem is AC, we can write its objective function as

$$\mathbb{E}\left[\min_{\pi \in \Pi} \sum_{k=0}^{K} C(s_k, X^{\pi}_k(s_k)) + \hat{z}\left(s_k, X^{\pi}_k(s_k)\right)\right] = \mathbb{E}\left[\min_{\pi \in \Pi^{\text{AC}}} \sum_{k=0}^{K} C(s_k, X^{\pi}_k(s_k)) + \hat{z}\left(s_k, X^{\pi}_k(s_k)\right)\right]$$
$$= \mathbb{E}\left[\min_{\rho \in \mathcal{R}(s_0)} \left\{\min_{\pi \in \Pi_\rho} \sum_{k=0}^{K} C\left(s_k, X^{\pi}_k(s_k)\right) + \hat{z}\left(s_k, X^{\pi}_k(s_k)\right)\right\}\right].$$

Restricting our search to the set of AC policies is especially convenient, because there are significantly fewer charging decisions to consider in the inner minimization.

Lemma 1 (Unimproved cost-to-go after balking a CS)**.** *Consider a vehicle in some state s_j at location j. The cost-to-go from a location j' as measured by the TSP static estimating policy is no greater if the vehicle travels directly from j to j' than if it travels j to $c \in \mathcal{C}$, balks c, then travels c to j'.*

Proof. Proof. Denote by $s_{k(j')}$ the resulting state of the vehicle that traveled directly j to j', and $s_{k(cj')}$ the resulting state of the vehicle that first balked at CS c. Recall that the TSP Static policy performs a single iteration of the outer minimization of equation (2.18), then solves the FRVCP for the resulting CL sequence. The CL sequence ρ^{TSP} resulting from a single solution of the master problem (2.19)-(2.25) will be the same for both $s_{k(j')}$ and $s_{k(cj')}$, so what we must show is that the value of the optimal policy produced by the solution to the subproblem for this sequence is no worse from state $s_{k(j')}$:

$$\min_{\pi \in \Pi_{\rho^{\text{TSP}}}} \mathbb{E}\left[\sum_{k=k(j')}^{K} C(s_k, X_k^{\pi}(s_k))\right] \leq \min_{\pi \in \Pi_{\rho^{\text{TSP}}}} \mathbb{E}\left[\sum_{k=k(cj')}^{K} C(s_k, X_k^{\pi}(s_k))\right]. \tag{2.49}$$

The left-hand side of (2.49) corresponds to the objective when traveling directly, and the right-hand side corresponds to the objective after balking. We proceed by contradiction.

For the statement (2.49) to be false, it must be the case that there is an action available *downstream* from state $s_{k(cj')}$ (in epochs $\{k(cj'), \ldots, K\}$) that yields a lower objective value and is not available downstream from state $s_{k(j')}$. As described in §2.4.3.2, the subproblem consists in finding the optimal charging decisions along ρ^{TSP} and can be modeled as a dynamic program with action space defined by (2.34)-(2.37). By the definition of this action space, the only actions exclusively available downstream from state $s_{k(cj')}$ are those in equation (2.35) that correspond to charging decisions to energy levels less than that with which the vehicle would arrive downstream from state $s_{k(j')}$. For such charging decisions to be in the set of feasible actions, it must be that the charge level is sufficient to reach the next stop in the CL sequence $n^{\star}$ and some subsequent CS c'. However, if this were the case, then – by the triangle inequality – the vehicle downstream from state $s_{k(j')}$ could simply skip the CS visit and instead proceed directly to $n^{\star}$, which would result in less incurred cost. Thus, it is not the case that there exists an action downstream from state $s_{k(cj')}$ that yields a lower objective value and is not available from state $s_{k(j')}$, so (2.49) holds. □

Chapter 3

Dynamic Ridehailing with Electric Vehicles[1]

3.1 Introduction

Governmental regulations as well as a growing population of environmentally conscious consumers have led to increased pressure for firms to act sustainably. This pressure is particularly high in the logistics domain, which accounts for about one third of emissions in the United States (Office of Transportation and Air Quality 2019). Ridehailing services offer a means to more sustainable transportation, promising to reduce the need for vehicle ownership, offering higher vehicle utilization (Lyft 2018), allowing transit authorities to streamline services (Bahrami et al. 2017), and stimulating the adoption of new vehicle technologies (Jones and Leibowicz 2019) In recent years, ridehailing services have seen rapid and widespread adoption, with the number of daily ridehailing trips more than quadrupling in New York City from November 2015 to November 2019 (New York City Taxi & Limousine Commission 2018).

Simultaneously, electric vehicles (EVs) are beginning to replace internal-combustion engine (ICE) vehicles, commanding increasingly more market share (Edison Electric Institute 2019). Coupling the pressures to act sustainably with EVs' promise of lower operating costs, ridehail companies are likely to be among the largest and earliest adopters of EV technology. Indeed most major ridehail companies have made public commitments to significant EV adoption (Slowik et al. 2019). However, EVs pose technological challenges to which their ICE counterparts are immune, such as long recharging times and limited recharging infrastructure (Pelletier et al. 2016).

In this work, we consider these challenges as posed to an operator of a ridehail company whose fleet consists of EVs. We further assume that the EVs in the fleet are centrally controlled and coordinated. While fleet control is somewhat centralized today, it is likely to become increasingly centralized as ridehail companies adopt autonomous vehicles (AVs).

[1] The research described in this chapter has been submitted for publication at *Transportation Science*. For a preprint, see

N. D. Kullman, M. Cousineau, J. C. Goodson, and J. E. Mendoza. Dynamic Ridehailing with Electric Vehicles. Working paper, Jan. 2020a. URL https://hal.archives-ouvertes.fr/hal-02463422

As with EVs, ridehail companies are likely to be among the earliest and largest adopters of AVs, as they stand to offer many benefits including reduced operating costs and greater efficiency and predictability (Fagnant and Kockelman 2015). For brevity, we refer to this problem as the Electric Ridehail Problem with Centralized control, or E-RPC.

Our contributions in this work are as follows: 1) We offer the first application of deep reinforcement learning (RL) to the E-RPC, developing policies that respond in real time to serve customer requests and anticipate uncertain future demand under the additional constraints of fleet electrification. The policies are *model-free*, meaning they learn to anticipate this demand without any prior knowledge of its shape. We compare these deep RL-based policies to a common heuristic in the taxi dispatching literature. 2) We evaluate these policies on instances constructed with real data reflecting ridehailing operations from New York City in 2018. 3) We establish a dual bound for the dynamic policies using a perfect information relaxation that we solve using a Benders-like decomposition. We compare this complex dual bound against a simpler dual bound and provide an analysis of when the additional complexity of the perfect information bound is valuable. 4) We show that our best deep RL-based solution significantly outperforms the benchmark dispatching heuristic and comes within 19% of an optimal policy with perfect information. 5) We further demonstrate that the best-performing deep RL-based policy can be scaled to larger problem instances without additional training. This is encouraging for operators of ridehail companies, as it suggests robustness to changes in the scale of operations: in the event of atypical demand or a change in the number of vehicles (e.g., due to fleet maintenance), the policy should still provide reliable service.

We begin by reviewing related literature in §3.2, then provide a formal problem and model definition in §3.3. We describe our solution methods in §3.4, the bounds established to gauge the effectiveness of these methods in §3.5, and demonstrate their application in computational experiments in §3.6. We offer brief concluding remarks in §3.7.

3.2 Related Literature

Ridehail problems (RPs), those addressing the operation of a ridehailing company, fall under the broader category of dynamic vehicle routing problems (VRPs). Within dynamic VRPs, they may be classified as a special case of the dynamic VRP with pickups and deliveries or, more precisely, a special case of the dynamic dial-a-ride problem. Recent technological advances in mobile communications have driven new opportunities in ridehailing and other *mobility-on-demand* (MoD) services, reinvigorating research in this area. As a result, there now exists a substantial body of literature specifically pertaining to MoD applications within the dial-a-ride domain. This literature is the focus of our review. For a broader survey of the dynamic dial-a-ride literature, we refer the reader to Ho et al. (2018), and, similarly, for the dynamic VRP literature, to Psaraftis et al. (2016).

Often studies of RPs focus their investigation on the assignment problem, wherein the operator must choose how to assign fleet vehicles to new requests. For example, Lee et al. (2004) propose the use of real-time traffic data to assign the vehicle which can reach the request fastest. A study by Bischoff and Maciejewski (2016) uses a variant of this assignment heuristic that accounts for whether demand exceeds supply (or vice versa), as well as

vehicles' locations within the service region. Seow et al. (2009) propose a decentralized heuristic that gathers multiple requests and allows vehicles to negotiate with one another to determine which vehicles serve the new requests. They show that it outperforms heuristics like those in Lee et al. (2004) and Bischoff and Maciejewski (2016). Hyland and Mahmassani (2018) introduce a suite of optimization-based assignment strategies and show that they outperform heuristics that do not employ optimization. Following suit, Bertsimas et al. (2019) describe ways to reduce the complexity of optimization-based approaches for problems with large fleets and many customers. They demonstrate their proposed methods on realistically-sized problem instances that reflect ridehailing operations in New York City.

The aforementioned studies largely ignore the task of repositioning idle vehicles in anticipation of future demand. This is in contrast to studies such as Miao et al. (2016), Zhang and Pavone (2016), and Braverman et al. (2019) who address their RP from the opposing perspective of the fleet repositioning problem. Miao et al. (2016) do so using learned demand data with a receding horizon control approach. Braverman et al. (2019) use fluid-based optimization methods for the repositioning of idle vehicles to maximize the expected value of requests served. They establish the optimal static policy and prove that it serves as a dual bound for all (static or dynamic) policies under certain conditions. Other studies such as Fagnant and Kockelman (2014) and Alonso-Mora et al. (2017) consider strategies for both the assignment and repositioning tasks. The former does so using rule-based heuristic strategies, evaluating them using an agent-based model; the latter using optimization-based methods.

A recent competing method to address ridehail problems is reinforcement learning (RL), predominantly deep reinforcement learning. As it is well-suited to address problems with larger fleet sizes, many studies employing RL take a multi-agent RL (MARL) approach (Oda and Tachibana (2018), Oda and Joe-Wong (2018), Holler et al. (2019), Li et al. (2019), Singh et al. (2019)). In work sharing many similarities to ours, Holler et al. (2019) use deep MARL with an attention mechanism to address both the assignment and repositioning tasks. They compare performance under two different MARL approaches: one in which a system-level agent coordinates vehicle actions to maximize total reward; and one in which vehicle-level agents act individually, each maximizing its own reward. Oda and Joe-Wong (2018) use a deep MARL approach to tackle fleet repositioning, but rely on a myopic heuristic to perform assignment. Oda and Tachibana (2018) do so as well, but employ special network architectures with soft-Q learning which they argue better accommodates the inherent complexities in road networks and traffic conditions. Li et al. (2019) employ a multi-agent RL framework, comparing two approaches that vary in what individual vehicles know regarding the remainder of the fleet; however, in contrast to most MARL applications to RPs, they address only the assignment problem, assuming that the vehicles are de-centralized and therefore not repositioned by the operator. Similarly assuming de-centralized vehicles and addressing only the assignment problem, Xu et al. (2018) combine deep RL with the optimization-based methods used in, e.g., Alonso-Mora et al. (2017), Zhang et al. (2017), and Hyland and Mahmassani (2018). In their work, Xu et al. use deep RL to learn the objective coefficients in a math program that is used in an optimization framework to assign vehicles to requests.

Missing from all aforementioned studies is fleet electrification — none take into account the technical challenges associated with electric vehicles, such as long recharging times or

limited recharging infrastructure. The number of studies that consider these constraints is limited but growing. Most (e.g., Jung et al. (2012), Chen et al. (2016), Kang et al. (2017), Iacobucci et al. (2019), La Rocca and Cordeau (2019)) use the heuristics and optimization methods previously mentioned. We focus here on three works closely related to ours that employ learning-based approaches. First, Al-Kanj et al. (2018) use approximate dynamic programming to learn a hierarchical interpretation of a value function that is used to determine whether or not to recharge vehicles at their current location, in addition to repositioning decisions to neighboring grid cells and the assignment of vehicles to nearby requests. The study considers the possibility for riders or the operator to reject a trip assignment at various costs, and it also considers the problem of fleet sizing. Second, Pettit et al. (2019) use deep RL to learn a policy determining when an EV should recharge its battery and when it should serve a new request. The study considers time-dependent energy costs. However, it only considers a single vehicle, and it ignores the task of repositioning the vehicle in anticipation of future requests. Finally, Shi et al. (2019) apply deep reinforcement learning to the RP with a community-owned fleet, in which they seek to minimize customer waiting times and electricity and operating costs. Similar to many of the RL studies cited previously, the study employs a multi-agent RL framework that produces actions for each vehicle individually which are then used in a centralized decision-making mechanism to produce a joint action for the fleet. The joint actions assign vehicles to requests and specify when vehicles should recharge; however the study also ignores repositioning actions.

Thus, ours marks the first application of deep reinforcement learning to the E-RPC in consideration of EV constraints alongside both fleet repositioning and assignment tasks. We also argue that we consider a more realistic problem environment than in most RL-based approaches to ridehail problems. Indeed, the vast majority of studies (e.g., Al-Kanj et al. (2018), Oda and Tachibana (2018), Holler et al. (2019), Shi et al. (2019), Singh et al. (2019)) rely on a coarse grid-like discretization to describe the underlying service region. This serves as the basis for their repositioning actions, which are often restricted to only the adjacent grid cells. This is in contrast to the current work, in which we allow repositioning to non-adjacent sites that correspond to real charging station locations. Time discretization can also be overly coarse. Many studies choose one minute between decision epochs, as in, for example, Oda and Joe-Wong (2018); but some choose far longer between decision epochs, such as six minutes in Shi et al. (2019) and 15 minutes in Al-Kanj et al. (2018). Given that current ridehailing applications provide fast, sub-minute responses, such an arrangement would likely lead to customer dissatisfaction. We are free of such a discretization here, allowing agents to respond to customer requests immediately.

3.3 Problem Definition and Model

We consider a central operator controlling a homogeneous fleet of electric vehicles $\mathcal{V} = \{1, 2, \ldots, V\}$ that serve trip requests arising randomly across a given geographical region and over an operating horizon T. Across the region are charging stations (CSs, or simply *stations*) $\mathcal{C} = \{1, 2, \ldots, C\}$ at which vehicles may recharge or idle when not in service. The operator assigns vehicles to requests as they arise. Additionally, the operator manages recharging and repositioning operations in anticipation of future requests. The objective is

to maximize expected revenue across the operating horizon.

Several assumptions and conditions connect the problem to real-world operations. When a new request arises, the operator responds immediately, either rejecting the request or assigning it to a vehicle. A vehicle is eligible to serve a new request if it can pickup within w time units, the amount of time customers are willing to wait. Each vehicle maintains at most one pending trip request. Thus, a vehicle either serves a newly assigned request immediately or does so after completing work-in-process. Assignments of requests to vehicles cannot be canceled, rescheduled, or reassigned. Each vehicle has a maximum battery capacity Q and known charging rate, energy consumption rate, and speed. The operator ensures assignments and repositioning movements are energy feasible.

We formulate the problem as a Markov decision process (MDP) whose components are defined as follows.

States.

A *decision epoch* $k \in \{0, 1, \ldots, K\}$ begins with a new trip request or when a vehicle finishes its work-in-process, whichever occurs first. The system state is captured by the tuple

$$s = (s_t, s_r, s_{\mathcal{V}}). \tag{3.1}$$

The current time $s_t = (t, d)$ is the time of day in seconds t and the day of week d. If the epoch was triggered by a new request, then $s_r = ((o_x, o_y), (d_x, d_y))$ consists of the Cartesian coordinates for the request's origin and destination, otherwise $s_r = \emptyset$. The vector $s_{\mathcal{V}} = (s_v)_{v \in \mathcal{V}}$ describes the state of each vehicle in the fleet. For a vehicle v, $s_v = (x, y, q, j_m^{(1)}, j_o^{(1)}, j_d^{(1)}, j_m^{(2)}, j_o^{(2)}, j_d^{(2)}, j_m^{(3)}, j_o^{(3)}, j_d^{(3)})$, consisting of the Cartesian coordinates of the vehicle's current position x, y; its charge $q \in [0, Q]$; and a description of its current activity (or *job*) $j^{(1)}$, as well as potential subsequent jobs $j^{(2)}$ and $j^{(3)}$, which may or may not exist, as determined by the operator. The description of a job $j^{(i)}$ consists of its type $j_m^{(i)} \in \{\text{idle, charge, reposition, preprocess, serve}, \emptyset\}$ (equivalently, $\{0, 1, 2, 3, 4, \emptyset\}$; "preprocess" refers to when a vehicle is en route to pickup a customer), as well as the coordinates of the job's origin $j_o^{(i)} = (j_{o,x}^{(i)}, j_{o,y}^{(i)})$ and destination $j_d^{(i)} = (j_{d,x}^{(i)}, j_{d,y}^{(i)})$. We limit the number of tracked jobs in the state to three, because this is the maximum number of scheduled jobs a vehicle can have given the constraint that a vehicle may have at most one pending trip request: if a vehicle is currently serving a request and has a pending request, then it will have jobs of type $j_m^{(1)} = 4$, $j_m^{(2)} = 3$, and $j_m^{(3)} = 4$. Note that we do not allow vehicles currently preprocessing for one request to be assigned a second (this would indeed require tracking a fourth job). With a sufficiently strict customer service requirement (i.e., the time between receipt of a customer's request and a vehicle's arrival to the customer), the number of opportunities to meet that requirement after undergoing the necessary (preprocessing, serving, preprocessing) sequence is small enough as to warrant the situation's exclusion. If a vehicle has $n < 3$ scheduled jobs, then we set $j_m^{(i)} = j_o^{(i)} = j_d^{(i)} = \emptyset$ for $n < i \leq 3$.

An *episode* begins with the system initialized in state s_0 in epoch 0 at time 0 on some day d with no new request ($s_r = \emptyset$) and all vehicles idling with some charge at a station

($v \in \mathcal{C}$, $q \in [0, Q]$, $j_m^{(1)} = \text{idle}$, $j_m^{(2)} = j_m^{(3)} = \emptyset$ for all $v \in \mathcal{V}$). The episode terminates at some epoch K in state s_K when the time horizon T has been reached.

Actions.

When the process occupies state s, the set of actions $\mathcal{A}(s)$ defines feasible assignments of vehicles to a new request as well as potential repositioning and recharging movements. An action $\mathbf{a} = (a_r, a_1, \ldots, a_v, \ldots, a_V)$ is a vector comprised of components $a_r \in \mathcal{V} \cup \{\emptyset\}$ denoting which vehicle serves new request s_r as well as a repositioning/recharging assignment a_v for each vehicle v indicating the station to which it should be repositioned and begin recharging $a_v \in \mathcal{C} \cup \{\emptyset\}$. $a_r = \emptyset$ denotes rejection of the new request (if it exists) and $a_v = \emptyset$ denotes the "no operation" (NOOP) action for vehicle v, meaning it proceeds to carry out its currently assigned jobs. We refer to the vehicle-specific repositioning/recharging/NOOP actions a_v as RNR actions. Jobs of type *idle* (0), *charge* (1), *reposition* (2), and $\emptyset$ are preemptable, meaning they can be interrupted by a new assignment from the operator, whereas jobs of type *preprocess* (3) and *serve* (4) are non-preemptable.

The following conditions characterize $\mathcal{A}(s)$. If there is no new request ($s_r = \emptyset$), then $a_r = \emptyset$. If there is a new request, then a vehicle is eligible for assignment if it can be dispatched immediately ($j_m^{(1)} \in \{0, 1, 2, \emptyset\}$) or if it is currently servicing a customer ($j_m^{(1)} = 4$) and can then be dispatched ($j_m^{(2)} \notin \{3, 4\}$). Further, a vehicle-request assignment must be energy feasible and must arrive to the customer within w time units, the maximum amount of time a customer is willing to wait. Let $f_q(s_v, s_r)$ be the maximum charge with which vehicle v can reach a station after serving request s_r. $f_q(s_v, s_r)$ is equal to the current charge of vehicle v, minus the charge required for the vehicle to travel to the origin of the new request (minus the charge needed to first drop off its current customer, if $j_m^{(1)} = 4$), to the destination of the new request, and then to the nearest $c \in \mathcal{C}$ from the destination. Let $f_t(s_v, s_r)$ be the duration of time required for vehicle v to arrive at the customer. $f_t(s_v, s_r)$ is equal to the time for vehicle v to travel to the origin of the new request from its current location (plus the time to first drop off its current customer, if $j_m^{(1)} = 4$). An assignment of vehicle v to request s_r is feasible if $f_q(s_v, s_r) \geq 0$ and $f_t(s_v, s_r) \leq w$. Repositioning and recharging movements must also be energy feasible and may not preempt existing service assignments. If a vehicle is currently serving or preparing to serve a request ($j_m^{(1)} \in \{3, 4\}$), or if the vehicle has been selected to serve the new request ($a_r = v$), then we only allow the NOOP action $a_v = \emptyset$. Otherwise, we allow repositioning to any station $c \in \mathcal{C}$ that can be reached given the vehicle's current charge level. Lastly, if a vehicle completes a trip request, triggering a new epoch, and that vehicle has not been assigned a subsequent trip request, then it must receive a repositioning action ($a_v \neq \emptyset$). This condition forces vehicles to idle only at stations. Eligible actions for an EV are summarized in Table 3.1.

Reward Function.

When the process occupies state s and action $\mathbf{a}$ is selected, we earn a reward $C(s, \mathbf{a})$ if we assign a vehicle to serve the new trip request s_r. The reward consists of a base fare plus a charge proportional to the distance of the request. Letting $d(s_r)$ be the distance from the

Table 3.1: Eligible actions for an EV v.

Action	Eligibility conditions
Serve request	Request exists, no pending service ($j_m^{(2)} < 3$), energy feasible, time feasible
Reposition to a_v	No active or pending service ($j_m^{(1)} < 3$), energy feasible
NOOP	Always, unless just finished serving and no pending service ($j_m^{(1)} = \emptyset$)

request's origin to its destination, then the reward is

$$C(s, \mathbf{a}) = \begin{cases} 0 & a_r = \emptyset \\ C_b + C_d d(s_r) & \text{otherwise,} \end{cases} \tag{3.2}$$

where C_b is the base fare, and C_d is the charge per unit distance.

Transition Function.

The transition from one state to the next is a function of the selected action and the event triggering the next epoch. The subsequent state s' is constructed by updating the current state s to reflect changes to vehicles' job descriptions based on selected action $\mathbf{a}$. Then, at time s'_t, we observe either a new request s'_r or a vehicle completing its work-in-process. At that time, we update the vehicle states $s'_{\mathcal{V}}$ to reflect new positions and charges. We also update vehicles' job descriptions. If a vehicle has completed n jobs between time s_t and s'_t, we left-shift vehicles' job descriptions by n ($j^{(i)} \leftarrow j^{(i+n)}$ for $1 \le i \le 3-n$) and backfill the vacated entries with null jobs ($j_m^{(i)} = j_o^{(i)} = j_d^{(i)} = \emptyset$ for $3-n < i \le 3$).

More formally, upon choosing action $\mathbf{a}$ from state s, we update vehicles' job descriptions as follows. Let the current location of vehicle v be (v_x, v_y).

- **NOOP.** Vehicles receiving the NOOP instruction ($a_v = \emptyset$) see no change to their existing job descriptions.

- **Repositioning/Recharging.** Let the location to which the vehicle is instructed to reposition be $a_v = c$ with location (c_x, c_y). We set the first job's type to repositioning ($j_m^{(1)} = 2$), its origin to the vehicle's current location ($j_o^{(1)} = (v_x, v_y)$), and its destination to c ($j_d^{(1)} = (c_x, c_y)$). We then set the second job type to charging ($j_m^{(2)} = 0$) which begins and ends at c ($j_o^{(2)} = j_d^{(2)} = (c_x, c_y)$). After recharging, the vehicle then idles at the station: $j_m^{(3)} = 0$ and $j_o^{(3)} = j_d^{(3)} = (c_x, c_y)$.

- **Serving Request.** Let the new request have origin (o_x, o_y) and destination (d_x, d_y).

  No work-in-process. If the vehicle is not already serving a request ($j_m^{(1)} \neq 4$), then its first job is updated to type *preprocess* ($j_m^{(1)} = 3$) with origin $j_o^{(1)} = (v_x, v_y)$ and destination $j_d^{(1)} = (o_x, o_y)$. Its second job is then serving the customer, so we set the type to serve ($j_m^{(2)} = 4$), the origin $j_o^{(2)} = (o_x, o_y)$, and the destination $j_d^{(2)} = (d_x, d_y)$. The vehicle's third job is then empty, so we set $j_m^{(3)} = j_o^{(3)} = j_d^{(3)} = \emptyset$.

Work-in-process. If the vehicle is currently serving an existing customer ($j_m^{(1)} = 4$), then the first job is unchanged. The second job is *preprocess* ($j_m^{(2)} = 3$), as the vehicle moves from the drop-off location of its current customer ($j_o^{(2)} = j_d^{(1)}$) to the pick-up location of the new customer ($j_d^{(2)} = (o_x, o_y)$). The vehicle's third job is then to serve the customer request, so we set the type to serve ($j_m^{(3)} = 4$), the origin $j_o^{(3)} = (o_x, o_y)$, and the destination $j_d^{(2)} = (d_x, d_y)$.

Objective Function.

Define a policy π to be a sequence of decision rules $(X_0^\pi, X_1^\pi, \ldots, X_K^\pi)$, where X_k^π is a function mapping state s in epoch k to an action $\mathbf{a}$ in $\mathcal{A}(s)$. We seek to provide the fleet operator with a policy $\pi^\star$ that maximizes the expected total rewards earned during the time horizon, conditional on the initial state:

$$\tau(\pi^\star) = \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0 \right], \tag{3.3}$$

where Π is the set of all policies. As is common in reinforcement learning, we will often refer to the user or implementer of a policy as an *agent*.

3.4 Solution Methods

We develop four policies to solve E-RPC, including a random policy that serves as a lower bound. We describe these policies in §3.4.2 after first providing a brief description of deep reinforcement learning (in §3.4.1) as it is the foundation of two of our policies.

3.4.1 Deep Reinforcement Learning

As defined in Sutton and Barto (2018), reinforcement learning (RL) refers to the process through which an *agent*, sequentially interacting with some environment, learns what to do so as to maximize a numerical reward signal from the environment; that is, learns a policy satisfying equation (3.3). In practice, the agent often achieves this by learning the value of choosing an action a from some state s, known as the state-action pair's Q-*value* ($Q(s, a)$), equal to the immediate reward plus the expected sum of future rewards earned by taking action a from state s. We can express this relationship recursively following from the Bellman equation:

$$Q(s, a) = C(s, a) + \gamma \mathbb{E}\left[\max_{a' \in \mathcal{A}(s')} Q(s', a') \middle| s, a \right], \tag{3.4}$$

where s' is the subsequent state reached by taking action a from state s and $\gamma \in [0, 1)$ is a discount factor applied to the value of future rewards. With knowledge of Q-values for all state-action pairs it may encounter, the agent's policy is then to choose the action with the largest Q-value. However, as the number of unique state-action pairs is too

large to learn and store a value for each, a functional approximation of these Q-values is learned. When deep artificial neural networks are used for this approximation, the method is called *deep reinforcement learning* or, more specifically, *deep Q-learning* (we use the terms interchangeably here). The neural network used in this process is referred to as the *deep Q-network* (DQN).

Beginning with arbitrary Q-value approximations, the agent's DQN improves through its interactions with the environment, remembering observed rewards and state transitions, and drawing on these memories to update weights θ defining its DQN. Specifically, with each *step* (completion of an epoch) in the environment, the agent stores a memory which is a tuple of the form (s, a, r, s'), consisting of a state s, the action a taken from s, the reward earned $r = C(s, a)$, and the subsequent state reached s'. Once a sufficient number of memories (defined by the hyperparameter M_{start}) have been accumulated, the agent begins undergoing *experience replay* every M_{freq} steps (Lin 1992). In experience replay, the agent draws a random sample (of size M_{batch}) from its accumulated memories and uses them to update its DQN via stochastic gradient descent (SGD) on its weights θ using a loss function based on the difference

$$r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a') - Q(s, a) \tag{3.5}$$

(or simply $r - Q(s, a)$ if s' is terminal), where the Q-values in equation (3.5) are estimated using the agent's DQN. The specific loss function and SGD optimizer used to update the weights may vary — we provide the specific implementations chosen for this work in §3.6.2.

We utilize the learned Q-values via an *ϵ-greedy* policy, wherein the agent chooses actions randomly with some probability ϵ and chooses the action with the highest predicted Q-value with probability $1 - \epsilon$. The value of ϵ is decayed from some initially large value ϵ_i at the beginning of training to some small final value ϵ_f over the course of some number ϵ_N of training steps. This encourages exploration when Q-values are unknown and exploitation as its predictions improve.

3.4.1.1 Implemented extensions to deep RL.

We adopt several well established extensions that improve the standard deep RL process just described. First, we utilize a more sophisticated sampling method during experience replay known as *prioritized experience replay* (PER) (Schaul et al. 2016). In PER, memories are more likely to be sampled if they are deemed more important, i.e., if the loss associated with a memory is not yet known (because the memory has not yet been sampled during replay) or if the loss is large (if the agent was more "surprised" by the memory). Second, we use the double DQN (DDQN) architecture (Hasselt et al. 2016), which employs a "target" DQN for action evaluation and a "primary" DQN for action selection. The target DQN is a clone of the primary DQN that gets updated less often (every M_{update} steps), which helps break the tendency for DQNs to overestimate Q-values. Third, we employ dueling DDQN (D3QN) architecture (Wang et al. 2016), in which the DQN produces an estimate both of the value of the state and of the relative *advantage* of each action. By producing these values independently (which together combine to yield Q-values), dueling architecture allows the agent to forgo the learning of action-specific values in states in

which its decision-making is largely irrelevant. Finally, we use n-step learning (Sutton 1988), which helps to reduce error in learned Q-values. With n-step learning, the rewards r and subsequent states s' stored in an agent's memories are modified: rather than using the state encountered one step into the future from s (s'), the subsequent state stored in the memory is that encountered n-steps into the future ($s^{(n)}$). Similarly, the reward r is replaced by the (properly discounted) sum of the next n rewards, i.e., those accumulated between s and $s^{(n)}$.

The process of choosing which deep RL extensions to implement and values for associated hyperparameters is not unlike traditional heuristic approaches in transportation optimization. While simple heuristics may suffice in simpler problems (see, e.g., Clarke and Wright (1964)), more complex problems often demand more complex (meta)heuristics specifically tuned for a particular application (see, e.g., Gendreau et al. (2002)). Analogously, off-the-shelf deep RL may be successful in simpler problems (e.g., Karpathy (2016)), but success in more complex problems requires enhancements (e.g., Hessel et al. (2018)) and is dependent on hyperparameter values (Henderson et al. 2018). The suite of deep RL extensions adopted here and our selection of hyperparameter values (§3.6.2) are guided by standard values where available and ultimately chosen empirically over the course of the work.

3.4.2 Policies

We begin by describing *Nearest*, a heuristic policy whose action selection is derived from distance-based rules commonly employed in dynamic taxi dispatching (Maciejewski et al. 2016). We then describe an augmentation of this policy that maintains the Nearest heuristic for RNR decisions but uses deep RL to determine which vehicle to assign to new trip requests. We refer to this policy as *deep ART* (for "Assigns Requests To vehicles") or just *Dart*. Finally, we describe the *deep RAFTR* ("Request Assignments and FleeT Repositioning") or *Drafter* agent, which uses deep RL both to assign vehicles to requests and to reposition vehicles in the fleet.

3.4.2.1 Nearest.

Given a state s with new request s_r and vehicle states $(s_v)_{v\in\mathcal{V}}$, the Nearest policy chooses an action $\mathbf{a}$ as follows. To serve the new request, it selects the vehicle which can reach the customer fastest: $a_r = \arg\min_{v\in\bar{\mathcal{V}}} f_t(s_v, s_r)$, where $\bar{\mathcal{V}}$ is the set of vehicles that are eligible to serve the request given time, energy, and job constraints. For vehicle-specific RNR decisions a_v, all vehicles that do not have work-in-process receive instructions to reposition to the nearest station, which we may denote $c_v^\star$ for vehicle v. This yields RNR actions

$$a_v = \begin{cases} c_v^\star & \text{if } j_m^{(1)} \notin \{3, 4\} \\ \emptyset & \text{otherwise.} \end{cases} \tag{3.6}$$

3.4.2.2 Dart.

The Dart agent uses a combination of heuristics and deep Q-learning to choose an action $\mathbf{a} = (a_r, (a_v)_{v \in \mathcal{V}})$ from state $s = (s_t, s_r, s_{\mathcal{V}})$. Specifically, Dart selects RNR actions a_v using the same logic as Nearest (equation (3.6)); however, the action a_r pairing a vehicle with the new request now employs a deep Q-network. We denote by $\mathcal{A}_{\text{Dart}} = \mathcal{V} \cup \{\emptyset\}$ the action space for Dart's DQN. The DQN takes as input the concatenation of a subset of features from the state s which we represent by $x_{\text{Dart}} = x_t \oplus x_r \oplus x_{\mathcal{V}}$ ("$\oplus$" is the concatenation operator). These components are defined as follows:

x_t **System time** : the concatenation of the relative time elapsed t/T and a *one-hot vector* indicating the day of the week d (e.g., for zero-indexed d with $d = 0$ corresponding to Monday, $d = 6$ corresponding to Sunday, and d currently equal to Thursday ($d = 3$), the vector $(0, 0, 0, 1, 0, 0, 0)$).

x_r **Request information** : the concatenation of a binary indicator for whether there is a new request $\mathbf{1}_{s_r \neq \emptyset}$; a one-hot vector indicating the location of the request's origin; a one-hot vector indicating the request's destination; and for each vehicle v, the distance v would have to travel to reach the request's origin (scaled by the maximum such distance so values are in $[0, 1]$ (ineligible vehicles are given value 1)).

$x_{\mathcal{V}}$ **Vehicle information** : the concatenation of x_v for all vehicles $v \in \mathcal{V}$ ($x_{\mathcal{V}} = x_1 \oplus \cdots \oplus x_V$), where the vehicle-specific x_v is itself the concatenation of the time at which its last non-preemptable job ends (scaled by $1/T$), the charge with which it will finish its last non-preemptable job (scaled by $1/Q$), and a one-hot vector indicating its current location.

One-hot vectors in x_{Dart} that indicate location rely on a discretization of the fleet's operating region, as is common in other dynamic ridehail problems in the literature (e.g., Al-Kanj et al. (2018), Holler et al. (2019)). We denote the set of discrete locations (*taxi zones* (TZs)) constituting the region by $\mathcal{L}$.

x_{Dart} is passed to the agent's DQN to produce Q-value predictions. It then uses an ϵ-greedy policy as described in §3.4.1 to choose an $a_r \in \mathcal{A}_{\text{Dart}}$ (infeasible vehicles are ignored). A schematic of Dart's DQN is shown at left in Figure 3.1.

3.4.2.3 Drafter.

From a state s, the Drafter agent uses a DQN to choose a vehicle to serve new requests a_r and to provide RNR instructions a_v for each vehicle $v \in \mathcal{V}$. Whereas the number of unique actions under the control of Dart's DQN is $V + 1$, for Drafter it is $(V + 1) \times (C + 1)^V$. For non-trivial fleet sizes V, this is intractably large. In response, we employ a multi-agent reinforcement learning (MARL) framework which avoids this intractability by making separate, de-centralized Q-value predictions for each vehicle. This reduces the number of actions under the control of Drafter's DQN to $C + 2$, corresponding to the vehicle-specific actions of relocating to each station $c \in \mathcal{C}$, serving the request, and doing nothing. We further reduce the number of actions by using the discretization into TZs $\mathcal{L}$ described in §3.4.2.2, yielding the action space $\mathcal{A}_{\text{Drafter}} = \{\text{serve}, \emptyset\} \cup \mathcal{L}$ for Drafter's DQN. These

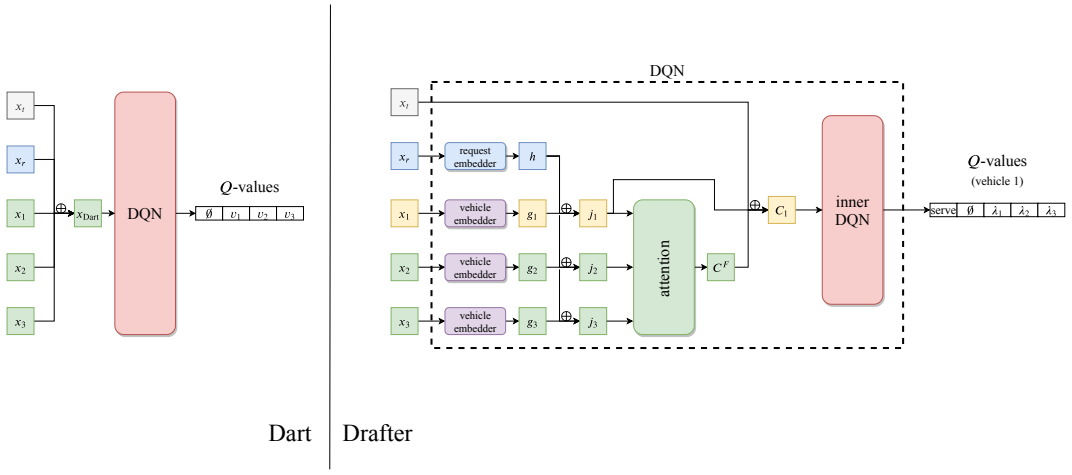

Figure 3.1: Schematics of the Dart and Drafter agents in the case of three vehicles and three TZs ($V = 3$, $\mathcal{L} = 3$). Elements are concatenated at intersections marked with $\oplus$. The Drafter schematic shows the case where Q-values are being predicted for vehicle 1 (in yellow). Note: in practice, a single forward pass with x_{Drafter} can be used to generate Q-values for all vehicles. We depict the process for a single vehicle here for the sake of clarity.

actions correspond respectively to serving the request, doing nothing, and relocating to each TZ $\lambda \in \mathcal{L}$. For vehicle v, we define its DQN input by $x^v_{\text{Drafter}} = (x_t, x_r, x_v, x_{\mathcal{V}})$ where components are as defined for Dart in §3.4.2.2 and the redundant x_v is included just to indicate the vehicle for which we are generating Q-values. With vehicle input x^v_{Drafter}, the DQN makes Q-value predictions $Q(x^v_{\text{Drafter}}, a)$ for $a \in \mathcal{A}_{\text{Drafter}}$. These predictions are collected for all vehicles and used to make a centralized decision. A schematic of Drafter's DQN is shown at right in Figure 3.1.

DQN and attention mechanism. Let us consider the prediction of Q-values for a vehicle $v^\star$. In the example in Figure 3.1, $v^\star = v_1$. We now also incorporate into the DQN an *attention mechanism* (Mnih et al. 2014) to improve the agent's ability to distinguish the most relevant vehicles in the current state. We expand on the attention mechanism used in Holler et al. (2019) by incorporating information about the current request, which is likely to influence vehicles' relevancy. Details of our attention mechanism can be found in §3.6.2.1. The attention is used to provide an alternative representation (an *embedding*) of each vehicle $g_v \in \mathbb{R}^l$ and the request $h \in \mathbb{R}^m$, as well as a description of the fleet $C^F \in \mathbb{R}^n$ known as the *fleet context* (here $l = n = 128$, $m = 64$; see §3.6.2.1). These components are used to form the vector $C_{v^\star} = C^F \oplus g_{v^\star} \oplus h \oplus x_t$ which is fed to an inner DQN. This inner DQN, which has a structure similar to Dart's DQN, outputs Q-values $Q(x^{v^\star}_{\text{Drafter}}, a)$ for $a \in \mathcal{A}_{\text{Drafter}}$. The process is repeated for each $v \in \mathcal{V}$, with each vehicle using the same attention mechanism and inner DQN (i.e., all weights are shared).

Drafter is made scalable both through its use of an MARL approach as described above, and also by the attention mechanism. While the MARL approach ensures that the output

of the DQN does not depend on the size of the fleet, the attention mechanism ensures the same for the input. More specifically, the attention mechanism makes it possible to predict Q-values (via the inner DQN) using a representation of the fleet C^F whose size is not determined by the number of vehicles in the fleet. This structure allows Drafter to be applied to instances of a different size than those on which it was trained. For example, given a Drafter agent trained to operate a fleet of size V, if a new vehicle is purchased and added to the fleet, or if a vehicle malfunctions and must be temporarily removed, Drafter is capable of continuing to provide service for this modified fleet of size $V' \neq V$. We assess the scalability of the Drafter agent in our computational experiments, §3.6.

Centralized decision-making. Given Q-value predictions for all vehicles, Drafter's centralized decision-making proceeds as follows. Let $\bar{\mathcal{V}}_{\text{Drafter}}$ be the set of vehicles for whom the service action has the largest Q-value. To serve the request, we choose the vehicle in $\bar{\mathcal{V}}_{\text{Drafter}}$ with the largest such Q-value; this vehicle then receives NOOP for its RNR action. The remaining vehicles perform the RNR action with the largest Q-value, with NOOP prioritized in the event of a tie (ties between repositioning actions are broken arbitrarily). Some consideration must be given to repositioning assignments a_v, as the DQN produces outputs in $\mathcal{L}$ (TZs), while actions a_v take values in $\mathcal{C}$ (stations). Given a repositioning instruction to some TZ $\lambda^\star \in \mathcal{L}$ for vehicle v, we set a_v to the station in $\lambda^\star$ that is nearest to the vehicle.

Given an action $\mathbf{a} = (a_r, (a_v)_{v\in\mathcal{V}})$, define a^v_{Drafter} to be the action in Drafter's action space $\mathcal{A}_{\text{Drafter}}$ assigned to vehicle v:

$$a^v_{\text{Drafter}} = \begin{cases} \text{serve} & a_r = v \\ L(a_v) & \text{otherwise,} \end{cases} \tag{3.7}$$

where $L(a_v)$ represents the TZ that contains station a_v.

Experience Replay. As described in §3.4.1.1, Drafter saves state-transition memories $(s, \mathbf{a}, r, s^{(n)})$ at each step which are later used to train its DQN via prioritized experience replay. Let $x^{v(n)}_{\text{Drafter}}$ be the DQN input for vehicle v from state $s^{(n)}$. Because Drafter makes V predictions at each step, our sample of M_{batch} memories leads to an effective sample size of $V * M_{\text{batch}}$, where each memory $(s, \mathbf{a}, r, s^{(n)}) = (x^v_{\text{Drafter}}, a^v_{\text{Drafter}}, r, x^{v(n)}_{\text{Drafter}})_{v\in\mathcal{V}}$. Note that the reward r is shared equally among all vehicles during training, with each vehicle receiving the full amount. This was chosen to encourage cooperation among the fleet. Thus, for Drafter, the difference that forms the basis of its loss function (c.f. equation (3.5)) is

$$r + \gamma \max_{a'\in\mathcal{A}_{\text{Drafter}}} Q(x^{v(n)}_{\text{Drafter}}, a') - Q(x^v_{\text{Drafter}}, a^v_{\text{Drafter}}). \tag{3.8}$$

3.4.2.4 Random.

Finally, we consider the *Random* policy which chooses a vehicle to serve the request (a_r) uniformly from $\mathcal{V} \cup \{\emptyset\}$ and chooses repositioning locations a_v uniformly from $\mathcal{C} \cup \{\emptyset\}$ for each $v \in \mathcal{V}$. We offer Random simply to serve as a lower bound.

3.5 Dual Bounds

Assessing policy quality is hampered by the lack of a strong bound on the value of an optimal policy, a dual bound. Without an absolute performance benchmark it is difficult to know if a policy's performance is "good enough" for practice or if additional research is required. Here we offer two dual bounds. First is the value of an optimal policy that can serve all requests (§3.5.1), and the second is the value of an optimal policy with perfect information, i.e., the performance achieved via a clairvoyant decision maker (§3.5.2).

3.5.1 Serve-All Bound.

We first consider the *Serve-all* (SA) dual bound in which we assume that new requests can always be assigned to some vehicle $v \in \mathcal{V}$. That is, we assume that the size of the fleet is sufficient to be able to meet demand. We know that, given a fleet of sufficient size, an optimal policy will serve all requests, because policies seek to maximize the sum of rewards, and rewards are non-negative and earned only by serving requests ($a_r = \emptyset \Rightarrow C(s, \mathbf{a}) = 0$)). Therefore, to compute the value of this policy, we can simply sum the rewards of all observed requests. The gap between the SA dual bound and the best policy is a reflection of the adequacy of fleet size relative to demand and can serve as justification for a ridehailing company to invest (or not) in additional vehicles.

3.5.2 Perfect Information Bound.

In practice, it is likely that not all requests can be feasibly served, making the Serve-all bound loose. In an attempt to establish a tighter dual bound, we consider the value of an optimal policy under a *perfect information* (PI) relaxation. Under the PI relaxation, the agent is clairvoyant, aware of all uncertainty a priori. In the E-RPC, to have access to PI is to know in advance all details regarding requests: their origins, destinations, and when they will arise. Denote such a set of known requests by $\mathcal{R} \in \mathrm{P}$, where P is the set of all request sets.

In the absence of uncertainty, we can rewrite the objective function as

$$\max_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0\right] = \mathbb{E}\left[\max_{\pi \in \Pi} \sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0\right]. \tag{3.9}$$

Notice that the *perfect information problem* (right-hand side of equation (3.9)) can be solved with the aid of simulation. We may rely on the law of large numbers — drawing random realizations of uncertainty (request sets $\mathcal{R} \in \mathrm{P}$), solving the inner maximization for each, and computing a sample average — to achieve an unbiased and consistent estimate of the true objective value. This value is the *perfect information bound*. It remains to solve the inner maximization.

In the absence of uncertainty that results from having access to PI, the inner maximization can be solved deterministically: with all information known upfront, no information is revealed to an agent during the execution of a policy. As a result, there is no advantage in making decisions dynamically (step by step) rather than statically (making all decisions

at time 0). This permits the use of an exact solution via math programming, which we pursue using a *Benders-based branch-and-cut* algorithm in which at each integer node of the branch-and-bound tree of the *master problem*, the solution is sent to the *subproblem* for the generation of Benders cuts. Here, the master problem is responsible for assigning vehicles to requests in a time-feasible manner, and the subproblem is responsible for ensuring the energy feasibility of these assignments. The use of the Benders-based decomposition enables the solution of large PI problems which may not be otherwise feasible. We discuss the master problem in more detail in §3.5.2.1, the subproblem and the generation of cuts in §3.5.2.2, and comment on the bound's tractability in §3.5.2.3. We then provide additional details on the solution method for the subproblem in §3.5.2.4 and §3.5.2.5.

3.5.2.1 Master problem.

The master problem, responsible for time-feasible assignments of customer requests to vehicles, is the mixed integer-linear program (MIP) defined by equations (3.10)-(3.16). In it, requests are represented as nodes in a directed graph $\mathcal{G}$. Request nodes i and j are connected by a directed arc (i, j) when a vehicle can feasibly serve request j after request i (ignoring energy requirements). The graph $\mathcal{G}$ also contains a dummy node for each vehicle. These dummy nodes are connected to request nodes for which the assignment of vehicles to requests is time-feasible. The problem involves choosing arcs in $\mathcal{G}$, starting from vehicles' dummy nodes, that form (non-overlapping) paths for the vehicles which indicate the sequence of requests that vehicles will serve. If a request i is a member of some vehicle's path, it contributes c_i to the objective function, its value as given by the reward function (equation (3.2)).

In the master problem, $\mathcal{R}$ is the set of all requests (known a priori given PI), $\mathcal{V}$ is the set of vehicles, c_i is the reward associated with serving request i, h_i is a binary variable taking value 1 if request i is assigned to any vehicle, y_{vi} is a binary variable taking value 1 if job i is the first request assigned to vehicle v, x_{ij} is a binary variable taking value 1 if a vehicle is assigned serve request j immediately after request i, z_i is a continuous non-negative variable equal to the time at which a vehicle arrives to pick up request i, t_i^r is the time at which request i begins requesting service, w is the maximum amount of time a customer may wait between when they submit their request and when a vehicle arrives, t_i^p is the travel time between the origin and destination of request i, and t_{ij}^s is the travel time between the destination of request i and the origin of request j. For vehicles, t_v^p is equal to the earliest time at which they can depart from their initial locations, and t_{vi}^s is equal to the travel time between their initial locations and the origin of request i. We formally

define the master problem as

$$\text{maximize} \quad \sum_{i \in \mathcal{R}} c_i h_i \tag{3.10}$$

$$\text{subject to} \quad \sum_{v \in \mathcal{V}} y_{vi} + \sum_{j \in \mathcal{R} \setminus \{i\}} x_{ji} \geq h_i, \quad \forall i \in \mathcal{R} \tag{3.11}$$

$$\sum_{j \in \mathcal{R} \setminus \{i\}} x_{ij} \leq h_i, \quad \forall i \in \mathcal{R} \tag{3.12}$$

$$\sum_{j \in \mathcal{R}} y_{vj} \leq 1, \quad \forall v \in \mathcal{V} \tag{3.13}$$

$$z_i \geq t_i^r + (t_v^p + t_{vi}^s - t_i^r) y_{vi}, \quad \forall i \in \mathcal{R}, \forall v \in \mathcal{V} \tag{3.14}$$

$$z_i - z_j \geq t_i^r - t_j^r - w + (t_j^p + t_{ji}^s - t_i^r + t_j^r + w) x_{ji}, \quad \forall i, j \in \mathcal{R}\,(i \neq j) \tag{3.15}$$

$$h_i, x_{ij}, y_{vi} \in \{0, 1\};\ t_i^r \leq z_i \leq t_i^r + w \tag{3.16}$$

The objective (3.10) maximizes the sum of rewards earned by assigning requests to vehicles. Equation (3.11) manages the binary assignment variable h_i, requiring that it be included in some vehicle's path to take value 1. Similarly, equation (3.12) manages the variables for the outgoing arcs from request i, forcing them to take value 0 if the request has not been assigned to a vehicle. Equation (3.13) ensures that each vehicle has at most one request assigned to be its first. Equation (3.14) sets a lower bound for the time at which vehicles' can arrive to their initial requests, and equation (3.15) sets a lower bound for the time at which vehicles' can arrive to subsequent requests. Finally, equation (3.16) defines variables scopes' and bounds the earliest and latest possible start times for requests z_i.

As mentioned, we only connect request nodes i and j via directed arc (i, j) if it is time-feasible to serve request j after request i; that is, if $t_i^r + t_i^p + t_{ij}^s \leq t_j^r + w$. While energy-feasibility is ultimately ensured by the subproblem, we can facilitate the master problem by eliminating additional arcs in $\mathcal{G}$ using known energy consumptions to perform stronger feasibility checks. Let ρ be the maximum rate at which vehicles acquire energy when recharging, q_i^p be the energy required to travel from the origin to the destination of request i, q_{ij}^s be the energy required to travel from the destination of request i to the origin of request j, q_i^d be the energy required to travel from the destination of request i to the nearest charging station, and q_i^o be the energy required to travel to the origin of request i from its nearest charging station. Then for arc (i, j) to exist, it must be that $t_j^r + w - (t_i^r + t_i^p + t_{ij}^s) \geq \frac{1}{\rho} \max\{0, (q_j^p + q_j^d - (Q - q_i^o - q_i^p))\}$, which states that the maximum down time between i and j (left-hand side) must be sufficient to accommodate any recharging that must occur between these requests (right-hand side). We provide similar feasibility checks for the arcs (v, i) connecting vehicle dummy nodes to requests. Specifically, the simpler time-feasible check requires that vehicle v can arrive to request i in time: $t_v^p + t_{vi}^s \leq t_i^r + w$. In consideration of energy requirements, we ensure $t_i^r + w - (t_v^p + t_{vi}^s) \geq \frac{1}{\rho} \max\{0, q_{vi}^s + q_i^p + q_i^d - q_v^p\}$, which states that the time for the vehicle to perform any required charging (right-hand side) must be no greater than the available time before it must arrive to request i. Finally, we force

the underlying graph to be acyclic, meaning we prohibit pairs of requests i, j such that $(t_i^r + t_i^p + t_{ij}^s < t_j^r + w) \wedge (t_j^r + t_j^p + t_{ji}^s < t_i^r + w)$.

3.5.2.2 Subproblem.

A solution to the master problem is a sequence of request assignments $r_v = (r_v^1, r_v^2, \ldots)$ for each vehicle $v \in \mathcal{V}$. r_v^1 is taken to be the element in the singleton $\{j | y_{vj} = 1\} \cup \emptyset$; subsequent entries r_v^i are similarly elements in singletons $\{j | x_{r_v^{i-1} j} = 1\} \cup \emptyset$ (r_v terminates with the first null element). Given sequences for each vehicle, the subproblem must ensure they are energy feasible. To do so, we use a modified version of the labeling algorithm developed by Froger et al. (2019) to solve the Fixed Route Vehicle Charging Problem (FRVCP) (Montoya et al. 2017). The FRVCP entails providing charging instructions (where to charge and to what amount) for an electric vehicle traversing a fixed sequence of customers. The algorithm takes as input a sequence like r_v and, if successful, terminates with the minimum duration path through the sequence that includes instructions specifying at which charging stations to recharge and to what amount. When unsuccessful, the labeling algorithm returns the first unreachable node in the sequence. In the original implementation customers did not have time constraints as they do here, where vehicles are required to arrive to requests in the window $[t_i^r, t_i^r + w]$. We further discuss the algorithm and our modifications to it to accommodate this difference in §3.5.2.4 and §3.5.2.5.

Unsuccessful termination of the algorithm for a sequence r_v indicates that the sequence cannot be traversed energy-feasibly. To remove it from the search space, we add the following *feasibility cut* to the master problem:

$$y_{v,r_v^1} + \sum_{i=2}^{j^\star} x_{r_v^{i-1}, r_v^i} < |r_v|, \tag{3.17}$$

where $j^\star \leq |r_v|$ is the index of the first unreachable request node in r_v to which the algorithm was unable to feasibly extend a label. This cut (3.17) enforces that not all variables defining the sequence be selected.

3.5.2.3 Tractability of the PI Bound.

The PI bound is often computationally intractable to obtain. This is because the estimation of the expected value with perfect information entails repeated solutions to the inner maximization of equation (3.9), a challenging problem despite the absence of uncertainty. Even if the Benders-based method of §3.5.2.1 and 3.5.2.2 does not return an optimal solution for a given realization of uncertainty, we may still get a valid bound by using the best (upper) bound produced by the solver (Gurobi v8.1.1). The solver's bound, typically attained via linear relaxations to the master problem, serves as a bound on the value of an optimal policy with PI, and therefore also as a bound on an optimal policy. This mixed bound, effectively combining both an information and a linear relaxation, is weaker than a bound based on the information relaxation alone. However, we show in computational experiments that it is still useful. In the experiments, when the bound results only from the information relaxation (i.e., we were able to solve all realizations of uncertainty to optimality), we denote the bound by an asterisk (*); otherwise, the bound is mixed.

3.5.2.4 Subproblem: algorithm overview.

We provide here additional details on the labeling algorithm from Froger et al. (2019) which is used to solve the subproblem described in §3.5.2.2. We will refer to the sequence of requests to be served by vehicle v, as determined by the master problem (§3.5.2), by $r_v = (r_v^1, r_v^2, \ldots)$.

To find the optimal recharging instructions given a request sequence r_v, the FRVCP is reformulated as a resource-constrained shortest path problem. The algorithm then works by setting labels at nodes on a graph $\mathcal{G}'_v$ (see example in Figure 3.2) that reflects the vehicle's assigned request sequence r_v and possible charging station visits. Labels are defined by *state-of-charge* (SoC) *functions*. SoC functions are piecewise-linear functions comprised of *supporting points* $z = (z^t, z^q)$ that describe a state of departure from a node in $\mathcal{G}'_v$ in terms of time z^t and battery level z^q. See Figure 3.3 for an example.

During the algorithm's execution, labels are extended along nodes in $\mathcal{G}'_v$. When extending labels, SoC functions are shifted by the travel time and energy between nodes, and supporting points resulting in infeasible (negative) charges are pruned. When extending a label to a charging station node, we create new supporting points that correspond to the *breakpoints* in the charging curve at that station – energy levels at which the charging rate changes. (Note that here, we assume linear charging until the vehicle reaches full battery, at which point it begins idling. This breakpoint corresponds to z_2 in the left graph of Figure 3.3.) We continue to extend labels along nodes in $\mathcal{G}'_v$ until either the destination node $r_v^{|r_v|}$ is reached or there are no feasible label extensions. If the destination node is reached, the algorithm returns the earliest arrival time to that node based on the label's SoC function and the sequence is deemed to be energy feasible; if the destination node cannot be reached, the sequence is deemed energy-infeasible and the algorithm returns the first request node to which it could not extend a label. Bounds on energy and time are established in pre-processing and are used alongside dominance rules during the algorithm's execution in order to improve its efficiency. For complete details on the algorithm, we refer the reader to Froger et al. (2019).

3.5.2.5 Subproblem: algorithm modifications.

We provide here additional details on our modifications to the labeling algorithm from Froger et al. (2019) to accommodate time constraints on the customers – constraints not present in the original implementation. We will again refer to the sequence of requests to be served by vehicle v by $r_v = (r_v^1, r_v^2, \ldots)$.

We assume that the vehicle is eligible to depart its initial location, denoted 0_v, at time t_v^p with charge q_v^p. Moving between locations a and b requires energy $q_{a,b}^s$ and time $t_{a,b}^s$ (if b is a request, the values reflect the time and energy to reach its destination via its origin). For this discussion, additional information about the algorithm beyond the overview in §3.5.2.4 may be necessary, for which we refer the reader to the description of Algorithm 3 in §5.3 and Appendix E of Froger et al. (2019).

The first modification serves to include the option of idling at charging stations. We add a point in the charging function at (∞, Q) as shown in Figure 3.3. This allows the

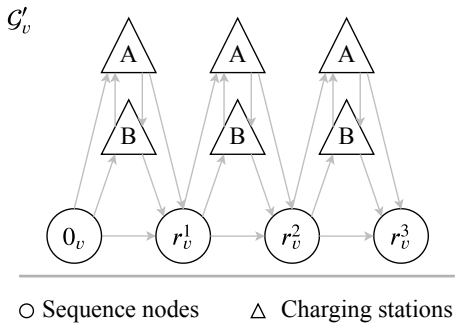

Figure 3.2: A depiction of the graph G'_v for a vehicle assigned to serve three customer requests. In the example, we assume there are two stations, A and B, at which the vehicle can recharge.

vehicle to idle at a CS after it has fully charged its battery. Second, when extending a label to a request, we prune supporting points at the request node based on the request's time constraints. Specifically, all supporting points that are too early are shifted later in time. This may result in multiple supporting points with the same time but different energy levels, in which case we keep only the supporting point with maximum energy. Supporting points whose times are too late are eliminated, and a new supporting point is established where the SoC function intersects the end of the customer's time window. These processes are shown in Figure 3.3. Finally, when extending a label from one request node directly to another, if a supporting point has been shifted later in time by some amount Δt (like z_1'' in Figure 3.3), then the point incurs a charge penalty Δq. This reflects the constraint that vehicles may not idle at request nodes, so we assume that the vehicle has continued driving (e.g., "circling the block") during the time Δt and has thus drained its battery by the amount Δq. Note that such penalties are not incurred when extending to or from charging station nodes in $\mathcal{G}'_v$ as these nodes have no time constraints.

3.6 Computational Experiments

To test the solution methods proposed in §3.4, we perform computational experiments modeled after business-day ridehailing operations on the island of Manhattan in New York City. We describe the experimental setup in §3.6.1, present our results in §3.6.3, and offer a brief discussion of the results in §3.6.4.

3.6.1 Experimental Setup

We generate problem instances (equivalently, *episodes* over which the agents act) using data publicly available for the island of Manhattan in New York City. New York City Taxi & Limousine Commission (2018) provide a dataset consisting of all ridehail, Yellow Taxi, and Green Taxi trips taken in 2018. We filter the data to only those trips that originate and terminate in Manhattan and those that occur on business days ($d \in \{0, 1, 2, 3, 4\} =$

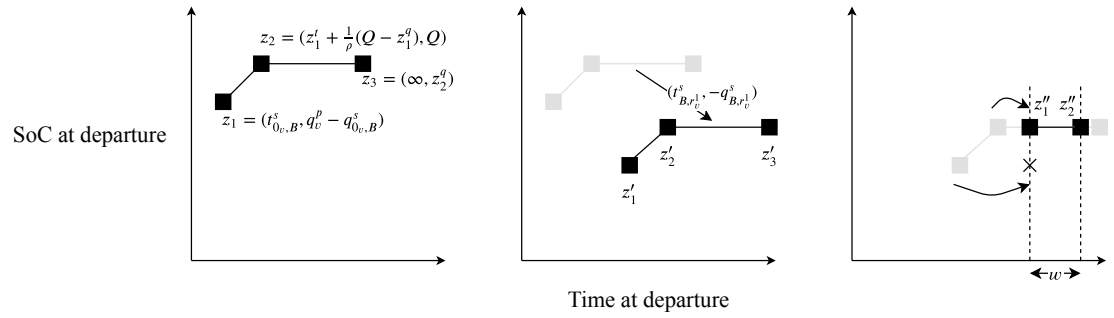

Figure 3.3: SoC functions for labels extended from 0_v to B to r_v^1. (Left) The SoC function at B after arriving directly from 0_v with supporting points corresponding to immediate departure without recharging (z_1), recharging completely (z_2), and subsequent indefinite idling (z_3). (Center) These points are then shifted by $(t^s_{B,r_v^1}, -q^s_{B,r_v^1})$ as we extend a label to r_v^1, resulting in points z'_1, z'_2, z'_3. (Right) Supporting points must lie within customer r_v^1's time window w. Point z'_1 is eliminated since it is dominated after this shift (indicated by the "x"), having less charge than the new point z''_1. Point z'_3 is eliminated and the new point z''_2 is created where the SoC function intersects the end of the time window. Note that there is no charge penalty incurred here, as the label is being extended from a charging station.

{Mon, Tues, Weds, Thurs, Fri}, excluding public holidays). The average daily profile for these data is shown in Figure 3.4. Based on this profile, we set the episode length T to 24 hours, with episodes beginning and ending at 3:00 am, as this time corresponds to a natural lull in demand from the previous day and gives agents time to perform proactive recharging before the morning demand begins. Data for each trip includes the request's pickup time as well as the location of its origin and destination. Requests' origins and destinations are given by their corresponding *taxi zones*, an official division of New York City provided by the city government (NYC OpenData 2019) that roughly divides the city into neighborhoods. We use this as the basis for our geographical discretization $\mathcal{L}$ as described in §3.4.2, resulting in $|\mathcal{L}| = 61$ TZs for Manhattan (see Figure 3.5). Coordinates $((o_x, o_y), (d_x, d_y))$ for requests' exact origins and destinations are drawn randomly from inside their corresponding TZs.

We fix the set of stations $\mathcal{C}$ to all CSs currently available or under construction in Manhattan, as listed by National Renewable Energy Laboratory (2019), yielding a set of $|\mathcal{C}| = 302$ CSs (blue marks in left map of Figure 3.5). We assume that all CSs dispense energy at a constant rate of 72kW, equal to that of a Tesla urban supercharger (Tesla 2017). Vehicles' batteries have similar traits to that of a mid-range Tesla Model 3, with a capacity of 62 kWh and an energy consumption of 0.15 kWh/km. Further, we assume vehicles travel at a speed of 16.1 km/hr (10 mi/hr), and, when serving a request, receive a fixed reward of $C_b = \$7.75$ and distance-dependent reward of $C_d = \$1.9$/km, under the assumption of Euclidean distances. We set the maximum time that customers are willing

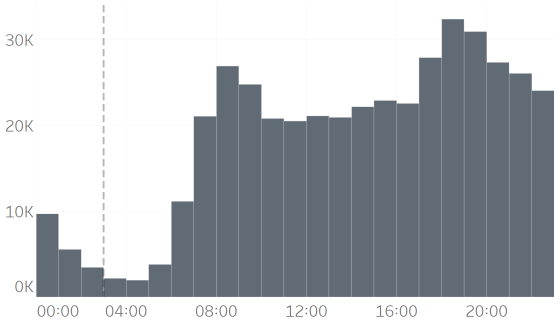

Figure 3.4: Mean number of requests by hour in Manhattan on a business day in 2018. The dashed vertical line indicates episodes' 03:00 start time.

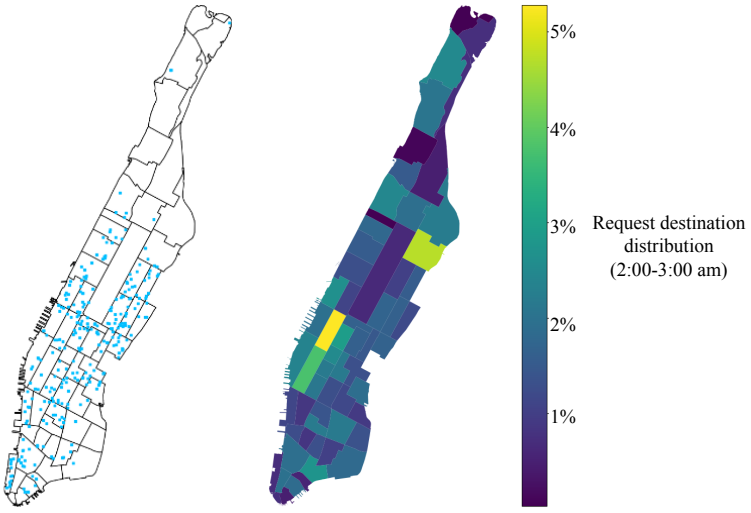

Figure 3.5: (Left) The island of Manhattan divided into taxi zones with charging station locations shown in blue. (Right) Distribution of requests' destinations by taxi zone during episodes' last hour (02:00-03:00).

to wait for a vehicle to be $w = 5$ minutes.

Vehicles' initial locations are determined by drawing a TZ randomly according to the distribution of drop-off locations during the final hour of the previous day's operation (2:00-3:00am). See right map in Figure 3.5. We then choose a charging station uniformly from within the TZ (TZs without CSs are excluded). Vehicles' initial charges are drawn uniformly from $[0, Q]$. To randomly generate a set of R requests for an instance, we first sample a business day from 2018 then draw R requests from that day.

We consider problem instances of three size classes as determined by the number of daily requests R and vehicles V; we will refer to these size classes by their ratio R/V. The first instance size 1400/43 was chosen such that the mean number of requests per hour during the busiest time of day (between 6:00-7:00pm, see Figure 3.4) is approximately 100. The number of vehicles in this set was then chosen to align with the (hourly) R/V ratio used in experiments in Bertsimas et al. (2019). Experiments on 1400/43 instances include results for all agents as well as the SA and PI bounds. The agents and SA bound are evaluated over a set of 200 episodes, while the PI bound — given its higher computational demand — is established using an average over 50 episodes. Prior to evaluation, the Dart and Drafter agents are first trained on a separate set of 750 episodes. We note that the PI bound can be solved to optimality for these instances, so we denote it by PI*.

Given a daily request set of size $R = 1400$, a fleet size of $V = 14$ is arguably more realistic than $V = 43$. This number is derived from a scaling of $R^\star/V^\star$, where $R^\star = 449,121$ is the total number of trip requests (taxi and ridehail) served on an average business day in Manhattan in 2018, and $V^\star = 4,400$ is an approximation of the number of simultaneously active Yellow Taxis in Manhattan at any given time in 2018 (see data aggregations in, e.g., Schneider (2019)). Consequently, the second size class of instances we consider is 1400/14. With the same demand but a reduced supply, these instances represent a more challenging problem environment for the agents than 1400/43. Experiments for this set include results for all agents and the SA and PI bounds; however, we can no longer solve the PI bound to optimality, so we use the mixed version of this bound as described in §3.5.2.3. Again, the SA bound and all agents are evaluated on a set of 200 episodes, with Dart and Drafter first trained on a set of 750 episodes, and the PI bound is an average over 50 episodes.

Finally, we consider an instance set ten times larger (14000/140) to assess Drafter's ability to scale and generalize. That is, we evaluate the Drafter agent directly on 200 instances of size 14000/140 without any additional training — it simply uses its DQN as trained on the 1400/14 instances. We compare the Drafter agent to the Random and Nearest agents, as well as the SA bound (we forgo the PI bound completely due to the size of the problem, $\sim 10^8$ variables and constraints). Dart is absent from this analysis as its scalability is inherently handicapped, with its DQN's output (that is $\mathcal{A}_{\text{Dart}}$) dependent on the number of vehicles in the instance. This is in contrast to Drafter, for which the size of its DQN output ($|\mathcal{A}_{\text{Drafter}}|$) is indifferent to the size of the fleet. Details on the hyperparameters used in the training of the Dart and Drafter agents for all instance size classes are provided in §3.6.2.

Table 3.2: Agents' hyperparameters.

Parameter	Dart	Drafter
Optimizer (learning rate)	Adam (0.001)	Adam (0.001)
Loss function	Huber	Huber
Discount factor γ	0.999	0.999
Memory capacity	1,000,000	1,000,000
Steps prior to learning M_{start}	2,000	2,000
Training frequency M_{freq}	100	100
Batch size M_{batch}	32	32
Initial epsilon ϵ_i	1	1
Final epsilon ϵ_f	0.01	0.1
Epsilon decay steps ϵ_N	75,000	75,000
PER type	None (uniform)	proportional
PER α	-	0.6
PER β_0	-	0.4
PER beta decay steps	-	600,000
Target network update frequency M_{update}	5,000	5,000
n-step learning	3	3
DQN activation functions	ReLU	ReLU*
Hidden layers (pre-dueling, advantage, value)	(1,2,2)	(2,2,2)**

* With the exception of the layers in the attention mechanism as described in §3.6.2.1.

** For the inner DQN (see Figure 3.1).

3.6.2 Agent Details

We provide here additional details about the training and implementation of the Dart and Drafter agents. Hyperparameters, chosen based on empirical results, are given in Table 3.2. We describe the attention mechanism used in Drafter's DQN in §3.6.2.1.

3.6.2.1 Attention Mechanism in Drafter

The attention mechanism used by the Drafter agent is depicted in Figure 3.1. We begin by creating an *embedding* g_v of each vehicle's representation x_v using a single dense layer called the "vehicle embedder." An embedding of some input is simply an alternative representation of that input. For example, for a vehicle representation $x_v \in [0,1]^2 \times \{0,1\}^{|\mathcal{L}|}$ (see §3.4.2.2) the embedder maps it to $g_v \in \mathbb{R}^{128}$ via $f(W_a x_v + \mathbf{b})$, where W_a is a $128 \times (2 + |\mathcal{L}|)$ matrix of trainable weights, $\mathbf{b}$ is a 128-length vector of trainable weights, and f is the activation function (here a rectified linear unit, $ReLU$). Similar to vehicle embeddings g_v, we create an embedding $h \in \mathbb{R}^{64}$ of the request x_r using another single dense layer (the "request embedder"). We then concatenate each vehicle's embedding g_v with the request embedding h to produce joint embeddings $j_v = g_v \oplus h$. The j_vs and g_vs are then used to perform

attention over the vehicles, creating a *fleet context* vector $C^F \in \mathbb{R}^{128}$ via

$$C^F = \sum_{v \in \mathcal{V}} \alpha_v g_v, \tag{3.18}$$

where $\alpha_v = \sigma(\mathbf{w} \cdot \tanh(W \cdot j_v))$ is a scalar weighting vehicle v, σ is the sigmoid activation function, $\mathbf{w}$ is a trainable vector of weights, and W is a trainable matrix of weights (the summation in equation (3.18) is performed element-wise). This attention mechanism is similar to the one used to produce the fleet context in Holler et al. (2019), with the notable difference that here we also incorporate the request. As the request s_r is likely to influence which vehicles are relevant in a given state, its inclusion in the attention mechanism should yield a more descriptive fleet context C^F. The fleet context vector is then used in the production of Q-values for each vehicle as described in §3.4.2.3.

3.6.3 Results

We divide our analysis of the results by instance size, beginning with 1400/43 in §3.6.3.2, 1400/14 in §3.6.3.3, and 14000/140 in §3.6.3.4. Prior to describing agents' performance on the instance sets, we first offer a comparison of the dual bounds in §3.6.3.1.

3.6.3.1 Comparison of Dual Bounds.

The first proposed dual bound, SA, is simple to compute but may be loose when not all requests can be feasibly served. In contrast, the PI bound promises to be tighter, but it is significantly more challenging to compute. Here we aim to quantify the value of the additional computation required for the PI bound.

Intuitively, with a larger fleet size V, more requests can be served. At some V, a policy with PI should be able to serve all requests, so the PI and SA bounds will be equal. Conversely, with decreasing fleet size, even with PI, it should become impossible to serve all requests, so the bounds will diverge. We test this intuition in an experiment comparing the value of these two bounds as a function of fleet size. The results are summarized in Figure 3.6. We begin with instances of size 1400/14, for which (as shown in §3.6.3.3) the gap between the PI and SA bounds is large. We then increment the fleet size by 2 ($V' = V + 2$), and resolve 20 episodes of the 1400/V' instances. We find that the results confirm intuition – the gap between the bounds decreases exponentially with increasing fleet size, reaching equivalence (100% of requests served by an optimal policy with PI) at a fleet size of 32. We note that the baseline PI values here reflect the mixed PI bound discussed in §3.5.2.3, implying that the gaps may actually be larger than these results suggest.

3.6.3.2 Instance size class 1400/43.

The Dart and Drafter *training curves*, showing their increasing objective performance over the 750 training episodes, are provided in Figure 3.7. We see that learning happens quickly for both agents, stabilizing after approximately 250 training episodes at which point they outperform the Nearest policy, whose performance over the training episodes is also

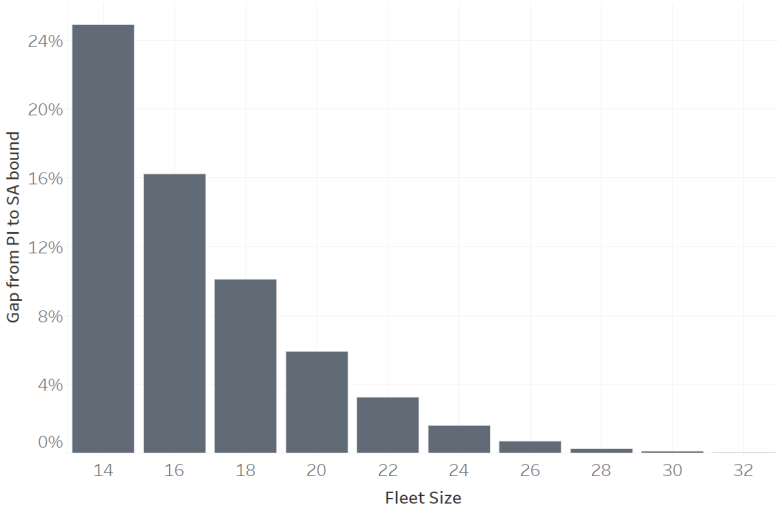

Figure 3.6: The mean percent difference of the SA bound relative to the PI bound over 20 episodes of an instance with 1400 requests for varying fleet sizes. Note: computed as $\frac{SA-PI}{PI}$, where SA is the revenue earned if all requests are served and PI is (the upper bound on) the revenue earned by an optimal policy with perfect information.

shown for reference in Figure 3.7. In Figure 3.8 we compare agents' performance on the 50 evaluation episodes for which the PI bound is available. For each of the 50 evaluation episodes, we are able to solve the PI bound to optimality. We find that the Drafter agent is the best performing with a mean daily revenue of \$14,642, although there remains an 18.8% gap between this policy and both the PI and SA dual bounds, whose values are identical. As anticipated by the results in §3.6.3.1, an optimal policy with perfect information is able to serve all requests. After Drafter, Dart is the second-best performing agent, earning a mean daily revenue of \$14,435 and beating Nearest (at \$14,036) by 2.8%.

We provide further comparison between the agents in Figure 3.9 which shows three valuable metrics for a fleet operator: as measures of customer service, we consider average customer waiting time and number of requests served; and as a measure of sustainability, we consider *fleet occupancy rates* (the percent of time, averaged across all vehicles, that they are either preprocessing or serving a request). We see that by always assigning the nearest eligible vehicle to serve a request, Nearest achieves the shortest average waiting time of 145 s. This is in contrast to the Dart and Drafter agents which have average customer waiting times of 198 and 200 s respectively. The similarity of Random's average wait time (202 s) to Drafter and Dart is due to the fact that it is in effect drawing a waiting time randomly from a distribution that is bounded below by Nearest (which always chooses the minimum) and above by the maximum waiting time $w = 300$ s. By being able to assign vehicles further away, we see that Drafter and Dart are both able to serve more requests on average, resulting in higher fleet occupancy rates and greater objective values.

Figure 3.7: Training curves for the Dart (orange) and Drafter (blue) agents over 750 training instances of size 1400/43. Nearest (red) is provided as a reference. Darker curves are smoothed representations of the lighter curves.

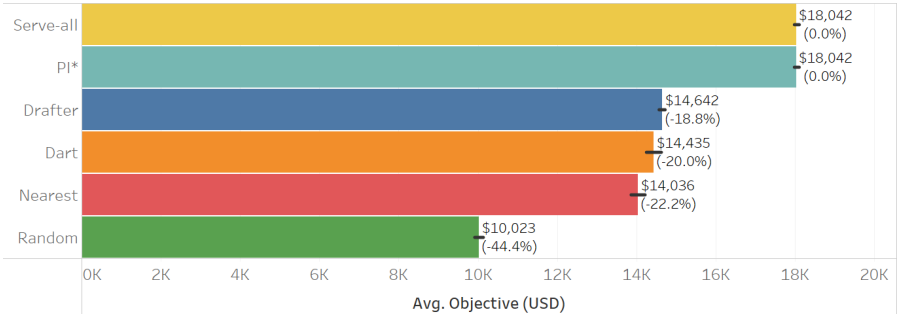

Figure 3.8: Agents' objective performance relative to the PI bound over the 50 evaluation instances of size 1400/43 for which the PI bound is available. Parenthesized values indicate the gap to the PI bound, which is equivalent to the SA bound. Black marks indicate 95% confidence intervals.

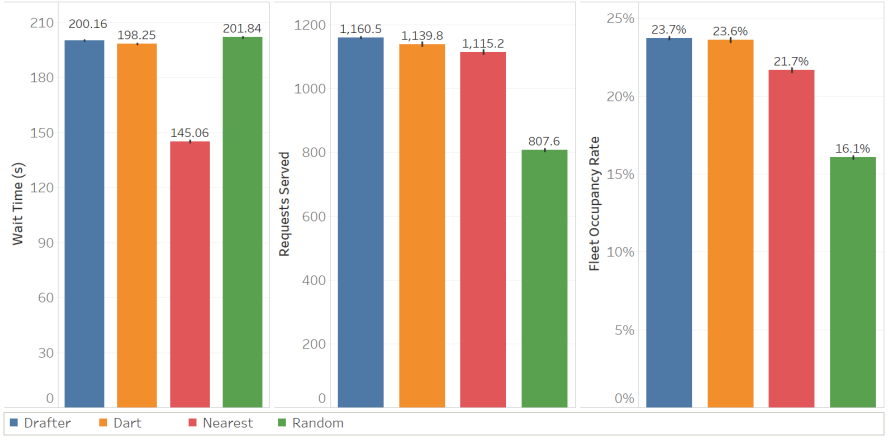

Figure 3.9: Over 200 evaluation instances of size 1400/43, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

Figure 3.10: Training curves for the Dart (orange) and Drafter (blue) agents over 750 instances of size 1400/14. Nearest (red) is provided as a reference. Darker curves are smoothed representations of the lighter curves.

3.6.3.3 Instance size class 1400/14.

Training curves for Dart and Drafter on the 1400/14 instances are provided in Figure 3.10. We see again rapid learning for both agents, plateauing after approximately 200 training episodes. Comparing agents' objective achievement over 200 evaluation episodes (Figure 3.11), we find that Drafter now achieves a significantly higher objective than both the Dart and Nearest agents, outperforming the former by \$1,597 (16.8%) and the latter by \$1,557 (16.4%). We find the agents' mean performance relative to the PI bound to be looser (34.3%) than for the 1400/43 instances, suggesting that access to information about the future becomes increasingly more valuable as supply shrinks relative to demand. We also find, as anticipated by the results in §3.6.3.1, that the gap between the PI and SA bounds has increased from 0% to 25%. Figure 3.12 offers further comparison of the agents. We again find that Nearest achieves the shortest waiting time of approximately 3 minutes, compared to approximately 3.3 minutes for Dart and Drafter. In contrast to before, this does not equate to a greater number of requests served by Dart, nor to an appreciable increase in its fleet occupancy rate. Drafter, however, serves on average 137 more requests per day than Dart and 134 more than Nearest. This results in the highest fleet occupancy rate of 46.7%, relative to 39.1% for Dart and 38.5% for Nearest.

3.6.3.4 Instance size class 14000/140.

We apply Drafter directly to the 14000/140 instances after undergoing no additional training after the 1400/14 instances, and we find that it is again the best performing agent, achieving an objective 4.5% (\$5,649) better than Nearest and coming within 26.6% of the SA dual bound (see Figure 3.13). Figure 3.14 offers a more detailed analysis, showing that on average Drafter serves 5.7% (570) more requests per day than Nearest and has a 5% higher fleet occupancy rate. It does so while offering a customer waiting time of 212 s, only 38 s longer than that of Nearest (174 s).

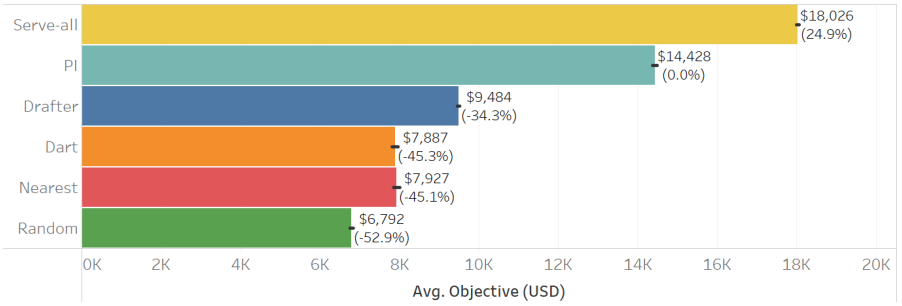

Figure 3.11: Agents' objective performance over 200 evaluation instances of size 1400/14 with the gap to the SA bound in parentheses. Black marks indicate 95% confidence intervals.

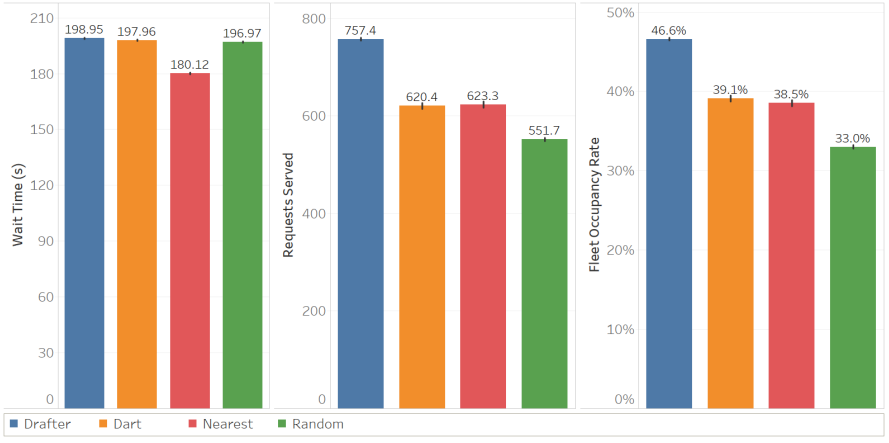

Figure 3.12: Over 200 evaluation instances of size 1400/14, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

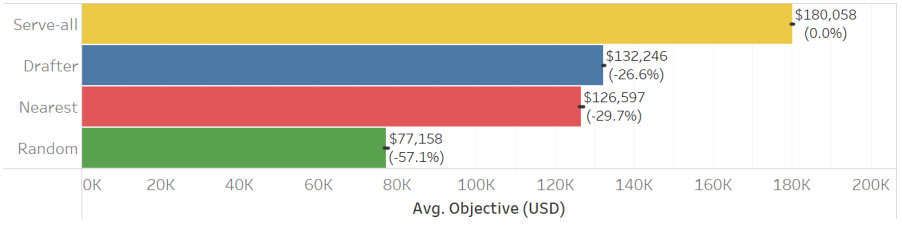

Figure 3.13: Agents' objective performance over 200 evaluation instances of size 14000/140 with the gap to the SA bound in parentheses. Black marks indicate 95% confidence intervals.

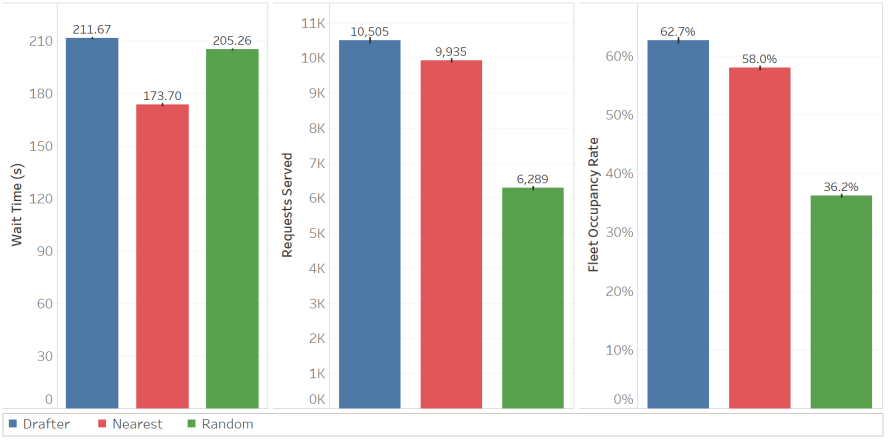

Figure 3.14: Over 200 evaluation instances of size 14000/140, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

3.6.4 Discussion

While often in dynamic optimization no dual bound is available against which to compare policy performance, we offer two here. Although the gaps from these bounds to our agents are sometimes large, the results serve to illustrate that even a loose bound has utility. For example, agents' widening gaps to the SA bound from the 1400/43 to the 1400/14 instances demonstrate the potential for the bound to gauge the size of the fleet relative to demand. Having such a measure allows the decision-maker to justify (or not) investments in additional vehicles. Comparative analysis of the bounds also suggests that access to perfect information is of significant value in the E-RPC: for instances with a V/R ratio of 1400/32 and greater, the bounds are equivalent, indicating that an optimal policy with PI is consistently able to serve all requests.

Most notable from the computational experiments is the performance of the Drafter agent. Drafter achieves the best objective value in all size classes, outperforming the second-best agent by 1.4% in the 1400/43 instances, 16.4% in the 1400/14 instances, and 4.5% in the 14000/140 instances. The results demonstrate Drafter's ability to conduct fleet repositioning actions that anticipate future demand and allow it to serve more customers. The increase in its performance relative to other agents between the 1400/43 and 1400/14 instances suggests that this anticipation is especially valuable in supply-limited regimes. Additionally, Drafter requires no prior knowledge or assumptions about the shape of the demand — in the language of reinforcement learning, it is *model-free*. Rather than relying on, e.g., historical data or domain knowledge regarding parameters describing the demand's spatial and temporal distributions, Drafter garners sufficient working knowledge of the demand during its training to determine which actions best allow it to serve future requests and ultimately achieve greater objective values. Being model-free also implies that Drafter is flexible to various relationships between actions and rewards, because this relationship is learned by the agent. That is, while we trained and evaluated Drafter on the reward

function defined by equation (3.2), we could have equally used, e.g., a stochastic reward function or one which incorporated additional revenue or cost metrics. Although displaying inferior performance than Drafter, Dart is also model-free and therefore enjoys these same benefits. Finally, we also highlight Drafter's scalability. After undergoing training on instances of size 1400/14, the agent was directly applied to the 14000/140 instances, ten times larger than those on which it was trained. Despite not receiving any training on instances of this size, Drafter still managed to outperform the other agents in this size class. This is encouraging for operators of ridehail companies, as it suggests that Drafter may be robust to changes in problem instances. That is, in the event that demand is slow on a particular day or one or more vehicles are unavailable (e.g., due to repairs), then Drafter should still provide reliable service.

3.7 Concluding Remarks

We considered the E-RPC, the problem faced by an operator of a ridehail service with a fleet composed of centrally-controlled electric vehicles. To solve the E-RPC, we developed two policies, Dart and Drafter, derived with deep reinforcement learning. To serve as a comparison, we consider a heuristic policy common in dynamic taxi dispatching, as well as a randomly-acting policy that serves as a lower bound. We develop two dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information. To establish this bound, we decompose a relaxed version of the E-RPC into time- and energy-feasibility subproblems, and solve them via a Benders-like decomposition. We offer an analysis of when this more complex dual bound offers value beyond that of a simpler proposed dual bound. The two dual bounds provide utility in assessing fleet size relative to demand and the value of perfect information.

We perform computational experiments on instances derived from real world data for ridehail operations in New York City in 2018. Across instances of varying sizes, we consistently find the deep RL-based Drafter agent to be the best performing. On the most realistic instances for which it was specifically trained, it achieved an objective 16.4% better than any other policy. We further show that Drafter is scalable, finding that it outperforms other policies by 4.5% on instances ten times larger than any on which it was trained. These results ultimately suggest there are opportunities for deep reinforcement learning in solving problems like the E-RPC. By more efficiently utilizing finite resources, these methods promote greater sustainability in the domain of transportation and logistics.

Chapter 4

Atari-fying the Vehicle Routing Problem with Stochastic Service Requests

4.1 Introduction

Deep reinforcement learning (RL) has seen much recent success in tasks that involve sequential decision making under uncertainty. These tasks span a variety of domains, including natural language processing (He et al. 2015), image recognition (Caicedo and Lazebnik 2015), healthcare (Liu et al. 2017), energy (Mocanu et al. 2018), taxi dispatching (Kullman et al. 2020a), and autonomous driving (Sallab et al. 2017). Perhaps the most well-known application domain is that of playing games. Deep reinforcement learning has been used to train agents capable of superhuman performance in games such as chess (Silver et al. 2018), Go (Silver et al. 2018), Doom (Lample and Chaplot 2017), Texas Hold'em Poker (Heinrich and Silver 2016), StarCraft II (Vinyals et al. 2019), and, of particular interest here, Atari (Mnih et al. 2015).

In the case of Atari, Mnih et al. established a single solution method — a single agent architecture — that was capable of outperforming humans on the majority of a set of 49 Atari games. The use of a single architecture to accomplish this feat is notable as these games differ in their appearance, goals, rewards, actions, etc. Given the diversity of the games on which this architecture was tested and shown to excel, one wonders whether it would perform comparably well on any game with a similar (Atari-like) format. Perhaps not, in which case it would suggest that there is something unique about the original set of Atari games that makes them susceptible to exploitation by this particular solution method. However, this seems unlikely. Rather, it seems more likely that given a game formatted similarly to those in the original set of Atari games from Mnih et al. (2015), a solution method like the agent architecture from that study would be able to conquer this new game as well.

Should this premise hold, it presents the opportunity of using this agent architecture to solve research problems, permitted that those problems can be properly formatted as an

Atari-like game.[1] We explore this opportunity here. For shorthand, we will refer to the process of modeling a research problem as an Atari-like game as *Atari-fying*. Our primary contribution in this work is demonstrating the Atari-fication of a well-known problem from the field of transportation, namely, the vehicle routing problem with stochastic service requests (VRPSSR). We discuss the merit of these efforts, their generalizability, the expected benefits and limitations of this approach, and we speculate on additional research problems for which Atari-fication may prove successful. Broadly, we hope to show that there is an opportunity to extend the groundbreaking achievements from the deep reinforcement learning community to research problems in other domains through reformulations of classical problem models.

4.2 Background & Related Work

We provide a brief background on deep reinforcement learning (§4.2.1) and vehicle routing problems (§4.2.2), then look at previous examples of the modeling of research problems as videogames (§4.2.3).

4.2.1 Deep Reinforcement Learning

Reinforcement learning (RL), as defined by Sutton and Barto refers to the process through which an agent, sequentially interacting with some environment, learns what to do (that is, a *policy* that determines how to map states to actions) so as to maximize a numerical reward signal from the environment (Sutton and Barto 2018). Often what the agent seeks to learn is the value of choosing a particular action a from some state s, known as the state-action pair's *Q-value* ($Q(s, a)$), equal to the immediate reward plus the expected sum of future rewards earned as a result of taking action a from state s. With knowledge of the Q-values for all possible state-action pairs, the agent's policy is then to choose the action with the largest Q-value. In practical problems, because the number of unique state-action pairs an agent could encounter in its environment is too large to learn and store a value for each, a functional approximation of these Q-values is learned. There are competing methods to learn this *value function approximation* (VFA). When deep artificial neural networks (ANNs) are used for this VFA, the method is called *deep reinforcement learning* or *deep Q-learning*, and the ANN used in this process is referred to as the *deep Q network* (DQN). While deep RL has a history dating back at least to Farley and Clark (1954), it has recently seen an uptake in usage, thanks to advances in computational performance, data availability, and subsequent methods development.

Deep RL offers flexibility in how the agent perceives the state, capable of effective VFA under a variety of representations (that is, under a variety of input formats to the DQN). Often the state representation is simply a list of relevant quantities describing, e.g., position and velocity. Such is usually the case in classical RL control problems and in applications such as robotics (see, e.g., those from Brockman et al. (2016)). An alternative representation is that of an image or set of images: the agent receives a visual description of

[1] We do not provide a precise definition of what makes a game *Atari-like*, but broadly mean that it is a two-dimensional third-player game with a total pixel count on the order of 10^5 or less.

the environment from which it makes decisions. This approach has a biological precedent, as it is how humans solve many problems — we process received visual information to understand and respond to a situation. This visual state representation is what was used in Mnih et al.'s work on Atari and, as such, is the representation we adopt here.

4.2.2 Vehicle Routing Problems

Vehicle Routing Problems (VRPs) are a broad class of NP hard problems that seek to determine the optimal routes for vehicles to follow to perform some task, such as the delivery of goods to a set of customers. A vast body of literature exists for VRPs, and VRPs remain an active research topic in operations research (see Toth and Vigo (2014) for a summary). VRPs come in many variants, accommodating various combinations of constraints and uncertainties, such as the VRP with a capacitated vehicle (CVRP), the VRP with customers that have time-windows (VRPTW), the VRP with an electric vehicle (E-VRP), the E-VRP with public charging stations (E-VRP-PP), the VRP with stochastic service requests (VRPSSR), with stochastic customer demands (VRPSD), with stochastic travel times (VRPSTT), with stochastic demands and time windows (VRPSDTW), etc.

Stochastic variants of the VRP are of particular interest here, because they more naturally lend themselves to Atari-fication: first, because the way that information is revealed over time in these problems is similar to the appearance of new obstacles or enemies in videogames; and second, they permit solution methods that allow for responding to this information (i.e., dynamic routing), similar to how players can react to new obstacles in videogames (e.g., dodge a bullet and return fire). Here we focus on the VRP with stochastic service requests (VRPSSR). The problem has been well studied in the literature, for example, in Gendreau et al. (1999a), Bent and Van Hentenryck (2004), and Ulmer et al. (2018). Inspiration for the exact problem addressed here, described in more detail in §4.3, comes specifically from Ulmer et al. (2018). While the problem has been addressed with multiple solution methods, the majority are based on *reoptimization*, a method in which a math program is constructed and solved every time a decision is needed. In addition to the Atari-fication of this problem, our approach here is also novel in that it marks the first application of deep RL to this problem.

4.2.3 Research Problems as Videogames

Successful attempts at modeling research problems as videogames exist, although these games have been developed for different purposes than that proposed here. Typically, these games have aimed to crowdsource human efforts to contribute to a research problem whose scale or complexity renders it difficult to solve via traditional algorithmic approaches. Human input in these games is then either used directly as a smaller component to a larger solution (as in Eyewire (Sterling 2012)), or it is used to guide an underlying algorithm, often by highlighting regions of the solution space in which to concentrate efforts (as in Phylo (Kawrykow et al. 2012), FoldIt (Khatib et al. 2011), and Quantum Moves (Lieberoth et al. 2015)). This is in contrast to our proposed approach in which the formulation as a videogame serves to translate the research problem so as to be amenable to solution by a different class of algorithms (i.e., deep RL instead of reoptimization or traditional VFA).

4.3 Atari-fying the VRPSSR

Here we demonstrate the Atari-fication of the vehicle routing problem with stochastic service requests (VRPSSR), as defined in Ulmer et al. (2018).

4.3.1 Problem description

The agent in this problem is responsible for dynamically routing a vehicle to serve customer service requests that arise randomly throughout some service region over the course of a work day of duration T. The locations of potential customer requests are known in advance, but it is not known which customers will request service or when they will do so. The problem begins with the vehicle at the depot at time 0, and it must return to the depot before the end of the day at time T. The agent moves the vehicle among the customers $\mathcal{N} = \{1, \ldots, N\}$ and the depot (defined to be node 0) via edges in some underlying graph (which varies by formulation, as discussed below). We assume known fixed travel times along the edges. At each step, beginning at time $t = 0$, the agent either instructs the vehicle to wait at its current location, or it selects an adjacent node in the graph to which to move the vehicle. The next step occurs either when the vehicle arrives to a new node (if the agent chose to move the vehicle) or after some predefined waiting time $\bar{t}$ (if the agent chose to wait). If the vehicle visits a node representing a customer that is currently requesting service, then the customer is marked as having been served, and the vehicle earns some reward. The agent's objective is to find a policy that dynamically routes the vehicle so as to maximize the expected sum of rewards.

4.3.2 Formulations' graphs

The abstraction used for the problem's underlying graph differs between the traditional and Atari-fied formulations. In the traditional formulation, the agent moves the vehicle among the customers and depot via edges in the complete graph G with vertices $V = \mathcal{N} \cup \{0\}$. In the Atari-fied formulation, the agent moves the vehicle along edges in the graph G', a Manhattan-style grid representation of G, where some nodes (intersections) represent customer locations. These representations are shown in Figure 4.1. While the problem definition is the same for both formulations, this difference in graph yields other consequential differences. For example, if Euclidean distances and travel times are used in the traditional formulation G (as is often the case), these would be Manhattan under the Atari-fied formulation G'. Further, while the size of the *action space* (the set of actions from which the agent can choose) is on the order of $N + 2$ under G (movement to each customer location, plus the depot and the wait option), it is simply 5 under G' (up, down, left, right, and wait). This difference in action spaces translates to additional dynamism and flexibility when using the Atari-fied formulation. This is because the movement of the vehicle from one customer to another in the traditional formulation can effectively be preempted under the Atari-fied formulation, since, with each step of the movement between the customers, the agent can alter the vehicle's path towards a different destination. See Figure 4.1 for an example.

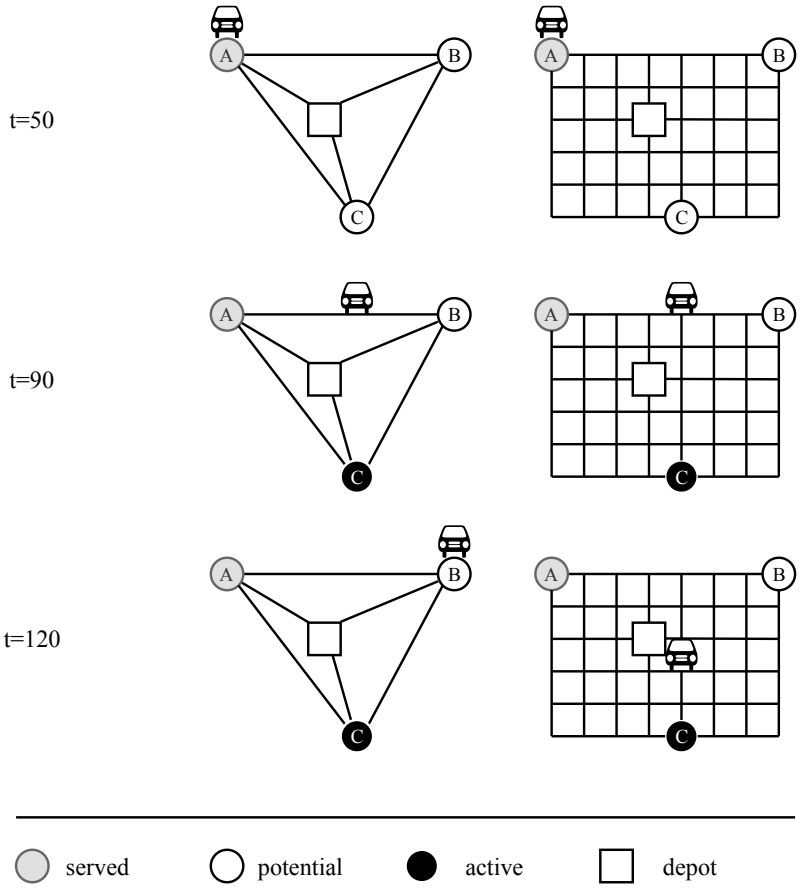

Figure 4.1: Graphs for the traditional (left) and Atari-fied (right) formulations of the VRPSSR. While the vehicle in the traditional formulation must continue moving directly along the arc connecting customers A and B, under the Atari-fied formulation this decision can effectively be pre-empted, as the vehicle can choose to head towards newly-requesting customer C.

4.3.3 State representations

The primary difference between the traditional and Atari-fied formulations is the observation of the state that the agent receives in order to perform decision-making. In the traditional formulation, the state is a tuple consisting of the vehicle's current location, the time t, the customers currently requesting service, and the customers that have not yet requested service. However, with Atari-fication, the agent receives a visual representation of the state, similar to what one might imagine a human operator would see on a control panel. The information about the state that is displayed in this visual state representation is the same as before: the vehicle's location (now represented on the Manhattan-grid graph G'), the customers currently requesting service, and the customers that may still request service. Time may also be provided in the visual representation, perhaps (as shown at left in Figure 4.2) using a bar, or it may also be provided to the agent simply as a scalar that accompanies the visual representation.

How this information is chosen to be displayed is up to the modeler: details may vary, such as the shapes and colors used to represent the objects in the state, and the size (in pixels) of the display. In addition, in the original Atari work (Mnih et al. 2015), the authors included in the state the four most recent game screens (*frames*), rather than just the most recent. This was to allow the agent to "see" the motion of certain objects in the game, such as the movement of the ball in Pong — with just one frame, the agent does not know with what speed the ball is moving or whether its movement is towards or away from them; however, this is immediately apparent with the inclusion of additional frames. Thus, the modeler may also decide the number of previous frames to include in the state so as to sufficiently capture movement in the game. A comprehensive study of the influence of these choices on the agent's ability to learn Atari-fied problems would be valuable, but is outside the scope of this work.

The visual rendering we use in our Atari-fication of the VRPSSR is shown in Figure 4.2. The basis of the rendering (the *playable area*) is a simplified depiction of G'; each pixel represents a node, bordered by its adjacent nodes from G'. Around the playable area is a thin border, and above the top border is a bar that displays the relative remaining time before the vehicle must return to the depot. The colors in the rendering are in grayscale. The depot and the customers are represented by individual pixels in the playable area. The customers currently requesting service are nearly white, while the depot and the potential customers are shades of gray. Customers that have already been served are not included in the render. The vehicle is represented by the location of the open central pixel in a white 3x3 pixel square. The drawing order of these objects, from bottom to top, is depot, potential customers, vehicle, active customers.

Based on results from early experiments, in practice we do not use the rendering directly as the state representation. Instead, we use what are known as *feature layers*, which show "elements of the game... isolated from each other, whilst preserving the core visual and spatial elements of the game" Vinyals et al. (2017). Here, we use three feature layers: one for the vehicle, one for the active customers, and one for the potential customers. Each feature layer is a pixel array with value one for the pixel(s) containing the relevant features for that layer, and zero otherwise. The stack of these three layers, together with a scalar representing the percent of the remaining time, comprise the state representation seen by

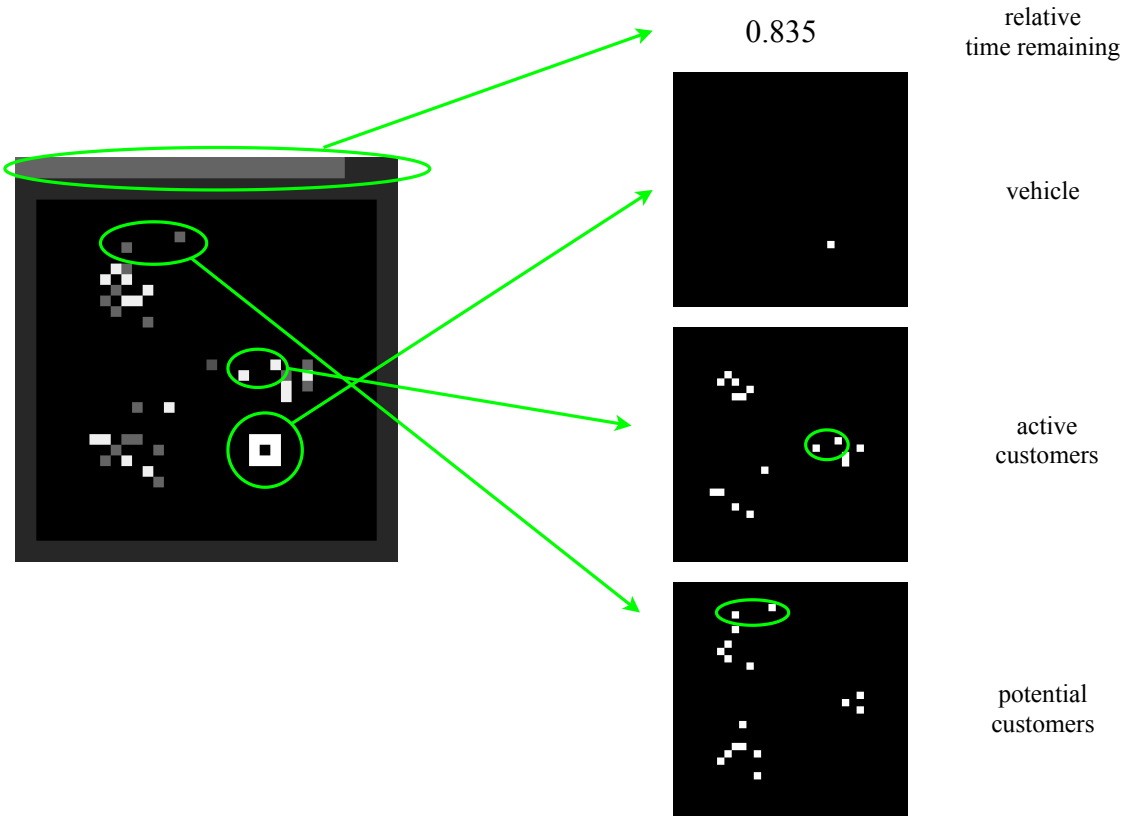

Figure 4.2: The rendering of the VRPSSR game (left) and its corresponding feature layer representation (right).

our agent. A depiction of the feature layer state representation is shown in Figure 4.2.

4.3.4 Computational Experiments

We test our approach on instances randomly generated in a manner motivated by Ulmer et al. (2018). We use a service region of size 256 km^2 (16km x 16km), divided into a grid with resolution 0.25 km^2 (grid size 0.5km x 0.5km), yielding a playable area of 32x32 pixels. We assume the vehicle travels at a constant speed of 30km/hr, yielding a time of one minute to traverse edges in G'. We use this as the default waiting time as well ($\bar{t} = 1$), and we use a workday duration of $T = 230$ minutes. When the remaining time is less than or equal to the time it would take the agent to return to the depot, we terminate the episode. This serves to ensure the resulting policy is admissible, and it also simplifies the learning process, as the agent then need only learn to serve customers and anticipate demand. The depot is centrally located, and customers are distributed among three clusters. If we take (0,0) to be coordinates of the lower left grid cell (in pixel count), then the first cluster is centered around (8,8), the second cluster around (8,24), and the third cluster around (24,16). When the vehicle visits a customer that is requesting service, it earns a reward of 10 units.

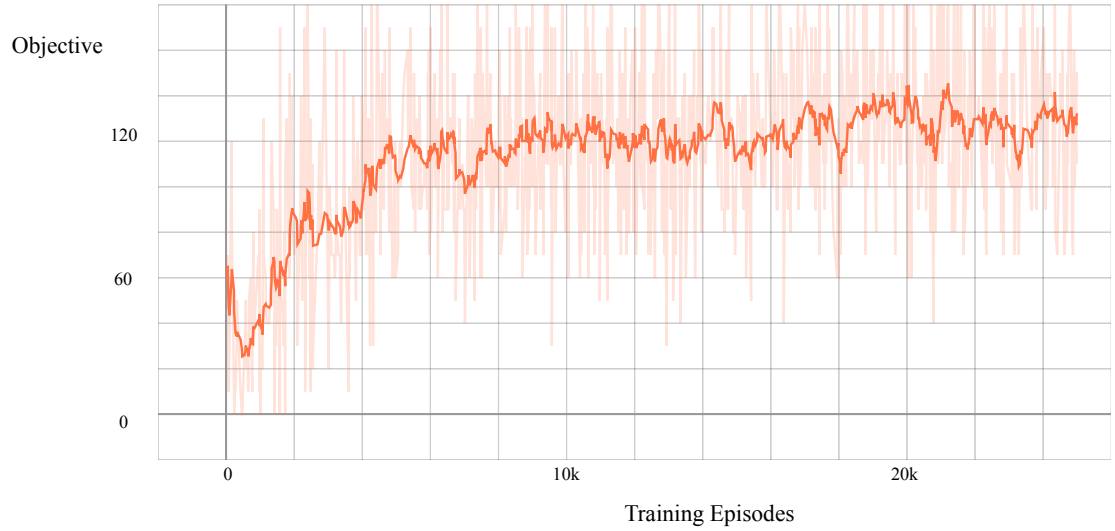

Figure 4.3: Agent's performance over 25000 training episodes.

For the customer placements and request times, we begin by producing a set that are requesting service at the beginning of the episode (at time $t = 0$). The number of such customers is sampled from a Poisson distribution with mean 15. Then, for each time step in $\{1, 2, \ldots, T-1\}$, we sample a number of customers that request service during that period according to a Poisson distribution with mean $15/(T-1)$. To place a customer, we first sample a cluster in which to locate it with probabilities $(0.25, 0.5, 0.25)$ for the first, second, and third cluster, respectively. We then sample a grid cell from around the chosen cluster's mean from a normal distribution with a standard deviation of $\sqrt{2}$. We accept the customer placement if that grid cell does not already hold a customer, otherwise we repeat the draw from the normal distribution.

The rendering for these instances has a playable area of 32x32 pixels surrounded by a 2px-thick border and a 2px-tall time bar across the top. This yields a total rendering size of 36x38. However, we use the feature-layer state representation as described above (see Figure 4.2), yielding a state that is a tuple consisting of a stack of three 32x32 pixel arrays for the vehicle, the active customers, and the potential customers, along with a scalar for the percent of time still remaining. This state is used as input to the agent's DQN, whose details are described in §4.3.4.1. We leverage three common deep RL enhancements: dueling (Wang et al. 2016) and double (Hasselt et al. 2016) DQN architecture (D3QN) with prioritized experience replay (Schaul et al. 2016).

The results of our computational experiments are summarized in Figure 4.3. With the described setup, we find that the agent is able to successfully learn and improve its performance in the VRPSSR environment, eventually achieving a mean score of 125.02 when averaged over the last 5000 training episodes. This score translates to serving 12.5 customers out of an average of 30. We suspect that these results do not yet compete with more traditional methods, although proper evaluation to reach this conclusion remains to be done. Ultimately, however, these preliminary results show promise in the process of Atari-fication, at least for the VRPSSR.

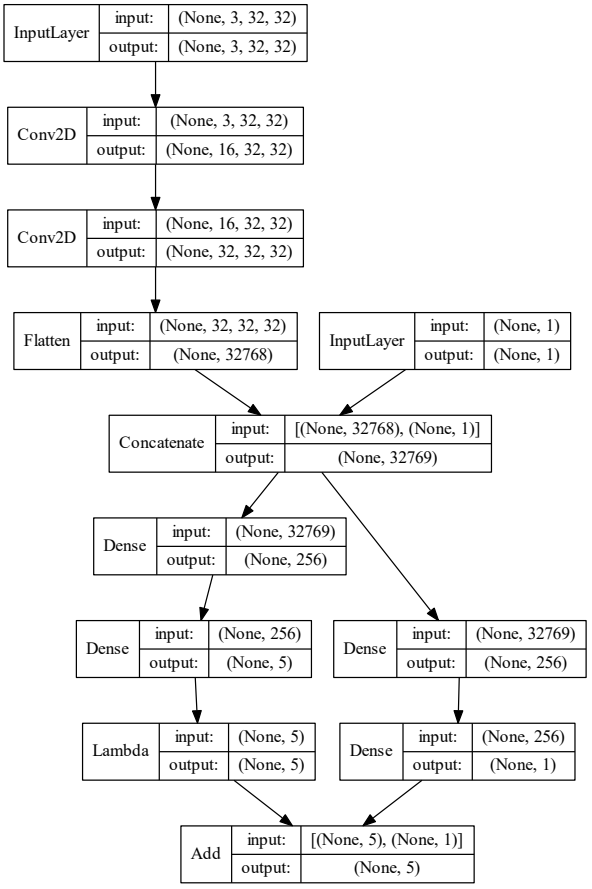

Figure 4.4: The agent's dueling DQN.

4.3.4.1 Agent Details

Policy. The agent follows an ϵ-greedy policy with an initial ϵ value of 1.0 (chooses actions totally at random) and a final ϵ value of 0.1, which is decayed linearly over 1 million time steps. Future Q-values are discounted using a discount factor of $\gamma = 0.99$.

Memory & training. The agent has a memory capacity of 1 million steps. It begins training after it has observed 10k steps, at which point it undergoes training every 16 steps using (proportional) prioritized experience replay (PER) with a batch size of 32. We use PER hyperparameters $\alpha = 0.6$ and $\beta_0 = 0.4$ with β_0 annealed to 1.0 over 600k steps.

DQN. We use a dueling double DQN (D3QN), where the primary and target networks are constructed as shown in Figure 4.4. We update the target network every 2000 steps. We use the RMSprop optimizer with a learning rate of 0.001.

4.4 Discussion

The Atari-fication of the VRPSSR produced results showing promise of the application of this approach to other research problems. Here we discuss its benefits and limitations, then speculate on other problems for which it may be useful.

4.4.1 Benefits & Limitations

We begin by highlighting the flexibility of the Atari-fication approach: any problem — not just those in vehicle routing or transportation — that can be visualized and formatted like an Atari-like game is a candidate for this solution method. Further, it is likely that Atari-fication efforts for one problem will largely extend to other problems in its class. For example, nearly all VRPs share a similar structure — typically involving a depot, vehicle, and some set customers — so the Atari-fied representation of one VRP can likely be used with only minor modifications for another. Consider the VRPSSR as Atari-fied here. To add time-window constraints for a customer, one could simply use a different color/pixel value for that customer, depending on the time remaining in their time window (e.g., use a pixel value of 0 for a customer if their time window has not yet opened or has already closed, 10 if it is open and more than 15 minutes remain in their window, and 20 otherwise). A repository like the VRP-REP (Mendoza et al. 2014) could be established to track and share Atari-fied formulations of research problems. This shareability reduces the upfront efforts needed to assess whether Atari-fication will be a viable solution method. In addition, much like exact commercial solvers and algorithm libraries are available for traditional VRPs and other OR problems, many libraries (e.g., Keras (Chollet et al. 2015), OpenAI Baselines (Dhariwal et al. 2017)) exist to build and execute a deep RL agent the proposed solution method, given an Atari-fied problem representation. The approach also lends itself naturally to dynamic decision making in the context of problems that involve frequent revealings of uncertainty. Atari-fication thus offers a new approach with which to solve such problems, which are often more difficult to solve using traditional methods (we give examples of such problems in §4.4.2).

The approach is not without limitations, however. First, it seems likely that this approach would not apply in multi-agent (e.g., multi-vehicle) contexts, since the pairing of agents' actions to specific controllable entities on the screen would be difficult for agents to interpret. While it is possible this approach would still work, such an environment would be quite different from that of the Atari-games on which the method has been tested and successfully demonstrated. Next, distance matrices may not always be preserved when converting from the traditional graph representation G to that required for the Atari-fied formulation G'. In such cases, the solutions to the Atari-fied formulation will serve as approximations or (if properly modeled) bounds for the traditional formulation of the problem. For some applications, this may prohibit the use of Atari-fication. Lastly, we note that the proposed approach is perhaps not as radical as its name may imply. As alluded to in §4.3, the approach may be more generally interpreted not as "Atari-fication," but rather as a reformulation of the state so as to be amenable to a specific class of (deep RL) solution methods. This is analogous to how, in operations research, a researcher may choose to remodel the math program for a particular problem so as to accommodate solution via, e.g.,

column generation or Benders decomposition. Under this interpretation, our work here simply highlights one such opportunity for problems modeled as Markov decision processes.

4.4.2 Additional Opportunities

Atari-fication may be applicable to many problems involving sequential decision making under uncertainty. In particular, problems that are naturally visualizable are strong candidates for Atari-fying. Several NP-complete problems that form the basis for many practical research problems have this characteristic of being visualizable, such as the traveling salesman problem (TSP) and the knapsack problem (KP). The visualizability of the TSP may be exploited to solve many stochastic and dynamic VRP variants, as we demonstrated here with the VRPSSR. One can also imagine intuitive Atari-fied formulations of VRPs involving, e.g., pickups and deliveries, time windows, or stochastic travel times. The KP and related problems in scheduling and bin packing may lend themselves to Atari-fications that resemble Tetris, where the player is responsible for arranging newly-arriving pieces (representing, e.g., jobs) in some area on the screen (representing one or more machines). Opportunities for additional constraints and uncertainties to be captured in these Atari-fications include machines' capacities and availabilities, as well as jobs' resource demands, objective values, and durations. Given the breadth of applications of TSP- and KP-like problems alone — arising in transportation, manufacturing, energy, and healthcare — Atari-fication may serve researchers in many fields.

4.5 Conclusion

We present a new general approach to modeling research problems as Atari-like videogames to make them amenable to recent groundbreaking solution methods from the deep reinforcement learning community. The approach is flexible, applicable to a wide range of problems. We demonstrate its application on a well known vehicle routing problem. Our preliminary results on this problem, though not transformative, show signs of success and suggest that Atari-fication may be a useful modeling approach for researchers studying problems involving sequential decision making under uncertainty.

Software Tools

Chapter 5

`frvcpy`: an Open-Source Solver for the Fixed Route Vehicle Charging Problem[1]

5.1 Introduction

Governmental regulations as well as a growing population of environmentally conscious consumers have led to increased pressure for firms to act sustainably. This pressure is particularly high in the logistics domain, which accounts for about one third of emissions in the United States (Office of Transportation and Air Quality 2019). Electric vehicles (EVs) offer a means to more sustainable transportation; however, they present technical challenges to which their conventional (i.e., internal-combustion engine) vehicle (CV) counterparts are immune. For example, because the distance EVs can travel on a single charge is often less than the distance an equivalent CV can travel on a tank of gas, EVs may demand more frequent recharging operations. This difficulty is compounded by the sparseness of EV recharging infrastructure relative to the network of refueling stations available to CVs, potentially forcing EVs to perform longer detours to recharge their batteries. Further, despite recent improvements to battery and charging station (CS) technology, recharging an EV still requires orders of magnitude more time than refueling a CV. The time required to recharge is also nonlinear with respect to the EV's *state of charge* (SoC), the relative amount of energy left in its battery, posing yet another challenge not applicable to CVs (Uhrig et al. 2015). Companies choosing to adopt EVs require optimization tools capable of handling these additional challenges.

[1]The research described in this chapter has been submitted for publication in the recently established "Software Tools" area of the *INFORMS Journal on Computing*. It has also been published as open-access software (Kullman et al. 2020b) available on the Python Package Index (PyPI; `https://pypi.org/project/frvcpy/`). Its source code is available on GitHub (`https://github.com/e-VRO/frvcpy`). For a preprint, see

N. D. Kullman, A. Froger, J. C. Goodson, and J. E. Mendoza. frvcpy: an Open-Source Solver for the Fixed Route Vehicle Charging Problem. Working paper, February 2020. URL `http://hal.archives-ouvertes.fr/hal-02496381`

The development of such optimization tools for conventional vehicles has commanded significant attention from the operations research (OR) community in the study of vehicle routing problems (VRPs). The incorporation of additional constraints that address the challenges posed by EVs has marked a new family of problems within VRPs known as electric vehicle routing problems (E-VRPs). One of the primary tasks in solving E-VRPs is making good charging decisions – namely, where to recharge and how long to do so. This is the crux of the fixed route vehicle charging problem (FRVCP), in which charging operations must be inserted into a fixed sequence of customers being visited by an EV so as to minimize the time for the EV to reach the end of the sequence in an energy-feasible manner. The FRVCP naturally arises as a subproblem in many E-VRPs, since its solution is required in order to determine the true duration or cost of a given route. Having a capable solution method for the FRVCP is thus crucial to the advancement of E-VRP research.

While FRVCP solution methods exist, they tend to suffer from one or more of the following issues: inexactness (e.g., heuristic methods that provide suboptimal solutions), inefficiency (e.g., mixed-integer programs (MIPs) whose solvers require significant run time), or a lack of robustness (e.g., exact algorithms that are limited to simplified versions of the FRVCP). With this work, we offer an implementation of a solution method that suffers from none of these issues. Our implementation provides optimal solutions in low runtime for FRVCPs with rich, realistic problem features. It is based on the labeling algorithm proposed in Froger et al. (2019), which, though capable, is notoriously difficult to implement.

In an attempt to remove the burden of implementation for future E-VRP researchers, we offer our implementation in an open-source Python package, `frvcpy`. `frvcpy` is designed to be easily embedded in more complex solution schemes for E-VRPs (such as in a (meta)heuristic or Benders decomposition): it requires minimal dependencies and inputs, can be accessed either via the command line or a Python API, and includes a translator to generate the required inputs from a common instance format in the VRP community (VRP-REP (Mendoza et al. 2014)). Our aim with `frvcpy` is to make it easier to solve E-VRPs, thereby stimulating additional research in this field which promises to bring about more sustainable practices in logistics.

The remainder of the paper is organized as follows. We first define the FRVCP in §5.2, then discuss some of the previous work on FRVCPs in §5.3. In §5.4 we give an overview of the algorithm implemented in `frvcpy`, then describe the package itself in §5.5. We conclude with brief comments in §5.6.

5.2 Defining the FRVCP

We consider an EV with a fixed route $\Pi = (\pi_1, \ldots, \pi_R)$ that begins at some node π_1 (usually the depot), has a sequence of stops at other nodes $(\pi_i)_{i=2}^{R-1}$ (customers to visit), and terminates at some node π_R (also usually the depot). The vehicle begins at π_1 with its battery at some initial energy level q_0, often taken to be equal to its maximum battery capacity Q. Let the set of nodes in the route be $I = \{\pi_i | \, i \in 1..R\}$. We also consider a set of charging stations C at which the EV may recharge between stops in Π. Each

CS $c \in C$ has some charging type (e.g., fast, slow) associated with a piecewise linear concave *charging function* $\Phi_c(t)$ specifying, for an empty battery, the resulting energy after charging for time t at CS c (see Figure 5.3). Additionally, define the related function $\bar{u}_c(q_1, q_2) = \Phi^{-1}(q_2) - \Phi^{-1}(q_1)$ to be the time required to charge from q_1 to q_2 at CS c. Let the set of breakpoints defining the charging function of CS c be B_c, where a breakpoint $b_i \in B_c$ is a (time, charge) pair: (b_i^t, b_i^q). When the EV travels between nodes $i, j \in I \cup C$, it incurs some known travel time t_{ij} and energy consumption e_{ij} (we assume the triangle inequality holds for both). At stops in Π, the EV may also incur some processing time (e.g., waiting). The objective of the FRVCP is to determine charging decisions – how much to recharge, at which CSs, between which stops in Π – that minimize the total time for the EV to traverse the route in an energy-feasible manner.

5.3 Related Literature

FRVCPs fall under the category of EV routing problems, which are themselves part of a larger body of research on VRPs. We focus our review here solely on FRVCPs. For an overview of E-VRPs we refer the reader to Pelletier et al. (2016), and similarly to Braekers et al. (2016) for an overview of VRPs.

Montoya et al. (2016) encounter an FRVCP in their work on a green vehicle routing problem. The FRVCP they consider assumes that vehicles may only visit one CS between stops, that they always fully restore their energy when recharging (that is, they follow a "full recharging strategy"), and that doing so requires constant time. To solve this FRVCP, they offer an exact algorithm. Roberti and Wen (2016) address an FRVCP in their work on the E-VRP with time windows (E-VRP-TW) and also offer an algorithm that solves this FRVCP exactly. Their solution accommodates a partial recharging policy, assuming that the time required to recharge is linear with the amount of energy. However, unlike in Montoya et al. (2016), they assume that the network of CSs is homogeneous; that is, that all CSs have the same charging technology. The FRVCP again arises in related works by Hiermann et al. (2016), Schiffer and Walther (2017), and Hiermann et al. (2019b). These studies offer exact algorithms for the FRVCP under the assumption that at most one CS may be visited between stops, that the CSs are homogeneous, and that recharging requires linear time. Hiermann et al. (2016) additionally assume a full recharging strategy while Schiffer and Walther (2017) and Hiermann et al. (2019b) allow partial recharging.

Montoya et al. (2017) then consider the first FRVCP that accommodates realistic (nonlinear) recharging times. In the study, they also demonstrate that the assumption of linear recharging times can lead to infeasible or suboptimal solutions. Their FRVCP allows for partial recharging and heterogeneous CSs but assumes that at most one CS may be inserted between stops. To solve their FRVCP, Montoya et al. offer both a heuristic and a MIP formulation. Koç et al. (2019) adopt the heuristic and MIP formulations from Montoya et al. (2017) to solve a similar FRVCP that arises in their work on the E-VRP with shared CSs and nonlinear charging. Baum et al. (2019) then offer a labeling algorithm to solve an FRVCP on real road networks that also accommodates realistic recharging times and allows for multiple CS insertions, although it is restricted to the special case where the route length is two (an origin-destination (OD) pair).

Finally, Froger et al. (2019) propose an exact labeling algorithm to solve the FRVCP from Montoya et al. (2017). Their algorithm is not restricted to OD pairs, and it additionally allows the EV to visit multiple CSs between stops in the route, making theirs the richest of the aforementioned FRVCP variants. Over a testbed of nearly 30,000 instances, they compare their labeling algorithm against a heuristic and a commercial solver for a MIP formulation. They find that the labeling algorithm matches the optimality of the MIP with a runtime comparable to the heuristic. The algorithm is thus state of the art for solving FRVCPs. However, the authors note that its performance is not without cost. They state that E-VRP researchers may ultimately prefer to adopt the heuristic solution, despite its inferior performance, given the complexity of implementing the labeling algorithm. Here, we offer an implementation of the algorithm in `frvcpy` in an attempt to ensure that its complexity does not prevent its adoption.

The algorithm from Froger et al. (2019) has also been adapted to accommodate FRVCPs with additional constraints. For example, in work on a stochastic E-VRP with public CSs, Kullman et al. (2019) adapt the algorithm to accommodate an FRVCP with discrete charging decisions and time-dependent waiting times at CSs. Similarly, Kullman et al. (2020a) adapt an early version of `frvcpy` to accommodate an FRVCP with customer time windows. In both cases, the algorithm's speed and exactness were required as it was called repeatedly to solve a subproblem in a Benders-based branch-and-cut procedure.

5.4 Overview of Labeling Algorithm from Froger et al. (2019)

Given the algorithm's complexity, we provide here a cursory overview and refer the reader to Froger et al. (2019) for additional details (see, in particular, their discussion of Algorithm 3 in §5.3 and Appendix E).

To find the optimal charging decisions for a given route Π, the FRVCP is reformulated as a resource-constrained shortest path problem. The algorithm then works by setting labels at nodes on a modified graph reflecting the vehicle's possible movements along Π (Figure 5.1, Inset 1). Labels are defined by *SoC functions* — piecewise-linear functions comprised of *supporting points* $z = (z^t, z^q)$ that describe a state of departure from a node in terms of time z^t and charge (SoC) z^q.

During the algorithm's execution, labels are extended along nodes in the graph. When a label is extended to a CS node c, we create new supporting points for each breakpoint in B_c to which we could charge (that is, breakpoints with a higher energy than that with which we arrived). Consider Figure 5.2, which depicts this process when extending a label along the edge from customer 33 to CS 48. When it arrives to CS 48, its SoC function has only one supporting point z_1 (assuming the EV has not yet stopped to recharge) depicted by the black square in the right graph of Figure 5.2. Then for each breakpoint in the CS's charging function to which the EV could recharge (b_2, b_3, b_4), we add a supporting point to the label's SoC function (z_2, z_3, z_4) whose time and charge reflect the decision to charge to that breakpoint. Figure 5.2 shows this explicitly for the new supporting point z_4, corresponding to the decision to recharge to the breakpoint b_4 (more specifically, to b_4^q).

We continue to extend labels along nodes in the graph until the destination node π^R is reached, whereat the algorithm returns the time of the first supporting point in the label's

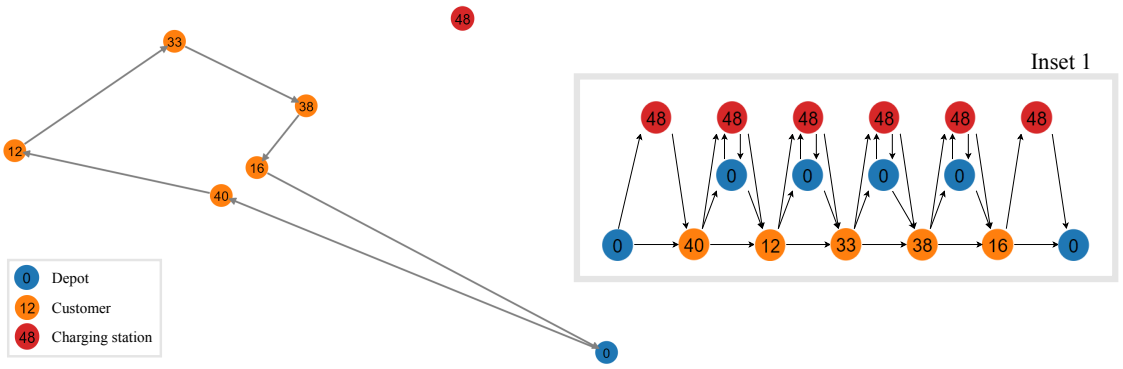

Figure 5.1: Instance excerpt depicting a fixed route in the original problem graph (left) and the modified problem graph for the FRVCP (Inset 1), which includes dummy nodes for CS insertions.

SoC function. Bounds on energy and time are established in pre-processing and are used alongside dominance rules during the algorithm's execution to improve its efficiency.

5.5 The `frvcpy` Package

`frvcpy` is an open-source Python-based implementation of the labeling algorithm from Froger et al. (2019) for solving the FRVCP. In this section we give an overview of its structure (§5.5.1), demonstrate its usage (§5.5.2), and briefly comment on its performance (§5.5.3).

5.5.1 Structure

`frvcpy` is a small package (approximately 1000 lines of code) built in the Python programming language; it is available on the Python Package Index and can be installed via "pip install frvcpy." It is comprised of three primary modules: core.py, solver.py, and algorithm.py. core.py consists of class definitions for ancillary objects required in the algorithm's execution such as nodes, labels, and the FRVCP problem instance. solver.py defines the user-facing `Solver` class which is responsible for pre-processing, calling the algorithm, and writing solutions to file. The algorithm itself and its accompanying functions are contained in algorithm.py. Additionally, the package contains the module translator.py. This module provides the ability to generate instance files compatible with `frvcpy` from instances formatted according to the VRP-REP specification. VRP-REP is a community-driven repository for vehicle routing problem data files; see Mendoza et al. (2014) for more details.

Input/output. Users can interact with `frvcpy` using a Python API or via the command-line interface (CLI). As input, `frvcpy` requires a compatible instance, the fixed route for the EV to travel, and the EV's initial energy. Compatible instances are JSON files (or equivalent Python dictionaries) following the schema available on `frvcpy`'s homepage

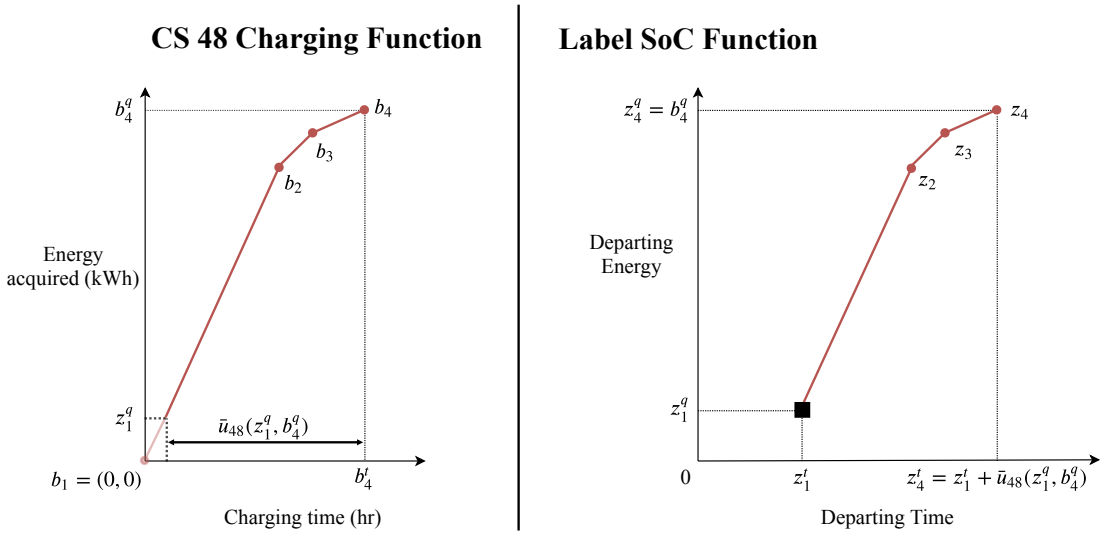

Figure 5.2: Depicting the creation of new supporting points at CS nodes for the case of CS 48 between customers 33 and 38 in Figure 5.1. Right shows the SoC function of the label extended to CS 48, with a black square for the initial supporting point (z_1^t, z_1^q). We create additional supporting points (z_2, z_3, z_4) for each breakpoint to which we could charge, shown by circles b_2, b_3, and b_4 in the CS's charging function (left). Axis labels for new supporting point z_4 (right) detail its creation from the decision to recharge to breakpoint b_4.

(Kullman et al. 2020b). After execution, the algorithm returns the energy-feasible route and its duration. The returned route is a list of tuples indicating stops' node IDs and the amount of energy to be recharged there (the latter given by the keyword 'None' for non-CS nodes; see Listing 5.1, Line 24).

Testing the installation. `frvcpy` provides simple testing to determine if its installation was successful. From the command line, users can execute the command `frvcpy-test` to run a suite of tests that performs an instance translation and solves 134 FRVCPs from the Froger et al. (2019) testbed. The same test suite can also be run in Python via

```
import frvcpy.test
frvcpy.test.runAll()
```

5.5.2 Example Usage

We provide an example demonstrating the use of `frvcpy` through the Python API.[2] Consider a user with the VRP-REP-compliant instance "vrprep-instance.xml," depicted

[2]Readers interested in recreating this example are encouraged to clone the repository from `frvcpy`'s homepage which contains the data discussed here (`https://github.com/e-VRO/frvcpy`). The example follows route "route_tc0c40s8cf0_23" for instance "tc0c40s8cf0" from Froger et al. (2019). The full testbed of instances from Froger et al. (2019) is available at `https://www.math.u-bordeaux.fr/~afroger001/documents/data-Improved_formulations_and_algorithmic_components.zip`

in Figure 5.4. The instance contains "fast," "normal," and "slow" CSs whose charging functions are shown in Figure 5.3. An EV, which begins at the depot with full battery, has been assigned the fixed route $\Pi = (0, 40, 12, 33, 38, 16, 0)$, depicted by the gray arrows in Figure 5.4. Because the EV does not have sufficient energy to traverse Π without recharging, we solve an FRVCP to determine the optimal insertion of charging operations. We can do this using `frvcpy` as follows:

Listing 5.1: Example `frvcpy` usage with Python API

```
1  from frvcpy.translator import translate
2  from frvcpy.solver import Solver
3
4  # translate the VRP-REP instance
5  frvcp_instance = translate("instances/vrprep-instance.xml")
6
7  route = [0,40,12,33,38,16,0]     # route to make energy feasible
8  q_init = frvcp_instance["max_q"] # EV begins with max battery capacity
9
10 # initialize solver with the instance, route, and initial charge
11 frvcp_solver = Solver(frvcp_instance, route, q_init)
12
13 # run the algorithm
14 duration, feas_route = frvcp_solver.solve()
15
16 # write a VRP-REP compliant solution file
17 frvcp_solver.write_solution("my-solution.xml", instance_name="frvcpy-instance")
18
19 print(f"Duration: {duration:.4}")
20 # Duration: 7.339
21
22 print(f"Energy-feasible route:\n{feas_route}")
23 # Energy-feasible route:
24 # [(0, None), (40, None), (12, None), (33, None), (48, 6673.379615520617), (38,
       None), (16, None), (0, None)]
```

The solution to the FRVCP instructs the EV to recharge at CS 48 between customers 33 and 38, as depicted by the black arrows in Figure 5.4 and the printed output on line 24 in Listing 5.1. This results in a total route duration of about 7.34 hours. We note that detouring to CS 48 actually requires more travel time than detouring to CS 41; however, given that CS 48 offers a faster charging speed, it is ultimately preferred over CS 41 (recharging at CS 41 instead of 48 results in an objective of 7.44 hrs).

The above example would be accomplished with the CLI via

Listing 5.2: Example `frvcpy` usage with CLI

```
# translate existing VRP-REP instance, write it to file
frvcpy-translate ./instances/vrprep-instance.xml new-frvcp-instance.json

frvcpy --instance=new-frvcp-instance.json --route=0,40,12,33,38,16,0
    --qinit=16000 --output=my-solution.xml
# Duration: 7.339
```

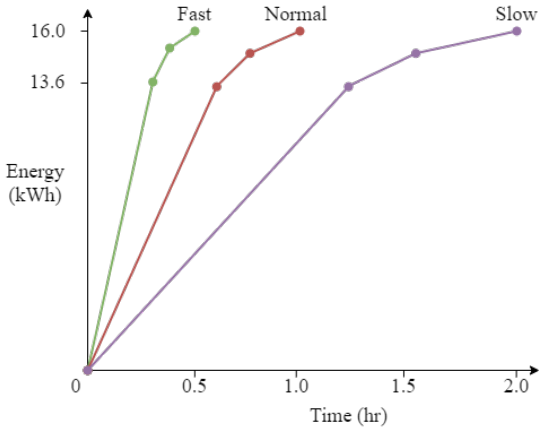

Figure 5.3: Piecewise linear charging functions for example instance "vrprep-instance.xml."

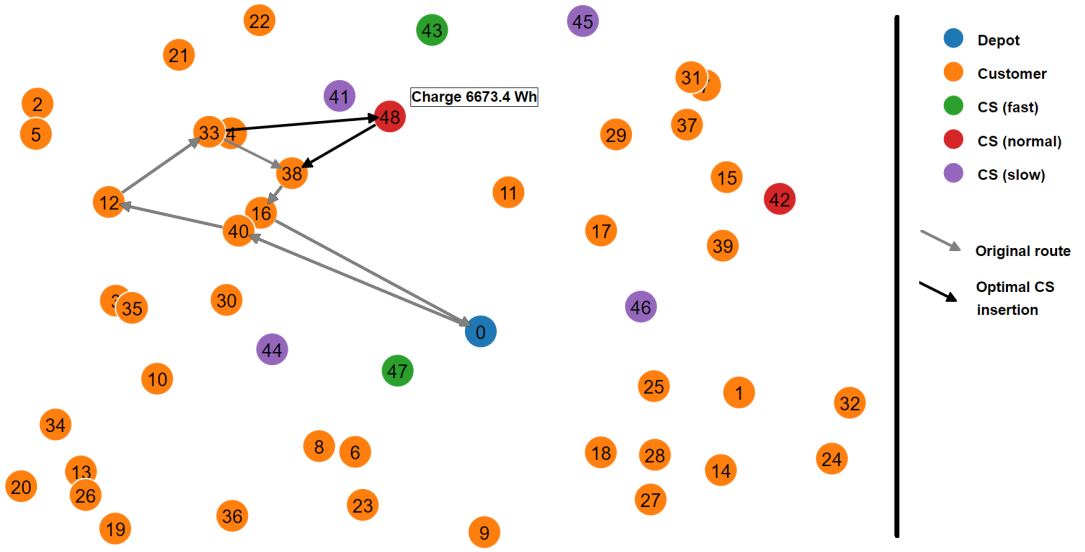

Figure 5.4: Depiction of example instance "vrprep-instance.xml." We consider an EV given the route shown by the gray arrows. The solution to the FRVCP for this route instructs the EV to recharge at CS 48 between customers 33 and 38 (black arrows).

```
# Energy-feasible route:
# [(0, None), (40, None), (12, None), (33, None), (48, 6673.379615520617), (38,
    None), (16, None), (0, None)]
```

5.5.3 Performance

We test the performance of `frvcpy` over the nearly 30,000 instances comprising the testbed from Froger et al. (2019). These instances have a median route length of 10 stops and a median of 18 CSs that may be inserted. On average over the tests, the algorithm has an average run time of 5.6 ms. In addition, in the same tests we find that our translator can translate instances from VRP-REP format in an average of 0.1 s. These results suggest that `frvcpy` requires sufficiently low runtime so as to be included in larger solution schemes for E-VRPs.

5.6 Conclusion

We introduced `frvcpy`, a Python-based open-source implementation of the labeling algorithm from Froger et al. (2019) for the fixed route vehicle charging problem. The algorithm and our implementation are flexible, able to accommodate realistic problem features such as non-linear recharging times, partial charging decisions, and heterogeneous charging station technologies. Because FRVCPs are often encountered as subproblems of more general EV routing problems, we designed `frvcpy` to be easily embedded in larger solution schemes. To that end, the package offers two modes of interaction, has minimal requirements, and is computationally efficient. Our hope is that `frvcpy` facilitates the solution of E-VRPs, lowering the barrier to entry in this field, and ultimately helping bring about a faster transition to more sustainable transportation practices.

Chapter 6

Mapper: An Instance Mapping Utility for the Vehicle Routing Problem Repository

6.1 Introduction to VRP-REP

The completion of VRP research commonly results in published resources that enable future researchers to build upon the completed study. In addition to journal articles, these resources often include data, such as problem instances and solution files. Given the size and rate of progress of the vehicle routing community, tracking the research that is being completed and knowing where to find its associated data files poses a challenge. If researchers are unable to find these resources, it may result in a duplication of efforts as they work to recreate them. In an attempt to solve this problem Mendoza et al. introduced the Vehicle Routing Problem Repository (VRP-REP; Mendoza et al. (2014)).

As its name suggests, VRP-REP is a repository for data associated with VRP research. It is a community-driven website that provides pages for VRP studies that contain their data files as well as links to published articles. VRP-REP also proposes a schema defining a common file format in which these data – primarily problem instances and solution files – can be stored. The format is intended to be sufficiently flexible to accommodate most VRP variants. As of February 2020, VRP-REP contains 90 datasets spanning 51 VRP variants that have been uploaded by 494 users from 61 countries. These datasets have garnered over 12,000 downloads.

6.2 Mapper

As an additional tool for users of VRP-REP, I developed Mapper.[1] Mapper is a web-based portal where users can upload and visualize VRP-REP-compliant problem instances and solutions. An example depicting an instance and solution plotted with Mapper can be seen

[1] https://vrp-rep.github.io/mapper/

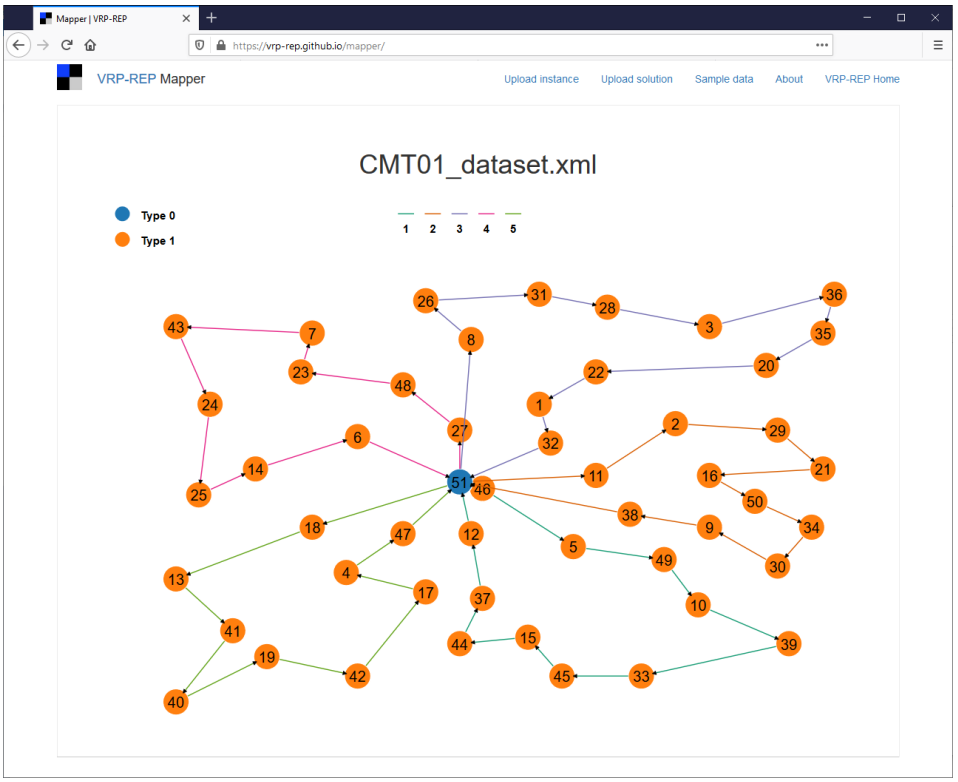

Figure 6.1: Screenshot of the Mapper utility with instance and solution uploaded.

in Figure 6.1.

VRP-REP-compliant instances and solutions are XML files that abide by a defined schema.[2] The VRP-REP instance format specifies that the locations a vehicle may visit be defined by `node` elements in the XML file. These `node` elements must at least contain information defining their location, their node ID, and their type (e.g., type 0 for a depot, type 1 for a customer). While additional information about nodes may be provided, it is currently ignored by Mapper. When a user uploads a problem instance, Mapper provides a plot of the nodes, with a unique color for each node type. Once an instance has been loaded, users then have the option of uploading a solution for that instance. VRP-REP-compliant solutions consist of one or more `route` elements, which Mapper also distinguishes by color. When solutions contain more than one route, users can choose whether to hide or display routes by clicking on their entries in the color legend.

Development Details Mapper is written in JavaScript, powered largely by the D3 library (Bostock et al. 2011). It runs in users' web-browsers, requiring no back-end server to generate the visualization. It is designed to provide simple yet general instance and solution visualizations that accommodate most any VRP variant. As a result, variant-specific details (such as nodes' demands, service times, time windows, etc.) are ignored.

[2] Schemas for instances and solutions are provided on VRP-REP's Resources page: `http://www.vrp-rep.org/resources.html`

Figure 6.2: Screenshot of the Mapper utility specifically adapted for the MP-E-VRP.

Mapper's development is open-source, and its code is available on VRP-REP's GitHub page.[3] Interested users are encouraged to fork the repository and build upon Mapper so as to better visualize their specific VRP variant. One example of a variant-specific adaptation of Mapper is for Echeverri et al.'s work on the multi-period E-VRP (MP-E-VRP; Echeverri et al. (2019)). For the MP-E-VRP, Mapper offers additional features such as 1) differentiating customer nodes by color based on the period in which they are to be serviced; 2) the ability to hide or display customer nodes based on the period in which they are to be serviced; 3) an additional panel showing a plot of EVs' battery levels over time; 4) a panel displaying charging operations across time; and 5) a panel showing the total instantaneous power draw from all charging stations across time. A depiction of Mapper as tailored for the MP-E-VRP is shown in Figure 6.2.[4]

[3] https://github.com/VRP-REP/mapper/

[4] The MP-E-VRP version of Mapper is available online at https://e-vro.github.io/visual-solution-checkers/MP-EVRP/

Bibliography

J. D. Adler and P. B. Mirchandani. Online routing and battery reservations for electric vehicles with swappable batteries. *Transportation Research Part B: Methodological*, 70:285–302, 2014.

L. Al-Kanj, J. Nascimento, and W. B. Powell. Approximate dynamic programming for planning a ride-sharing system using autonomous fleets of electric vehicles. *arXiv preprint arXiv:1810.08124*, 2018.

J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

Anheuser-Busch. Anheuser-Busch Drives the Future of Beer Delivery with One of Largest Reported Pre-Orders of Tesla Electric Self-Driving Trucks, Dec. 2017. https://www.anheuser-busch.com/newsroom/2017/12/anheuser-busch-drives-the-future-of-beer-delivery-with-one-of-la.html, Accessed on September 4, 2019.

P. Avella, M. Boccia, and A. Sforza. Solving a fuel delivery problem by heuristic and exact approaches. *European Journal of Operational Research*, 152(1):170–179, 2004.

O. Bahrami, J. Gawron, J. Kramer, and B. Kraynak. Flexbus: Improving public transit with ride-hailing technology, Dec. 2017. http://sustainability.umich.edu/media/files/dow/Dow-Masters-Report-FlexBus.pdf, Accessed December 2019.

J. Barco, A. Guerra, L. Muñoz, and N. Quijano. Optimal routing and scheduling of charge for electric vehicles: A case study. *Mathematical Problems in Engineering*, 2017, 2017.

M. Baum, J. Dibbelt, A. Gemsa, D. Wagner, and T. Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. *Transportation Science*, 53(6):1627–1655, 2019.

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

G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

D. Bertsimas, P. Jaillet, and S. Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.

J. Bischoff and M. Maciejewski. Simulation of city-wide replacement of private cars with autonomous taxis in berlin. *Procedia computer science*, 83:237–244, 2016.

M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

A. Braverman, J. G. Dai, X. Liu, and L. Ying. Empty-car routing in ridesharing systems. *Operations Research*, 67(5):1437–1452, 2019.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

D. B. Brown, J. E. Smith, and P. Sun. Information relaxations and duality in stochastic dynamic programs. *Operations research*, 58(4-part-1):785–801, 2010.

J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2488–2496, 2015.

A. Campbell and B. Thomas. Challenges and advances in a priori routing. In B. Golden, S. Raghavan, and E. Wasil, editors, *The vehicle routing problem: latest advances and new challenges*. Springer, New York, NY, 2008.

T. D. Chen, K. M. Kockelman, and J. P. Hanna. Operations of a shared, autonomous, electric vehicle fleet: Implications of vehicle & charging infrastructure decisions. *Transportation Research Part A: Policy and Practice*, 94:243–254, 2016.

F. Chollet et al. Keras. `https://keras.io`, 2015.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

E. Denardo. *Dynamic programming: models and applications*. Dover Publications, Mineola, NY, 2003.

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

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

L. C. Echeverri, A. Froger, J. E. Mendoza, and E. Néron. A matheuristic for the Multi-period Electric Vehicle Routing Problem. In *13th Metaheuristics International Conference*, Cartagena de Indias, Colombia, July 2019. URL `https://hal.archives-ouvertes.fr/hal-02279106`.

Edison Electric Institute. Electric vehicle sales: Facts and figures, Oct. 2019. `https://www.eei.org/issuesandpolicy/electrictransportation/Documents/FINAL_EV_Sales_Update_Oct2019.pdf`, Accessed January 2020.

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

Etalab. Fichier consolidé des bornes de recharge pour véhicules électriques, Oct. 2014. `https://www.data.gouv.fr/en/datasets/fichier-consolide-des-bornes-de-recharge-pour-vehicules-electriques/`, Accessed on September 4, 2019.

D. J. Fagnant and K. Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

D. J. Fagnant and K. M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, 2014.

B. Farley and W. Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, 1954.

FedEx. FedEx Introduces Zero-Emission All-Electric Nissan e-NV200 Vehicles in Belgium, Apr. 2017. https://about.van.fedex.com/newsroom/fedex-introduces-zero-emission-electric-nissan-e-nv200-vehicles-belgium/, Accessed on September 4, 2019.

J. V. C. Ferrándiz, M. A. S. Gabaldón, and Ó. C. Font. The use of an electric vehicle fleet for the domiciliary hospitalization unit of the Hospital of Alcoy. *Transportation research procedia*, 18: 411–418, 2016.

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

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

M. Gendreau, G. Laporte, C. Musaraganyi, and É. D. Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 26(12): 1153–1173, 1999b.

M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated vrp. In P. Toth and D. Vigo, editors, *The vehicle routing problem*, chapter 6, pages 129–154. SIAM, 2002.

M. Gendreau, O. Jabali, and W. Rei. Chapter 8: Stochastic vehicle routing problems. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 213–239. SIAM, 2014.

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

B. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

J. Goodson, J. W. Ohlmann, and B. Thomas. Rollout policies for dynamic solutions to the multi-vehicle routing problem with stochastic demand and duration limits. *Operations Research*, 61 (1):138–154, 2013.

H. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2094–2100. AAAI Press, 2016.

J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.

J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *32nd AAAI Conference on Artificial Intelligence*, pages 3207–3214. AAAI Press, 2018.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *32nd AAAI Conference on Artificial Intelligence*, pages 3215–3222. AAAI Press, 2018.

G. Hiermann, J. Puchinger, S. Ropke, and R. F. Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, 252(3):995–1018, 2016.

G. Hiermann, R. F. Hartl, J. Puchinger, and T. Vidal. Routing a mix of conventional, plug-in hybrid, and electric vehicles. *European Journal of Operational Research*, 272(1):235 – 248, 2019a. ISSN 0377-2217.

G. Hiermann, R. F. Hartl, J. Puchinger, and T. Vidal. Routing a mix of conventional, plug-in hybrid, and electric vehicles. *European Journal of Operational Research*, 272(1):235–248, 2019b.

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

J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *arXiv preprint arXiv:1911.11260*, 2019.

M. Hyland and H. S. Mahmassani. Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign avs to immediate traveler demand requests. *Transportation Research Part C: Emerging Technologies*, 92:278–297, 2018.

R. Iacobucci, B. McLellan, and T. Tezuka. Optimization of shared autonomous electric vehicles operations with charge scheduling and vehicle-to-grid. *Transportation Research Part C: Emerging Technologies*, 100:34–52, 2019.

O. Jabali, T. Van Woensel, and A. De Kok. Analysis of travel times and co2 emissions in time-dependent vehicle routing. *Production and Operations Management*, 21(6):1060–1074, 2012.

E. C. Jones and B. D. Leibowicz. Contributions of shared autonomous vehicles to climate change mitigation. *Transportation Research Part D: Transport and Environment*, 72:279–298, 2019.

J. Jung, R. Jayakrishnan, and K. Choi. Shared-taxi operations with electric vehicles. *Institute of Transportation Studies Working Paper Series, Irvine, Calif*, 2012.

N. Kang, F. M. Feinberg, and P. Y. Papalambros. Autonomous electric vehicle sharing system design. *Journal of Mechanical Design*, 139(1):011402, 2017.

A. Karpathy. Deep reinforcement learning: Pong from pixels, May 2016. http://karpathy.github.io/2016/05/31/rl/, Accessed January 2020.

A. Kawrykow, G. Roumanis, A. Kam, D. Kwak, C. Leung, C. Wu, E. Zarour, L. Sarmenta, M. Blanchette, J. Waldispühl, et al. Phylo: a citizen science approach for improving multiple sequence alignment. *PloS one*, 7(3):e31362, 2012.

F. Khatib, S. Cooper, M. D. Tyka, K. Xu, I. Makedon, Z. Popović, and D. Baker. Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, 108(47):18949–18953, 2011.

Ç. Koç, O. Jabali, J. E. Mendoza, and G. Laporte. The electric vehicle routing problem with shared charging stations. *International Transactions in Operational Research*, 26(4):1211–1243, 2019.

N. D. Kullman, J. Goodson, and J. E. Mendoza. Electric Vehicle Routing with Public Charging Stations. working paper or preprint, Sept. 2019. URL https://hal.archives-ouvertes.fr/hal-01928730.

N. D. Kullman, M. Cousineau, J. Goodson, and J. E. Mendoza. Dynamic Ridehailing with Electric Vehicles. working paper or preprint, Jan. 2020a. URL https://hal.archives-ouvertes.fr/hal-02463422.

N. D. Kullman, A. Froger, J. E. Mendoza, and J. C. Goodson. e-vro/frvcpy v0.1.0, Feb. 2020b. URL https://doi.org/10.5281/zenodo.3677583.

La Poste. La Poste et l'environnement, 2011. https://www.laposte.fr/legroupe/Nos-engagements/Developpement-durable/La-Poste-et-l-environnement%20?, Accessed on September 4, 2019.

C. R. La Rocca and J.-F. Cordeau. Heuristics for electric taxi fleet management at teo taxi. *INFOR: Information Systems and Operational Research*, 0(0):1–25, 2019.

G. Lample and D. S. Chaplot. Playing FPS games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

G. Laporte. Fifty years of vehicle routing. *Transportation science*, 43(4):408–416, 2009.

D.-H. Lee, H. Wang, R. L. Cheu, and S. H. Teo. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record*, 1882(1):193–200, 2004.

M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference*, pages 983–994. ACM, 2019.

A. Lieberoth, M. K. Pedersen, A. C. Marin, T. Planke, and J. F. Sherson. Getting humans to do quantum optimization-user acquisition, engagement and early results from the citizen cyberscience game quantum moves. *arXiv preprint arXiv:1506.08761*, 2015.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Y. Liu, B. Logan, N. Liu, Z. Xu, J. Tang, and Y. Wang. Deep reinforcement learning for dynamic treatment regimes on medical registry data. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 380–385. IEEE, 2017.

Lyft. Why 130,000 lyft passengers were ready to ditch their personal cars in less than 24 hours, Dec. 2018. https://blog.lyft.com/posts/ditchyourcardata, Accessed December 2019.

M. Maciejewski, J. Bischoff, and K. Nagel. An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems*, 31(1):68–77, 2016.

I. Markov, S. Varone, and M. Bierlaire. Integrating a heterogeneous fixed fleet and a flexible assignment of destination depots in the waste collection vrp with intermediate facilities. *Transportation research part B: methodological*, 84:256–273, 2016.

J. E. Mendoza, C. Guéret, M. Hoskins, H. Lobit, V. Pillac, T. Vidal, and D. Vigo. VRP-REP: the vehicle routing community repository. In *Third Meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog). Oslo, Norway*, 2014.

F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering*, 13(2):463–478, 2016.

V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pages 2204–2212. MIT Press, 2014.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

E. Mocanu, D. C. Mocanu, P. H. Nguyen, A. Liotta, M. E. Webber, M. Gibescu, and J. G. Slootweg. On-line building energy optimization using deep reinforcement learning. *IEEE Transactions on Smart Grid*, 2018.

A. Montoya, C. Guéret, J. E. Mendoza, and J. G. Villegas. A multi-space sampling heuristic for the green vehicle routing problem. *Transportation Research Part C: Emerging Technologies*, 70:113–128, 2016.

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

P. Morrissey, P. Weldon, and M. O'Mahony. Future standard and fast charging infrastructure planning: An analysis of electric vehicle charging behaviour. *Energy Policy*, 89:257–270, 2016.

B. Nag, B. L. Golden, and A. Assad. Vehicle routing with site dependencies. *Vehicle routing: methods and studies*, pages 149–159, 1988.

National Renewable Energy Laboratory. Alternative fuels data center, 2019. Data retrieved 09/2019 from https://www.nyserda.ny.gov/All-Programs/Programs/Drive-Clean-Rebate/Charging-Options/Electric-Vehicle-Station-Locator.

New York City Taxi & Limousine Commission. Tlc trip record data, 2018. Data retrieved 09/2019 from https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

NYC OpenData. Nyc taxi zones, 2019. Data retrieved 09/2019 from https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc.

T. Oda and C. Joe-Wong. Movi: A model-free approach to dynamic fleet management. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2708–2716. IEEE, 2018.

T. Oda and Y. Tachibana. Distributed fleet control with maximum entropy deep reinforcement learning. In *NeurIPS 2018 Machine Learning for Intelligent Transportation Systems Workshop*, 2018.

Office of Transportation and Air Quality. U.S. Transportation Sector Greenhouse Gas Emissions 1990-2017, June 2019. URL https://nepis.epa.gov/Exe/ZyPDF.cgi?Dockey=P100WUHR.pdf. EPA-420-F-19-047.

A. Omidvar and R. Tavakkoli-Moghaddam. Sustainable vehicle routing: Strategies for congestion management and refueling scheduling. In *Energy Conference and Exhibition (ENERGYCON), 2012 IEEE International*, pages 1089–1094. IEEE, 2012.

Orlando Utilities Commission. Championing the EV charge, 2018. https://www.ouc.com/environment-community/green-initiatives/championing-the-ev-charge, Accessed on September 4, 2019.

S. Pelletier, O. Jabali, and G. Laporte. 50th anniversary invited article - goods distribution with electric vehicles: review and research perspectives. *Transportation Science*, 50(1):3–22, 2016.

J. F. Pettit, R. Glatt, J. R. Donadee, and B. K. Petersen. Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning. *arXiv preprint arXiv:1912.03408*, 2019.

H. N. Psaraftis, M. Wen, and C. A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

D. Reyes, M. Savelsbergh, and A. Toriello. Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, 80:71–91, 2017.

H. Ritchie and M. Roser. Co2 and greenhouse gas emissions. *Our World in Data*, 2020.

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

R. Roberti and M. Wen. The electric traveling salesman problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 89:32–52, 2016.

A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

M. W. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *4th International Conference on Learning Representations*, 2016.

M. Schiffer and G. Walther. An adaptive large neighborhood search for the location-routing problem with intra-route facilities. *Transportation Science*, 52(2):331–352, 2017.

M. Schiffer, M. Schneider, and G. Laporte. Designing sustainable mid-haul logistics networks with intra-route multi-resource facilities. *European Journal of Operational Research*, 265(2):517 – 532, 2018. ISSN 0377-2217.

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

T. W. Schneider. Taxi and ridehailing usage in new york city, 2019. https://toddwschneider.com/dashboards/nyc-taxi-ridehailing-uber-lyft-data/, Accessed September 2019.

N. Secomandi. Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers & Operations Research*, 27(11-12):1201–1225, 2000.

K. T. Seow, N. H. Dang, and D.-H. Lee. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering*, 7(3):607–616, 2009.

J. Shi, Y. Gao, W. Wang, N. Yu, and P. A. Ioannou. Operating electric vehicle fleet for ride-hailing services with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

A. Singh, A. Alabbasi, and V. Aggarwal. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning. In *NeurIPS 2019 Machine Learning for Autonomous Driving Workshop*, 2019.

P. Slowik, L. Fedirko, and N. Lutsey. Assessing ride-hailing company commitments to electrification, Feb. 2019. https://theicct.org/sites/default/files/publications/EV_Ridehailing_Commitment_20190220.pdf, Accessed January 2020.

K. Sörensen and M. Sevaux. A practical approach for robust and flexible vehicle routing using metaheuristics and monte carlo sampling. *Journal of Mathematical Modelling and Algorithms*, 8(4):387, 2009.

A. Sterling. Eyewire, Dec 2012. URL https://blog.eyewire.org/about/. Accessed on November 11, 2019.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN 9780262352703.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1): 9–44, 1988.

T. M. Sweda, I. S. Dolinskaya, and D. Klabjan. Adaptive routing and recharging policies for electric vehicles. *Transportation Science*, 51(4):1326–1348, 2017.

Tesla. Supercharging cities, 2017. https://www.tesla.com/blog/supercharging-cities, Accessed December 2019.

Tesla. Service, 2018. https://www.tesla.com/service, Accessed on September 4, 2019.

A. Toriello, W. B. Haskell, and M. Poremba. A dynamic traveling salesman problem with stochastic arc costs. *Operations Research*, 62(5):1107–1125, 2014.

P. Toth and D. Vigo. *The vehicle routing problem*. SIAM, 2014.

Y.-y. Tseng, W. L. Yue, M. A. Taylor, et al. The role of transportation in logistics chain. Eastern Asia Society for Transportation Studies, 2005.

M. Uhrig, L. Weiß, M. Suriyah, and T. Leibfried. E-mobility in car parks–guidelines for charging infrastructure expansion planning and operation based on stochastic simulations. In *EVS28 International Electric Vehicle Symposium and Exhibition*, pages 1–12, 2015.

M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and B. W. Thomas. On modeling dynamic vehicle routing problems. working paper.

M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and M. Hennig. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, 53 (1):185–202, 2018.

UPS. UPS To Deploy New, State-Of-The-Art Electric Vehicles In London And Paris, May 2018. https://pressroom.ups.com/pressroom/ContentDetailsViewer.page?ConceptType=PressReleases&id=1525867012405-924, Accessed on September 4, 2019.

T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

J. G. Villegas, C. Guéret, J. E. Mendoza, and A. Montoya. The technician routing and scheduling problem with conventional and electric vehicle. working paper, June 2018. URL https://hal.archives-ouvertes.fr/hal-01813887.

O. Vinyals, S. Gaffney, and T. Ewalds. DeepMind and Blizzard open StarCraft II as an AI research environment, Aug 2017. URL https://deepmind.com/blog/announcements/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment. Accessed on November 11, 2019.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913. ACM, 2018.

L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2151–2159. ACM, 2017.

R. Zhang and M. Pavone. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research*, 35(1-3):186–203, 2016.

L. Zhen, M. Li, G. Laporte, and W. Wang. A vehicle routing problem arising in unmanned aerial monitoring. *Computers & Operations Research*, 105:1–11, 2019.

Résumé :

Cette thèse présente trois études menées sur des problèmes de tournées dynamiques. En particulière, elle se concentre sur les challenges résultants de l'utilisation de véhicules électriques dans les systèmes logistiques et de transports. Dans la première étude, nous introduisons le problème de tournées de véhicules électriques avec des bornes de recharge publiques et privées. Dans ce contexte, les véhicules peuvent recharger leurs batteries en route, dans des bornes publiques, ainsi qu'au dépôt (bornes privées). Pour se protéger contre l'incertitude de la disponibilité des bornes publiques, nous présentons des politiques de routage qui anticipent la dynamique des files d'attente des bornes. Nos politiques se basent sur une décomposition du problème en deux phases : routage et planification des opérations de recharge. Grâce à cette décomposition, nous obtenons la politique statique optimale, ainsi qu'un certain nombre de politiques dites « anticipatoires » et une borne inférieure. Des tests numériques effectués sur des instances réelles fournies par une entreprise, monter que nos politiques sont capables de livrer des solutions avec un gap d'optimalité de moins de 5%. Nos tests montrent aussi que permettre aux véhicules de charger en dehors du dépôt (même en présence d'incertitude sur la disponibilité des bornes) se traduit par des économies considérables dans la durée des routes.

Dans la deuxième étude, nous considérons le problème d'un opérateur contrôlant une flotte de véhicules de tourisme avec chauffeur (VTCs) électriques. L'opérateur, qui cherche à maximiser ses revenus, doit affecter les véhicules aux demandes au fur et à mesure de leur apparition ainsi que charger et repositionner les véhicules en prévision des demandes futures. Pour attaquer ce problème, nous utilisons des approches basées sur l'apprentissage par renforcement profond. Pour mesurer la qualité de nos approches, nous avons développé aussi une heuristique proche de celle typiquement utilisée dans l'affectation de taxis, ainsi que des bornes supérieures. Nous testons nos approches dans des instances construites à partir de données réelles de l'île de Manhattan. Nos tests montrent que notre meilleure politique basée sur l'apprentissage profond livre des résultats supérieurs à ceux livrés par l'heuristique. Les tests montrent aussi que cette stratégie passe facilement à l'échelle et peut être déployée sur de plus grandes instances sans entrainement supplémentaire.

La dernière étude introduit une nouvelle approche générique pour modéliser des problèmes d'optimisation dynamique sous la forme de jeux vidéo de type Atari. L'objectif est de les rendre abordables à travers de méthodes de solution issus de communauté d'apprentissage par renforcement profond. L'approche est flexible et applicable à un large éventail de problèmes. Pour illustrer son application, nous nous attaquons à un problème bien établie dans la littérature : le problème de tournées de véhicules avec des requêtes de service stochastiques. Nos résultats préliminaires sur ce problème sont très encourageants et montrent que « l'Atari-fication » peut être la voie pour résoudre des problèmes d'optimisation dynamique qui s'avèrent difficiles pour les approches basées sur les outils classiques de la recherche opérationnelle.

Les derniers chapitres présentent deux logiciels développées pour supporter nos recherches. Le premier, nommé frvcpy, permet de déterminer l'insertion optimal des opérations de recharge dans une tournée prédéterminée. Ce logiciel et son code source, présenté comme une bibliothèque Python, a été mis à disposition de la communauté scientifique. Le deuxième outil, VRP-REP Mapper, est un outil web pour visualiser et analyser des solutions

pour les problèmes de tournées de véhicules. Cette outil a été intégré a www.vrp-rep.org, la plateforme de référence pour le partage de données scientifiques dans le domaine.

Mots clés :

routage dynamique, incertitude, optimisation, processus de décision markovien, logistique, véhicules électriques

Abstract :

This thesis details three problems and two software tools related to dynamic decision making under uncertainty in vehicle routing and logistics, with an emphasis on the challenges encountered when adopting electric vehicles. We first introduce the electric vehicle routing problem with public-private recharging strategy in which vehicles may recharge en-route at public charging infrastructure as well as at a privately-owned depot. To hedge against uncertain demand at public charging stations, we design routing policies that anticipate station queue dynamics. We leverage a decomposition to identify good routing policies, including the optimal static policy and fixed-route-based rollout policies that dynamically respond to observed queues. The decomposition also enables us to establish dual bounds, providing a measure of goodness for our routing policies. In computational experiments using real instances from industry, we show the value of our policies to be within five percent of the value of an optimal policy in the majority of instances and within eleven percent on average. Further, we demonstrate that our policies significantly outperform the industry-standard routing strategy in which vehicle recharging generally occurs at a central depot. Our proposed methods for this problem stand to reduce the operating costs associated with electric vehicles, facilitating the transition from internal-combustion engine vehicles.

We then consider the problem of an operator controlling a fleet of electric vehicles for use in a ridehailing service. The operator, seeking to maximize revenue, must assign vehicles to requests as they arise and recharge and reposition vehicles in anticipation of future requests. To solve this problem, we employ deep reinforcement learning, developing policies whose decision making uses Q-value approximations learned by deep neural networks. We compare these policies against a common taxi dispatching heuristic and against dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information which we establish using a Benders-based decomposition. We assess performance on instances derived from real data for the island of Manhattan in New York City. We find that, across instances of varying size, our best policy trained with deep reinforcement learning outperforms the taxi dispatching heuristic. We also provide evidence that this policy may be effectively scaled and deployed on larger instances without retraining.

We then present a new general approach to modeling research problems as Atari-like videogames to make them amenable to recent solution methods from the deep reinforcement learning community. The approach is flexible, applicable to a wide range of problems. Here, we demonstrate its application on the well-studied vehicle routing problem with stochastic service requests. Our preliminary results on this problem, though not transformative, show signs of success and suggest that Atari-fication may be a useful modeling approach for researchers studying problems involving sequential decision making under uncertainty.

We then introduce *frvcpy*, the first of our two proposed software tools. In the routing of electric vehicles, one of the most challenging tasks is determining how to make good charging decisions for an electric vehicle traveling a given route. This is known as the fixed route vehicle charging problem. An exact and efficient algorithm for this task exists, but its implementation is sufficiently complex to deter researchers from adopting it. Our proposed tool, *frvcpy*, is an open-source Python package implementing this algorithm. Our

aim with the package is to make it easier for researchers to solve electric vehicle routing problems, facilitating the development of optimization tools that may ultimately enable the mass adoption of electric vehicles.

Finally, we introduce the second software tool, *Mapper*. *Mapper* is a simple web-based visualizer of problem instances and solutions for vehicle routing problems. It is designed to accompany the suite of tools already available to users of the vehicle routing community's website, *The Vehicle Routing Problem Repository*.

Keywords :

dynamic routing, uncertainty, optimization, Markov decision process, logistics, electric vehicles