Routing and Scheduling of Electric and Alternative-Fuel Vehicles

by

Jonathan D. Adler

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2014 by the
Graduate Supervisory Committee:

Pitu Mirchandani, Chair
Ronald Askin
Esma Gel
Guoliang Xue
Muhong Zhang

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

Vehicles powered by electricity and alternative-fuels are becoming a more popular form of transportation since they have less of an environmental impact than standard gasoline vehicles. Unfortunately, their success is currently inhibited by the sparseness of locations where the vehicles can refuel as well as the fact that many of the vehicles have a range that is less than those powered by gasoline. These factors together create a “range anxiety” in drivers, which causes the drivers to worry about the utility of alternative-fuel and electric vehicles and makes them less likely to purchase these vehicles. For the new vehicle technologies to thrive it is critical that range anxiety is minimized and performance is increased as much as possible through proper routing and scheduling.

In the case of long distance trips taken by individual vehicles, the routes must be chosen such that the vehicles take the shortest routes while not running out of fuel on the trip. When many vehicles are to be routed during the day, if the refueling stations have limited capacity then care must be taken to avoid having too many vehicles arrive at the stations at any time. If the vehicles that will need to be routed in the future are unknown then this problem is stochastic. For fleets of vehicles serving scheduled operations, switching to alternative-fuels requires ensuring the schedules do not cause the vehicles to run out of fuel. This is especially problematic since the locations where the vehicles may refuel are limited due to the technology being new.

This dissertation covers three related optimization problems: routing a single electric or alternative-fuel vehicle on a long distance trip, routing many electric vehicles in a network where the stations have limited capacity and the arrivals into the system are stochastic, and scheduling fleets of electric or alternative-fuel vehicles with limited locations to refuel. Different algorithms are proposed to solve each of the three problems, of which some are exact and some are heuristic. The algorithms are tested on both random data and data relating to the State of Arizona.

DEDICATION

To my grandmother Betty.

ACKNOWLEDGMENTS

I am thankful to my wife gen for helping me out whenever graduate school proved to be particularly annoying, and for following me across the country (twice!) over the years. I am thankful for all of the times she called me out when I was wrong even though it might have taken hours, days, or months before I realized she was right.

I would like to express my deepest gratitude to Professor Pitu Mirchandani for immediately providing me with a great research topic for my dissertation the moment I arrived at ASU, and for giving constant help and guidance during the program. The fact that his door was always open was instrumental to the completion of this document, and he always “had my back.” I would also like to thank the rest of my committee: Professors Ronald Askin, Esma Gel, Guoliang Xue, and Muhong Zhang for helping me with my dissertation as well as helping me navigate the perils of a PhD program. I would also like to thank Professors Michael Kuby and Subbarao Kambhampati for providing guidance with my research.

Thank you to my friends Stephanie Bell, Talon Burns, Eric and Katie DeMarco, Charlie and Rebecca Everett, Chester Ismay, Arthur Mitrano, Joshua Shimko, Courtney and Michael Tallman, and Christopher Wishon for turning this inhospitable wasteland into a place I enjoy being. Thank you to my family who always stressed the importance of education and taught me the skills I needed to succeed at Arizona State University.

This work has been financially supported by the National Science Foundation grant #1234584 as well as the U.S. Department of Transportation Federal Highway

Administration through the Dwight David Eisenhower Transportation Fellowship Program.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the above organizations or persons.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Chapter 1

INTRODUCTION

The environmental, geopolitical, and financial implications of the global dependence on oil are well known and documented, and much is being done to lessen our use of the substance. One focus of this issue has been on replacing gasoline powered automobiles with vehicles that used electric motors or alternative-fuels. Governments and automotive companies have recognized the value of these vehicles in helping the environment (Hacker et al., 2009) and are encouraging the ownership of electric and alternative fuel vehicles through economic incentives (Gallagher and Muehlegger, 2011). Cities and private companies are assisting the owners of these new types of vehicles by creating the infrastructure to allow them to refueling in well trafficked areas (Senart et al., 2010; Vaughan, 2011), however the refueling stations are still sparsely located when compared to gas stations.

For many electric vehicles, such as the Nissan LEAF or Tesla Model S Sedan, the primary method of recharging the vehicle battery is to plug the battery into the power grid at places like home or the office (Bakker, 2011; Kurani et al., 2008). Because the battery has a low capacity and cannot travel far before requiring a recharge, this method has the implicit assumption that the vehicle will be used only for driving short distances. *Range anxiety*, when the driver is concerned that the vehicle will run out of charge or fuel before reaching the destination, is a major hindrance for the market penetration of electric and alternative-fuel vehicles (Jeeninga et al., 2002; Sovacool and Hirsh, 2009; Yu et al., 2011). These inherent problems, combined with a current lack of refueling infrastructure, are inhibiting a wide-scale adoption of electric and alternative-fuel vehicles. The problems are especially apparent during intercity trips, where the vehicle needs to recharge or refuel several times before arriving at the destination.

For the case of electric vehicles, companies are trying to overcome this limited range requirement in two different ways. *Rapid recharge station*s are locations where a vehicle can be charged in only a few minutes to near full capacity. Besides being much more costly to operate rapid recharge stations than standard electric power outlets, the vehicles still take more time to recharge than a gasoline vehicle would take to refuel (Botsford and Szczepanek, 2009). Another refueling infrastructure design is to have *battery-exchange stations*. These stations remove a pallet of batteries that are nearly depleted from a vehicle and replace the battery pallet with one that has already been charged (Shemer, 2012). This method of refueling has the advantage that it is reasonably quick since the vehicle does not need to wait for the batteries to be charged. The unfortunate downside is that all of the vehicles serviced by the battery-exchange stations are required to use the identical pallets and batteries so that they can be interchanged without issue. In conjunction to the battery-exchange concept, it is assumed that there is a viable business model that provides a reasonable profit for companies that establish battery-exchange facilities for the public. Battery-exchange stations have been implemented in the countries of Israel and Denmark by the company Better Place (Better Place, 2013), but unfortunately they have yet to take off (Kershner, 2013) and this has forced the company into bankruptcy. The electric vehicle company Tesla Motors Inc., which currently sells plug-in electric vehicles, has recently shown their cars working with battery-exchange technology (Motavalli, 2013).

There are several different types of alternative-fuel vehicles that are in development and production (Ogden et al., 1999). Some alternative-fuel vehicles are in production for use in limited geographical locations, such as the Honda Civic GX powered by compressed or liquefied natural gas for use in the Los Angeles metropolitan area (Kuby et al., 2013). Toyota also plans on putting a hydrogen powered vehicle into production for use in California starting in 2015 (Lavrinc, 2014). These vehicles are required to refuel at specialized stations since they cannot use gasoline. The lack of availability of these stations still cause range anxiety to be a concern, even though the range of the vehicles is typically longer than that of electric vehicles.

This dissertation will focus on addressing the routing and scheduling questions that arise from the implementation of electric and alternative-fuel vehicles. Specifically it will focus on several types of situations depending on the decision making entity. In the simplest case, the vehicle is controlled independently by the user, such as with a consumer purchased alternative-fuel car where the driver plans the route the vehicle will take. In the fleet setting, the vehicles are fully coordinated by a central planning agency, such as with electric buses being routed around a city. There can also be cases of partially centralized control, where a central agency plans a subset of the decisions. An example of this case would be a group of consumer electric vehicles where the drivers decide the origin, destination, and departure time, while the central agency decides at which stations the vehicles swap their batteries based on battery availability at the stations.

Since the problems inherent in the implementation of electric and alternative-fuel vehicles are similar, this dissertation will in most cases be applicable to both types of vehicles. The main exception to this is with electric vehicles using battery exchange stations, since the infrastructure has to handle the demand so that the stations do not run out of charged batteries. Throughout the document there will be instances where we use the term electric vehicles or alternative-fuel vehicles, but they are interchangeable unless otherwise specified. Note that throughout this document the term *refueling station* will be used to mean rapid recharging stations, battery-exchange stations, and/or alternative-fuel stations depending on the context, and similarly the term *refuel* will be used to mean visiting one of those stations to recharge a battery, exchange a battery, or refuel the vehicle respectively.

The dissertation is broken down into four main chapters. Chapter 2 will be on how to route a single electric or alternative-fuel vehicle given constraints on the distance the vehicle can travel before refueling and where it can refuel. Chapter 3 will extend the work in the first section to allow for routing many vehicles at once in a network with limited availability at the refueling stations. This primarily concerns the case of routing electric vehicles with battery swapping, since the refueling stations will have a limited number of available charged batteries on hand. The chapter will assume that the when routing a vehicle what vehicles will arrive in the future is unknown. Chapter 4 focuses on how to schedule a fleet of electric or alternative-fuel vehicles that are to be assigned to trips with specified starting and ending times. This chapter is relevant for cases such as using alternative-fuel buses to serve different routes in a city. Finally, Chapter 5 presents some additional results for each of the previous three chapters. This includes adding stochastic edge lengths to the single vehicle routing problem, adding variable arrival rates to the network routing problem, and adding the decision on where to locate refueling stations when planning fleet schedules. Below we give a more in depth overview of each of the main topics.

1.1. Routing issues for a single vehicle

Taking a trip, especially one through lowly populated areas, requires the driver to plan when the vehicle will need to be refueled. Given the abundance of gasoline stations for standard vehicles, drivers usually consider refueling only when their fuel tank is low. In the case of electric and alternative-fuel vehicles, planning when to refuel is more important than for gasoline vehicles, since there are few places to refuel which increases the possibility of completely running out of fuel. Likewise, understanding routing would be even more critical for planning facility placement, since the infrastructure would gradually involve so at least initially the density of the refueling stations would be very low. The low density of refueling stations makes it unlikely a station would fall directly on the shortest route for a vehicle, therefore one needs to develop models which look for the routes from origins to destinations that include detouring to refueling stations. Objectives for these models could be to (a) minimize the total detouring distances and (b) minimize the total number of refueling stops. Such shortest route models relate to the constrained shortest path problem (Beasley and Christofides, 1989; Desrochers and Soumis, 1988; Handler and Zang, 1980; Xiao et al., 2005). However, it is surprising that detouring is not a consideration in these models.

Related to this detouring issue is managing the anxiety of the driver. The anxiety function is the propensity of a driver to delay refueling the vehicle while increasing the risk of possibly running out of fuel. For example, a driver with an extremely high anxiety will refuel their vehicle at every possible instance, to completely minimize the possibility of running out of fuel. A driver with a low anxiety will refuel the vehicle as rarely as possible, to minimize the number of instances where she needs to stop and refuel. Most drivers fall somewhere in between these two extremes, balancing a desire to travel as quickly as possible with a desire to lower the risk of getting stuck.

Chapter 2 of the dissertation provides methods for finding the shortest walks for vehicles with a limited range. These algorithms handle the case where the number of stops is restricted, as well as when the objective is to minimize driver anxiety rather than driving time. Using randomly generated data, the algorithms are tested and compared to finding the shortest path without being concerned about fuel. This chapter was previously published in the journal *Networks and Spatial Economics* (Adler et al., 2014), and the copyright is held by Springer. Section 5.1 of the dissertation modifies this problem by adding stochastic edge lengths to it, which may require the driver to change the route once they set out.

1.2. Routing issues for multiple vehicles

An extension of the problem of routing a single electric vehicle that uses battery swapping stations is to consider the case where the routes are planned by a central agency based on the number of batteries available at each location. If each driver is planning their route independently and they are each taking the shortest route possible, a highly visited battery-exchange station could run out of batteries leaving drivers stranded. Thus if the routes are planned by a central agency they can possibly avoid having vehicles arriving at battery swapping stations that have no charged batteries by having drivers take routes that are not necessarily the shortest.

Since each vehicle arrives into the routing system when the driver turns it on and requests a route, the central planning agency will not know at exactly what time each vehicle will need to be routed. The agency may at most have the rate at which vehicles are expected to arrive to travel between each origin and destination pair. The decision of which route to send a vehicle on must be made at the time the vehicle requests it, before knowing what other vehicles will turn on in the future. If the objective is to minimize the travel time of all vehicles, this problem is an online stochastic optimization problem, since the decisions have to be made before full routing demand is revealed.

Chapter 3 of this dissertation covers the topic of how to route electric vehicles in a network with capacitated battery-exchange stations and unknown future arrivals. The system is modeled as a Markov chance-decision process, which allows for the status of the batteries at the stations to be considered as the state of the system. Time over the course of the day is broken into discrete units, and during each time unit there is the possibility of a single car arriving. This assumes that the probability that two or more vehicles arriving during the same time unit is negligible, which is valid if the time units are sufficiently small. The vehicle arrival process is modeled as a Poisson process, and how they are routed are decisions which change the state of the system. Since finding the optimal policy for the Markov chance-decision process is intractable for all but trivial cases, approximate dynamic programming techniques are used. The approximate dynamic programming algorithm is tested on data involving the Arizona highway system. Section 5.2 of the dissertation extends the problem by removing some assumptions from Chapter 3, such as assuming that vehicles drop off batteries having zero charge remaining.

1.3. Scheduling issues

The problems involved in scheduling a fleet of vehicles to service routes at specific times, for example a fleet of buses in a city, are broadly referred to as vehicle scheduling problems (VSP) which have been well studied and documented (Bodin et al., 1983; Dror, 2000; Freling et al., 2001; Golden, 1988; Laporte, 2009). In these problems there are typically a fleet of vehicles each which is stored at a depot, and there are a set of trips to serve throughout the day each with specific start and end times and locations. Traveling between the locations has a specific cost, and the objective of the problem is to serve all of the trips with a minimum cost. In the case of a fleet of buses in the city, the tasks would be bus trips with specific start and end times that need to be served for picking up and dropping off passengers.

Changing the fleet to one that uses electric or alternative-fuel vehicles increases the complexity of the problem because of the limited range of the vehicles. This range limit will require the vehicles to refuel several times during their use throughout the day, sometimes having to detour significantly from the original route due to the limited number of refueling stations. Refueling is also complicated by the fact that the vehicle cannot refuel during an assigned trip, only before or after a trip is completed. Thus, despite the fact that electric and alternative-fuel vehicles have less of an environmental impact than gasoline ones, some inefficiency is added by requiring frequent stops. An explicit requirement of refueling, sometimes several times during a set of trips, adds not only the refueling requirements for the vehicles but also a detouring component to planning that is not present in the standard VSP problems. Cases of the vehicle scheduling problem with time and fuel constraints have been covered before, but not with the specific ability to refuel at specialized stations.

In Chapter 4 of this dissertation we present the alternative-fuel multiple depot vehicle scheduling problem. This problem takes the classic multiple depot vehicle scheduling problem and adds a constraint on the distance a vehicle can travel before refueling. The refueling can be done at a fixed set of specified locations. An exact column generation algorithm is proposed to solve the problem, and several heuristic methods are discussed. The column generation algorithm and one heuristic algorithm are tested on both randomly generated data and data from Valley Metro, the organization that provides public transportation for the Phoenix Arizona metropolitan area. In Section 5.3 of the dissertation we modify the problem to allow for the decision of where to place the refueling stations to be made in addition to the scheduling decisions.

Parts of this introduction chapter were included in Mirchandani et al. (Mirchandani et al., 2014).

Chapter 2

THE ELECTRIC VEHICLE SHORTEST WALK PROBLEM

This chapter focuses on only one aspect of the overall electric and alternative-fuel vehicle design problem: finding the shortest walk from an origination point s to a destination t with minimum cost. This is route referred to as a "walk" as opposed to a "path" since a detour to exchange or charge a battery may include repeat arcs which are normally not included in shortest paths. In this chapter for simplicity we will assume the vehicles are electric vehicles (EV) and the stations are battery-exchange stations, however this problem applies to any scenario where there only a limited number of places for "refueling." This is true not only for electric vehicles but also other alternative fuel vehicles where a refueling infrastructure still needs to be developed, for example hydrogen powered vehicles (Ogden et al., 1999) where empty tanks or canisters are exchanged for full ones at special stations.

Consider the underlying scenario. Taking a trip, especially one through lowly populated areas, requires the driver to plan when the vehicle will need to be refueled. Given the abundance of gasoline stations for standard vehicle, drivers of these vehicles usually consider refueling only when their fuel tank is low. The search for a good refueling point can be further aided by navigation systems and smart phone apps, such as Google Maps, that provide motorists the location of gasoline stations in the vicinity. In the case of EVs, planning refueling is more important than for gasoline vehicles, since there are few places to recharge. Hence, one needs to develop models which look for the shortest routes from origins to destinations that include detouring when necessary.

The problem of finding the shortest path for an electric vehicle was originally discussed by Ichimori et. al. (Ichimori et al., 1981) where a vehicle has a limited battery and is allowed to stop and recharge at certain locations. This chapter improves on that work by adding a limit to the number of times the vehicle can stop, as well as running empirical tests to validate the algorithm and describing several special case modifications (such as minimizing the maximum anxiety). We also provide an illustrated example of the algorithm. The problem presented by Ichimori et al. was discussed by Lawler (Lawler, 2001) who sketched a polynomial algorithm for its solution. If each arc requires an amount of fuel that does not depend on the length of the arc, and the goal is to find the shortest path constrained on the amount of fuel used (and the vehicle cannot stop to refuel), then the problem is exactly the shortest weight-constrained path problem (Garey and Johnson, 1979). This problem is NP-hard and has been discussed extensively in the literature (Beasley and Christofides, 1989; Desrochers and Soumis, 1988; Handler and Zang, 1980). Adding refueling stations to the shortest weight-constrained path problem has been discussed by Laporte and Pascoal (Laporte and Pascoal, 2011) and Smith et al. (Smith et al., 2012), although since the fuel and length components of the arcs are not related the problem is still NP-hard. The problem described in this chapter can be solved in polynomial time since the fuel and length components of each arc are equal; this allows for more efficient solutions. Sachenbacher (Sachenbacher, 2010) has studied route planning for electric vehicles on a single battery use where parts of the trip can return energy to the battery.

There is a complementary location problem, which is not addressed in this chapter, where we wish to locate “refueling” stations in a region where there are currently none. The problem of optimally locating such refueling stations has been investigated by several researchers (Capar and Kuby, 2012; Kuby and Lim, 2007; Kuby, 2005; Lim and Kuby, 2010; Upchurch et al., 2009). Typically, they use modifications of flow capturing or flow interception models (Berman et al., 1992; Hodgson, 1990; Rebello et al., 1995), to cover as many Origin-Destination routes as possible with a given number of stations. To compare proposed models, standard p-median and p-center problems (e.g. (Mirchandani and Francis, 1990)) have been used as proxies for maximizing proximity to stations and coverage by stations, respectively, for locating the stations. The developed models have been compared empirically for specific scenarios in order to choose one location model over another (Lim and Kuby, 2010). However, these models do not take into consideration the likely possibility of vehicles making detours to refuel; therefore the direct consideration of locating facilities to minimize detouring distances and or minimizing detouring stops have not been included in the model developments.

Another location issue relates to the effect of locating battery exchange-stations on origin and destination traffic patterns of the driving population. If travelers from origins to destinations do not choose the shortest routes but instead choose routes that minimize their detouring costs due to refueling, then the addition of battery exchange stations would change traffic patterns to a new equilibrium. There has been some recent consideration of the effect of EVs on traffic assignment and traffic equilibrium, but the research is only on restricting the distances electric vehicles can travel and assumes no refueling. (Jiang et al., 2012).

2.1. The shortest electric vehicle walk problem

We will assume a network model. The prototypical problem is to find the "best" walk from a given origin to a given destination, possibly making battery exchanges (we will refer this generically to "refueling" when the context is clear). Any solution walk must have no segment without refueling with a length greater than ω, where ω is the distance the vehicle can travel on a full battery. The vehicle must make no more than p battery exchanges, where $p > 0$ (when $p = 0$ this is the trivial shortest path problem). We use the general term route to refer to the movement of an electric vehicle on the network that includes a set of refueling stops along the electric vehicle walk. While there are many different criteria that could be used, initially consider the objective being optimized is the minimization of travel distance of the route, i.e. the length of the walk the electric vehicle route takes.

Let $G = (V, E)$ be an undirected network with node set V and edge set E, and let $n = |V|$ and $m = |E|$ be the number of vertices and edges respectively. Let vertices $s, t \in V$represent the starting and ending points of a trip by the electric vehicle. Let d_{ij} denote the length each edge $(i, j) \in E$ and let $B \subseteq V$ be the given set of charging locations. We define the electric vehicle shortest walk problem (EV-SWP) as the problem of finding the shortest walk in G starting at s and ending at t such that any walk contained in the path starting and ending at nodes in $\{s, t\} \cup B$ has length at most ω and the vehicle stops to recharge at most p times. In the unconstrained-stops version of the problem as many refueling stops make be taken as necessary ($p = \infty$).

The EV-SWP can be formulated as a linear integer program. In our solution, the electric vehicle will likely make several trips between different refueling stations. Let these walks between $\{s, t\} \cup B$ be called subtrips. The vehicle will make at most $p + 1$ of these subtrips, since the vehicle can only stop at p refueling stations. Define the decision variable x_{ijk} for $(i, j) \in E$ and $k \in \{1, \dots, p + 1\}$ as whether or not the electric vehicle will travel from i to j during its kth subtrip. Let y_{ik} for $i \in B \cup \{t\}$ and $k \in \{1, \dots, p\}$ represent the decision for whether or not the vehicle stops at station i after subtrip k. The index i can also take the value of t to represent the vehicle reaching the destination without having to exchange batteries the full p times and so it artificially exchanges at the end point. The integer program is now the following:

$$\text{Minimize } \sum_{k=1}^{p+1} \sum_{i,j:(i,j)\in E} d_{ij} x_{ijk} \tag{1}$$

$$\text{s.t. } \sum_{j:(j,i)\in E} x_{jik} - \sum_{j:(i,j)\in E} x_{ijk} = 0 \quad \forall i \in V \setminus (B \cup \{s,t\}),\ k = 1, \dots, p+1 \tag{2}$$

$$\sum_{j:(j,i)\in E} x_{jik} + y_{ik-1} - \sum_{j:(i,j)\in E} x_{ijk} - y_{ik} = 0 \quad \forall i \in B \cup \{t\},\ k = 2, \dots, p \tag{3}$$

$$\sum_{j:(j,i)\in E} x_{ji1} - \sum_{j:(ij)\in E} x_{ij1} - y_{i1} = 0 \quad \forall i \in B \cup \{t\} \tag{4}$$

$$\sum_{j:(i,j)\in E} x_{ijp+1} + y_{ip} - \sum_{j:(j,i)\in E} x_{jip+1} = 0 \quad \forall i \in B \tag{5}$$

$$\sum_{j:(s,j)\in E} x_{sjk} - \sum_{j:(j,s)\in E} x_{jsk} = 0 \quad k = 2, \dots, p+1 \tag{6}$$

$$\sum_{j:(s,j)\in E} x_{sj1} - \sum_{j:(j,s)\in E} x_{js1} = 1 \tag{7}$$

$$\sum_{j:(j,t)\in E} x_{jtp+1} + y_{tp} - \sum_{j:(t,j)\in E} x_{tjp+1} = 1 \tag{8}$$

$$\sum_{i\in R\cup\{t\}} y_{ik} = 1 \quad k = 1, \dots, p \tag{9}$$

$$\sum_{i\in V} \sum_{j:(i,j)\in E} d_{ij} x_{ijk} \le \omega \quad k = 1, \dots, p+1 \tag{10}$$

$$x_{ijk} \in \{0,1\} \quad \forall i,j \in V: (i,j) \in E, k = 1, \dots, p+1 \tag{11}$$

$$y_{ik} \in \{0,1\} \quad \forall i \in B \cup \{t\}, k = 1, \dots, p \tag{12}$$

Constraints (2) ensure the conservation laws hold for each vertex not in $B \cup \{s,t\}$. Constraints (3) through (6) ensure that conservation laws hold for the vertices in $B \cup \{s,t\}$, since special care is need to ensure that when a battery is exchanged the next subtrip starts. Constraints (7) and (8) ensure that the vehicle starts at origin s and ends at destination t. Constraint (9) ensures that between each subtrip exactly one battery exchange station (or t) is visited. Constraint (10) ensures that the vehicle can indeed traverse each subtrip without running out of battery power. Because the problem is formulated as a binary integer program here, using an off-the-shelf optimization program to solve it will likely take non-polynomial time.

If there is no limit on the number of stops (i.e. the vehicle can make up to $|B|$ stops) and the only concern is to minimize total distance, then it can be shown that the problem can be easily solved in polynomial time using the standard shortest path labeling algorithm (Ahuja et al., 1993), albeit up to $|B|$ times. We will first describe the algorithm with an illustration before we analyze it. Suppose we wish to travel from vertex s to vertex t in the network of Fig. 1. The triangular nodes in the figure are refueling stations.

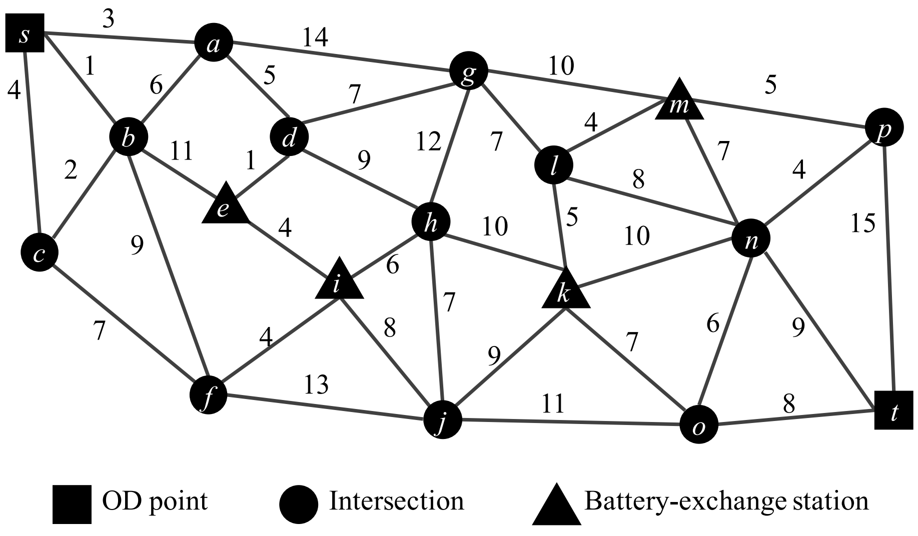


Fig. 1. The example road network.

When the range ω is large, say greater than 50, then the shortest path from s to t can be found using a shortest path algorithm such as Dijkstra's (Ahuja et al., 1993); the bold path shown in Fig. 1 is the shortest path of length 39. Note that this path does not pass any refueling points.

If the range was 20 then the vehicle will have to refill at least once to reach vertex t. The shortest path tree from s to all reachable nodes within distance $\omega = 20$ is shown in Fig. 2. As shown in the figure, two stations are reachable from s, the station at vertex e and the station at vertex i.

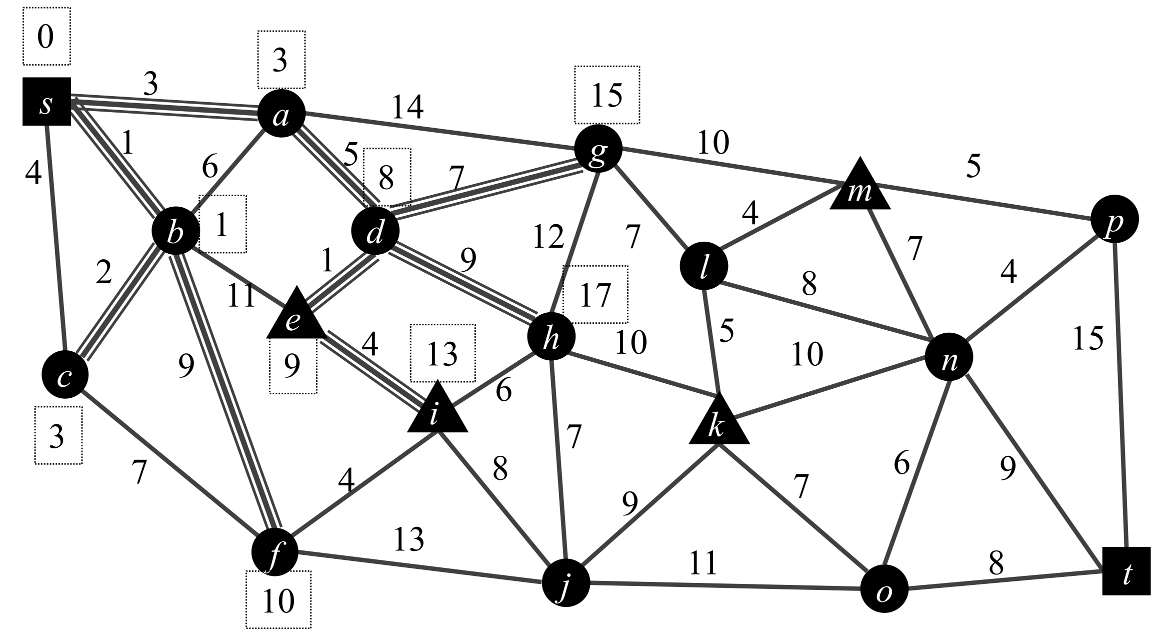


Fig. 2. The example network with range-limited shortest path tree from s and the distances to reachable nodes from s.

We then can do the same with these two stations as the starting points. Fig. 3 shows the range limited shortest path tree starting at the green station. Now from the station e we can reach three stations, i, k and, m at distances 4, 20, and 18 as shown in the boxed labels of Fig. 3.

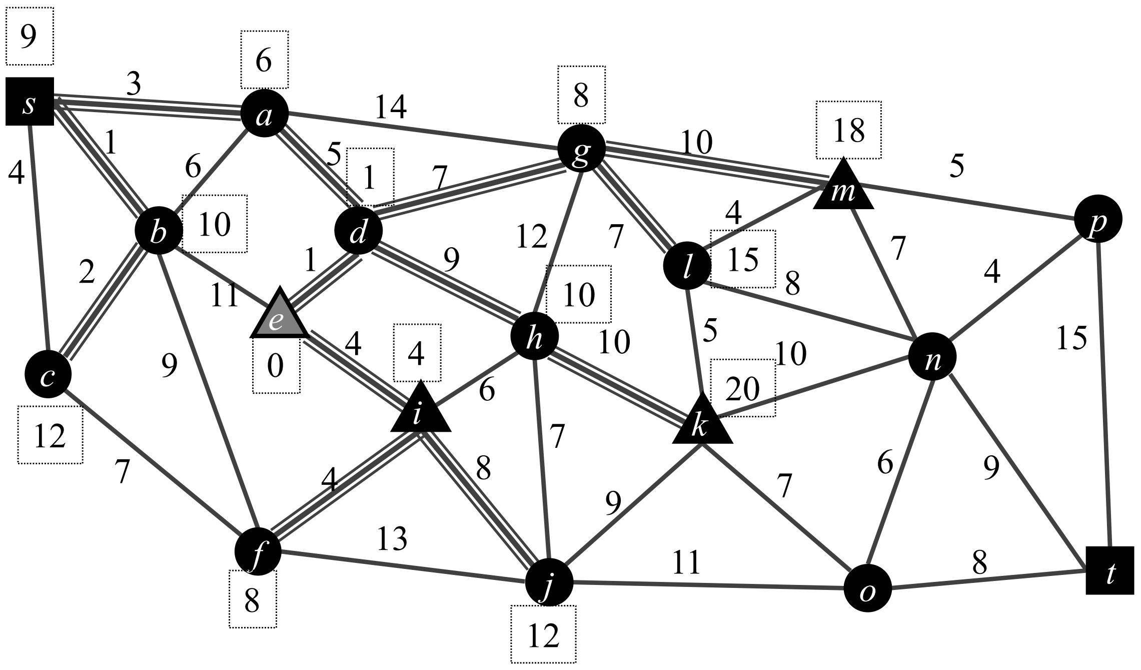


Fig. 3. The example network with range-limited shortest path tree from e and distances to nodes reachable from e.

We can repeat this for each of the reachable stations and will come up with the following network of range-limited shortest paths between stations, origin and destination, where each of the edges correspond to a path in a range-limited shortest path tree. We refer to this undirected network as the refueling shortest path network (RSPN) denoted by $G' = (V', E')$, and let $n' = |V'|$ and $m' = |E'|$. Observe that RSPN can be obtained in, at most, $(|B| + 1)$ iterations of the shortest path algorithm: one iteration for the starting node and one for each of the stations. Fig. 4 shows the RSPN for the example.

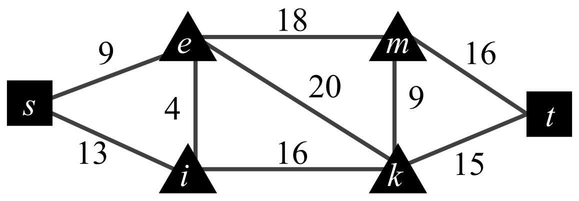


Fig. 4. Refueling Shortest Path Network (RSPN) for the example.

This corresponds to electric vehicle walk in the original network s-a-d-e(refuel)-d-g-m(refuel)-n-t. The paths s-e, e-m, and m-t are each subtrips in the solution. Note that the walk includes a cycle d-e-d and a detour g-m-n as compared to the shortest segment g-l-m in the shortest path from s to t (see Fig. 5).

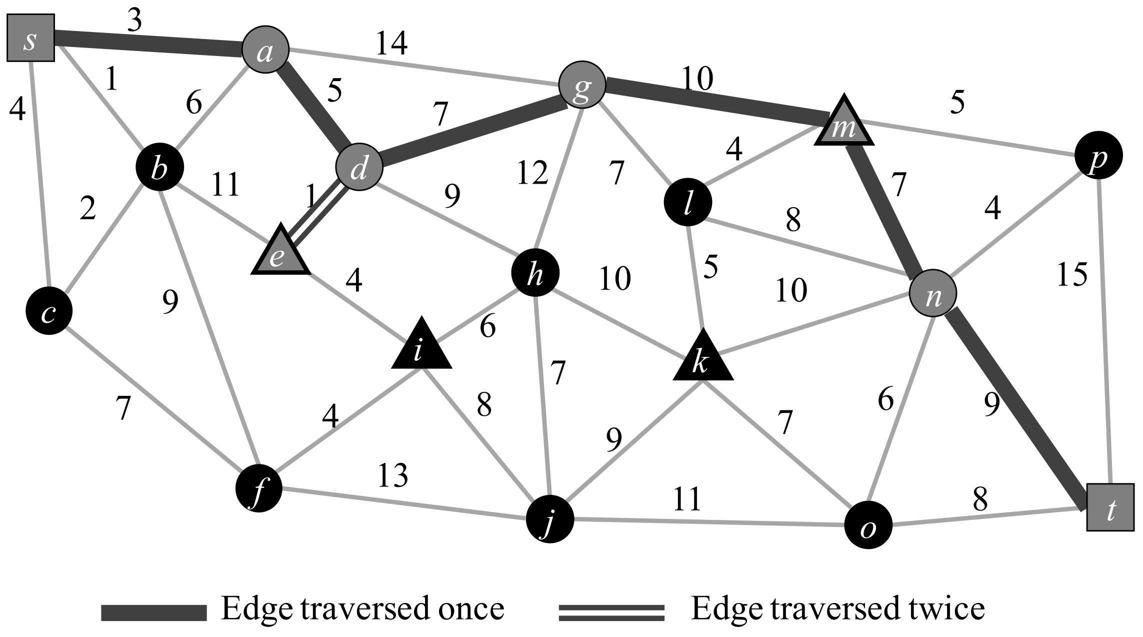


Fig. 5. The EV shortest walk for the example problem.

Theorem: The EV shortest walk problem can be solved in $O\big(|B|\,(n\log_2 n + m)\big)$ time.

Proof: First note that we solved $|B| + 1$ shortest path problems for starting node s and one for each station in set B. The best known bound for a shortest path algorithm is $n\log_2 n + m$ where n is the number of vertices and m the number of edges in the original network. Finally, there is the step that solves the problem on the refueling shortest path network which has complexity $O\big(|B|(\log_2|B| + m')\big)$ if solved by the best known algorithm; this is asymptotically dominated by $\mathcal{O}\big(|B|\,(n\log_2 n + m)\big)$. The theorem follows. ■

2.2. The restricted EV shortest walk problem

Up to now the number of stops to reach the destination has not been restricted. In the restricted case $p < |B|$ stops, we need to find the solution to a stop-limited walk in RSPN. That is, we need the solution to the shortest walk between vertices s and t that has at most $p + 1$ edges in RSPN. It is not immediately clear if this problem still polynomial time solvable. The structure of the problem allows us to develop a preprocessing polynomial network transformation to the classical shortest path problem which of course is polynomial solvable.

Notice that for a given graph G and $b_1, b_2 \in B$, there is a single shortest path to get between the two refueling stations b_1 and b_2 that does not depend on s and t. If we know that the vehicle is going to refuel at b_1 then refuel next at b_2, we do not need to know any other information to find the path the vehicle will take between these two refueling stations. If the length of the shortest path between b_1 and b_2 is greater than ω, then no feasible solution can have the vehicle refuel at b_1 then refuel next at b_2. So now we will create a directed graph where the vertices represent the refueling stations as well as s and t, and directed edges represent paths from stations and the start/end nodes to other stations and start/end nodes that are reachable with fuel w. If we are restricted to p refueling stops, then $p + 2$ copies of these directed arcs are needed. One may think of each network copy signifies the reachable nodes with a fully-charged battery (or with a full fuel tank). The RSPN for the example gives the multi-level network shown in Fig. 6

when we are restricted to a maximum of 2 refueling stops (edge lengths are removed for readability).

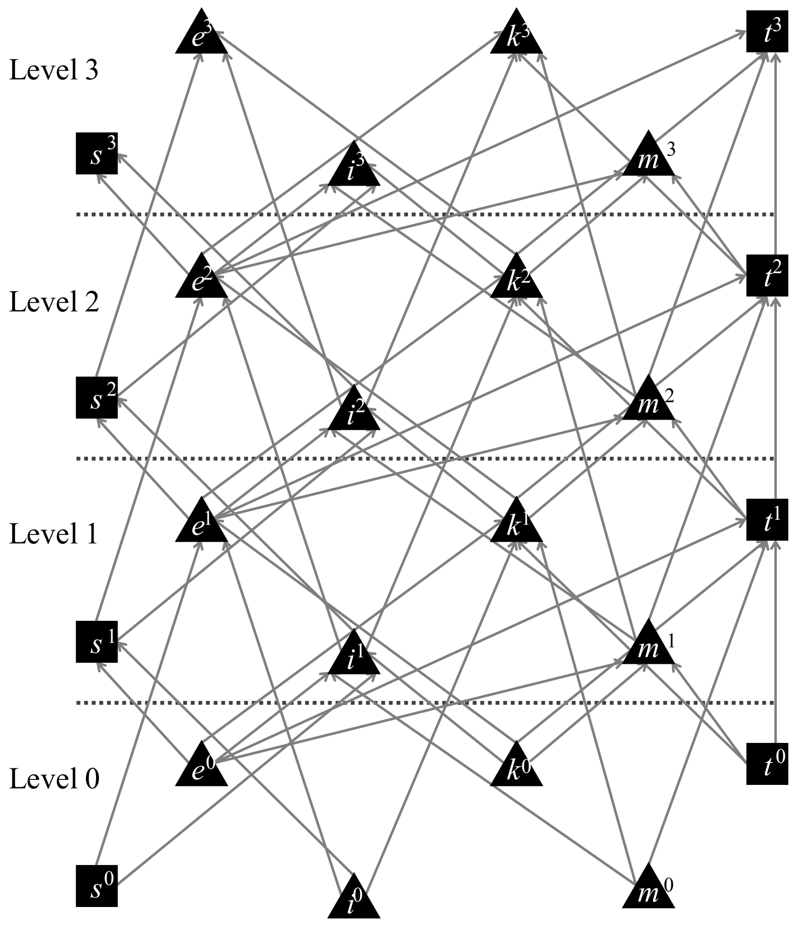


Fig. 6. Multilevel network when the number of refueling stops is restricted to a maximum of two.

We create a directed graph $G'' = (V'', E'')$ that is a transformation of RSPN, $G' = (V', E')$. The vertex set $V'' = \left\{x^{[i]}: x \in \{s, t\} \cup B, i \in \{0,1, \dots, p+1\}\right\}$ has $p+2$ copies of each vertex in $\{s, t\} \cup B$. We define the arc set as $E'' = E_1'' \cup E_2''$ where $E_1'' = \left\{\left(a^{[i]}, b^{[i+1]}\right): (a', b') \in E', i \in \{0,1, \dots, p\}\right\}$ and $E_2'' = \left\{\left(t^{[i]}, t^{[i+1]}\right): i \in \{0,1, \dots p\}\right\}$. The distance mapping is defined for arcs in E_1'' as $d''\left(a^{[i]}, b^{[i+1]}\right) = d'(a', b')$ and for arcs in

E_2'' as $d''\left(t^{[i]}, t^{[i+1]}\right) = 0$. Thus, the distances in this new graph between levels are the same as those between vertices in the RSPN, except for the addition of zero distance edges which allow the vehicle to go from any of the $t^{[i]}$ nodes to the $t^{[p+1]}$ node penalty free. This graph has the property that any path from $s^{[0]}$ to $t^{[p+1]}$ contains exactly $p + 1$ arcs, and corresponds to a path in G' that travels from s to t in at most $p + 1$ edges. Thus, to find the stop-limited shortest walk, we need to find the shortest path in G'' from $s^{[0]}$ to $t^{[p+1]}$. The shortest path will contain exactly p intermediate nodes, and the number of refueling stations the vehicle will stop at corresponds to the number of intermediate nodes in the path until reaching the first termination node $t^{[i]}$. The bold edges in Fig. 6 show this shortest path for the example: s^0 to e^1 (then stopping to exchange batteries), e^1 to m^2 (again stopping to exchange batteries), and then to destination t^3, with the travel distance 43 as discovered earlier.

Theorem: The p-stops limited EV shortest walk problem can be solved in $O\left(p|B|\left(n \log_2 n + m\right)\right)$ time.

Proof: Let T_1 represent the time required to transform G' into G'', and let T_2 represent the time required to find the shortest path in G'' from $s^{[0]}$ to $t^{[p+1]}$. Note that $T_1 = O\left(p(n' + m')\right)$ since it creates $p + 1$ copies of V' and p copies of E'. Since $n' = |B| + 2$, this means $T_1 = O\left(p(|B| + m')\right)$. Also, by the construction of G'' note that $|V''| \leq (p + 2)(|B| + 2)$ and $|E''| \leq (p + 1)m'$. Thus finding the shortest path in G'' will take $T_2 = O\left(p(|B| \log_2 p|B| + m')\right)$ time.

The overall run time T of the p-stops limited EV walk algorithm therefore is from first finding G', the RSPN, whose complexity is $\mathrm{O}\big(|B|\,(n\log_2 n + m)\big)$, plus $T_1 = \mathcal{O}\big(p(|B| + m')\big)$ and plus $T_2 = \mathcal{O}\big(p(|B|\log_2 p|B| + m')\big)$ time. All terms are dominated by $\mathcal{O}\big(p|B|\,(n\log_2 n + m)\big)$. The theorem follows. The p-stops limited case has a higher complexity due to having to make $p + 2$ copies of the RSPN graph, which is not needed if the number of stops is unlimited. ■

2.3. Experimental results

We tested the algorithm on randomly generated data to determine effectiveness of the algorithm. When generating random networks, we wanted to ensure that the random networks reasonably resembled potential real world road networks. This required the network to be planar, and for vertices in the network to be positioned in $\mathbb{R}^2$. Further the edges between vertices had to relate to the distances between the vertices in $\mathbb{R}^2$. Thus, the random network data was generated in the following way. On input n, the nodes V such that $|V| = n$ were randomly selected from $\{1,2,\dots,100\}^2$, representing points in a discretized plane. The edges E were defined as the Delaunay triangulation of the points V, and for each edge $(v_i, v_j) \in E$, the length of the edge was set to be the L_1 (Manhattan) distance between v_i and v_j. This method of generating the edges was used since it would allow for the edges to be both planar and reasonably sized in relation to the node locations. The vehicle was set to have a distance capacity $\omega = 35$.

To determine which vertices should be the refueling stations, the following algorithm was implemented:

1. Initialize B as a random vertex $v \in V$.
2. For each $v \in V$, find the minimum distance $\delta(v)$ between v and any element of $B \setminus \{v\}$.
3. Let $\dot{v}$ be the vertex with the maximum distance. If $\delta(\dot{v}) \leq \omega$, then stop. Otherwise, let $\ddot{v} \in V \setminus B$ such that $\ddot{v}$ has the maximum distance of all vertices that are not already refueling stations. Add $\ddot{v}$ to B and repeat step 2.
4. Picking station locations in this manner has the advantage that all of the vertices in the network will be within battery distance of a refueling station. After B is determined, s and t are selected randomly from the vertices $V \setminus B$.

Once G, B, s and t are generated, the graph is checked to ensure there exists a feasible way for the vehicle to get from s to t using the refueling stations of B. If there is not, the randomly generated road network is thrown out. While this method generated data that had many of the properties of real-world road networks, it did not allow for direct control of the number of refueling stations and the number of edges. Fig. 7 shows an example random network.

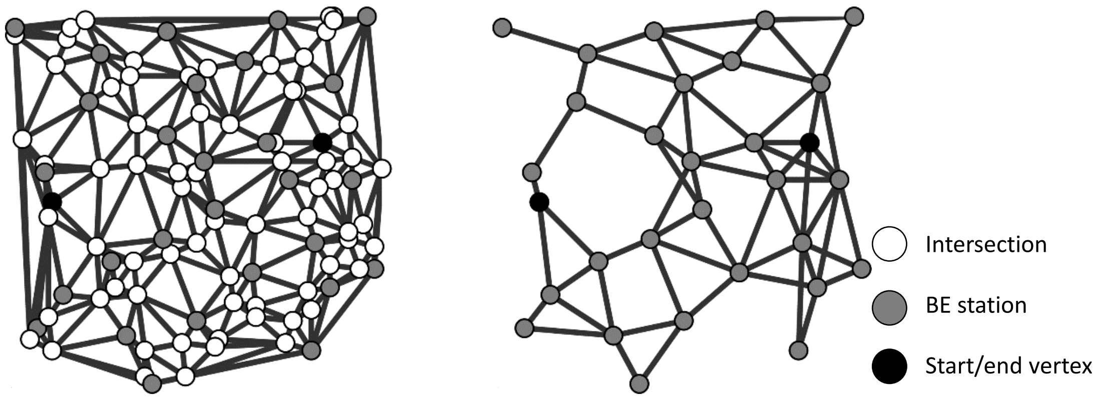


Fig. 7. An example randomly generated network and its corresponding RSPN.

We generated 1,000 random networks with $n = 100$, and compared three different shortest paths:

1. The path if there was no distance constraint, i.e. the shortest direct path between s and t.
2. The path if the vehicle had a distance constraint, but had no limit on the amount of times it could stop.
3. The path if the vehicle was allowed to make one less stop than that in path (2).

In the vast majority of casts (985) there was no feasible stop-restricted path, i.e. the shortest path with distance constraints already also had the least number of stops. Fig. 8 shows the runtime of the shortest walk algorithm without stop limit as a function of the number of refueling stations in the network. Each trial had a different number of edges, which explains some of the variance in the runtime. Overall, the runtime increases linearly as the number of charging stations increases, which is expected.

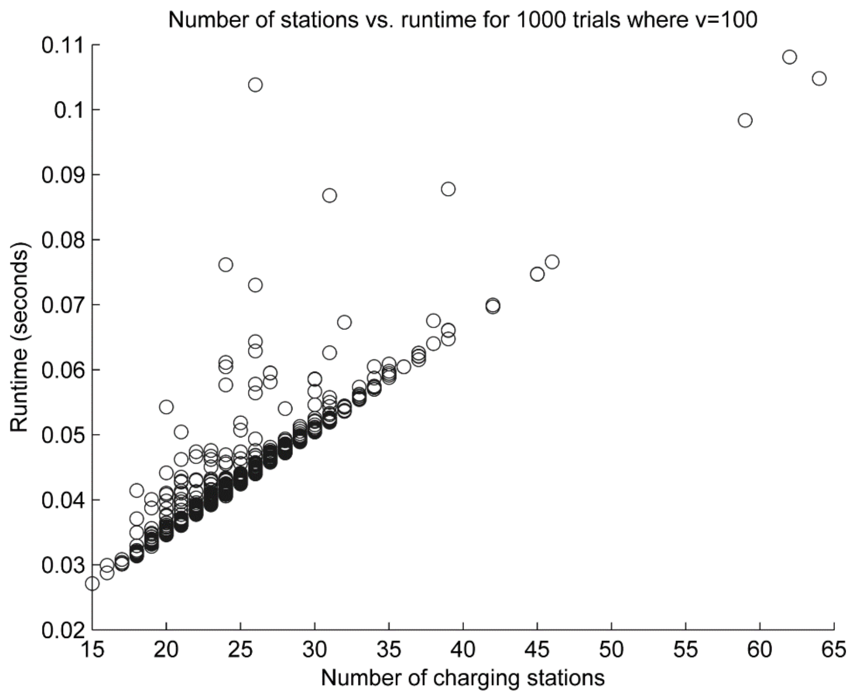


Fig. 8. The number of charging stations in the network versus the runtime of the algorithm.

For the stop-restricted algorithm, Fig. 9 shows the number of allowed stops versus the runtime of the algorithm with a fitted 3rd degree polynomial curve. This graph only includes generated instances where the number of allowed stops was at least 1. There does not appear to be a strong relationship between the number of stops versus the runtime. This is due to the fact that finding the RSPN takes substantially more time than finding the shortest path in the multi-level network.

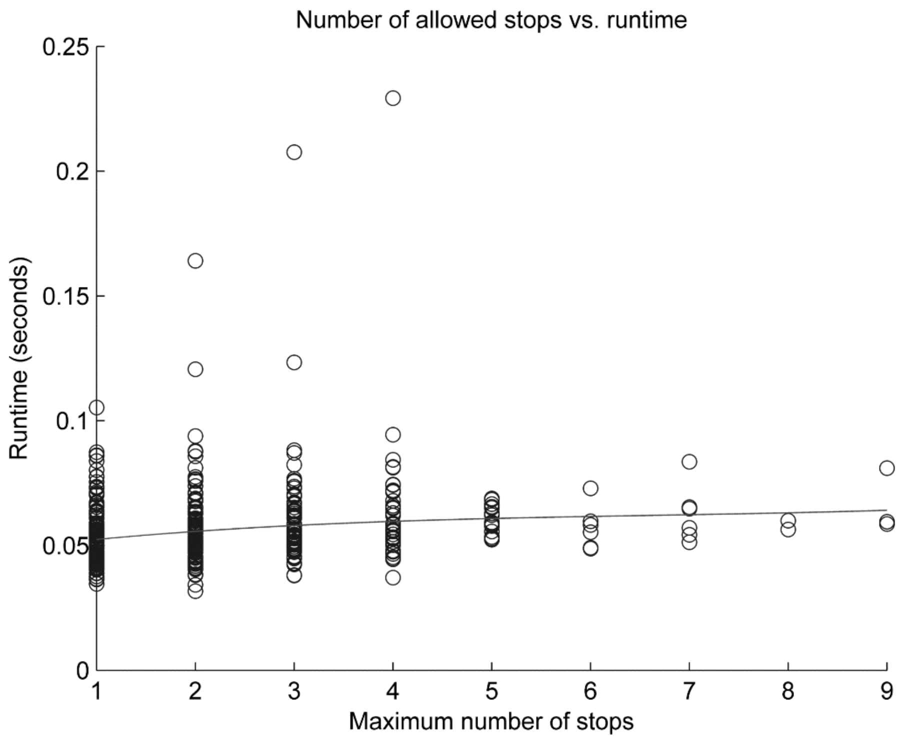


Fig. 9. The number stops allowed in the restricted algorithm versus the runtime of the algorithm.

Fig. 10 shows the length of the fuel constrained shortest walk compared to the unconstrained shortest path between the starting and ending nodes. In most cases the increase in route length due to detouring is less than 10%, however it can be over twice as long due to unfortunate refueling station placement.

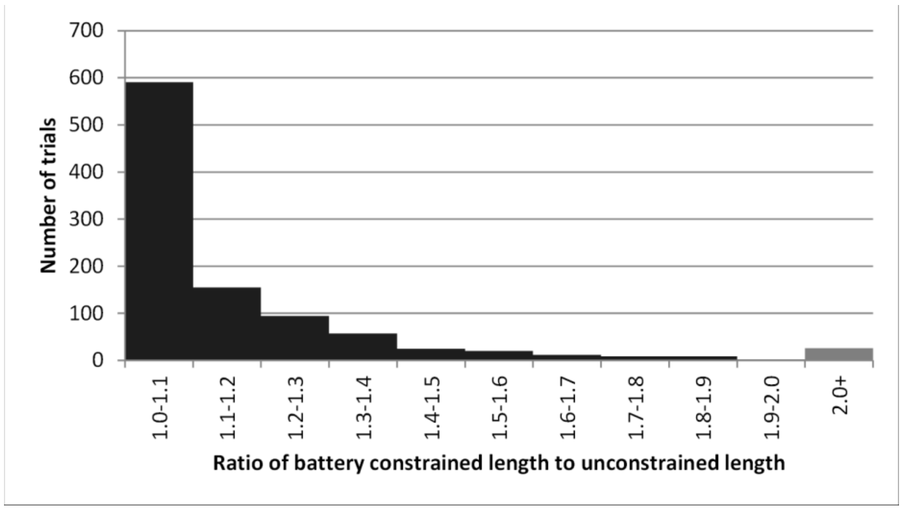


Fig. 10. A comparison of path lengths between the unconstrained shortest path and the EV shortest walk.

We also compared the solutions from our algorithm to the solutions generated by using CPLEX to solve the integer programming formulation in equations (1)-(12). We took each of the 1,000 randomly generated networks used earlier and solved them using the IBM ILOG CPLEX IDE. We found that the CPLEX solver gave the same solution for each of the problems, typically in a matter of seconds, provided a solution existed. In the event that there was no feasible solution the algorithm did not seem to halt, even after twenty minutes. Thus to ensure a solution existed, for each of the 1,000 problems we set p to be either the number of stops used in the unconstrained-stops version of the problem, or 1 if the solution to the unconstrained-stops problem was a route without any battery-exchanges. A comparison of the runtimes of the integer program on the 1,000 problems is shown in Fig. 11. While the vast majority of the problems ran in under 5 seconds, several of the sample problems took well over 20 seconds and four had to be cut off after running for 10 minutes. This is compared to our algorithm in MATLAB which consistently ran under a tenth of a second. Table 1 shows the runtimes and route lengths for the EV-SWP algorithm in MATLAB compared to the CPLEX implementation (when it successfully completed) and the unconstrained shortest path algorithm solved in MATLAB.

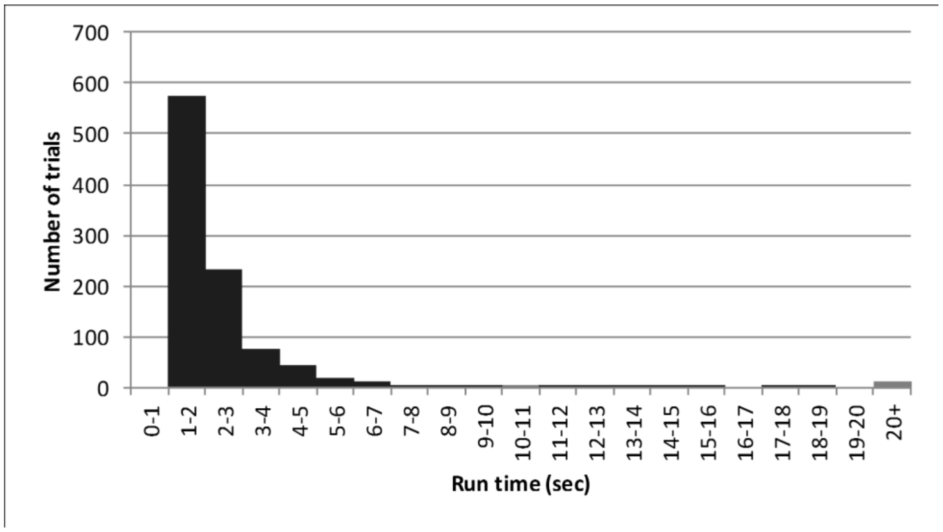


Fig. 11. A histogram of the runtimes for the CPLEX formulation for the EV-SWP.

Table 1
A comparison of the runtimes for the EV-SWP in MATLAB and the formulation solved in CPLEX, along with the runtimes for unconstrained shortest path algorithm in MATLAB

Measure	Runtime (seconds)			Route length	
	EV-SWP (MATLAB)	EV-SWP (CPLEX)	Shortest path (MATLAB)	EV-SWP	Unconstrained
Mean	0.0599	3.5755	0.0004	79.403	65.889
Median	0.0567	1.857	0.0004	75.500	65.500
95th percentile	0.0641	5.7682	0.0005	162.000	118.000

2.4. Special cases

The algorithm can be adjusted to handle the case that the vehicle begins the trip with only ω_L fuel, where $\omega_L < \omega$. The only change required is to instead of using a given distance function d, use a distance function d_L defined as

$$d_L(e) = \begin{cases} d(e) + (\omega - \omega_L) & e \text{ adjacent to } s \\ d(e) & \text{otherwise} \end{cases}$$

Using this distance function will cause the vehicle to artificially travel a less than distance $\omega - \omega_L$ after the trip begins, which is equivalent to starting with only ω_L fuel.

The algorithm can also be altered for the case when the vehicle needs to go from s to t and back to s. Given $G = (V, E)$, s, t, B, d, ω, p, create a new undirected instance $G^* = (V^*, E^*)$, s^*, t^*, B^*, d^*, ω^*, and p^* where:

$$V^* = \{v_i : v \in V, i \in \{1,2\}\}$$

$$E^* = \{(a_1, b_1) : a, b \in V, (a, b) \in E\} \cup \{(a_2, b_2) : a, b \in V, (a, b) \in E\} \cup \{(t_1, t_2)\}$$

$$d^*(a_i, b_i) = d(a, b) \quad \forall a, b \in V, (a, b) \in E, i = \{1,2\}$$

$$d^*(t_1, t_2) = 0$$

$$s^* = s_1, \quad t^* = s_2, \quad B^* = \{b_i : b \in B, i \in \{1,2\}\}, \quad \omega^* = \omega, \quad p^* = p.$$

This new graph G^* is two copies of the original graph G, where the vertices and edges are labelled by which copy they are in. The starting point is the vertex s in the first copy and the ending point is the vertex s in the second copy. The graph also has a distance zero edge between t_1 and t_2 (the vertex t in the two copies) which forces the vehicle to travel through the point t_1, the original destination, on its way from s_1 to s_2. After the shortest walk is found, the $\{1,2\}$ labels can be removed and the shortest route corresponds to the optimal solution in the original graph. The algorithm for going to and from a destination is the same as the original problem on a graph that is twice as large.

2.5. Routes that minimize anxiety

We now define a related problem shortest route problem that considers driver's anxiety. Here we define the anxiety level of the driver as a monotonically increasing function of the charge (fuel) used from the battery from full; the lower the charge (fuel) the higher the anxiety. We assume that like in previous sections the level of charge in the battery decreases as the vehicle travels, until it reaches a refueling station. Hence, the level of anxiety monotonically increases with distance traveled since fully charged. Every time the vehicle recharges the driver's anxiety drops to its minimum. Fig. 12 gives a typical trajectory of anxiety as driver travels from s to t.

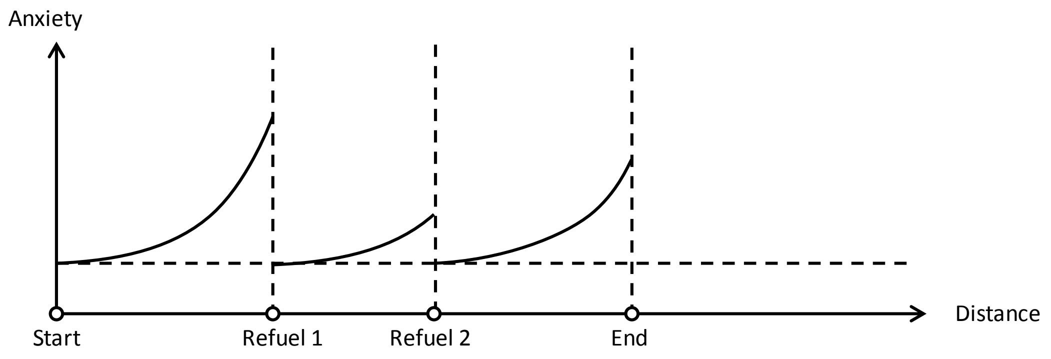


Fig. 12. Typical anxiety trajectory for a trip. Here the highest anxiety is just before first refuel.

Notice that since anxiety is monotonic the driver's anxiety is lowest when the vehicle is fully charged and reaches the highest point when the vehicle has traveled furthest after refueling. Therefore the problem of minimizing the maximum anxiety is simply that of finding a walk from s to t with the minimum maximum edge in the RSPN. Because the RPSN is an undirected graph, finding a minimum spanning tree (MST) in the RSPN gives all the min-max paths in RSPN, in particular the min-max path from s to t (Ahuja et al., 1993); the best complexity of finding MST is simply $O(m' \log_2 n')$ using Kruskal's algorithm. As an example, the MST for the example RSPN is given in the bold lines in Fig. 13.

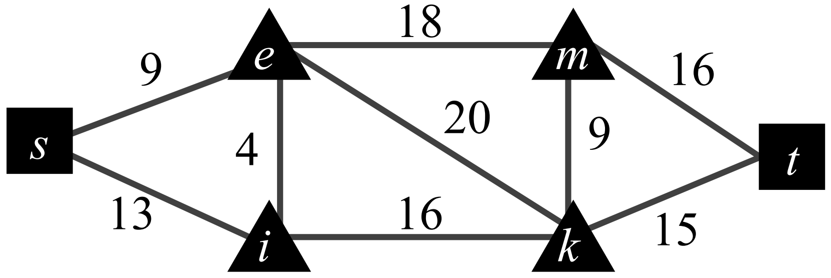


Fig. 13. MST of the example RSPN.

Now the min-max anxiety path is s-a-d-e(refuel)-i(refuel)-h-k(refuel)-o-t with the total travel distance of 44 whereas the walk that minimized distance had distance length of 43. Notice that this route has three refueling stops with the maximum anxiety corresponding to the arc of length 16 from station i to station k. The overall complexity for the unconstrained-stop case, including the time for building the RSPN, is $O(|B|(n \log_2 n + m))$ since finding the minimum spanning tree is dominated by complexity in creating the RSPN.

If we were restricted to at most p stops on the s-t route, then the same solution approach will apply the the problem as to that of the restricted case for the shortest walk problem. Again create a $(p+2)$-level directed RSPN network using the same procedure as Section

3 and look for a min-max directed path from s to t in this network. Such a problem was referred to as the bottleneck shortest path by Kaibal and Peinhardt (Kaibel and Peinhardt, 2006), where the bottleneck corresponds to the largest arc in the path. Such a path can be found by modifying Dijkstra's algorithm so that the temporary labeling updates to next node j from node i uses the operation:

$$\text{New temporary label on node } j = \max(\text{permanent label on } i, \text{new arc length } d_{ij})$$

Hence, the overall complexity is the same as for the shortest walk case: $O\big(p|B|(n\log_2 n + m)\big)$. Kaibal and Peinhardt propose an algorithm for the bottleneck shortest path problem which runs in $O(m' \log\log m')$ time, and if implemented would give a complexity of $O(|B|(n\log_2 n + m) + m' \log\log m')$, which may or may not be lower depending on the graph. Returning to the example, if we were restricted to maximum of 2 stops, then the solution min-max path is s^0-9^1-11^2-t^3 which also has a max arc length of 16 and a distance of 44 units.

This approach to finding the min-max anxiety can also handle the special cases from Section 5. In the case where the trip begins with partially charged battery, simply add a dummy arc (s_o, s) of distance corresponding to charge $\omega - \omega_L$ and repeat above procedure to find min-max anxiety path with or without stop-restrictions. To find the shortest route from s to t and back, generate a new graph which has two copies of the original graph using the procedure in Section 5, then run the min-max anxiety path algorithm on it.

2.6. Conclusion

Although electric vehicles have been around for a while, they have neither been widely accepted by commuters nor by organizations with service fleets. The benefits for owning EVs are many, which include little or no emissions, energy efficient, less dependent on non-renewable resources and rechargeable at homes and offices. It is predominately the “range anxiety” that discourages people and organizations from owning EVs, while electric-gasoline hybrid vehicles are becoming increasingly attractive since there is no range anxiety; because when batteries lose their charge vehicles can use back-up gasoline power. Unlike gasoline refueling stations which are practically everywhere, battery exchange or quick-charging facilities are hardly anywhere. Hence, establishing and operating a battery exchange (or recharging) infrastructure is essential for EVs to have a larger market share. The design of such an infrastructure requires one to minimize the detouring necessary for battery recharging through proper vehicle routing, and requires minimize waiting times at the stations to pick up recharged batteries by locating and sizing battery exchange network. This chapter addressed the first problem of routing to minimize detouring; the research team is conducting further research on sizing of network of facilities to minimize wait times.

For minimizing detouring, the EV shortest walk problem was defined to determine the route from a starting point to a destination; this route may include cycles for detouring to recharge batteries. Two such problem scenarios were studied: one is the problem of traveling from an origin to a destination to minimize the travel distance when one or more battery recharge/exchange stops may be made; the other is to travel from origin to destination when a maximum of p stops can be made. It was shown that both of these problems are polynomially solvable. The first problem requires several runs of a shortest path algorithm for determining shortest path trees totaling $O\big(|B|\,(n \log_2 n + m)\big)$ elementary operations, where $|B|$, n, and m are the number of located stations, number of nodes in the network, and number of arcs, respectively. The second problem requires an additional network transformation and also several shortest path trees, totaling $O\big(p|B|\,(n \log_2 n + m)\big)$ operations. The algorithms were tested on randomly generated data, and it was shown that they are quick and efficient to run. Other cases of routing with battery-exchange stops were analyzed, specifically the case when the vehicle starts partially charged, the case when the vehicle needs to make a round trip, and the case where the goal is to minimize driver anxiety.

One large area of future work is to adjust the problem to make it stochastic instead of deterministic. This could involve having the edge lengths be random variables as well as having a random variable for the distance the vehicle can travel before being stranded. We discuss this problem further in Section 5.1.

Chapter 3

ONLINE ROUTING AND BATTERY RESERVATIONS FOR ELECTRIC VEHICLES IN A NETWORK WITH BATTERY SWAPPING

One possible method for powering electric vehicles that drive long distances is to have the vehicles swap their batteries at battery-exchange stations on the route. These stations would be run by a company with central control, and would charge the vehicles to have a fresh battery replace their empty ones. Since the construction of battery-exchange stations and their infrastructure is very expensive, they have only been placed in a limited number of locations so far. In addition, the extra batteries that are stored in the station are also expensive, so to keep inventory costs low it is best to stock the stations with as few batteries as required. The number of batteries needed by vehicles visiting the station throughout the day depends on the location of the station, the day of the week, the weather, and many other variables. Further, the manner in which the vehicles arrive affects the number of batteries needed; if the vehicles all tend to arrive around the same time then more batteries are needed since there will not be enough time for them to be recharged after being dropped off. Since the exact distribution of cars that will pass through a station is unknown when the stations are constructed, each station has to be designed to handle near the maximum demand that station is expected to receive at any given time. In the event that no full batteries are available at a station, an arriving vehicle will have to wait until a battery is charged before it can leave the station. If a driver of a vehicle is informed in advance that there will be no available batteries at a station they planned to stop at, then that station could be avoided by taking a circuitous route involving a station having charged batteries. The decision on how many batteries to place at a station has to balance the company's desire to minimize inventory costs with need for drivers to not have to wait when there are no batteries available. It is possible that by balancing the vehicles needing battery swaps across many stations, fewer vehicles would arrive at the stations when no batteries are available which would improve the service they receive.

The goal of this chapter is to devise an algorithm for real-time routing of electric vehicles with swappable batteries that balances the desire for drivers to have quick trips with the need for the operating company to balance the battery swap loads across the stations. Further, the algorithm will make reservations for each vehicle at all of the battery-exchange stations on the selected route. Making reservations will remove the possibility that the batteries the vehicle expected to receive are unavailable due to other vehicles taking them. The objective of the routing and reservation algorithm is to minimize the total expected travel times of not only the particular vehicle being routed at each decision, but of future vehicles as well. Because of this objective, part of the routing and reservation process is to understand how a set of battery reservations could affect future arrivals into the system.

This chapter assumes that there is a viable business model for the development and operation of a network of battery-exchange systems. Thus, for our purposes we assume that the stations have already been located and built to hold a set number of batteries that are constantly being charged. In practice, an algorithm like this would require an onboard computer unit in each vehicle that communicates with a central server that does the routing and reservations. When a fleet of cars is produced to be compatible with a set of battery-exchange stations, the operating company of the stations will likely be involved in the design and production of the vehicles. In the case of the company Better Place, they collaborated with Renault Fluence Z.E. so that their vehicles could be used in the network (Kershner, 2013). Because of this involvement, the operating company can ensure that each vehicle has a compatible computer unit installed that communicates with the operating company's central server. The battery-swap operating company could implement such a routing and reservation software system that provides to each vehicle a route from its origin to its destination, along with where to stop to swap the battery so the vehicle does not run out of charge; the system would also make reservations for battery swaps at the stations where the vehicle intends to stop.

The routing and reservation system would make the route suggestion based on the current battery charge levels at each station along with the pre-existing reservations made by earlier vehicles, which would be stored in the central server. The model in this chapter assumes that when the vehicle turns on and requests a route to a destination, it will be provided with a single route from the routing and reservation system, and that the driver will take the given route exactly. The envisioned steps in this routing and reservation process for each vehicle are:

1. When the electric vehicle is turned on, the driver would input a destination into the vehicle's system unit. This, combined with the origin of the trip determined by the GPS location of the vehicle, would be sent to the central server.

2. The central server receives this origin and destination (OD) pair and, using the current battery levels at the stations and the reservations already made, determines which route the vehicle should take and when and where the vehicle should stop to swap its battery.
3. The central server makes reservations for the batteries at each of the stations for the most convenient times the vehicle would require it, subject to availability.
4. The central server sends the selected route and reservation times to the unit onboard the vehicle, and the driver begins to travel the route.

We assume that since the batteries are reserved the vehicles will always find them available precisely when they were reserved for. We do not allow for any randomness in the amount of time it takes to traverse a road, nor that the driver of a vehicle can change its route partway through its route. Thus the assumed road network and driver behavior are fully deterministic, although the arrivals of vehicles into the system are stochastic. Since many different vehicles will be routed, this is an online system that is fully aware of all of the routes taken by the vehicles that have previously arrived in the system. Although future routing and reservation decisions are not known at the time a new route is given, the rate of demand for each origin and destination pair is known and assumed to be fixed over time.

In this research we assume that the goal is to find routes for vehicles that take the least amount of time (as opposed to other metrics such as distance travelled). The amount of time it takes to travel a route is dependent on three components:

1. the time spent by the vehicle driving along roads,

2. the time spent having the battery swapped at each station, and
3. the time spent having the vehicle wait at a station for a battery to become available.

Although the summation of these times can be found directly to compute the travel time, it is also possible that some of these time components are considered worse for drivers than others and a weighted sum is more appropriate. For instance, waiting at a station for a battery to become available could be more frustrating than driving, so for example every minute waiting for a battery could be worth two minutes of driving time. We refer to the weighted sum of the travel time as the *value* of the route.

The simplest routing and reservation software could be designed using a *greedy algorithm* which would find the best route for each particular driver, using for example the shortest walk algorithm presented in Chapter 2. It may however be in the best interest of the company to suggest that some vehicles take slightly longer routes so that other vehicles have much shorter routes. In this case the company is trying to minimize the average route values across all vehicles, rather than trying to sequentially minimize each trip individually. For example consider simple the road network in Fig. 14 where the edge lengths denote travel times. In this network the electric vehicles have a range of 12 time units, and battery swapping is instantaneous. Here there are two OD pairs for vehicles to take. For vehicles driving from origin 1 to destination 1 there is a single route of value 20 that stops at station 1. For vehicles driving from origin 2 and destination 2 there are two routes: one stopping at station 1 which has a value of 20 and one stopping at station 2 which has a value of 22. Also assume that station 1 has a single battery on hand, while station 2 has many batteries available for swaps.

Suppose a vehicle arrives wanting to travel from origin 2 to destination 2 and there is an available battery at both stations 1 and 2. The shortest route for the vehicle would be to stop at station 1, however that would require the vehicle to take the only battery at the station. If immediately after routing this vehicle a second vehicle arrived wanting to travel between origin 1 and destination 1, it would have no choice but to wait for the battery at station 1 to become available. Thus, the best global decision may be to have the first vehicle stop at station 2 taking the slightly longer path, which allows a future vehicle to get a battery at station 1 without having to wait. This problem of online routing of vehicles to minimize total route value is the focus of this chapter.

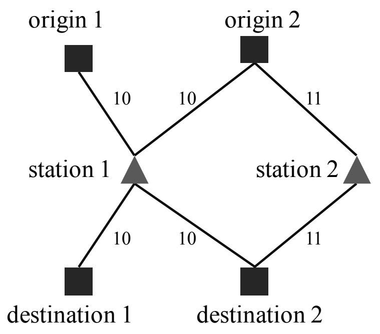


Fig. 14. An example route network with two origin and destination pairs.

We will model the system as a *Markov Chance-Decision Process* (MCDP) (Hentenryck et al., 2009) where the states describe the current reservations at the stations and the actions are routing the vehicles that arrive. An MCDP is a modification of the standard *Markov Decision Process* (MDP) (Russell and Norvig, 2009). In a MDP, an *agent* moves between *states* by taking *actions* which move the agent to each state with a specified probability. When the agent is in a state a *policy* determines an action for the agent to take. In an MDP the uncertainty occurs after an action is selected and is involved with determining which state the agent will end up in. In an MCDP, when the agent is at a state first a random outcome is selected. The outcomes determine which possible states the agent is allowed to transition to, and the agent may transition to any of them with certainty. Thus in an MCDP the uncertainty occurs before action is taken by randomly selecting which possible actions are available to the agent, and there is no uncertainty once the action is chosen.

In the case of routing electric vehicles through a network with reservations, during each interval of time it is unknown whether or not a vehicle will arrive needing to be routed, and if it does arrive what its OD pair will be. When a vehicle does arrive, its OD pair becomes known and the algorithm routes it precisely, and given the routing and reservations made the new state of the system is then exactly known. Then, in the next time step a new vehicle may arrive and request a route and reservations; this process is repeated until the end of the day. Note that the time periods are chosen to be small enough so that two or more vehicles do not arrive at the same time, as in a Poisson process.

The optimal policy for routing may be very complicated and depend on not only if batteries are available at the different stations, but how many batteries are available and when they are available. Finding the optimal policy for routing vehicles is intractable due to the large number of possible options for routing and reservations over the course of a day, and thus our best hope is to find an approximate solution. We provide a solution using the approximate dynamic programming technique of linear temporal differencing (Powell, 2011).

The problem of routing a single electric vehicle through a network where the vehicle has a limited range and must stop at a charging or battery-exchange station was first discussed by Ichimori et al. (Ichimori et al., 1981). The paper solves for the case where the amount of battery charge consumed by the vehicle is proportional to distance it travels. The work was extended to limit the number of stops in the previous chapter of this dissertation. In the case that the battery consumed along a stretch of road is not dependent on the distance, for example in the case when it also depends on the speed limit of the road, then the problem becomes NP-complete (Laporte and Pascoal, 2011; Smith et al., 2012). The problem of routing an electric vehicle has also been discussed for the situation where the vehicle can recharge on certain roads, like when it is going downhill (Sachenbacher et al., 2011). We note that when the station sizes are not limited by the number of available batteries, then each arriving electric vehicle will be routed using the shortest path methods discussed above since they will not need to compete for available batteries.

The problem of routing multiple electric vehicles to stations as they arrive has been discussed by de Weerdt et al. (de Weerdt et al., 2013) where they route the vehicles based on the forecasted future demand for the day as well as the current vehicles in the system. The algorithm they suggest still only finds the optimal route for each vehicle individually rather than trying to minimize the total route times of all the vehicles. The problem of placing and sizing the battery-exchange stations while simultaneously finding the routes the vehicles will take between OD pairs has been discussed by Mak et al. (Mak et al., 2013). That analysis assumes that the route between an OD pair is fixed after construction of the stations, while we allow for vehicle routes to change throughout the day as different stations run out of batteries. Approximate dynamic programming has been used to find when to charge the batteries at a battery-exchange station (Worley and Klabjan, 2011). Here vehicles arrive at a single battery-exchange station and the batteries are charged at different times to minimize the cost of electricity used.

The chapter is arranged as follows. The problem is formalized in Section 3.1 where the problem input and output are defined. In Section 3.2.1 we show how the problem can be transformed into a Markov chance-decision process. Since the MCDP painfully suffers from the curse of dimensionality, Section 3.2.2 shows how the problem can be approximated using temporal differencing of linear models. Finally, in Section 3.3 we illustrate the use of our algorithm to find the approximate solution for the Arizona road network assuming different amounts of demand.

3.1. Problem description

We will discretize time into small units of constant length. At the start of each time unit, there is a chance a vehicle will arrive wanting to travel from a particular origin to a particular destination. At each time period at most one vehicle could arrive, and the probability that a vehicle would arrive for any particular OD pair is constant throughout the day. For each possible origin and destination pair, there is a predetermined list of possible routes for the vehicle to take.

A vehicle is assigned a route before embarking, and when the route is selected each station that will be visited will have a battery reservation made at it. The reservations are made at the stations in the order that the stations will be visited, since the vehicle may have to wait at earlier stations for a charged battery and that should be taken into account when batteries are being reserved later in the route. When a vehicle arrives at a station it will wait until a battery is available for it to take. In addition, the vehicle cannot take a battery that is already reserved for another vehicle unless the battery this vehicle drops off will be fully charged and ready for pick up when the second vehicle arrives. This prevents battery reservations from being snatched away. If there is a battery that meets these criteria, the vehicle will take it as early as possible – we will never allow a vehicle to wait at a station when it could take a battery.

3.1.1. Problem input

The road network is represented as an undirected graph $G = (N, E)$ where N is a set of nodes representing intersections and E is a set of edges representing the roads in the network. Placed at certain predefined intersections in the network are β battery-exchange stations. Let $\{b_1, b_2, \dots, b_\beta\} = B \subseteq N$ be the intersections that have battery-exchange stations. Each station b_i has n_i batteries for $i = 1, \dots, \beta$. It is likely that stations placed in more heavily trafficked locations will have more batteries, so we do not assume that all of the stations have the same number of batteries. We are interested in routing many different vehicles that have different possible origins and destinations. Let $\{(o_1, d_1), (o_2, d_2), \dots, (o_\sigma, d_\sigma)\} \subseteq N \times N$ be the set of σ possible origin and destination pairs, so each pair represents one possible combination of a starting point and an ending point. Let $O = \{o_1, o_2, \dots, o_\sigma\}$ and $D = \{d_1, d_2, \dots, d_\sigma\}$. We assume that $o_j \neq d_j$ for any OD pair j, since in that case the vehicles would not travel at all. For an example network see Fig. 15 which contains three OD pairs. Notice that while in Fig. 15 each node is used in at most one OD pair, it is possible that a node can be in multiple OD pairs and it can also be the location of battery exchange station simultaneously.

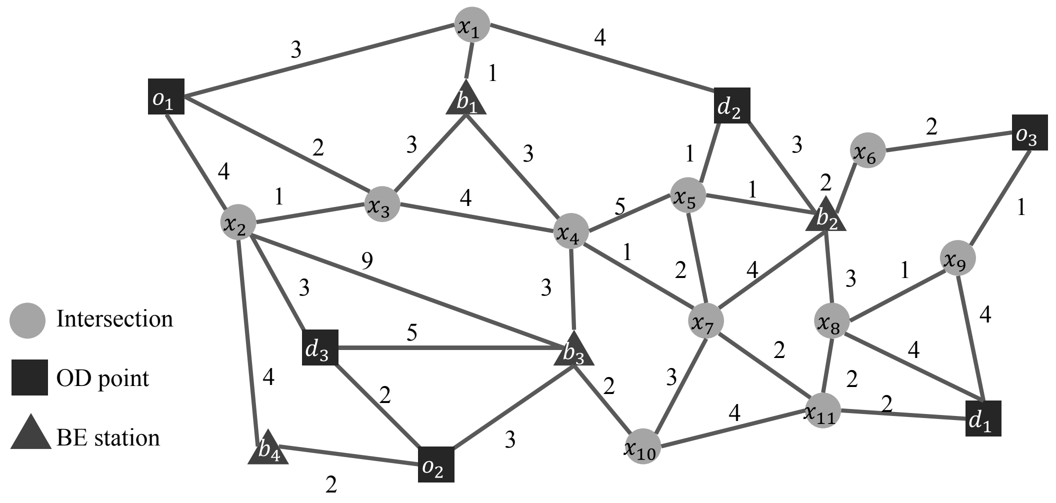


Fig. 15. An example network with OD pairs and battery exchange stations.

The time period in which the battery-exchange stations are open is split into $T + 1$ discrete units $\{0, \dots, T\}$. Associate with each edge $e \in E$ a nonnegative integer length $l(e)$ representing the number of time units it takes for a vehicle to traverse the road. Let $l'(v_1, v_2)$ for $v_1, v_2 \in N$ be a function that returns the minimum path length in G between two vertices, which can be computed using Dijkstra's algorithm. The battery capacity of the electric vehicles in the network is an integer ω representing the number of time units the vehicle can travel between battery swaps. Each vehicle starts at its origin with a full battery. For simplicity we assume that when a vehicle drops off a battery at a charging station it is fully depleted. The act of swapping the battery takes g time units, and when a battery is dropped off at a station it takes h time units to fill to fully charged. Only once the battery is full may it be swapped into a new vehicle. At the start of each time interval at most one vehicle can arrive in the system. Assume that no vehicle arrives into the system with probability p_0, and a vehicle arrives wanting to travel from o_j to d_j with probability p_j for each OD pair $j = 1, \dots, \sigma$. Since these probabilities cover all of the possible arrival cases, we have that $\sum_{j=0}^{\sigma} p_j = 1$. Also assume that each OD pair will require stopping to exchange batteries at least once on the way, otherwise they do not need to be considered in the model. Let $\rho_1 \geq 0$ be the penalty multiplier to swapping time incurred and let $\rho_2 \geq 0$ be the penalty multiplier to waiting time incurred. If a vehicle arrives at time t and assigned a route with ψ_t^{drive} time units driving, $\psi_{t,i}^{swap}$ time units swapping batteries at station i, and $\psi_{t,i}^{wait}$ time units waiting at station i, we say the route assignment has value $\psi_t = \psi_t^{drive} + \rho_1 \sum_{i=1}^{\beta} \psi_{t,i}^{swap} + \rho_2 \sum_{i=1}^{\beta} \psi_{t,i}^{wait}$.

For each OD pair j, let $\mathcal{R}_j$ be the set of routes that the vehicle is allowed to travel between o_j and d_j. Each route $R \in \mathcal{R}_j$ is a sequence $\left(b_1^R, b_2^R, \dots, b_{|R|}^R\right)$ of battery exchange stations that the vehicle will visit between the origin and destination in the order the stations will be visited. Thus the length of the shortest path between the stations in the sequence $l'(b_i^R, b_{i+1}^R)$ for $i = 1, \dots, |R| - 1$ must each be at most ω since the vehicle must be able to travel between the stations without running out of charge. Further, the shortest path between o_j and the first station in the sequence and the shortest path between d_j and the last station in the sequence must both have length at most ω. If a vehicle is assigned a route where the vehicle travels between two battery-exchange stops, then there is a single shortest path in G for the vehicle to take between those two stations. Thus the travel times between stations can be computed in advance and used for any routes that contain that pair of exchange-stations, and when making routing decisions the travel times between stations can be found by doing a table lookup. For example for the network in Fig. 15 with $w = 10$, for the OD pair (o_1, d_1) one possible set $\mathcal{R}_1$ is $\mathcal{R}_1 = \{(b_1, b_2), (b_3)\}$. In this case the first sequence represents taking the path $o_1 - x_1 - b_1$ with minimal length $l'(o_1, b_1) = 4$, swapping for the first time, followed by path $b_1 - x_1 - d_2 - b_2$ with minimal length $l'(b_1, b_2) = 8$, swapping for the second time, then getting to the destination by $b_2 - x_8 - d_1$ with minimal length $l'(b_2, d_1) = 7$.

The set $\mathcal{R}_j$ is a user input for each OD pair j and may be a subset of all feasible routes. The route sets may be restricted to avoid allowing routes that have too high values or too many stops. For example with the network in Fig. 15 another possible value for $\mathcal{R}_1$ is $\{(b_1, b_2), (b_3), (b_4, b_3)\}$, which has the additional sequence where the vehicle stops at station b_4 before visiting b_3. Since the sequence (b_4, b_3) is more circuitous than (b_3) which visits station b_3 directly, it may be best to exclude it. Let the single element set $\mathcal{R}_0 = \{\emptyset\}$ represent the possible routes to take when no vehicle arrives to be routed: since there are no decisions to make when no vehicles arrive, the only possible choice is to not visit any stations.

For each OD pair j and corresponding route set $\mathcal{R}_j$, let L_j be the minimal value of routes in set $\mathcal{R}_j$ assuming each station has an available battery at any time. For example assume that for the network in Fig. 15 we have that $g = 2$, $\rho_1 = 2$, and $\rho_2 = 4$. Using set

$\mathcal{R}_1$ defined above, the value $L_1 = 9 + 2 \cdot 2 + 8 = 21$, since the minimal time under the best conditions would be found using the second sequence (b_3) which has a shortest path of length 9, one swap taking 2 time units (weighted twice as heavily by ρ_1), then a final path of length 8. The value L_j can be thought of as the best case value for a vehicle to travel between OD pair (o_j, d_j). For a vehicle traveling OD pair j assigned at time t with a route value of ψ_t, the *delay penalty* $\psi_t - L_j$ is the increased value of the route that is assigned compared to the best case value.

3.1.2. Reservation policy

When a vehicle arrives into the system wanting to traverse OD pair j, it is immediately given a route $R \in \mathcal{R}_j$ to take from o_j to d_j. When the vehicle is assigned the route a battery reservation will be made at each of the battery-exchange stations it will visit. The reservation will state at what time the vehicle will be picking up the battery. A battery can only be reserved at a time when (1) there is a battery available and (2) the battery to be picked up will not be needed by any vehicle before the battery to be dropped off will be refilled. The second condition ensures that the reservation does not conflict with one already in place. For example, suppose that in the network in Fig. 15 both station b_1 and b_2 have a single battery and again $g = 2$ and $h = 6$. If a vehicle using pair (o_1, d_1) wants to reserve the battery at station b_1 at $t = 120$, then that battery must be available not only at $t = 120$ but also $t = 121,\ t = 122, \dots, t = 127$. Only at time $t = 128$ will the old battery of this vehicle be ready to be put into a new vehicle. Having the battery be available between $t = 120$ and $t = 127$ will ensure that no other vehicle can reserve the battery when it is being swapped or when it is charging. The reservation need not be made for the exact time the vehicle arrives, only the earliest available time after a vehicle arrives. That is to say the vehicle may reserve a battery for a later than when it would arrive at the station if taking a battery that is available earlier would violate a previous reservation; however we assume a vehicle will never voluntarily wait if there is a battery available that would not cause a conflict with another reservation. If a vehicle arrives at a station after time T, which for instance could occur if the vehicle turns on to be routed at time $t = T - 1$, assume there will be a battery immediately available at the station for it.

For a more in depth example consider the network in Fig. 15 and a vehicle travelling from o_1 to d_1 wanting to use route (b_1, b_2) and suppose the vehicle is being routed at time $t = 33$. Assume that the two stations each have a single battery, and station b_1 already has a reservation at time 32 and station b_2 has a reservation at time 52. At the time of the assignment the reservations for batteries of the vehicle being routed have to be made at the two stations. Leaving at $t = 33$, the vehicle will arrive at vertex b_1 at time 37. Since the battery at station b_1 is already reserved by another vehicle at time 32, it is therefore not available between times 32 through 39 (due to swapping and charging time of the battery being dropped off by the earlier reservation). Thus a reservation is made at time 40, and with this delay plus the 2 time periods to swap the battery, the vehicle will be done at station b_1 at the end of time period 41. Thus with the 7 time units it takes to travel between b_1 and b_2, the vehicle will arrive at station b_2 at the start of time period 48. The battery at station b_2 is full and available at time 48, but another vehicle already has a reservation at time 52. Thus a reservation cannot be made at time 48 since the battery would then be unavailable to the car arriving at time 52. Since it takes 8 time units to swap and charge, the reservation has to be made at time 60. Therefore the vehicle will leave the station at location b_2 at the start of time period 62 (since it will swap during periods 60 and 61) and will arrive at the destination at time 69. If the waiting penalty and swap penalty are 2 and 4 respectively, the value of this trip is:

$$\psi_{33} = \psi_{33}^{drive} + \rho_1 \sum_{i=1}^{\beta} \psi_{33,i}^{swap} + \rho_2 \sum_{i=1}^{\beta} \psi_{33,i}^{wait}$$

$$= 18 + 2 \cdot (2 + 2 + 0 + 0) + 4 \cdot (3 + 12 + 0 + 0) = 86.$$

This calculation can be done entirely before the route has begun to be traversed, and so once the vehicle sets out it is exactly known how long the trip will take including waiting at stations.

3.1.3. Problem output

The solution to the problem is a function that, upon arrival of a vehicle at time t wanting to travel between an OD pair j_t, takes the current battery levels of the stations and the already made reservations and outputs a route that minimizes

$$\mathbb{E}_t\left[\psi_t + \mathbb{E}_{t+1}\left[\psi_{t+1} + \mathbb{E}_{t+2}[\cdots + \mathbb{E}_T[\psi_T]]\right]\right].$$

Thus the algorithm makes decisions not only to minimize the value of the trip the vehicle is about to take, but also of the trips future vehicles may take. Notice that this is equivalent to minimizing the delay penalties, the differences between the trip values and the best case trip values, on each trip between t and T:

$$\mathbb{E}_t\left[\left(\psi_t - L_{j_t}\right) + \mathbb{E}_{t+1}\left[\left(\psi_{t+1} - L_{j_{t+1}}\right) + \mathbb{E}_{t+2}\left[\cdots + \mathbb{E}_T\left[\psi_T - L_{j_T}\right]\right]\right]\right].$$

We will use this equivalent objective function. Unfortunately finding the exact solution is intractable for all but the smallest cases, and so we will provide an approximate solution.

3.2. Solution theory

3.2.1. Markov chance-decision process

For a vehicle wanting to traverse between an OD pair there may be many different routes the vehicle should take depending on the station conditions. When a vehicle arrives all of the reservations that have already been made by other vehicles are known, so the actual routes the previous vehicles have taken are irrelevant. That is to say the best route for the vehicle to take only depends on the current reservations and battery levels at the station at the time of routing and not on previous decisions made. In this way the system is Markovian. Thus, we will model the system using a specialized Markov Decision Process (MDP) called a Markov Chance-Decision Problem (MCDP). While MCDPs and MDPs have the same expressive power, in certain situations the MCDP has easier computations. The MCDP model fits especially well into the framework of this problem since the chance and the decision have clear analogs to the vehicles being routed.

A Markov Chance-Decision Problem (MCDP) is formally defined as a tuple $(\mathcal{S}, S_0, \Xi, Z, C)$ where:

- $\mathcal{S}$ is a finite set of states;
- $S_0 \in \mathcal{S}$ is the initial state;

- $\Xi = (\xi_t)_{t\geq 0}$ is the input process. It is a Markov chain with a state set J, an initial probability distribution $\mu \in \mathrm{prob}(J)$ and a $J \times J$ transition matrix Q;
- $Y: \mathcal{S} \times J \to 2^{\mathcal{S}}$ is a transition mapping with $Y(S, j) \neq \emptyset$ for all $S \in \mathcal{S}$ and $j \in J$; and,
- $C: S \times J \times S \to \mathbb{R}$ is the cost map.

In a MCDP the agent starts at initial state S_0, and at the start of each time period t, the agent is in state S_t. At time t the Markov chain Ξ transitions from state $j_{t-1} \in J$ by one step to a new state $j_t \in J$ using matrix Q. The agent then has the choice to move to any state $S_{t+1} \in Y(S_t, j_t)$ and receives a cost $C(S_t, j_t, S_{t+1})$. A visual representation of the progression of an MCDP can be seen in Fig. 16. In this figure the agent starts in a state S_0 and then a random outcome j_0 occurs. This leads to a set of new possible states $Y(S_0, j_0)$ of which $S_1 \in Y(S_0, j_0)$ is chosen for the next state. This repeats indefinitely for the infinite version of an MCDP, and for the finite time version of a MCDP, T transitions occur. The initial state of the input process is randomly distributed by μ. A *policy* for a MCDP is a mapping $\pi: \mathcal{S} \times J \to \mathcal{S}$ such that $\pi(S, j) \in Y(S, j)$ for all $S \in \mathcal{S}$ and $j \in J$. For a finite MCDP the objective is to find a policy that minimizes the expected sum of the costs taken from transitioning from S_0 over the course of the T transitions. Formally our objective is to find the policy π that is the solution to:

$$\min_{\pi}\Bigg(\mathop{\mathbb{E}}_{j_1}\Bigg[\min_{S_1 \in \pi(S_0, j_1)}\Bigg(C(S_1, j_1, S_0) \\ + \mathop{\mathbb{E}}_{j_2}\Bigg[\min_{S_2 \in \pi(S_1, j_2)}\Bigg(C(S_2, j_2, S_1) + \cdots \mathop{\mathbb{E}}_{j_T}\Bigg[\min_{S_T \in \pi(S_{T-1}, j_T)}\big(C(S_T, j_T, S_{T-1})\big)\Bigg]\Bigg)\Bigg]\Bigg)\Bigg]\Bigg).$$

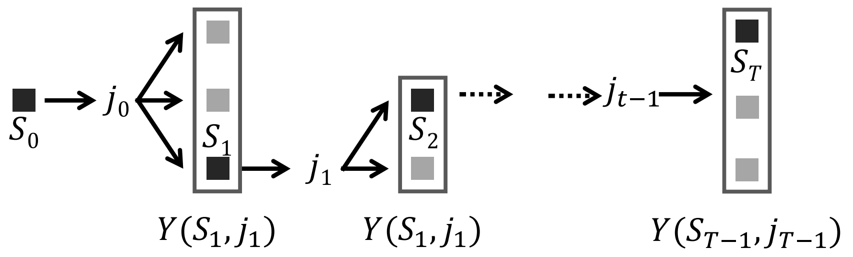


Fig. 16. The progression of states for a finite time MCDP.

An optimal policy for a finite time MCDP can be found using *dynamic programming* (DP). In dynamic programming, the value of each state is determined by starting at the latest state in time and working backwards until the first time period. Define the value of a state $V_t(S_t, j_{t-1})$ for states $S_t \in \mathcal{S}_t$ for $t = 1, \ldots, T-1$ as

$$V_t(S_t, j_{t-1}) = \mathop{\mathbb{E}}_{j_t}\left[\min_{S_{t+1} \in Y(S_t, j_t)} \left(C(S_t, j_t, S_{t+1}) + V_{t+1}(S_{t+1})\right) \middle| \, j_{t-1}\right],$$

and for $S_T \in \mathcal{S}_T$ let $V(S_T) = \mathop{\mathbb{E}}_{j_T}[\min_{S_T' \in Y(S_T, j_T)} C(S_T, j_T, S_T')]$. These values can be calculated by first finding the values for $S_T \in \mathcal{S}_T$, then computing the values $S_{T-1} \in \mathcal{S}_{T-1}$ and so on. For a more in depth overview of dynamic programming see (Bertsekas, 2007).

In the case of this network routing problem, the state of the system depends on how many batteries are unavailable at each station during each time period future. Let t be the current time. At any time period $t, \ldots, T$ for a station b_i, between 0 and n_i batteries may be unavailable. As time passes less information needs to be stored since fewer time periods remain until the end of the day. Fig. 17 shows the state for a network with a single battery-exchange station with four batteries at time $t = 10$, and then the state after a vehicle makes a reservation and time passes to $t = 11$. In this figure it takes two time units to swap the battery and six to fully charge an empty one. Notice that what may have happened is a vehicle wanted a reservation at time 17, but the only two batteries available at time 17 would be needed by two other vehicles at times 19 and 20. Thus the reservation had to be postponed until time 22, and the vehicle would have to wait at the station for 5 extra time units. The reservation takes up 8 time units, 2 to actually do the swapping and 6 to charge.

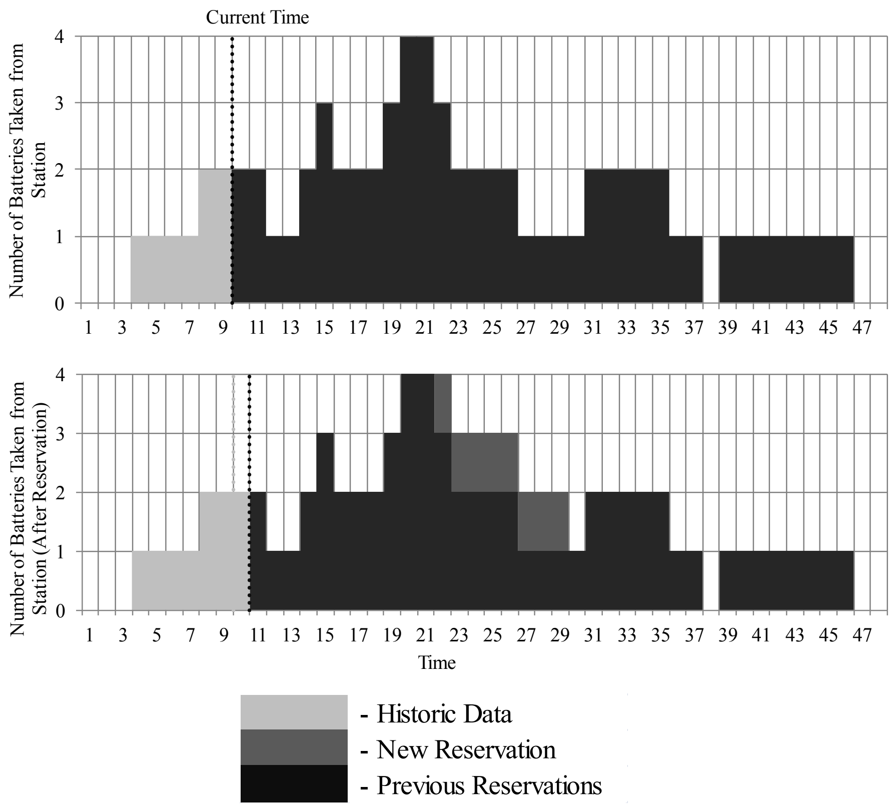


Fig. 17. An example of the state of a single station and the station after a vehicle makes a reservation and time steps by one unit.

At time period t for $t = 0, \ldots, T$, the possible states of station i are

$$\mathcal{S}_t^i = \{0, \ldots, n_i\}^{T-t},$$

which represents the number of unavailable batteries at station i during each time period from t until the end of the day. We define the possible states of the system at time t for

$t = 0, \dots, T$ as $\mathcal{S}_t = \mathcal{S}_t^1 \times \mathcal{S}_t^2 \times \cdots \times \mathcal{S}_t^\beta$. This represents all possible combinations of states of each station. The set $\mathcal{S}$ represents the possible states of the entire system

$$\mathcal{S} = \bigcup_{t=0}^{T} \mathcal{S}_t$$

where each state of the system encompasses the unavailable batteries of each of the battery-exchange stations from the current time until the end of the day. For convenience, for a state $S_t \in \mathcal{S}_t$ to refer to the number of batteries available at station i at time $\tau \geq t$ we use the notation $S_t(i, \tau)$. Also for convenience, define the function $\Lambda: \mathcal{S} \rightarrow \mathcal{S}$ as the function that maps $\mathcal{S}_t \rightarrow \mathcal{S}_{t+1}$ (progressing a state one step forward in time) by removing the first element of each $\mathcal{S}_t^i$. So for an example state $(2,3,1,0,0,1)$ where there is a single station and 7 time periods until the end of the day, $\Lambda\big((2,3,1,0,0,1)\big) \mapsto (3,1,0,0,1)$. This represents the state that the system would be in after no vehicles are routed but time steps by one unit.

The initial state S_0 represents the stations before any vehicles arrive, and thus S_0 is a tuple of $T \cdot \beta$ zeros. The input process $\Xi = (\xi_t)_{t \geq 0}$ determines which vehicles arrive at each time period. Thus the state set J is the possible desired OD pairs a vehicle could arrive wanting to travel: $J = \{0,1,2, \dots, \sigma\}$. Here the values $j = 1, \dots, \sigma$ represents a vehicle arriving wanting to travel from origin o_j to destination d_j, and the value $0 \in J$ represents no vehicle arriving at that time period. Since the vehicle arrivals at different times are independent, we can simply define the transition matrix Q such that $Q_{jj'} = p_j$ for all $j, j' \in 0,1, \dots, \sigma$. The μ vector is unnecessary, also due to the fact that the arrivals are independent during each time period.

The transition mapping Y will describe the possible ways a vehicle could be routed. An (S_t, j_t) pair represents the state of the system at time t and which vehicle has just arrived, and $Y(S_t, j_t)$ is the set of all possible states the system can transition to. Each $S_{t+1} \in Y(S_t, j_t)$ has a one-to-one correspondence with the routes in $\mathcal{R}_{j_t}$ since each possible route gives a new state that the system could transition to. In the case that $j_t = 0$, there is a single element in $Y(S_t, j_t)$, the state where no new reservations are made to the current state but time steps forward by one interval. Thus, when $j_t = 0$ We have that $Y(S_t j_t) = \{\Lambda(S_t)\}$.

The exact mathematical formulation of transition function Y is cumbersome to define explicitly since it has to encompass the logic of when is the earliest a reservation could be made after a vehicle arrives as described in Section 3.1.2. It is however fairly straightforward to write an algorithm that on input (S_t, j_t) gives set $Y(S_t, j_t)$. That algorithm can be seen in Fig. 18. This algorithm also returns a matrix Z where each column corresponds to a route in $\mathcal{R}_{j_t}$. The column is a $(\beta + 1)$-vector where the ith entry is in the column is the penalty incurred for waiting at station i and the $(\beta + 1)$th entry is the time travelled by the vehicle plus the penalty for all of the battery swapping times. This matrix Z will be used by later calculations in the MCDP. In the algorithm, variable a tracks the time at which the vehicle arrives at each station in the route, f tracks how long the vehicle will wait at the current station.

The algorithm iterates through the stations along each route, and determining how long the vehicle will have to wait when it arrives at the station before it can leave with a new battery. Starting with the first station in the sequence, the algorithm checks if there will be no available batteries at any time between when the vehicle arrives assuming no wait, $(a + w)$ for $w = 0$, and up to when the battery the vehicle swapped out would be fully charged, which is $(a + w + g + h)$. If indeed there is a point in that interval where no batteries will be available then the vehicle must wait to get a battery. The value w' is added to the waiting time, where w' is the count of the number of instances in the time sequence where no batteries are available. The value w' provides an amount of time that is necessary but not sufficient to ensure the vehicle has waited enough for a battery to be available. The algorithm then checks if the updated wait time w is sufficient, and if not the process is repeated until a valid waiting time is found. Once the algorithm determines how long the vehicle will wait at the station, the arrival time at the next station is calculated and the process continues for that station.

```
On input $S_t, j_t$
Initialize $Y \leftarrow \emptyset, a \leftarrow 0, w \leftarrow 0, i \leftarrow 0,$
Initialize $S_t^{'}$ as a $(T-t)$-tuple of zeros
Initialize $Z$ as a $(\beta+1) \times |\mathcal{R}_{j_t}|$-matrix
For each $R = (b_1^R, b_2^R, \ldots, b_{|R|}^R) \in \mathcal{R}_{j_t}$
        Set $i \leftarrow i + 1$
    Set $a \leftarrow l^{'}(o_{j_t}, b_1^R)$
    Set $S_t^{'} \leftarrow S_t$
    Set $Z(\beta+1, i) \leftarrow l^{'}(o_{j_t}, b_1^R)$
    For $x \leftarrow 1, \ldots |R|$
        Set $f \leftarrow 0$
        While $S_t(b_x^R, y) \geq n_x^R$ for any $y \in \{a+w, \ldots, \min(a+w+g+h-1, T)\}$
            Set $w \leftarrow w + w'$ where $w'$ is the count of times $S_t(b_x^R, y) \geq n_x^R$ for $y \in$
            $\{a+w, \ldots, \min(a+w+g+h-1, T)\}$
        End while
          Set $a \leftarrow a + w + g$
        Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}(b_x^R, b_{x+1}^R) + \rho_1 g$
        Set $Z(b_x^R, i) \leftarrow \rho_2 w$
          For each $y \in \{a+w, \ldots, \min(a+w+g+h-1, T)\}$
            Set $S_t^{'}(b_x^R, y) \leftarrow S_t^{'}(b_x^R, y) + 1$
        End for
        If $x < |R|$ then
            Set $a \leftarrow a + l^{'}(b_x^R, b_{x+1}^R)$
            Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}(b_x^R, b_{x+1}^R)$
                    End if
          End for
    Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}(b_x^R, d_{j_t})$
    Replace column $i$ of $L$ with $z$
        Set $Y \leftarrow Y \cup \Lambda(S_t^{'})$
End for
Return $Y, Z$
```

Fig. 18. An algorithm for the mapping $Y: S \times J \rightarrow 2^S$ that provides the state of the system after a vehicle has been routed and also returns the route length and waiting times.

Finally, we define the cost function $C: S \times J \times S \rightarrow \mathbb{R}$, which describes the cost of taking a transition from (S_t, j_t) to $Y(S_t, j_t)$. For (S_t, j_t) where $S_t \in \mathcal{S}_t$ and $j_t \in \{0, \ldots, \sigma\}$, let $S_t' \in Y(S_t, j_t)$ correspond to taking path R which is column i in Z. The value of the of the route, denoted $\psi_t(R)$ is the value in the Fig. 18 found by summing the column i of matrix Z, since the first β entries are the weighted wait times at the stations and the $(\beta + 1)$th entry is the weighted time travelling and swapping. The cost function mapping is $C(S_t, j_t, S_t') \mapsto \psi_t(R) - L_{j_t}$, which represents the delay penalty the vehicle will take if that route is assigned. If the vehicle were to take the shortest possible route and had no delays, the resulting cost would be the minimum value 0. Technically if $S_t' \notin Y(S_t, j_t)$ we then assign the cost $C(S_t, j_t, S_t') \mapsto \infty$, although this is important only for ensuring C is a proper mapping. Further we also define cost function $C^1: S \times J \times S \to \mathbb{R}$ as $C^1(S_t, j_t, S_t') \mapsto \psi_t^{drive}(R) + \psi_t^{swap}(R) - L_{j_t}$: the cost attributed to the vehicle taking a longer route and making addition battery swaps beyond the minimum (but not associated with waiting). Also define the cost functions $C_i^2: S \times J \times S \to \mathbb{R}$ as $C_j^2(S_t, j_t, S_t') \mapsto \psi_{t,i}^{wait}(R)$ for $j = 1, \dots, \beta$: the cost attributed to the vehicle having to wait at station i. Thus cost C is equal to the summation $C^1 + C_1^2 + C_2^2 + \cdots + C_\beta^2$, which is the cost of travelling time and swapping time, plus the waiting incurred at each station. Each of these costs is stored in a different row in Z; the cost C^1 is the $(\beta + 1)$th entry of Z and C_i^2 is the ith row of Z for $i = 1, \dots, \beta$. The component cost functions $C^1, C_1^2, C_2^2, \dots, C_\beta^2$ will be used for calculations later, but with only C we now have a full description of the MCDP for the network routing problem. The optimal policy for the MCDP will correspond to the function that takes as input the state of the system and the vehicle that arrives and outputs the optimal route for that vehicle to take.

3.2.2. MCDP approximation using linear temporal differencing

As stated previously, for a Markov decision process the methods for finding the optimal policy typically rely on dynamic programming. These techniques compute the value of being in a particular state, where the value of a state is the expected cost incurred after arriving in that state and having the agent continue in the Markov decision process. Once the values of each state are found, each decision can be made to minimize the sum of the expected value of the action plus the expected value of the next state. While value iteration and other techniques can be used on MCDPs as well as MDPs, in both cases the algorithms that find the optimal policy become intractable for a large number of states. The problem of routing and reserving batteries for electric vehicles has an immense number of states. In fact at time period t there are $\prod_{i=1}^{\beta}(b_i+1)^{T-t+1}$ possible states, so for example in a system with 10 stations each having 5 batteries with 20 time periods before the end of the day there are $(6^{20})^{10} \approx 4.27 \cdot 10^{155}$ states.

Thus, we need to find an approach that will grant a solution that may not be optimal but is still better than using a naïve policy such as the greedy routing of each vehicle. For that we turn to approximate dynamic programming. Here we will still attempt to find the value of each state of the MCDP, only now we will accept approximate solutions for the values. Notice that in the case of the network routing and reservation problem the value of a state represents the estimated delay penalties of all of the future arrivals.

3.2.2.1. Approximate dynamic programming

In *Approximate Dynamic Programming* (ADP) (Powell, 2011), rather than computing the exact value for each state, the values of the states are approximated. Unlike the standard dynamic programming algorithm which computes the values of each state backwards, typically in approximate dynamic programming the state values are given approximations and using the approximations the algorithm steps forward in time. Here we present the Q-learning ADP algorithm (Russell and Norvig, 2009) but modified for MCDPs instead of the classic MDP. Let $V_t(S_t, j_t)$ be the value of being in state S_t at time t and then having the random outcome j_t occur. This represents the value of the state after the uncertain outcome j_t has happened. We could also measure the value of the state before the uncertainty (and decision making), which we will denote $V_t(S_t|j_{t-1})$, since it depends on the previous state of the Markov chain. Let $\bar{V}_t^{m-1}(S_t, j_t)$ be an approximation of $V_t(S_t, j_t)$, and let $(j_0^m, j_2^m, \dots, j_{T-1}^m)$ be a randomly generated realization of Ξ. Then we can compute a new and hopefully closer approximation $\bar{V}_t^m(S_t, j_t)$ for $V_t(S_t, j_t)$ by finding for each $t = 0, \dots, T$

$$\hat{v}_t^m = \min_{S_{t+1}^m \in Y(S_t^m, j_t^m)} \left(C(S_t^m, j_t^m, S_{t+1}) + \sum_{k \in J} \bar{V}_{t+1}^{m-1}(S_{t+1}, k) P(k|j_t^m) \right),$$

and letting S_{t+1}^m be the value that minimizes S_{t+1}. Then using a stepsize parameter $0 < \alpha \leq 1$,

$$\bar{V}_t^m(S_t, j_t) = \begin{cases} (1 - \alpha_m)\bar{V}_t^{m-1}(S_t, j_t) + \alpha_m \hat{v}_t^m & S_t = S_t^m \\ \bar{V}_t^{m-1}(S_t, j_t) & \text{otherwise.} \end{cases}$$

The algorithm works by using an approximation of the value of the possible next states to determine which action to take. Since in an MCDP which chance outcome will occur at the next state is unknown, the possible outcomes are averaged using the probabilities of the outcomes. The values of the states before the uncertainty can then be computed as $\bar{V}_t^m(S_t|j_{t-1}) = \sum_{k \in J} \bar{V}_{t+1}^m(S_{t+1}, k) P(k|j_{t-1})$. As new approximations are generated for the values of the states the approximations should hopefully get more accurate.

3.2.2.2. Temporal difference updates

Suppose that the system is in state S_t^m (having previously seen j_{t-1}^m) and we are already given a policy π. Also suppose there is a given future set of outcomes $(j_0^m, j_{t+1}^m, \dots, j_{T-1}^m)$. One way to compute the value of the state S_t^m would be to follow the policy until the end of the horizon at time T, this would give us an approximation for $V_t(S_t^m|j_{t-1}^m)$ of

$$\hat{v}_t^m = C(S_t^m, j_t^m, S_{t+1}^m) + C(S_{t+1}^m, j_{t+1}^m, S_{t+2}^m) + \cdots + C(S_{T-1}^m, j_{T-1}^m, S_T^m),$$

where for $\tau = t, \dots, T$, $S_{\tau+1}^m = \pi(S_\tau^m, j_t^m)$. Here the value $\hat{v}_t^m$ is sum of all of the future costs that will be incurred during the time horizon. The value of the $\bar{V}_t^{m+1}(S_t|j_t^{m+1})$ could be set directly by $\bar{V}_t^{m+1}(S_t^m|j_t^m) = (1-\alpha_{m+1})\bar{V}_t^m(S_t^m|j_t^m) + \alpha_{m+1}\hat{v}_t^{m+1}$. Alternatively, define the *temporal difference* as

$$\delta_\tau^{\pi,m} = C(S_\tau^m, j_\tau^m, S_{\tau+1}^m) + \bar{V}_{\tau+1}^{m-1}(S_{\tau+1}^m|j_\tau) - \bar{V}_\tau^{m-1}(S_\tau^m|j_{\tau-1}^m).$$

Then because the sum of the temporal differences are telescoping, it is equivalent to write

$$\hat{v}_t^m = \bar{V}_t^{m-1}(S_t^m|j_t^m) + \sum_{\tau=t}^{T} \delta_\tau^{\pi,m}$$

So the updated value is the sum of all of the differences to the current values. However this weighs the changes to the approximation that occur far into the future just as heavily as the changes that occur immediately after t. Since the earlier changes are likely more dependent on the particular action we take (since the decision has more of a direct impact), the differences can be geometrically discounted with rate λ and thus:

$$\bar{V}_t^m(S_t^m|j_t^m) = \bar{V}_t^{m-1}(S_t^m|j_t^m) + \alpha_m \sum_{\tau=t}^{T} \lambda^{T-\tau} \delta_\tau^{\pi,m}$$

This method of updating the approximations is the classic temporal difference algorithm $\text{TD}(\lambda)$ (Powell, 2011), with the slight modification due to using an MCDP instead of an MDP. We will use this algorithm for our particular MCDP.

3.2.2.3. Temporal differences with a linear model

While it is possible to find the optimal policy to a finite horizon MCDP using dynamic programming $\text{TD}(\lambda)$, the amount of states in the network routing MCDP makes this technique still intractable. Dynamic programming requires the calculation of the value of the system in each possible state, and each state can have a unique and independently calculated value. It is likely the case however that the values of the states are closely related. For instance in the case of the network routing problem the value of states S_t and $\Lambda(S_t)$ (i.e. no vehicle arriving and time progressing by one) should be fairly close to each other. If the values of the states are approximated in such a way that we utilize the similarities of the states, the calculation should become tractable. Thus by approximating the values of each state future by using linear functions the problem becomes feasible to solve.

For simplicity, since in our model the random variable ξ_t does not depend on ξ_{t-1}, we will denote the value of a state as $\bar{V}_t^m(S_t)$ instead of $\bar{V}_t^m(S_t|j_{t-1})$. For our problem we will approximate the state values using a linear model. The approximations of the state values will take the form of a linear model

$$\bar{V}_t^m(S_t) = \sum_{f \in \mathcal{F}} \theta_{tf}^m \phi_f(S_t),$$

where $\phi_f(S_t)$ are *basis functions* used to describe the state and θ_{tf}^m is a set of coefficients for the functions, which depend on the time t of the system and the approximation iteration m. The set $\mathcal{F}$ is the indexes all of the basis functions. Now instead of finding the best approximation for each state at each time, we are instead interested in finding the optimal coefficients $\theta_{f,t}^m$ which best approximate each state at each time. Notice that because this is a finite horizon model, there is a different set of coefficients for each time interval and hence the index t.

For each station $b_i \in B$ and for each value $q = 1, \dots, n_i$, define the function $\phi_{(i,q)}$ which maps state $S \in \mathcal{S}$ to the number of time periods in which station b_i has at least q batteries reserved. Functions $\phi_{(i,1)}, \phi_{(i,2)}, \ \dots, \phi_{(i,n_i)}$ capture the total minutes that the batteries at station i are reserved for. For example, if at station b_2 has two batteries in it ($n_2 = 2$), and at time $t = 30$ there is exactly one battery reservation which makes the battery unavailable during times $41,42, \dots, 48$, then $\phi_{(i,1)} = 8$ and $\phi_{(i,2)} = 0$. When using the functions to describe the station, we lose exactly when the batteries are reserved: we may know that there are 8 time periods in the future where 1 battery is unavailable, but we do not know when those 8 periods fall (and the time may be split into multiple segments). Also define the function $\phi_0(S_t) = T - t$, which determines the number of time periods left until the end of the day. Therefore for this network problem $\mathcal{F} = \{0, (1,1), \ \dots, (1, n_1), (2,1), \ \dots, (\beta, n_\beta)\}$. Additionally, since we know that the costs associated with routing a vehicle increase when some batteries are unavailable, as the values for these basis functions increase so too should the value of the state. Having fewer batteries available is always worse than having more batteries available, so we can restrict the coefficients so that $\theta_{tf}^{m} \geq 0$ for all $t \in T$, $f \in \mathcal{F}$ and all m.

This selection of basis functions used to describe the states have that added property that they are scale appropriately with time. For example if between a certain time t and the end of the day there are 20 time periods where station 3 has all 5 of its batteries reserved, what the particular value of t is does not matter for the sum of the vehicle delays that are expected to occur. Thus, instead of computing parameters θ_{tf}^{m} for each $t \in \{0, \dots, T\}$ and $f \in \mathcal{F}$ we can instead compute a single value θ_{f}^{m} for each $f \in \mathcal{F}$ and assume that $\theta_{tf}^{m} = \theta_{f}^{m}$ for all $t \in \{0, \dots, T\}$. This dramatically reduces the number of coefficients that need to be calculated.

There is one further complication to the computation. The equation $\bar{V}_t^m(S_t) = \sum_{f \in \mathcal{F}} \theta_f^m \phi_f(S_t)$ estimates the entire value of the state using each of the basis functions. While these basis functions should work, the network routing problem has an added advantage that much of the value of the state can be attributed to certain battery-exchange stations (and thus to particular basis functions). Recall that the cost of a transition is due to the amount of time travelling and swapping batteries beyond the minimum time for any possible route in an empty system, and the amount of time waiting at a station for a battery. Define *base value* $\bar{V}_t^{m,1}(S_t)$ as the value of the state S_t at time t due to vehicle travel time and swap time. Define the *sub-values* $\bar{V}_{t,i}^{m,2}(S_t)$ for $i = 1, \dots, \beta$ as the value of the state S_t at time t due to vehicles waiting at station i. Thus,

$$\bar{V}_t^m(S_t) = \bar{V}_t^{m,1}(S_t) + \sum_{i=1}^{\beta} \bar{V}_{t,i}^{m,2}(S_t).$$

Further, we can estimate these values by using:

$$\bar{V}_t^{m,1}(S_t) = \sum_{f \in \mathcal{F}} \theta_f^{m,1} \phi_f^m(S_t)$$

$$\bar{V}_{t,j}^{m,2}(S_t) = \sum_{z=0}^{n_i} \theta_{(i,z)}^{m,2} \phi_{(i,z)}^m(S_t) \quad \text{for all } i = 1, \dots, \beta.$$

Here $\theta_f^{m,1}$ and $\theta_f^{m,2}$ are nonnegative real numbers for each $f \in \mathcal{F}$ and we force $\theta_0^{m,2} = 0$ since it is not used in any of the sub-value approximations. The base value is still an approximation using all of the basis functions and coefficients $\theta_f^{m,1}$. The sub-value for each station i is approximated using only the basis functions related to that station, i.e. the functions that count the number of minutes at each battery level at that station. By using these multiple values we decrease the likelihood that a delay caused by a station would be attributed to the battery level at a station on the other side of the network that just happened to also have a high battery level at that time.

The algorithm to compute the values $\theta_f^{m,1}$ and $\theta_f^{m,2}$ can be seen in Fig. 19. The algorithm runs U progressions through the horizon, and each state taking the action that provides the lowest cost using the previous approximation. The temporal differences are then calculated for the base value and sub-values separately, and a new set of coefficients are generated by finding that minimize the L_2 distance of the approximate values from the TD values of each state, while ensuring that the coefficients are nonnegative. The coefficients are generated for the base values and then for each of the sub-values separately, although they are all stored in a single vector. Thus we are computing $\beta + 1$ nonnegative least squares regressions, one for the base value and then one for each sub-value associated with a station. The non-negativity ensures that highly correlated values do not cause any of the coefficients to be negative, which would suggest that having fewer batteries available *decreases* the travel time. For example, if in most of the runs a station i rarely has exactly 4 batteries taken, if negative values were allowed it would be possible for $\theta_{t(i,4)}^{m,2} \ll 0$ and $\theta_{t(i,5)}^{m,2} \gg 0$. This also intuitively makes sense, since decreasing the number of available batteries should never make a state more attractive. The nonnegative least squares problem can be solved using standard algorithms (Lawson and Hanson, 1974). The algorithm requires parameters M, U, α_m, and λ_m to be tuned manually.

```
Initialize $\theta_f^{0,1} \leftarrow 0$ and $\theta_f^{0,2} \leftarrow 0$ for $f \in \mathcal{F}$
For $m \leftarrow 1, \dots, M$
    For $u \leftarrow 1, \dots, U$
        Set $y \leftarrow \emptyset$
        Generate random sequence of car arrivals $(j_0^u, \dots, j_{T-1}^u)$
        Set $S_0$ to be the state where all of the stations are empty
        For $t \leftarrow 1 \dots T$
            Set $S_t^u \leftarrow \underset{S_t' \in Y(S_{t-1}, j_{t-1}^u)}{\operatorname{argmin}} \left(C(S_{t-1}^u, j_{t-1}^u, S_t') + \sum_{f \in \mathcal{F}} (\theta_f^{m-1,1} + \theta_f^{m-1,2}) \phi_f(S_t')\right)$
        End for
        For $t \leftarrow 0, \dots, T-1$
            Set $\delta_t^{1,u} \leftarrow C^1(S_t^u, j_t^u, S_{t+1}^u) + \sum_{f \in \mathcal{F}} \theta_f^{m-1,1} \phi_f(S_{t+1}^u)$
                $- \sum_{f \in \mathcal{F}} \theta_f^{m-1,1} \phi_f(S_t^u)$
            For $j = 1, \dots, \beta$
                Set $\delta_{t,j}^{2,u} \leftarrow C_j^2(S_t^u, j_t^u, S_{t+1}^u) + \sum_{z=0}^{n_j} \theta_{(j,z)}^{m-1,2} \phi_{(j,z)}(S_{t+1}^u)$
                    $- \sum_{z=0}^{n_j} \theta_{(j,z)}^{m-1,2} \phi_{(j,z)}(S_t^u)$
            End for
        End for
        For $t \leftarrow 0, \dots, T-1$
            Set $y_t^{u,1} \leftarrow \sum_{f \in \mathcal{F}} \theta_f^{m-1,1} \phi_f(S_t^u) + \sum_{\tau=t}^{T} (\lambda_m)^{T-\tau} \delta_\tau^{1,u}$
            For $j \leftarrow 1, \dots, \beta$
                Set $y_{t,j}^{u,2} \leftarrow \sum_{z=0}^{n_j} \theta_{(j,z)}^{m-1,2} \phi_{(j,z)}^m(S_t^u) + \sum_{\tau=t}^{T} (\lambda_m)^{T-\tau} \delta_{\tau,j}^{2,u}$
            End for
        End for
    End for
    Set
        $$\theta^{m,1} \leftarrow \underset{\theta \geq 0}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{t=0}^{T-1} \left(y_t^{u,1} - \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t^u)\right)^2$$
    For $j \leftarrow 1, \dots, \beta$
        Set
            $$\theta_j^{m,2} \leftarrow \underset{\theta \geq 0}{\operatorname{argmin}} \sum_{u=1}^{U} \sum_{t=0}^{T-1} \left(y_t^{u,2} - \sum_{z=0}^{n_j} \theta_{(j,z)}^{m-1,2} \phi_{(j,z)}(S_t^u)\right)^2$$
    End for
    Set $\theta^{m,1} \leftarrow \alpha_m \theta^{m,1} + (1 - \alpha_m) \theta^{m-1,1}$
    Set $\theta^{m,2} \leftarrow \alpha_m \left[0, \theta_1^{m,2}, \theta_2^{m,2}, \cdots, \theta_\beta^{m,2}\right]' + (1 - \alpha_m) \theta^{m-1,2}$
End for
```

Fig. 19. A TD(λ) algorithm with linear approximations for finding a policy for the network routing problem.

With the successful implementation of the algorithm in Fig. 19, we will be provided a set of coefficients $\theta_f^{M,1}$ and $\theta_f^{M,2}$ for $f \in \mathcal{F}$. These coefficients can be used by the computer routing the vehicle to determine the best route to take. When the state of the system is S_t and a vehicle turns on at time t wanting to travel OD pair j_t taking one of the possible routes in $\mathcal{R}_{j_t}$, then for each route $R \in \mathcal{R}_{j_t}$ the computer should calculate

$$v_t = \psi_t^{travel}(R) + \sum\nolimits_{f \in \mathcal{F}} \left(\theta_f^{M,1} + \theta_f^{M,2}\right)\phi_f(S_{t+1}),$$

where S_{t+1} is the state of the system after the vehicle takes route R. The route selected should be the one that minimizes v_t. Finding the minimum v_t can be done extremely quickly since it only requires one run of the algorithm in Fig. 18 for each route and a few summations.

3.3. Methods

We tested the algorithm in Fig. 19 to determine the amount of savings the algorithm would provide compared to the greedy policy and the run time of the algorithm. The test data was the Arizona state highway network from (Upchurch et al., 2009), shown in Fig. 20 (again from (Upchurch et al., 2009)). In that paper Upchurch, Kuby, and Lim had a charging station located in each of the 25 cities in the network plus an addition 25 stations located on longer roads between cities. They also used a gravity model to represent demand of vehicles wanting to traverse routes between each OD pair of cities. Here the demand was a function of the population of the OD cities and the length of the shortest path between them. They assumed that the electric vehicles would have a battery that allows them to travel 100 miles before recharging.

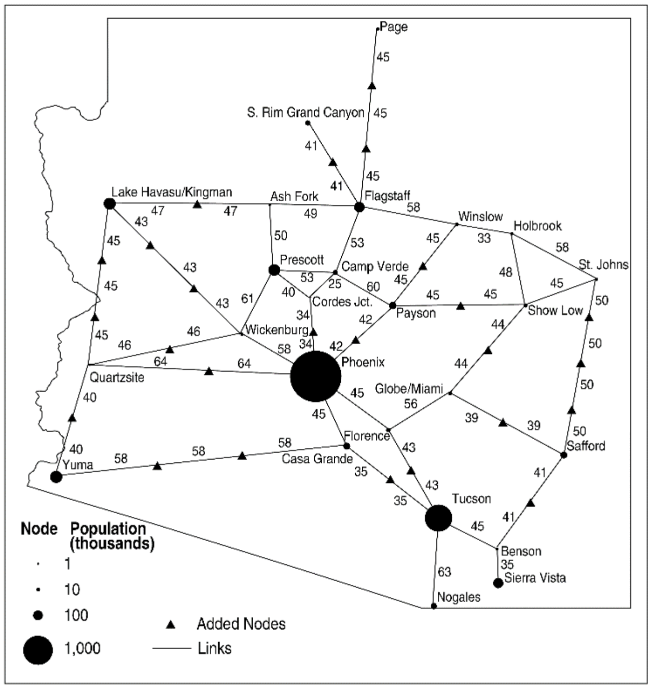


Fig. 20. The Arizona road network with battery-exchange stations from Upchurch et al. (Upchurch et al., 2009).

3.3.1. Network parameters

For this analysis we assumed that there were 18 hours of demand in the day (since people would not be driving late at night) and split the time span into 10 second intervals. Therefore in our implementation $T = 6480$. We assumed that each vehicle travels at 65 miles per hour and thus could travel for 554 time units before requiring a recharge. We assumed that each battery would take 120 seconds to swap, since the Tesla Model S Sedan was shown to take 90 seconds to swap, plus we included an additional 30 seconds for the driver to get out of the car and pay. The battery charging time was set to 4 hours based on charging a Nissan Leaf to full using a 220 volt outlet (Edmunds.com, 2013).

Each of the battery exchange stations located in a city had 48 batteries stored in it. Each of the stations located along side of the highway had only 12 batteries. Given that a vehicle arrived in the system in need of a route, the probability that a vehicle would arrive for a particular OD pair was proportional to the gravity between the two cities used in the model from (Upchurch et al., 2009). For each OD pair j the set of possible routes R_j was generated in the following way. First, we computed the shortest time route from o_j to d_j possible, assuming the vehicle never had to wait for a battery to become available. Then we found all other routes that had at most the same number of battery-exchange stops than the shortest route.

The probability of a vehicle arriving in each time period was taken to be values in $\{0.025, 0.05, \ldots, 0.15\}$ and the algorithm was run on each value. The reason for testing many arrival rates was that if the arrival probability was sufficiently low then the greedy policy would be the optimal policy, since the stations would never run out of batteries (as was the case when the arrival probability was 0.025). Similarly, if the arrival probability was too high then regardless of routing policy the vehicles would be waiting at the stations until the end of the day, and so the greedy policy would just as effective (which occurred when the arrival probability was 0.15). By varying the arrival probability we were able to find at what point the MCDP based policy is the most effective.

For each of the six arrival probabilities the algorithm was run to find approximations for the value coefficients. The algorithmic policy was there compared to the greedy policy by simulating 10 days of vehicles traversing the network, and for each day using the greedy policy and the policy generated by the algorithm. The difference in the amount of delays between the two policies was used to determine which was more effective.

3.3.2. Parameter values

The four parameters for the algorithm were:

- M, the number of times the θ^m coefficients were tuned;
- U, the number of runs through the full time horizon for each run in M;
- α_m the smoothing parameter for θ^m;
- and λ_m the decay parameter for the temporal differencing.

For our trials we set $M = 24$, $U = 8$, $\alpha_m = \frac{20}{20+m}$, $\lambda_m = \alpha_{m-1}\left(\frac{1-\alpha_m}{\alpha_m}\right)$. There were many possible reasonable choices, especially for function definitions for α_m and λ_m, however in our experiments these seemed to prove sufficient.

3.3.3. Results and discussion

The calculations were done on a computer with an Intel Core 2 Duo 2.4Ghz (x2) processor with 4GB of RAM running Windows 7 Ultimate. The algorithm was coded in MATLAB 2012b. The nonnegative least squares calculation was done using the "nnls" package by Whiten on the MATLAB Central File Exchange (Whiten, 2012).

Fig. 21 shows a comparison of the amount of delay incurred due to detours and waiting in the policy generated by the algorithm versus the greedy policy. For arrival probabilities between 0.05 and 0.1 the algorithm substantially lowered the amount of delays. When the arrival probability was 0.025 there were no delays at all (and thus the greedy policy was optimal). When the arrival probability was greater than 0.1 there were too many cars so all of the vehicles had to wait until the end of the day before a battery was ready, and so the algorithm was ineffective.

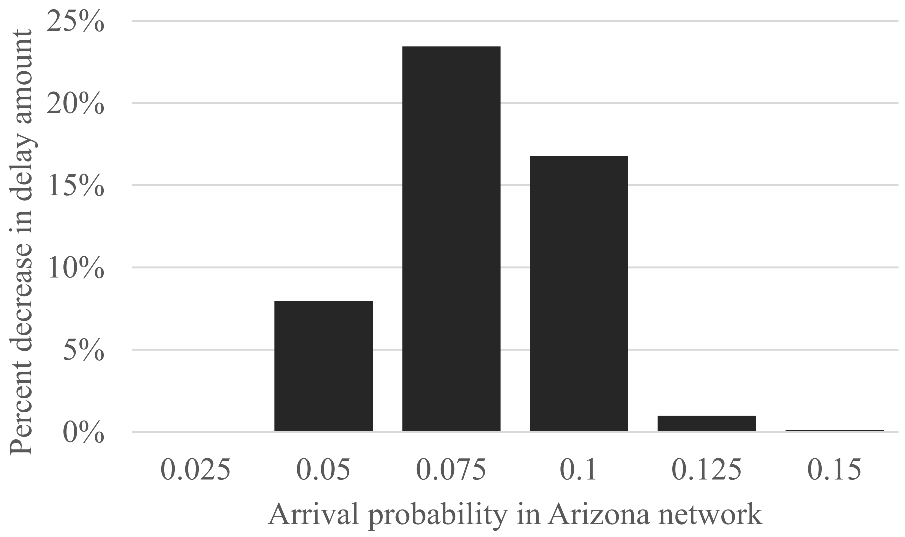


Fig. 21. A comparison of the algorithm vs. the greedy policy runs on different random networks.

The results are further described in Table 2. In the table each row corresponds to the Arizona network with a different arrival probability. The last row shows the runtime in seconds of the algorithm to compute the coefficients. In all of the cases it took less than two hours to run. The θ^m coefficients would be pre-computed so that when the vehicles are routed they only need to find the value of each possible route by computing the value of the resulting state. The amount of time it took to route each individual vehicle during the 10 simulated days is shown in the second to last row. So while the computation of the coefficients took a substantial amount of time, actually routing the cars would take fractions of a second.

In Table 2 the 1st and 2nd rows describe the Arizona network. Rows 3 through 6 describe the greedy policy averaged over the 10 runs, and rows 7 through 10 describe the policy generated by the algorithm. The units for the routes and delay are weighted hours: hours either driving or being delayed, weighted by the coefficients ρ_1 and ρ_2 which penalize the time swapping batteries and waiting for a battery. The 11th row denotes the number of times in a day that the action taken by a vehicle using the policy generated by the algorithm was different than that of the greedy policy, and the 11th row shows the percent decrease in delays between the two policies.

Table 2
The algorithm and greedy policy results for each randomly generated network over ten runs

Measure	Arrival probability in each time period					
	0.025	0.050	0.075	0.100	0.125	0.150
Network						
Number of batteries across stations	1500	1500	1500	1500	1500	1500
Average number of vehicles in day	159.8	326.3	489.6	640.2	812.8	967.6
Greedy policy						
Average total route value of all vehicles	355	728	1113	1488	2550	5338
Average route value of each vehicle	2.2	2.2	2.3	2.3	3.1	5.5
Average total delay for all vehicles	0	1	25	71	754	3197
Average number of cars with any delays	0.0	4.2	76.7	178.7	392.0	549.8
Algorithm policy						
Average total route value of all vehicles	355	728	1107	1476	2542	5334
Average route value of each vehicle	2.2	2.2	2.3	2.3	3.1	5.5
Average total delay for all vehicles	0	1	19	59	747	3193
Average number of cars with any delays	0.0	4.2	77.2	178.9	390.4	548.2
Comparison						
Average times policies differed	0	73.5	119.3	167.7	215.9	424.9
Percent decrease in delay time	--	8.0%	23.4%	16.8%	1.0%	0.1%
Runtimes						
Routing per vehicle (10^{-2} sec)	5.7	3.1	2.2	1.8	1.5	1.4
To generate coefficients (10^{3} sec)	3.94	4.11	4.29	4.43	4.52	4.71

3.4. Conclusion

This research provides a novel framework for routing many electric vehicles through a network with battery exchange technology allowing for battery reservations. We have provided an online algorithm which routes each vehicle in a manner which tries to minimize the total route value of all vehicles then makes the reservations at the stations to be visited. This is done using an approximation of the expected future costs associated with taking each route possible route using an MCDP with linear temporal differencing. The algorithm has been shown to successfully decrease the amount of delays vehicles will have which should lead to better adoption of electric vehicle technology.

There are several ways in which the model could be improved in future work. Better approximation functions for the state values could be found to improve the benefit of the algorithm. The algorithm assumes that transit times are fixed and all vehicles take their assigned routes, both assumptions that could be relaxed. Whether or not to charge a non-full battery at a station can be modeled as a decision, rather than assuming the batteries will always be charged whenever possible. This could potentially reduce costs since less electricity for charging batteries may be used during peak hours. In Section 5.2 additional extensions to the model are discussed: allowing for the demand rate between OD pairs to vary over time, accounting for batteries not being empty when dropped off at a station, and having vehicles start along routes without full batteries.

Chapter 4

THE ALTERNATIVE-FUEL VEHICLE SCHEDULING PROBLEM

Scheduling fleets of vehicles is an important aspect of designing many logistics systems. There are numerous examples of logistics problems where a fleet of vehicles have to perform several spatial-temporal tasks with given resource constraints, such as passengers that must be transported by a bus fleet from origins to destinations, or cargo that must be moved from warehouses to businesses using a fleet of cars or trucks. A large class of these problems have been defined, which are referred to as *Vehicle Scheduling Problems* (VSP). These problems have been analyzed by researchers and their findings have been reported in the literature of Operations Research, Industrial Engineering, Computer Science and related fields. Additionally, vehicle scheduling has many close parallels to classic graph theoretic optimization problems such as the *Traveling Salesman Problem*, *min-cost network flow problems*, and the *Capacitated Arc Routing Problem* (CARP) (Diestel, 2000; Dror, 2000; Heineman et al., 2008).

The multiple depot vehicle scheduling problem (MDVSP) is conceptually straight forward: given a set of n *service trips*, with costs and times assigned for vehicles to travel from the end location of one service trip to the start location of another (called *dead-heading trips*), and a set of vehicles located at d depots, what is the minimum cost required to complete all the service trips with the vehicles? Here the service trips are predefined in terms start and end times and locations, and the services being performed during the trips could be something like picking and dropping off passengers. The multiple depot vehicle scheduling problem as well as generalizations and special cases of it have been analyzed over the past 30 years (Bodin et al., 1983; Bunte and Kliewer, 2009; Ceder, 2002; Daduna and Paixão, 1995; Desaulniers and Hickman, 2007; Desrosiers et al., 1995).

Recent developments in alternative-fuel vehicles have created new questions in transportation. These vehicles, such as electric vehicles, hydrogen-gas vehicles, and bio-fuel based vehicles, do not require the use of gasoline and are an approach to lowering the global dependence on oil. Alternative-fuel vehicles have special requirements on refueling; in the case of hydrogen-gas and bio-fuel vehicles they can only refill at special fueling stations (Ogden et al., 1999). An electric vehicle can either be charged by plugging the vehicle into the electric grid, or by swapping the spent battery of the vehicle with a fresh one at a battery swapping station (Botsford and Szczepanek, 2009). These electric vehicles are being actively researched and experimented with (Folkesson et al., 2003), for example a fleet of electric vehicles was used at the Shanghai Expo in 2010 (Zhu et al., 2012). Because these vehicles types are new, there is very little specialized infrastructure available for them and the number of refueling stations is limited. Additionally, the use of alternative-fuel can considerably limit the distance a vehicle can travel before requiring to refuel, particularly in the case of electric vehicles (Bakker, 2011).

This chapter focuses on the problem where a fleet of vehicles, in particular buses, is changed to a fleet of alternative-fuel vehicles for environmental, sustainability, or government regulation reasons, and where it is necessary to analyze the impacts on cost and fleet sizing. The major difficulty in scheduling is that since the technology is new, there will be few refueling stations and thus the fuel level of the vehicles must be taken into account in vehicle scheduling. Since the range is constrained, more vehicles may be needed for covering the same service trips. Constraining the range that each vehicle may travel in a day in vehicle scheduling problems has been discussed before (Desrosiers et al., 1995; Haghani and Banihashemi, 2002) as the *vehicle scheduling problem with length of path considerations* (VSPLPR). VSPLPR is similar to the problem discussed in this chapter but does not allow for the vehicles to refuel and continue traveling. This is also similar to the *vehicle scheduling problem with time constraints* (Freling and Paixão, 1995) where the time a vehicle may travel is constrained. The situation where the vehicles require refueling but may only refuel at their home depots has been heuristically solved using ant colony algorithms (Wang and Shen, 2007; Wei et al., 2010). The problem in this chapter differs from their work by allowing the vehicles to have several options on where to refuel, instead of only being able to refuel at the vehicle's home depot.

Instances of the multiple depot vehicle scheduling problem, especially when there are additional constraints due to time, often are solved using column generation techniques as first shown by Ribeiro and Soumis (Ribeiro and Soumis, 1994). Other researchers have extended the column generation approach by adding time constraints (Desaulniers et al., 1998), allowing for degenerate problems (Oukil et al., 2007), and integrating crew scheduling into the problem (Steinzen et al., 2010). In these settings, the problem is first formulated as an integer program and the linear relaxation of the integer program is referred to the *master problem*. Most of the decision variables are removed from the master problem to generate a new problem, called the *restricted master problem*. The dual of the restricted master problem is solved, and the solution of the dual is then used as the basis for a subproblem which determines additional decision variables to add to the restricted master problem. This is repeated until the optimal solution to the restricted master problem is found to be the same as the optimal solution to the master problem. In the event the solution is noninteger, a branch and bound technique is used until an integer solution is found. For an in-depth introduction to column generation see (Desrosiers and Lübbecke, 2005). In regards to vehicle scheduling problems, each decision variable typically refers to one feasible set of trips a vehicle will serve after leaving a depot. The column generation algorithms provide a set of feasible service trip selections for each vehicle, as well as the depots the vehicles will use.

In this chapter, we will consider how the limited refueling locations and limited range of alternative fuel vehicles changes the multiple depot vehicle scheduling problem. The classic MDVSP will be modified by adding new constraints where service trips and dead-heading trips require a certain amount of fuel, each vehicle has a limited fuel capacity, and there exists s fuel stations such that a vehicle may refuel as required to continue traveling. The dead-heading trips now include the trips from the fuel stations to the start locations of service trips, and from the end locations of service trips to the fuel stations (as well as between the depots and fuel stations). In this model we use term “fuel” to represent the energy consumed by the use of an alternative-fuel vehicle, however the research could apply to electric vehicles by simply considering “battery charge” to be “fuel level” and “recharging” to be “refueling.” Our solution will be based on using a column generation algorithm to repeatedly determine new possible sets of schedules for vehicles to serve. Because of the fuel limitation for the vehicles, determining feasible sets of schedules a single vehicle can serve is non-trivial, and is in fact an instance of the weight constrained shortest path problem with replenishment arcs (Smith et al., 2012).

This chapter is organized as follows: Section 4.1 formally defines the problem and discusses how to expand the problem to handle cases that violate the assumptions made in the definition. Section 4.2.1 formulates the problem as a master problem and subproblem to be solved using the column generation approach. Section 4.2.2 discusses how to solve the subproblem for the column generation technique, and Section 4.2.3 discusses the branch and bound approach. In Sections 4.3 and 4.4 heuristic approaches are presented and discussed. In Section 4.5 results are generated from applying the algorithms to both randomly generated data and data from Valley Metro, the bus service for the Phoenix Arizona metropolitan area.

4.1. Formal definition of the problem

We define the Alternative-Fuel Multiple Depot Vehicle Scheduling Problem (AF-VSP) as follows. The problem is defined for a given n service trips $T = \{\tau_1, \tau_2, \dots, \tau_n\}$, b fuel stations $B = \{\sigma_1, \sigma_2, \dots, \sigma_b\}$, and d depots $D = \{\theta_1, \dots, \theta_d\}$. For each service trip τ_i there is a start location s_i, and end location e_i, start and end times st_i and et_i, and fuel requirement $f_\tau(\tau_i)$. For each depot θ_j there is a capacity r_j representing how many vehicles are stationed at the depot. For any pair $z_1, z_2 \in T \cup B \cup D$ with $z_1 \neq z_2$, there are travel times $t(z_1, z_2)$, costs $c(z_1, z_2)$, and fuel requirements $f(z_1, z_2)$ representing the respective values for the dead-heading trip from z_1 to z_2. Note that the costs, travel times, and fuel requirement functions take as input the service trips themselves and not their starting and ending points, since any dead-heading trip leaving a service trip must depart from the service trip's end point, and any dead-heading trip heading to a new service trip must arrive at the service trip's start point. The fuel capacity of each vehicle in the fleet is a given value ω. For these given inputs the objective is to find a feasible minimum cost assignment of vehicles from depots to service trips (and fuel stations between service trips) such that each service trip is visited by exactly one vehicle, each vehicle must start and end at the same depot and obey the depot capacity, and for any path a vehicle takes between two fuel stations or a depot the sum of the fuel requirement of the service trips and dead-heading trips in the path must be at most ω.

Without loss of generality assume that the vehicles have no capital cost, there is no cost associated with the act of refueling, and vehicles are refueled instantly on arrival. Also, assume that after refueling, a vehicle must take a least one service trip before refueling again. Later in this section it will be shown why these assumptions can be made. In addition each vehicle starts at the depot with ω fuel, however the depots themselves are not a fuel stations; if that is desired then an additional fuel station can be located at each depot. Also without loss of generality, assume that the service trips are ordered by their start time, so $st_i \leq st_j$ for $1 \leq i \leq j \leq n$. A visual representation of a sample AF-VSP with a single depot can be seen in Fig. 22.

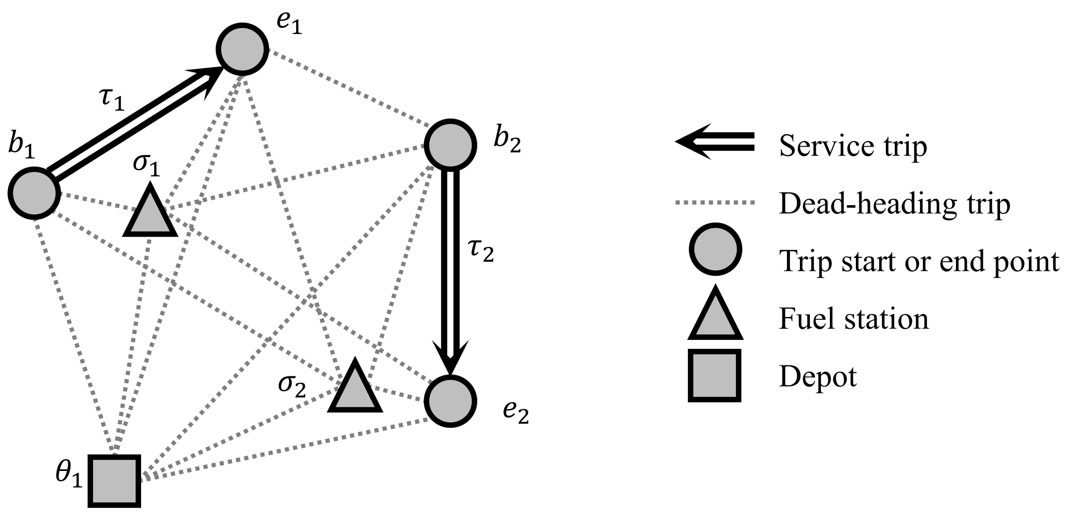


Fig. 22. An example AF-VSP where $n = 2$, $b = 2$, and $d = 1$.

Two service trips $\tau_i, \tau_j \in T$ are *compatible* if $et_i + t(\tau_i, \tau_j) \leq st_j$, meaning a single vehicle may serve both trips without violating time constraints. Two service trips $\tau_i, \tau_j \in T$ are *compatible with fuel station* $\sigma_k \in B$ if $et_i + t(\tau_i, \sigma_k) + t(\sigma_l, \tau_j) \leq st_j$, meaning a vehicle can visit both trips and stop for fuel at σ_k between them without violating time constraints. We write the relationship for compatibility of τ_i and τ_j as $\text{comp}(\tau_i, \tau_j)$ and τ_i and τ_j are compatible with fuel station σ_k as $\text{comp\_fuel}(\tau_i, \sigma_k, \tau_j)$. All trips are both compatible and compatible with fuel with any depot. We use the term *schedule* to refer to a set itinerary a vehicle would take in a day, thus a schedule is an assignment of a vehicle to a depot and service trips and fuel stations as an ordered sequence of elements of $T \cup B \cup D$. In a schedule the first and last elements in the sequence are the same element of D (and no other elements of the sequence are in D), no two fuel stations may occur in a row, and the sequence must include at least one service trip. The route is considered *feasible* if it can be satisfied without the vehicle running out of fuel and the pairs of service trips in the sequence must each be compatible or compatible with a fuel station if the vehicle refuels between them.

The AF-VSP is a generalization of the multiple depot vehicle scheduling problem since VSP is a special case of AF-VSP where $\omega = \infty$ and $b = 0$. Without the fuel constraint the multiple depot vehicle scheduling problem is NP-hard, and thus the AF-VSP is NP-hard as well. When the problem is restricted to a single depot, without the fuel constraint the problem can be solved in polynomial time. With the fuel constraint the problem is still NP-hard since it is an extension of the VSPLPR which was shown to be NP-hard in (Ball, 1980). AF-VSP with no fuel stations and a single depot (i.e. $B = \emptyset$ and $d = 1$), is exactly the VSPLPR. It can also be shown that finding a solution to AF-VSP that has a cost of less than 150% of the optimal solution cost is NP-hard, by using a proof similar to that of CARP in (Golden and Wong, 1981).

As previously stated, this definition of the AF-VSP makes several major assumptions: that at most one fuel station may be visited between pairs of service trips, that there are no costs associated with the number of vehicles, and that refueling is instantaneous and has no cost. In the case when situation violates the assumptions, then by making a few adjustments to it the problem can be fit into the standard framework. Below we discuss how to make those adjustments.

If we allow a vehicle to visit multiple fuel stations between service trips, then we can create an alternate version of the problem that does not allow multiple refueling stops between service trips. First note that the number of fuel stations visited between service trips is bounded by b since a vehicle would never want to visit the same fuel station twice between two service trips. In fact for any sequence of $2 \leq k \leq b$ refueling stations $\{m_1, m_2, \dots, m_k\}$ for $m_1, \dots, m_k \in B$, and $f(m_j, m_{j+1}) \leq \omega$ for all $j < k$, the amount of fuel required for a vehicle to deadhead from a service trip to the beginning of the sequence of stations is uniquely determined by the fuel required to reach m_1. A new proxy fuel station $\sigma_{\{m_1,m_2,\dots,m_k\}}$ can be created representing this sequence of refueling stations, where the dead-heading trips from each service trip to $\sigma_{\{m_1,m_2,\dots,m_k\}}$ have the same costs, travel times, and fuel requirements as the dead-heading trips to fuel station m_1. The dead-heading trips from $\sigma_{\{m_1,m_2,\dots,m_k\}}$ to each service trip $\tau_i \in N$ have travel times $t\left(\sigma_{\{m_1,m_2,\dots,m_k\}}, \tau_i\right) = t(m_k, \tau_i) + \sum_{x=2}^{k} t(m_{x-1}, m_x)$, have the cost $c\left(\sigma_{\{m_1,m_2,\dots,m_k\}}, \tau_i\right) = c(m_k, \tau_i) + \sum_{j=2}^{k} c(m_{j-1}, m_j)$, and the fuel requirement of $f\left(\sigma_{\{m_1,m_2,\dots,m_k\}}, \tau_i\right) = f(m_k, \tau_i)$. So the proxy station $\sigma_{\{m_1,m_2,\dots,m_k\}}$ represents the vehicle traveling using all of the fuel stations in that sequence. The values for the dead-heading trips between the refueling station and the depots can be generated in a similar manner.

While there are at most $b!$ possible fuel stations added, the number of additional fuel stations can be lowered by removing dominated sequences. Each sequence of stations starting with station $\bar{m}$ and ending with station $\bar{\bar{m}}$ has a corresponding total cost and total traversal time. Consider two sequences $M_1 = \left\{\bar{m}, m_1^1, m_2^1, \dots, m_{k_1-1}^1, \bar{\bar{m}}\right\}$ and $M_2 = \left\{\bar{m}, m_1^2, m_2^2, \dots, m_{k_2-1}^2, \bar{\bar{m}}\right\}$. For the two sequences, $f\left(\sigma_{M_1}, z\right) = f\left(\sigma_{M_2}, z\right)$ and $f\left(z, \sigma_{M_1}\right) = f\left(z, \sigma_{M_2}\right)$ for any $z \in T \cup B \cup D$, since they share the same start and end stations. Similarly, $c\left(z, \sigma_{M_1}\right) = c\left(z, \sigma_{M_2}\right)$ and either $c\left(\sigma_{M_1}, z\right) \leq c\left(\sigma_{M_2}, z\right)$ or $c\left(z, \sigma_{M_1}\right) \geq c\left(z, \sigma_{M_2}\right)$ holds for all z. That is to say either the costs of the dead-heading trips from first sequence are all at least as much as the costs of those from the second, or the costs from the second sequence are all at least as much as the costs of those from the first sequence. This is also true for time in addition to cost. Thus, any two sequences which are not Pareto optimal with respect to cost and time can be removed; this corresponds to sequences of fuel stations between $\bar{m}$ and $\bar{\bar{m}}$ which are dominated by other sequences.

If the use of each vehicle has an additional capital cost c_{vehicle} that is incurred if the vehicle is used at all, then the cost can be added to the arcs. For each dead-heading trip that leaves a depot $\theta_j$heading to service trip or fuel station z having cost $c(\theta_j, z)$, change the cost of the service trip to $c(\theta_j, z) + c_{\text{vehicle}}$. Since a vehicle will take exactly one of these dead-heading trips if and only if it is required for the solution, the final solution will have an additional cost of $x \cdot c_{\text{vehicle}}$ where x is the number of vehicles used. Because each depot itself is not a fuel station, a vehicle will not traverse more than one of the trips leaving a depot. Similarly to the situation of the additional cost of vehicles, if refueling has a cost of c_{fuel} for each visit to a fuel station, every dead-heading trip leaving a fuel station should have an additional cost of c_{fuel} added to it. If there is a time t_{fuel} associated with refueling, then the travel time for each dead-heading trip leaving the fuel station should have t_{fuel} added to it.

4.2. The column generation algorithm

Before using a column generation algorithm on the AF-VSP, the problem must first be formulated as a binary integer programming problem. The integer program will then be relaxed and solved using column generation. To find a solution for the unrelaxed integer version, a branch and bound method will be used. To formulate the binary integer programming problem, we first generate a graph corresponding to the problem and then make a formulation based on the graph. Let $G = (V, E)$ be a directed graph, let $c', f': E \to \mathbb{R}^+$ be cost and fuel requirement functions, and a let w be constant value representing the amount of fuel the vehicle can hold. Let $h(\tau_i, \sigma_k)$ represent visiting fuel station σ_k immediately after service trip τ_i, where $i \in \{1, \dots, n\}$ and $k \in b$. Similarly, let $h(\theta_j, \sigma_k)$ represent visiting fuel station σ_k immediately after starting from depot θ_j. Define the set H as:

$$H = \left\{h(\tau_i, \sigma_k): i \in \{1, \dots, n\};\ k \in \{1, \dots, b\}\right\} \cup \left\{h(\theta_j, \sigma_k): j \in \{1, \dots, n\}, k \in \{1, \dots, b\}\right\},$$

which represents all possible ways of visiting a fuel station following a service trip or a depot. This set will be used to determine whether or not a fuel station is compatible with the next service trip given what the vehicle visited immediately before. Let vertex set V be defined as $V = D \cup T \cup H$. In this representation each fuel station in the problem is represented by multiple vertices since each vertex corresponds to stopping at the station after a particular service trip or depot. This representation is to keep track of the service trips that are taken before and after the refueling, and to ensure that no loops are formed that do not go through the depot. The arc set E is defined as the union of arc sets $E_1, \dots, E_7$ given in Table 3.

Table 3
The arcs in graph $G = (V, E)$, where $E = \bigcup_{i=1}^{7} E_i$

Set	Definition	Fuel f'	Cost c'	Description
E_1	$\{(\tau_i, \tau_j): \tau_i, \tau_j \in T, \mathrm{comp}(\tau_i, \tau_j)\}$	$f_\tau(\tau_i) + f(\tau_i, \tau_j)$	$c(\tau_i, \tau_j)$	service trip to service trip
E_2	$\{(\tau_i, \theta_j): \tau_i \in T, \theta_j \in D\}$	$f_\tau(\tau_i) + f(\tau_i, \theta_j)$	$c(\tau_i, \theta_j)$	service trip to depot
E_3	$\{(\tau_i, h(\tau_i, \sigma_j)) : \tau_i \in T, \sigma_j \in B\}$	$f_\tau(\tau_i) + f(\tau_i, \sigma_j)$	$c(\tau_i, \sigma_j)$	service trip to fuel station
E_4	$\{(h(z, \sigma_j), \tau_k): z \in T \cup D, \sigma_j \in B, \tau_k \in T, \mathrm{comp\_fuel}(z, \sigma_j, \tau_k)\}$	$f(\sigma_j, \tau_k)$	$c(\sigma_j, \tau_k)$	fuel station to service trip
E_5	$\{(h(z, \sigma_j), \theta_k): z \in T \cup D, \sigma_j \in B, \theta_k \in D\}$	$f(\sigma_j, \theta_k)$	$c(\sigma_j, \theta_k)$	fuel station to depot
E_6	$\{(\theta_i, \tau_j): \theta_i \in D, \tau_j \in T\}$	$f(\theta_i, \tau_j)$	$c(\theta_i, \tau_j)$	depot to service trip
E_7	$\{(\theta_i, h(\theta_i, \sigma_j)) : \theta_i \in D, \sigma_j \in B\}$	$f(\theta_i, \sigma_j)$	$c(\theta_i, \sigma_j)$	depot to fuel station

Arc set E_1 represents a vehicle taking a dead-heading trip between two service trips. The fuel requirement for the origin service trip is added to the fuel cost of the dead-heading trip so that we do not need to associate fuel requirements with vertices. Arc sets E_2 and E_3, between a service trip and a fuel station and depot respectively; again we add the fuel cost for the origin service trip. Arc sets E_4 and E_5 represent traveling from fuel stations to service trips and depots, and arc sets E_6 and E_7 represent traveling from depots to fuel stations and service trips. An example G is given in Fig. 23, representing the AF-VSP in Fig. 22. Note that in Fig. 23 it is assumed that service trips τ_1 and τ_2 are compatible with each other and with station σ_1.

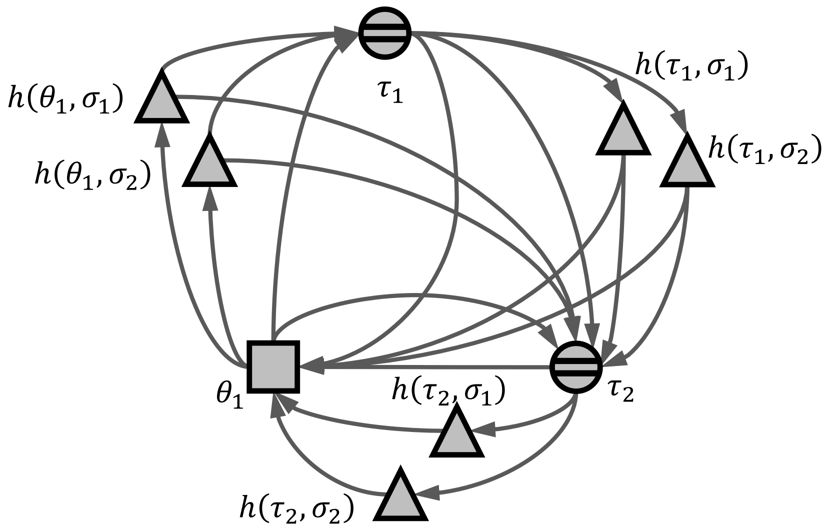


Fig. 23. A graph G corresponding to the AF-VSP in Fig. 22.

Given graph G, functions c' and f', and scalar w, the AF-VSP is now the following: find a minimum cost set of cycles C in G such that each node in $T \subseteq V$ is included in exactly one cycle, each cycle includes exactly one depot vertex, no subpaths of cycles in C between vertices in the set $D \cup H$ have a fuel requirement greater than ω, and at most r_j cycles contain depot θ_j. There are several advantages to this approach to formulating the problem. First, we no longer have to consider the time component of the service trips; time is fully captured by the location of arcs in graph G. Two service trips have a path between them if and only if they are compatible, and similarly an edge exists from τ_i to $h(\tau_i, \sigma_j)$ and from $h(\tau_i, \sigma_j)$ to τ_m if and only if comp_fuel$(\tau_i, \sigma_j, \tau_m)$. Second, each service trip is associated with a single vertex, so the different start and end locations of each service trip are accounted for in the arcs. Further, all fuel requirements are associated with arcs despite the fact that service trips are represented by vertices. Finally, every cycle in in G corresponds to a schedule that could be taken by a vehicle, and the schedule is feasible if no subpath between vertices in $D \cup H$ has a fuel requirement greater than w. The cost of assigning a vehicle to a particular schedule is the sum of the edges in the associated cycle in G.

4.2.1. The column generation master problem

Recall that a schedule is a sequence starting and ending with a depot and containing fuel stations and service trips. Let Ω represent the set of all feasible schedules for an instance of the AF-VSP. For a particular schedule $p \in \Omega$, let variable v_p^i be 1 if schedule p contains service trip τ_i and 0 otherwise. Similarly, let variable u_p^j be 1 if schedule p starts and ends at depot θ_j and 0 otherwise. For each schedule p let q_p be the sum of the cost associated with the dead-heading trips of schedule p. With these indicator variables, we can define an integer programming problem to solve the AF-VSP. Let decision variable x_p be 1 if we include schedule p in the solution and 0 otherwise. The integer program is then:

$$\text{Minimize} \sum_{p \in \Omega} q_p x_p \tag{13}$$

$$\text{subject to} \quad \sum_{p \in \Omega} v_p^i x_p = 1 \quad i = 1, \dots, n \tag{14}$$

$$\sum_{p \in \Omega} u_p^j x_p \leq r_j \quad j = 1, \dots, d \tag{15}$$

$$x_p \in \{0,1\} \quad \forall p \in \Omega \tag{16}$$

Here, the objective (13) is to minimize the sum of all of the costs of the schedules used in the solution. Constrain (14) ensures that each service trip is traversed exactly once. Constraint (15) ensures that no depot will have more schedules using it than the number of vehicles stored at that depot. This integer programming problem is of the *set partitioning type* of multiple depot vehicle scheduling formulation, since it is based on finding a partitioning of the service trips by the feasible schedules that can cover them. This integer program is essentially the same as that used by Ribeiro and Soumis (Ribeiro and Soumis, 1994), except the set Ω has changed from all schedules that satisfy time constraints to all schedules that satisfy both time and fuel constraints. Unfortunately, the number feasible schedules is intractable for problem with all but the smallest number of service trips, and thus it would not be possible to even generate this integer program, never mind solve it.

We can use a column generation approach for this problem to instead solve a linear relaxation of the integer program. We refer to the main linear programming problem as the master problem, and when the problem is solved using only a subset $\Omega' \subseteq \Omega$ of all possible schedules we refer to it as the restricted master problem. We start by solving the duel of the linear relation for a small set of possible candidate schedules, and then use the dual of the solution to find a set of new schedules to add to the restricted master problem.

Let π_i for $i = 1, \dots, n$ be the solutions for the dual variables associated with constraint (14) of the restricted master problem, and let ρ_j for $j = 1, \dots, d$ be the solutions associated with constraint (15). The next schedule to add to the restricted master problem is the schedule that is the solution to: $q^* = \min_{p \in \Omega} \{ q_p - \sum_{i=1}^{n} v_p^i \pi_i + \sum_{j=1}^{d} u_p^i \rho_j \}$. This is equivalent to finding the lowest cost feasible schedule with the additional cost of $-\pi_i$ for traversing service trip τ_i and additional cost ρ_j when starting from depot θ_j. When the new column is found, if $q^* < 0$ then adding the column to the restricted master problem will improve the solution. Thus, when $q^* < 0$ the column is added to the restricted master problem and problem is re-solved. Again a new column is added and the process continues. If $q^* \geq 0$, then no further columns added to the restricted master problem could improve the solution, and thus the solution to the restricted master problem is also the solution to the master problem.

An initial set of n dummy columns are used to start the algorithm. For each service trip τ_i let $\bar{p}_i$ be a dummy schedule that serves only service trip τ_i, does not start at a depot, and has a cost of M where M is a sufficiently large number to ensure it is quickly removed from the solution. Therefore $v^i_{\bar{p}_i} = 1$ and $v^{i'}_{\bar{p}_i} = 0$ for $i' \neq i$, and $u^j_{\bar{p}_i} = 0$ for all j.

4.2.2. Column generation subproblem

To implement a column generation approach for the master problem, we need a way of finding new variables to add to the restricted master problem with minimal values of q^*. This is equivalent to finding lowest cost feasible schedules for vehicles given a particular set of costs. Recall that in graph G, each schedule corresponds to a cycle containing a single depot vertex where if a vehicle follows that cycle it will not run out of fuel. The lowest cost feasible schedule is the lowest cost cycle in this graph that does not have the vehicle run out of fuel. Therefore, given that the vehicle is based at depot θ_j to find the lowest cost feasible path of a vehicle that starts at the depot we can solve the Weight

Constrained Shortest Path Problem with Replenishment (WCSPP-R) (Smith et al., 2012) on graph G with weights f'' and costs c''. Here the both the origin and destination are θ_j, the vertices in set H are replenishment nodes, the weight capacity is w, and for edge $(v_1, v_2) \in E$, the weight values are $f''(v_1, v_2) = f'(v_1, v_2)$ and the cost values are $c''(v_1, v_2)$ are

$$c''(v_1, v_2) = \begin{cases} c'(v_1, v_2) - \pi_i & \text{for } \tau_i \in T, v_2 \in V \\ c'(v_1, v_2) & \text{for } v_1 \in H, v_2 \in V \\ c'(v_1, v_2) + \rho_j & \text{for } v_1 \in D, v_2 \in V. \end{cases}$$

For simplicity we add a copy of the depot θ_j and label it $\bar{\theta}_j$, and label the original as the start node and the new one as the end node. We also remove all vertices $\theta_{j'} \in D \setminus \{\theta_j\}$. All of the outgoing edges of θ_j are associated with the start node, and the all of the incoming edges are associated with the end node. Since this only solves the problem when the vehicle is based at depot θ_j, the problem needs to be solved for each $j = 1, \ldots, d$ separately.

Several solution methods for the WCSPP-R were presented by Smith et al. in their paper. Note that while their paper assumed replenishment is done over arcs, it is trivial to reformulate the problem where replenishment occurs on the vertices, as is the case in our problem. The solutions they present fall into two categories: label correcting algorithms and meta-network methods. Labeling correcting algorithms rely on storing at each vertex a set of possible partial paths that could be used to arrive at that vertex. When determining a set of labels for a vertex, partial paths that have both a higher cost and leave less fuel than an alternate partial path are removed. For an in-depth discussion of label correcting algorithms for the weight constrained shortest path algorithm see

Desrosiers et al. (Desrosiers et al., 1995). Meta-network methods rely on creating a new network which contains only vertices that represent the replenishment nodes and the start and end nodes. The edges in this meta-network have a cost associated with the shortest path between the nodes in the original graph that can be traversed without replenishment. Thus, once the meta-network is computed the solution can be found to the WCSPP-R by solving a standard unconstrained shortest path problem on the meta-network.

Smith et al. found that the meta-network methods were not as efficient on their data sets. For the purposes of the AF-VSP the meta-networks are especially bad, since the meta-network would have $b + 2$ vertices and to build the meta-network the weight constrained shortest paths would have to be found between each pair of vertices. The label correcting algorithms however are especially well suited for the AF-VSP since the graph in the column generation subproblem has a clear ordering in how to determine the labels for each node.

The partial ordering of the vertices in graph G from Fig. 23 can be seen in Fig. 24. For each fuel station σ_k, there is only a single edge ending at vertex $h(z, \sigma_k)$. That edge starts at vertex z, which is implicit in the design of $h(z, \sigma_k)$ since it represents visiting station σ_k after being at either trip or depot z. The vertex representing service trip τ_i can only be reached from depot vertex θ_j, a vertex $h(\theta_j, \sigma_k)$ for each k, a vertex representing service trip τ_j for $j < i$, or from a vertex $h(\tau_j, \sigma_k)$ for each k and $j < i$,. This comes from the fact that the service trips are ordered by their start times, and so to reach a vertex representing a service trip you must take an edge from a vertex representing a depot, visiting a station after a depot, an earlier trip, or visiting a station after an earlier trip.

Finally, the terminal node $\bar{\theta}_j$ can be reached from any of the nodes except θ_j and $h(\theta_j, z)$ for all z. This is because a vehicle must serve at least one service trip after leaving the depot. Therefore we can generate the labels for the vertices by analyzing them in the sequence:

$$\{\theta_j, h(\theta_j, \sigma_1), \dots, h(\theta_j, \sigma_b), \tau_1, h(\tau_1, \sigma_1), \dots,$$

$$\dots, h(\tau_1, \sigma_b), \tau_2, h(\tau_2, \sigma_1), \dots, h(\tau_2, \sigma_b), \tau_3, \dots, h(\tau_n, \sigma_b)\}.$$

After labeling each vertex in $T \cup H$, we can find the length the paths associated with each label and immediately returning to the original depot. If that completed path is both feasible and shorter than any previous path it can be stored as the new best solution.

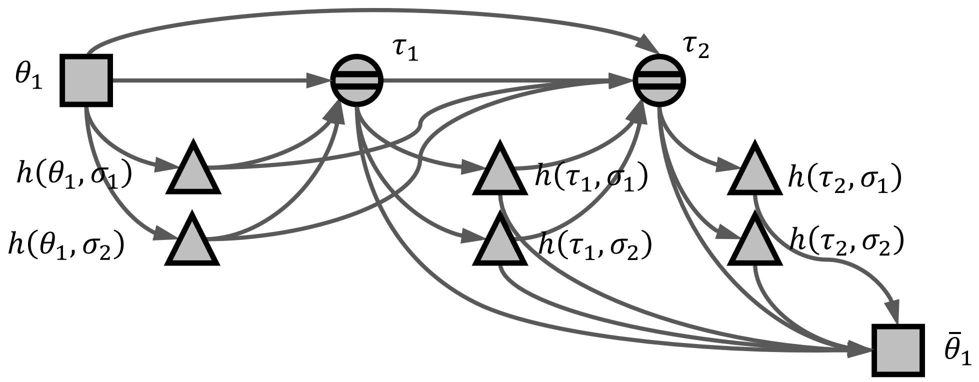


Fig. 24. The graph G from Fig. 23 with replicated node θ_1, ordered from left to right.

For our purposes, with each vertex $v \in H \cup T$ we will associate a set of labels $L(v)$, where each label corresponds to one possible partial schedule to get to that vertex. A label $r \in L(v)$ is a tuple $r = (c, f, sch)$ where c is the cost of the schedule to get to that vertex, f is the remaining fuel at the point along the schedule, and sch is a sequence of elements of $T \cup B \cup D$ representing the actual schedule the vehicle would take during the day (although it will not be a complete schedule since the vehicle will not have returned to the depot at this point). For convenience we refer to the cost component of a label as l_c, the fuel component as l_f, and the schedule component as l_{sch}; for example $l_c(r)$ is the cost component of label $r \in L(v)$. For two schedules $r_1 = (c_1, f_1, sch_1)$ and $r_2 = (c_2, f_2, sch_2)$ where $r_1, r_2 \in L(v)$ we say r_2 is *dominated* by r_1 if either $c_1 \leq c_2$ and $f_1 < f_2$, or $c_1 < c_2$ and $f_1 \leq f_2$. When r_2 is dominated by r_1, then label r_2 can be removed from the label set since there is no reason to take that subschedule: schedule sch_2 results in both higher costs and leaves the vehicle with less fuel than schedule sch_1. Therefore, the algorithm for the shortest path subproblem will involve generating sets of labels and removing the dominated ones.

The labeling algorithm can now be defined. For each service trip in sequence do the following:

1. Generate the labels for the vertex associated with the service trip by taking the union of:
   a. the label associated with the schedule coming directly from the depot,
   b. the labels associated with the schedules coming from previous service trips, and
   c. the labels associated with the schedules coming from refueling stations following earlier service trips or the depot.
2. Remove the labels that involve schedules exceeding the fuel constraint of the vehicle and delete dominated labels.

3. Generate the labels associated with visiting each refueling station following the service trip.
4. Check to see if any of the schedules associated with labels at the service trip or at the refueling station following the service trip can reach the end depot vertex given the remaining fuel. If they are feasible, check if the schedules are shorter than the currently stored lowest cost solution, if any are then the lowest cost one is the new best solution.

Because of the nature of this problem, several steps can be made particularly efficient. First, since at each fueling station the fuel of the vehicle is fully refreshed, all of the labels associated with the fueling station have a fuel value of 0. Thus there can be a single label that dominates all other labels (in the case that both the cost and fuel remaining are equal with several labels, one can be chosen arbitrarily). Each fueling station label set will contain a single element. Further for a service trip τ_j, traveling from $h(z, \sigma_k)$ for some station σ_k to τ_j has the same cost and fuel requirement for all z. Therefore there will be only a single label associated with previously stopping at a fuel station that is not dominated (again choosing arbitrarily if fuel and costs are equal) since the labels at each fuel station have 0 fuel. Thus for each fueling station the single label to add to the service trip's set can be chosen in linear time by finding the one with the minimum cost.

The full algorithm is given in Fig. 25 with a subfunction in Fig. 26. This algorithm returns the shortest schedule from the depot θ_j and the cost of the schedule. First, the algorithm generates the labels for the stations visited immediately after the depot. Then for each service trip in order, the algorithm generates the labels for that service trip using the ones from the previous service trips and from visiting stations immediately before the service trip. For each service trip the labels for the stations following the service trip are generated, and the cost returning to the depot is compared to the current shortest schedule. The subfunction DeleteLabels first removes any labels that are associated with schedules that have the vehicle using more than ω fuel between refueling stations. The remaining labels are sorted in ascending order by remaining fuel and schedule cost, and then the labels are checked to see if any are dominated. The dominated labels are removed and the set of non-dominated labels is returned.

This algorithm finds the shortest path with replenishment for a vehicle that leaves from a given depot. Every time a new column is generated, the algorithm needs to be run for each depot independently. While running the algorithm for each starting depot increases the runtime of the algorithm, there are two positive notes. First, finding the shortest path for each depot can be done in parallel which speeds up the algorithm considerably. Second, rather than only adding the schedule with the minimum value of q^* to the restricted master problem, the shortest schedules for each depot can *all* be added as columns to the restricted master problem, provided the columns have negative solution costs. This will allow for the restricted master problem to more quickly converge on a solution. Note that it may be possible that some of the solutions generated by using different depots may already be in the restricted master problem, so care should be taken to ensure they are not duplicated.

```
Function ShortestPathWithReplenishment
On input of graph $G = (V, E)$ where $V = \theta_j \cup T \cup H$ and $c''$ and $f''$ are functions on $E$
Initialize $best_{sch} \leftarrow \{\}$, $best_{cost} \leftarrow \infty$, $L(v) \leftarrow \emptyset$ for each $v \in H \cup T$
For $k = 1$ to $b$ \\ Initialize the labels of the stations following the depot.
    If $f''\left(\theta_j, h(\theta_j, \sigma_k)\right) \leq \omega$
        $L\left(h(\theta_j, \sigma_k)\right) \leftarrow \left\{\left(c''\left(\theta_j, h(\theta_j, \sigma_k)\right), 0, \{\theta_j, \sigma_k\}\right)\right\}$
    End if
End for
For $i = 1$ to $n$ \\ For each trip in order
    $L(\tau_i) \leftarrow L(\tau_i) \cup \left(c''(\theta_j, \tau_i), f''(\theta_j, \tau_i), \{\theta_j, \tau_i\}\right)$
        For $k = 1$ to $b$ \\ Find the best label for each station.
        $a = \underset{a}{\operatorname{argmin}}\left\{ l_c(r) + c''(h(a, \sigma_k), \tau_i) \,\middle|\, \begin{array}{c} \{r\} = L(h(a, \sigma_k)), \\ a \in \{\theta_j, \tau_1, \ldots, \tau_{i-1}\}, \text{comp\_fuel}(a, \sigma_k, \tau_i) \end{array} \right\}$
        $L(\tau_i) \leftarrow L(\tau_i) \cup \left\{ \begin{pmatrix} l_c(r) + c''(h(a, \sigma_k), \tau_i), \\ l_f(r) + f''(h(a, \sigma_k), \tau_i), \\ \{l_{sch}(r), \tau_i\} \end{pmatrix} \,\middle|\, \{r\} = L(h(a, \sigma_k)) \right\}$
    End for
    For $i' = 1$ to $i - 1$ \\ Join the labels from the previous trips.
        If $\text{comp}(\tau_{i'}, \tau_i)$ Then $L(\tau_i) \leftarrow L(\tau_i) \cup$
$\left\{\left(l_c(r) + c''(\tau_{i'}, \tau_i), l_f(r) + f''(\tau_{i'}, \tau_i), \{l_{sch}(r), \tau_i\}\right) \middle| r \in L(t_{i'})\right\}$
    End for
    $L(\tau_i) \leftarrow \text{DeleteLabels}\left(L(\tau_i)\right)$
    For $k = 1$ to $b$ \\ Find the best labels for each station following the trip
        $r = \underset{r' \in L(\tau_i)}{\operatorname{argmin}}\left\{l_c(r') \middle| l_f(r') + f''\left(\tau_i, h(\tau_i, \sigma_k)\right) \leq \omega\right\}$
        $L\left(h(\tau_i, \sigma_k)\right) \leftarrow \left\{\left(l_c(r) + c''\left(\tau_i, h(\tau_i, \sigma_k)\right), 0, \{l_{sch}(r), \sigma_k\}\right)\right\}$
        If for $\{r\} = L\left(h(\tau_i, \sigma_k)\right)$, $l_f(r) + f''\left(h(\tau_i, \sigma_k), \theta_j\right) \leq \omega$ and $l_c(r) +$
$c''\left(h(\tau_i, \sigma_k), \theta_j\right) < best_{cost}$
            $best_{sch} \leftarrow \{l_{sch}(r), \theta_j\}$
            $best_{cost} \leftarrow l_c(r) + c''\left(h(\tau_i, \sigma_k), \theta_j\right)$
        End if
    End for
    $r = \underset{r' \in L(\tau_i)}{\operatorname{argmin}}\left\{l_c(r') \middle| l_f(r') + f''\left(\tau_i, \theta_j\right) \leq \omega\right\}$
    If $l_c(r) + c\left(\tau_i, \theta_j\right) < best_{cost}$ Then $best_{sch} \leftarrow \{l_{sch}(r), \theta_j\}$, $best_{cost} \leftarrow l_c(r) +$
$c''\left(\tau_i, \theta_j\right)$
End for
Return $(best_{sch}, best_{cost})$
```

Fig. 25. A function to solve the weight constrained shortest path problem for a fixed depot.

```
Function DeleteLabels
On input set $L$
$L \leftarrow L \setminus \{r | r \in L, L_f(r) > \omega\}$
Sort $L$ in ascending order of $L_f$ then by ascending order of $L_c$
$L^{'} \leftarrow \emptyset$
$\bar{c} \leftarrow \infty$
For each $r \in L$ (in sorted order)
    If $L_c(r) \geq \bar{c}$
        $L^{'} \leftarrow L^{'} \cup \{r\}$
    Else
        $\bar{c} \leftarrow L_c(r)$
    End if
End for
$L \leftarrow L \setminus L^{'}$
Return $L$
```

Fig. 26. A subfunction of the function in Fig. 25 which deletes dominated and invalid labels.

4.2.3. The branch and bound algorithm

The column generation approach finds a solution to the relaxed formulation of the AF-VSP, so the solution it provides may have a noninteger number of vehicles assigned to a schedule (but between zero and one). Thus, we will use a branch and bound approach when the solution is noninteger. For an introduction to branch and bound techniques see Winston and Venkataraman (Winston and Venkataraman, 2003). We start by storing an upper bound of ∞ for the solution. When a new integer solution is found, if it has a cost lower than the current best cost then it is stored as the new best solution and its cost is used as a new upper bound. When a noninteger solution is generated, if the relaxed cost is greater than the current upper bound the problem is pruned since no integer solutions found by branching will be as good as the current best. If the noninteger solution has a relaxed cost less than the current best upper bound, the problem branches into two new problems. This continues until all of the branches have either ended in integer solutions or have been pruned since their lower bound is too high.

To define the branching procedure, note that in a noninteger solution to the relaxed AF-VSP there must be either a pair of service trips (τ_j, τ_i) or a depot and a service trip (θ_k, τ_i) such that the schedules in the solution that contain the pair have a noninteger sum of solution values. A pair is contained in a schedule if either both elements fall next to each other in the schedule, or there is only a fuel station between them in the schedule. In the case that it is a pair of service trips (τ_j, τ_i) the problem can be branched into two new problems. In one branch, any schedule that serves trip τ_i must serve trip τ_j as the previous service trip, although stopping to refuel between the service trips is allowed. Thus a schedule that has a vehicle serve trip τ_j, refuel at station σ_k, then serve trip τ_i would be acceptable, but a schedule that has a vehicle serve trip τ_k for $k \neq j$ then trip τ_i would not be allowed. In the other branch, a schedule must not have service trip τ_j followed by service trip τ_i, even with a refueling stop between them. In this sense we are branching on the service trip pair (τ_j, τ_i) where one branch must include the pair (τ_j, τ_i) in a schedule in the solution, and in the other branch all schedules in the solution must exclude pair (τ_j, τ_i). In the event that the pair is a depot and a service trip (θ_k, τ_i) the branching uses the pair. In one branch any vehicle serving trip τ_i must be stationed at depot θ_k and serve trip τ_i as its first service trip in the schedule. In the other branch a vehicle cannot serve trip τ_i as the first trip after leaving depot θ_k.

As the branch and bound algorithm progresses the problems to be solved will have a longer and longer list of pairs that must be included or excluded. This data can be entirely stored as a subset $\Gamma \subseteq (T \cup D) \times T$ representing which pairs should be excluded from any schedule in the problem. In the case a pair (a, b) is excluded then that pair is an element of Γ. If a pair (a, b) is included in the solution then any pair with the second element of b but the first element not equaling a is in Γ; i.e. $\big((T \cup D) \setminus \{a\}\big) \times \{b\} \subseteq \Gamma$. Thus, when the algorithm branches on pair (a, b) at a problem with excluded set Γ then two new problems need to be solved: one with excluded pair set $\Gamma_0 = \Gamma \cup (a, b)$ representing removing pair (a, b) from the possible solutions and one with excluded pair set $\Gamma_1 = \Gamma \cup \big((T \cup D) \setminus \{a\}\big) \times \{b\}$ representing forcing (a, b) to be in the solution.

When the two new problems are set up to be solved after branching at a previous problem, the column generation algorithm must be run on both of them. However rather than starting with only the initial dummy variables in the solution as discussed in Section 4.2.1, the columns from the previous problem can be used. The only exception is the any of the columns in the previous problem that violated the excluded pair sets need to be removed. It is likely that only a small amount of additional columns need to be generated before each of the new problems will find a solution. To generate columns that do not violate the excluded pairs, the labeling algorithm from Section 4.2.2 needs to be adjusted. This can be done simply by when aggregating the sets of labels for each service trip, excluding labels from trips that would include excluded pairs. For example, if (τ_j, τ_i) is an excluded pair, then when finding the label set $L(\tau_i)$ none of the labels from $L(\tau_j)$ nor $L\left(h(\tau_j, \sigma_k)\right)$ for any k can be used.

4.3. A heuristic algorithm for AF-VSP

Due to the fact that both AF-VSP and WCSPP-R are NP-hard problems, the column generation algorithm will become prohibitively time consuming to run for problems with large numbers of trips. Thus, it is important to have heuristic algorithms available, especially since modern fleets may need to serve thousands of trips in a day. Here we present a heuristic algorithm for the AF-VSP based on the *concurrent scheduler algorithm* (Bodin et al., 1978). The concurrent scheduler algorithm is a heuristic algorithm used to solve constrained vehicle scheduling problems such as the VSPLPR. The algorithm assigns the first service trip to vehicle 1, then iterates through the remaining service trips. For each service trip, if it is feasible to assign it to an existing vehicle, assign it to the vehicle that adds the least cost to the total problem. If no vehicle can serve the service trip given their current assignments, or if it would be cheaper to assign it to a vehicle that is currently assigned to no service trips, assign the trip to a new vehicle stationed at one of the depots that have unassigned vehicles. We will need a method for taking trips assigned to a vehicle and determining if they are feasible to serve given fuel constraints, and if so where the vehicle should stop to refuel. We will refer to such a method as the Fuel Scheduling Algorithm (FSA), which we will devise later. Given such an algorithm exists, we can use it to assign refueling stops to schedules and determine the costs of serving the schedules with alternative fuel vehicles. Using the FSA the concurrent scheduler heuristic is as follows:

1. Assign the first service trip τ_1 to vehicle 1. For each depot with available vehicles, use the FSA to determine which stations to refuel at if the vehicle were stored at

that depot. For the depot with the minimum cost assignment, assign the vehicle to that depot. Set $i = 2$.

2. Determine which assigned schedules are compatible with service trip τ_i (i.e. which vehicles assigned to routes could serve trip t_i without violating time constraints). Find the cost of assigning the trip to each compatible vehicle using FSA. Also, use the FSA to find the cost of assigning a vehicle stored at each of the depots with available capacity. Assign the trip to either the already assigned vehicle with a minimum cost, or if it is cheaper assign the service trip to a new vehicle from a depot with available capacity,
3. Set $i = i + 1$. If $i > n$ stop and return the assignment, otherwise go to 2.

While this algorithm is likely to provide solutions that are suboptimal, it has the advantage of being extremely fast. Each service trip has to be analyzed only once, and for each service trip the fuel sequencing problem has to be solved only once for each depot with an available vehicle and once for each vehicle currently assigned to schedules where the most recently assigned trip is compatible with the current one. Further, this computation can be made especially quickly by storing information on the schedules after assigning each trip. Since this algorithm runs quickly, it can be used to generate a feasible starting point for more effective algorithms such as large neighborhood search.

The concurrent scheduler algorithm requires that we have the fuel scheduling algorithm: a method of taking a vehicle assigned to a sequence of service trips and determining where and when the vehicle should stop to refuel with minimal cost. Such a method would be useful for other algorithms as well since it would allow for first solving a problem by ignoring the fuel constraints, and then including them later. It is possible that there are several ways to assign fuel stations to a sequence of trips to make the trips feasibly serviced by a vehicle with limited fuel capacity, and thus we desire the one with minimal cost. It also is possible that the set of service trips a vehicle is serving cannot be feasibly visited by using any combination of fuel stations, for instance if there is a trip that has a fuel requirement greater than the fuel capacity of a vehicle. The problem of finding which fuel stations to stop at is a special case of AF-VSP where $d = 1$ and $r_1 = 1$, since the problem is to find the lowest cost schedule for a single vehicle to serve a set of routes from a given starting depot. This problem is not unlike the *simple fixed-route vehicle refueling problem* (Lin et al., 2007), where a vehicle is on a fixed schedule and the amount to refuel at each stop needs to be determined. In this case however, a vehicle needs to detour to refuel and has to choose a single station out of a set of possible ones, and so the choice of where to refuel is more constraining.

Since this is a special case of the AF-VSP with a single vehicle, we can simplify the column generation subproblem algorithm to solve it. Because only a single vehicle can be used, the vehicle must be assigned to all of the service trips, and thus the objective is to find a weight constrained shortest path with replenishment, similarly to the column generation subproblem from Section 4.2.2. In fact the only difference is that in the fuel station sequence problem, each service trip *must* be visited by the vehicle being assigned to the path, while the general column generation subproblem may allow for service trips to be skipped. The graph G for the column generation subproblem can be used here as well. An example of a special case of G for an AF-VSP network where $d = 1$ and $r_1 = 1$

can be seen in Fig. 27. Thus, the fuel station sequence problem can be solved using a similar algorithm to the column generation subproblem, with only a minor change. Again we use a labeling algorithm and the same labeling system as Section 4.2.2 where each label is of the form (c, f, sch). Now the labeling algorithm on page 96 can be performed after changing step 1 for each service trip to:

1. Do one of the following:
    a. In the case that this is the first service trip, generate the labels for the vertex associated with the trip by taking the union of the label associated with the schedule coming directly from the depot and the labels from vertices representing coming from refueling stations immediately after the depot.
    b. In the case that this is the not the first service trip, generate the labels for the vertex associated with the trip by taking the union of the set of labels from the previous trip in the sequence and the labels on vertices representing refueling stations visited after the previous trip.

This updated algorithm is the FSA. The change to step 1 forces the minimum length path to visit each service trip in the sequence, but otherwise the algorithm remains the same. Labels are associated with each vertex, and the vertices associated with each refueling station have a single label that dominates the rest. The algorithm now either returns the minimum cost path which contains the stations the vehicle should visit (or the algorithm finds no solution if there is no way to satisfy the fuel constraints while serving the set of trips). The algorithm can also store the labels for later use. This can be useful if additional service trips may be added to the end of the schedule, since all of the labels from earlier service trips will stay the same.

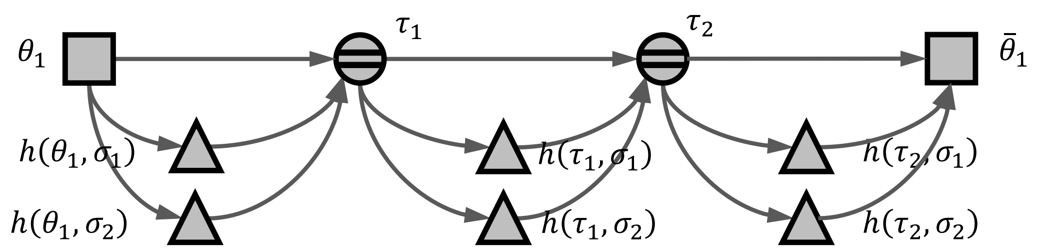


Fig. 27. Graph G for an example AF-VSP where $d = 1$ and $r_1 = 1$ for finding where on a sequence of service trips to stop and refuel.

Theorem: This algorithm solves the fuel station sequencing problem in $\mathcal{O}(bn^2 \log(bn) + b^2 n)$ time.

Proof: Recall that the label sets associated with the vertices in H each have a single label since the vehicle refuels at the station so the lowest cost label will dominate the rest. Thus for vertex τ_i the label set $L(\tau_i)$ has a size at most $|L(\tau_{i-1})| + b$ for $i > 1$, and when $i = 1$, $|L(\tau_1)| = b + 1$ since the only labels are from visiting the fuel station or arriving directly from the depot. This bounds the label set size to $|L(\tau_i)| \leq b(i-1) + 1$, and for each trip vertex the labels need to be sorted. Since there are n service trip labels each needing to be sorted, finding the labels for all of the service trips will require $\mathcal{O}(bn^2 \log(bn))$ computations. Since the labels associated with refueling stations do not need to sort the labels and instead only need to find the minimum cost label, a label for a station following service trip τ_i will require $\mathcal{O}(bi)$ computations. Since there are $b \cdot (n + 1)$ vertices in H, when all of the calculations attributed to the refueling station vertices are combined the computations will require $\mathcal{O}(b^2 n)$ time. Therefore the total complexity of the algorithm is $\mathcal{O}(bn^2 \log(bn) + b^2 n)$. ■

Additionally, in the case that the minimum fuel requirement of a service trip is bounded below by $\gamma > 0$, then the FSA solves the problem in $\mathcal{O}(b^2 n)$ time. Since the fuel requirement is bounded, a vehicle cannot serve more than $\left\lfloor \frac{\omega}{\gamma} \right\rfloor$, trips before refueling, and thus any schedule that has q trips between two stations is infeasible where $q = \left\lfloor \frac{\omega}{\gamma} \right\rfloor + 1$. Therefore for any trip τ_i, the labels associated with the trip must not include any partial schedule where the vehicle has not stopped at any of the last q possible refueling stations. Suppose that $L(\tau_i)$ does include a label r where the schedule includes vertex $h(z, \sigma_k)$ for some z and k, and that the schedule includes not refueling station vertices after that one. In this case we know that r is the only label that has $h(z, \sigma_k)$ as the most recent fuel stop, since all other labels of that type will be dominated. Therefore the number of labels for $L(\tau_i)$ is bounded and is a constant with respect to n. Thus, the computational complexity of finding the labels associated with service trip vertices is only $\mathcal{O}(bn)$. Since for each service trip the following vertices representing fuel stations need to find the minimum cost labels and there are s fuel station vertices for each trip, the fuel station labels in total require $\mathcal{O}(b^2 n)$ time. $\mathcal{O}(b^2 n)$ dominates the computational complexity in this case.

4.4. Heuristic algorithms for AF-VSP based on MDVSP

There are many different heuristics that have been proposed for the multiple depot vehicle scheduling problem, see the work of Pepin et al. (Pepin et al., 2009) for a recent comparison of five of them on different sets of data. One possible method for finding good heuristics for the AF-VSP would be to modify MDVSP heuristics to add the fuel constraint. Three of the algorithms Pepin et al. analyzed are based on set-partitioning formulations of the MDVSP, and because the AF-VSP formulation we proposed is of the set partitioning type they provide good candidates for modifying into AF-VSP algorithms. The three heuristic algorithms are:

1. truncated column generation,
2. large neighborhood search, and
3. tabu search.

We will briefly discuss the merits of each of these algorithms individually as applied to the AF-VSP. For more in-depth explanations for the heuristics see the Pepin et al. paper. Each of these methods holds potential to be used with the AF-VSP, however since they each have their own advantages and disadvantages investigating their use would be a good area for future research.

4.4.1. Truncated column generation

When using truncated column generation for the MDVSP, rather than repeating the column generation algorithm until the next candidate variable to add has a value of $q^* \geq 0$, the algorithm instead stops when the value of the solution to the restricted master problem has not decreased by more than z_{min} over a length of I iterations where z_{min} and I are predetermined parameters. Since the column generation algorithm tends to converge to a solution slowly when nearing completion, this alteration will cause the algorithm to terminate when the solution seems to be close to optimal which dramatically lowers computation time. Once a solution to the relaxed problem is found, if the solution is noninteger then one of the nonzero and noninteger decision variables can be set to 1 and the problem can be resolved. So rather than branching by attempting to both include and exclude noninteger decision variables, instead the heuristic solution will assume some of the variables need to be included.

The truncated column generation algorithm can easily be applied to AF-VSP by modifying the stopping criterion for the algorithm in Section 4.2.1. Unfortunately, in the AF-VSP the column generation subproblem is NP-hard, so for large numbers of service trips the time to compute which variables to add to the restricted master problem could make this heuristic infeasible, even for large values of z_{min} and I. For the truncated column generation method to be effective for large numbers of service trips the subproblem of the weight constrained shortest path problem with replenishment would need to be approximated as well.

4.4.2. Large neighborhood search

The large neighborhood search heuristic begins with a suboptimal solution for the MDVSP, then repeatedly takes subsets of the solution and reoptimizes the service trips in that subset. For the MDVSP this involves first finding an assignment of the service trips to vehicles, then taking a the set of schedules assigned to a subset of all vehicles and solving the MDVSP exactly on the service trips in those schedules, using the vehicles assigned to those trips plus whatever other vehicles remain unassigned. Finding the exact MDVSP solution for those subset of trips can be done using an algorithm of the user’s choice such as a column generation approach. For the AF-VSP, the large neighborhood search algorithm involves first finding a suboptimal solution for the entire problem. This can be done using the concurrent scheduler algorithm. Then subsets of the service trips can be reoptimized using the exact column generation algorithm. The number of schedules to reoptimize at each iteration needs to balance the desire to have more schedules to better optimize the problem and the desire to have fewer schedules so that the exact solution is found faster. Rather than finding the exact solution to the subset of the service trips, it may make sense for large problems to approximate the subset solution using a different heuristic such as truncated column generation.

4.4.3. Tabu search

The tabu search technique involves searching the space of all solutions to the MDVSP (including infeasible solutions that have vehicles taking service trips too late) to find the lowest cost feasible one. For an overview of tabu search see (Glover, 1990). Starting at an initial element in the solution space, other neighboring solutions are considered, where a solution is a neighbor if it can be achieved by making a single alteration, or *move*, from the current solution. The lowest valued neighboring solution is selected as the next solution, and the move used to arrive at that solution is added to a temporary list of unavailable moves. The list of unavailable moves is used to avoid repeating solutions. The value of a neighboring solution is a function of the cost of that solution, a penalty depending on if the time constraints of the problem are broken by the solution, a penalty for if the solution was visited before, and a random value to cause the state space to be explored. In the case of the tabu search based MDVSP heuristic in Pepin et al., the two moves considered were reassigning a single service trip to a different vehicle and swapping two service trips between vehicles.

For the AF-VSP again the initial solution can be found using the concurrent scheduler. The two moves from Pepin et al. can also be used in the case of AF-VSP to generate solutions. When calculating the cost change for moving a service trip to a new schedule, the FSA algorithm can be used to compute the cost of the schedule that the service trip is assigned to, and the new cost of the schedule that no longer contains the service trip. Unfortunately to generate the full set of neighboring solutions every possible move must be considered, and so the FSA algorithm may need to be run $O(n)$ times, which may cause the procedure to become burdensome for a problem with a large number of service trips.

4.5. Empirical results

We tested the column generation and concurrent scheduler algorithms in two different ways: we generated random networks to see how the algorithms preformed on a number of different sized problems, and we tested the method on real world bus data from Valley Metro, the regional transit system of the Phoenix Arizona metropolitan area. Valley Metro is a unified organization containing the transit systems of different cities in the Phoenix area, and includes buses, light rail, and ride sharing. For our analysis we focused only on the bus service of Valley Metro. While the exact column generation algorithm was only used on a small subset of the Valley Metro bus service trips, we also tested the concurrent scheduler heuristic algorithm on a large set of Valley Metro trips.

4.5.1. Randomly generated data

First, we tested the two algorithms using randomly generated data created using a modification on the method of (Dell’Amico et al., 1993). Their method was altered so we only considered short service trips since our vehicles would not have enough fuel to serve a long service trip, and we removed the cost of waiting between service trips. To generate a problem for a given n number of service trips, b number of fuel stations, and d depots, a set of v *relief points* was randomly selected, where each relief point was uniformly chosen from a 60 by 60 grid. The relief points served as a set of potential starting and ending locations for the service trips, as well as potential locations for the depot and fuel stations. The number of relief points v was an integer randomly chosen from a uniform distribution spanning set $\left\{\left\lceil\frac{n}{3}\right\rceil, \left\lceil\frac{n}{3}+1\right\rceil, \ldots, \left\lceil\frac{n}{2}\right\rceil\right\}$. The fuel stations were located at randomly selected relief points without replacement, and d of the fuel stations were chosen to also be depots (and thus there must be at least as many fuel stations as depots). For a problem with n service trips and d depots, each depot was given a uniformly random integer number of vehicles in the set $\left\{\left\lceil 3+\frac{d}{2.5d}\right\rceil, \ldots, \left\lfloor 3+\frac{d}{3.5d}\right\rfloor\right\}$. The starting and ending locations of each service trip were also randomly selected from the relief point locations, but unlike fuel stations multiple service trips could share relief points, including relief points that are fuel stations. Let $\theta(a, b)$ represent the Euclidean distance between relief points a and b. The travel time between relief points was taken to be the same as the distance between the points, and refueling was set to require requires 5 units of time with a cost of 150. The fuel capacity of a vehicle ω is set as $\omega = 150$.

For a given service trip $\tau_j \in N$ between relief points a and b, the starting time st_j was an integer chosen randomly with a probability of 15% of being uniformly chosen from in $[420,480)$, a 70% probability of being uniformly chosen from $[480,1019)$, and a 15% probability of being uniformly chosen from $[1020,1080)$. The ending time et_j was chosen as a uniformly random integer from $[st_j + \theta(a,b) + 5, \dots, st_j + \theta(a,b) + 40]$. The fuel spent on service trip τ_j was set as $f(\tau_j) = \theta(a,b)$. For all dead-heading trips, the fuel and costs requirements are given in Table 4. The costs were different than in Dell'Amico et al. since we added a cost of refueling and have removed a cost for a vehicle waiting between service trips. The cost of waiting was removed since it would not be possible to factor in the time for edges between service trips and fueling stations. We also added an additional capital cost of 2000 for each bus to be used.

Table 4
The cost and fuel requirements for dead-heading trips in the randomly generated data

Origin	Destination	Fuel	Cost	Description
$\tau_i \in T$	$\tau_j \in T$	$\theta(\tau_i, \tau_j)$	$10\theta(\tau_i, \tau_j)$	service trip to service trip
$\tau_i \in T$	$\sigma_j \in B$	$\theta(\tau_i, \sigma_j)$	$10\theta(\tau_i, \sigma_j)$	service trip to fuel station
$\tau_i \in T$	$\theta_j \in D$	$\theta(\tau_i, \theta_j)$	$10\theta(\tau_i, \theta_j)$	service trip to depot
$\sigma_i \in B$	$\tau_j \in N$	$\theta(\sigma_i, \tau_j)$	$150 + 10\theta(\sigma_i, \tau_j)$	fuel station to service trip
$\sigma_i \in B$	$\theta_j \in D$	$\theta(\sigma_i, \theta_j)$	$150 + 10\theta(\sigma_i, \theta_j)$	fuel station to depot
$\theta_i \in D$	$\tau_j \in N$	$\theta(\theta_i, \tau_j)$	$2000 + 10\theta(\theta_i, \tau_j)$	depot to service trip
$\theta_i \in D$	$\sigma_j \in B$	$\theta(\theta_i, \sigma_j)$	$2000 + 10\theta(\theta_i, \sigma_j)$	depot to fuel station

All of the results were computed on a personal computer having an Intel core 2 duo 2.4Ghz dual core processor and 4GB of memory that was running Windows 7 Ultimate. All of the algorithms were coded in MATLAB version 2012b. For the column generation subproblem, the shortest routes between the different depots were solved in parallel using the MATLAB Parallel Computing Toolbox.

We used various combinations of numbers of service trips, stations, and depots. For each set number of service trips, stations, and depots, we randomly generated 10 feasible runs and averaged the results of the runs for each of the algorithms. Table 5 shows the results for the column generation algorithm where each row corresponds to the 10 randomly generated instances of a particular problem size.

Table 5
Results from the exact column generation solution of the AF-VSP from Section 4.2 used on the randomly generated data

Number of service trips	Number of stations	Number of depots	Mean runtime (sec)	Mean cost (10^4)	Mean number of buses	Mean relaxation gap (10^{-3})	Mean number of nodes	Max number of nodes	Mean max level	Max level	Mean number of columns
10	2	2	4	3.52	3.2	0.23	5	13	2.5	5	50
10	4	2	2	3.70	3.4	0.01	2	3	1.5	2	33
10	4	4	2	3.86	3.6	0.03	3	7	1.7	3	53
10	8	4	3	3.65	3.4	0.17	3	5	1.8	3	55
20	2	2	76	6.02	5.4	0.50	147	977	7.2	14	693
20	4	2	17	6.32	5.8	0.38	20	57	4.9	11	174
20	4	4	13	5.84	5.4	0.16	11	43	3.5	6	186
20	8	4	10	5.55	5.2	0.06	6	15	2.8	5	154
30	2	2	89	8.00	7.3	0.28	102	371	8.0	12	736
30	4	2	49	9.15	8.4	0.15	54	271	5.6	13	411
30	4	4	42	8.24	7.7	0.09	26	81	5.1	10	398
30	8	4	34	8.17	7.6	0.07	18	75	4.4	11	307
40	2	2	1091	10.52	9.5	0.29	1100	7001	12.7	20	7677
40	4	2	183	11.08	10.1	0.21	127	315	8.6	14	974
40	4	4	68	10.01	9.2	0.12	25	91	4.8	11	483
40	8	4	52	9.87	9.1	0.07	12	61	3.0	8	368
50	2	2	3769	13.43	12.0	0.07	3577	11925	16.3	25	16516
50	4	2	1616	13.44	12.1	0.05	730	5835	9.0	22	5523
50	4	4	1159	13.49	12.4	0.28	583	4931	13.0	37	6188
50	8	4	273	12.77	11.8	0.23	61	141	8.4	15	1000

As expected, the runtime of the algorithm increased as the number of service trips increased. Interestingly, the instances with the fewest numbers of stations and depots tended to have the longest runtime, since the most columns needed to be generated by the algorithm before a solution was found. The set of sample problems with 50 service trips, 2 stations, and 2 depots took the longest to run, on average over an hour. This average hides much of the variability: within those 10 runs the fastest took only 75 seconds and the slowest took over 2.5 hours. Table 6 shows a comparison of the concurrent scheduler heuristic to the column generation algorithm. The solution provided by the heuristic algorithm had on average a 3.8% higher cost than the optimal solution, and the gap increased with the number of service trips. However the runtime of the heuristic solution was dramatically lower than that of the exact solution: while the heuristic never take more than a second to run the runtimes for the exact algorithm went into minutes on the larger sample problems. Out of the 200 $(20 \cdot 10)$ runs, in only 12 of them did the heuristic solution use more buses, and in each of those runs the heuristic solution used only a single additional bus.

Table 6
A comparison of the exact column generation algorithm to the concurrent scheduler heuristic algorithm on the randomly generated data

			Column generation			Concurrent scheduler				
Number of service trips	Number of stations	Number of depots	Mean runtime (sec)	Mean cost (10^4)	Mean number of buses	Mean runtime (sec)	Mean cost (10^4)	Mean number of buses	Increase in cost for heuristic	Increase in number of buses for heuristic
10	2	2	4	3.52	3.2	0.02	3.65	3.3	3.6%	3.1%
10	4	2	2	3.70	3.4	0.03	3.85	3.5	4.2%	2.9%
10	4	4	2	3.86	3.6	0.04	3.95	3.6	2.4%	0.0%
10	8	4	3	3.65	3.4	0.05	3.72	3.4	2.2%	0.0%
20	2	2	76	6.02	5.4	0.05	6.16	5.4	2.4%	0.0%
20	4	2	17	6.32	5.8	0.06	6.47	5.8	2.5%	0.0%
20	4	4	13	5.84	5.4	0.09	6.12	5.5	4.9%	1.9%
20	8	4	10	5.55	5.2	0.11	5.71	5.2	2.9%	0.0%
30	2	2	89	8.00	7.3	0.09	8.32	7.4	3.9%	1.4%
30	4	2	49	9.15	8.4	0.11	9.55	8.6	4.4%	2.4%
30	4	4	42	8.24	7.7	0.15	8.62	7.8	4.7%	1.3%
30	8	4	34	8.17	7.6	0.18	8.41	7.6	3.0%	0.0%
40	2	2	1091	10.52	9.5	0.15	10.88	9.5	3.4%	0.0%
40	4	2	183	11.08	10.1	0.16	11.39	10.1	2.7%	0.0%
40	4	4	68	10.01	9.2	0.21	10.43	9.2	4.2%	0.0%
40	8	4	52	9.87	9.1	0.25	10.35	9.2	4.8%	1.1%
50	2	2	3769	13.43	12.0	0.21	14.11	12.2	5.0%	1.7%
50	4	2	1616	13.44	12.1	0.22	13.93	12.2	3.6%	0.8%
50	4	4	1159	13.49	12.4	0.30	13.97	12.4	3.6%	0.0%
50	8	4	273	12.77	11.8	0.33	13.40	11.9	4.9%	0.8%

4.5.2. Data from the Valley Metro bus service

All the necessary information about the service trips served by Valley Metro buses is publicly available on the city of Phoenix website (City of Phoenix, 2013). This includes the start and end locations, trip times, and the trip paths themselves (of which the length of the service trip can be calculated). Four depot locations were found based on publicly available information on where the Valley Metro contractors First Transit and Veolia have facilities located. The four depots were chosen to be fueling stations, and four additional fueling station locations were selected. The four Valley Metro Transit Centers: Chandler Fashion Center, Metrocenter, Paradise Valley Mall, and Superstition Springs were selected as likely candidates to have refueling infrastructure due to their locations relative to the depots.

The Valley Metro trips are organized into routes, where a route was a set of service trips all which shared origin and destination points (although they could be traveled in either direction). For example Valley Metro route 30 traverses University Dr. in Tempe and Mesa in both the East and West directions. For our analysis we used routes numbered 1-575 in the Valley Metro data. This contained all of the standard bus routes, but excluded some others such as the Orbit routes which have specialized buses, and the highway rapid transit buses. This amounted to a total of 72 routes and 4,373 service trips. Unfortunately the Valley Metro data only contained information about the service trips themselves and not about the dead-heading trips between them, so to obtain this data we used the Bing Maps API (Microsoft, 2013). A map of the 72 routes and the stations and depots can be seen in Fig. 28.

The buses were assumed to have a range of 120 kilometers before needing to be refueled, and refueling would take 10 minutes. Each bus was assigned a cost of 500 and refueling had a cost of 50, which was added to the problem formulation in the method from Section 4.1. The cost and fuel requirement of each service trip and dead-heading trip was taken to be the physical distance of the trip taken on roads. The travel time of each service trip was found in the Valley Metro schedule and the travel time of each dead-heading trip was found using the Bing Maps API.

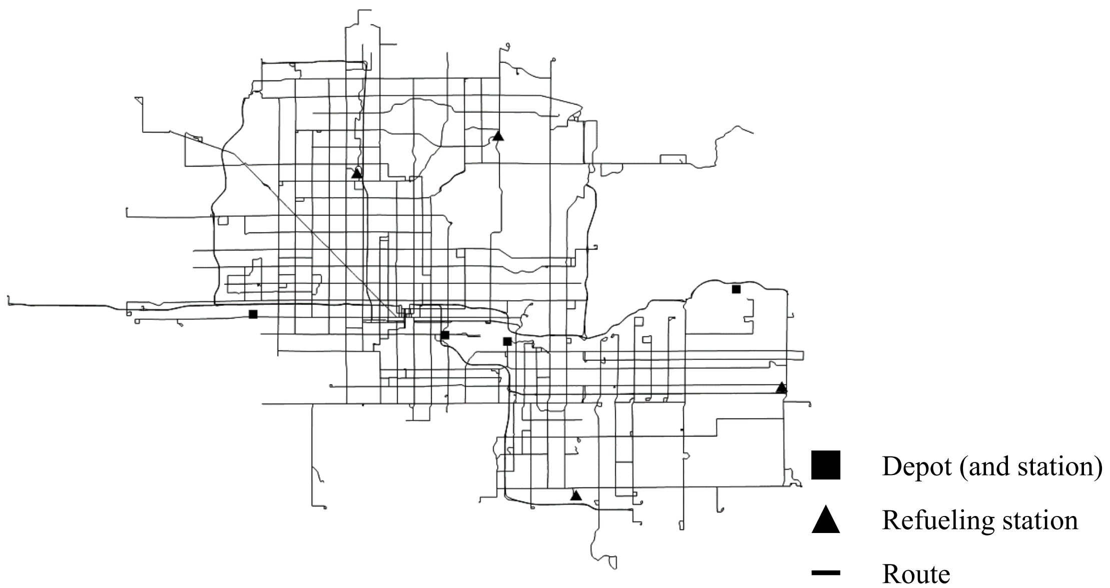


Fig. 28. The Valley Metro network and station and depot locations.

4.5.2.1. Exact vs heuristic algorithms on subset of Valley Metro data

The column generation and concurrent scheduler algorithms were tested on a subset of the 4,373 Valley Metro service trips, since the column generation algorithm would be too time consuming to run on the full set of service trips. To lower the total number of trips, one service trip was randomly selected from all of the service trips associated with each route, which generated a problem with 72 service trips, 8 stations, and 4 depots. This random selection was done 5 times to create 5 distinct problems to test on. Each depot was assumed to have 10 buses available at it. The results of the algorithm can be seen in Table 7, where each row corresponds to one of the five randomly selected sets of service trips. The cost of the heuristic solution provided by the concurrent scheduler algorithm was between 8.1% and 18.7% higher than the cost of the exact solution from the column generation algorithm, however the concurrent scheduler took less than a second to run and the column generation algorithm took between 2 and 12 hours. Interestingly, both algorithms used the same number of buses for each solution in all but one case.

Table 7
The results of the testing the column generation and concurrent scheduler algorithms on a subset of the Valley Metro data

	Column generation							Concurrent scheduler			
Run number	Runtime (sec)	Cost	Number of buses	Relaxation gap (10^{-3})	Number of nodes	Max level	Number of columns	Runtime (sec)	Cost	Number of buses	Increase in cost for heuristic
1	9971	8964	13	1.77	1941	29	28153	0.63	9883	13	10.3%
2	14901	8077	11	3.03	1707	32	33408	0.60	9591	12	18.7%
3	24134	10239	16	1.59	5385	58	61578	0.74	11333	16	10.7%
4	32851	10803	17	3.52	7467	45	81609	0.85	11998	17	11.1%
5	37183	10614	17	3.66	7153	46	80948	0.79	11468	17	8.1%

4.5.2.2. Heuristic algorithm on all Valley Metro service trips for routes 1-575

The concurrent scheduler algorithm was used on the full 4,373 service trips. In 2012, Valley Metro had a total of 889 buses in its fleet (Valley Metro, 2013), however it is unclear how many of those were assigned to serve the 72 routes in this problem and how many were assigned to other routes. Thus, for this problem we assumed there were 600 available buses assigned evenly across the four depots. The results of the algorithm can be seen in Table 8. Of the 600 buses, only 477 of them were assigned, and it took 290 seconds to run the algorithm.

Table 8
Results from applying the concurrent schedule algorithm to the 4,373 service trips from the Valley Metro data

Metric	Value
Number of service trips	4373
Number of stations	8
Number of depots	4
Cost	359827
Number of available buses	600
Number of buses used	477
Runtime (sec)	290

4.6. Conclusion

Scheduling is an important problem in transportation systems operating a fleet of vehicles, and switching fleets to use alternative-fuels can create new logistical challenges in those systems. We have introduced the alternative-fuel multiple depot vehicle scheduling problem, a new generalization to the MDVSP where each vehicle has a limited fuel capacity and there are limited stations available for the vehicle to refuel. We have formulated the problem as a binary integer program, and an exact column generation algorithm and a heuristic algorithm to solve the problem were developed. These algorithms were compared on randomly generated data and real world instances from Valley Metro, which provided promising results for both algorithms. Further research areas include finding heuristic algorithms that provide better solutions and modifying the vehicle assignments to a particular problem when there is a sudden change in the network due to breakdowns or accidents. Additionally, the AF-VSP can be modified to also solve the decision problem of deciding where to place a limited number of fuel stations, which is discussed in Section 5.3.

Chapter 5

EXTENSIONS

In this chapter we will provide additional work on the topics from each of the three previous chapters. These additions should both provide new and interesting extensions for the previous chapters, as well as set up areas for future work. They take the concept of each chapter and alter it slightly to provide results that should be practical. Each section in this chapter reflects the extension for one of the previous chapters and includes results tested on sample data.

5.1. Electric vehicle shortest walk extensions

The most obvious way to extend the electric vehicle shortest walk problem is to add stochasticity to the problem. In Chapter 2, the arc lengths of the network were assumed to be deterministic. This is unrealistic since the time it takes to traverse a road is dependent on varying factors such as traffic and the weather. There are many possible ways of extending the problem to allow for stochastic shortest edges. Possibilities include having the arc lengths be random variables that have known distributions and reset every time the edge is traversed, or having a set of possible configurations for the arc lengths where one configuration is selected before routing begins (but what configuration the system is in is still unknown). In the stochastic case, the route the vehicle plans to traverse may change as it moves through the network due to the driver gaining more information or having an unexpected amount of battery charge left. Since the route can be altered during traversal due to a change in the state of the system or in the vehicle fuel status, instead of a walk the solution is now a policy that maps the current state of the system and vehicle to the current route the vehicle should take.

Adding stochastic arc lengths to the problem creates an unusual situation since it requires handling the possibility that the vehicle could run out of battery. In the deterministic case it was known with certainty whether or not a walk could be taken without the vehicle running out of battery. In the stochastic case of the problem, since the true amount of battery charge an edge requires may be unknown, the driver either has to accept some risk of running out of battery or never take any route that has a nonzero probability of running out of battery. The problem must be formulated in such a way that it factors into account the driver's appetite for risk, if any.

The problem of finding shortest paths in a stochastic network has a large body of research behind it. Frank presented the problem of finding the probability that paths had a length less than a parameter l (Frank, 1969). Mirchandani (Mirchandani, 1976) discussed the shortest path problem with discrete arc lengths and Eiger et al. (Eiger et al., 1985) addressed the same problems that take into account traveler's attitude towards uncertainty in travel times. Sigal et al. (Sigal et al., 1980) discuss the situation of trying to plan a route along arcs with stochastic lengths before knowing the realization of the arc lengths. In this case the authors were interested in finding the path with the highest probability of being the shortest path. The difficulty in this approach is that the lengths of each path are not independent, since two different paths may share arcs. In their paper they give an analytic method to calculate the path with the highest probability of being the shortest. In the electric vehicle setting this problem is complicated since a path set before the realization of the random variables may not be feasible when the true lengths are revealed. Thus when comparing two paths j and k in graph G, not only would $P(L_j < L_k)$ need to be considered where L_i is the length of path i, but also $P(B_j < B_k)$ would need to be considered, where B_i is the probability that the car will run out of fuel if it takes path i.

Polychronopoulos and Tsitsiklis (Polychronopoulos and Tsitsiklis, 1996) discuss the problem where the arcs are randomly distributed but not necessarily independent; the arc lengths are determined before the path is found however the agent taking the path does not know the arc lengths. It is assumed that the set of feasible realizations of the arc lengths is finite. Thus the agent has to balance taking the path that has the shortest expected length with the ability to explore and gain more information about arcs that have not yet been traversed. This corresponds with decreasing the cardinality of the set of feasible realizations given the known arc lengths. This work is also related to the work of Waller and Ziliaskopoulos (Waller and Ziliaskopoulos, 2002) where they modify the problem to limit the relationships between arcs to those which are adjacent.

The work of Fan et al. (Fan et al., 2005) covers the dynamic policies in a stochastic network to try and maximize the probability of arriving on time. In this problem a vehicle is trying to get from a starting point to a destination in a graph G with probabilistic arc lengths in a way that maximizes the probability of arriving before a certain time b. After traversing each arc, the vehicle can alter the path given the previously realized arc lengths. The ability to alter the path is relevant because it allows the driver to weigh the volatility of the current path against the remaining amount of time before b. In the electric vehicle setting, this problem would also need to add the fact that the driver has to base his or her decision on how much battery is remaining in addition to the remaining time.

In Kolobov et al. (Kolobov et al., 2012) the authors present a framework for stochastic shortest path Markov decision processes where there is the possibility of arriving at a failure state so the path cannot be completed. This is not unlike the situation for electric vehicles, since when the vehicle is stranded due to running out of battery charge it is in a "failure state". They discuss the case when the failure state is given a fixed cost and an infinite cost. Modeling an object as it moves through a system using a Markov decision process has been done before, for example in routing jobs through a factory (Mirchandani and Veatch, 1986).

5.1.1. Problem setup

In this section we will formulate the electric vehicle shortest route problem for the case where the edge lengths are independent of each other with known discrete distributions. The vehicle has the option of recourse: the driver can choose a new path after traversing each edge. The edge lengths are randomly chosen from a given discrete distribution each time the vehicle traverses an edge, so there is no knowledge gained by taking actions when traversing the graph. Let $G = (V, E)$ be a directed graph, where V is the set of vertices and E is the set of arcs. For each $e \in E$, let L_e be a nonnegative integer random variable representing the time required to traverse the edge. Assume that all of the L_i random variables are independent, and that each traversal an edge will correspond with a new realization of the edge length. In this sense the edge lengths "reset" each time they are travelled, and may have a new length the next time they are traversed. Let $B \subseteq V$ be a set of recharging stations on the graph. Let $s \in V$ be the starting point of a vehicle and let $t \in V$ be the desired end point of the vehicle. The charge level of the battery is represented as the amount of time the vehicle can travel for until the battery is empty. When the vehicle battery is full it can travel for ω time, and each time it traverses an edge e it uses L_e battery. If the vehicle battery charge is ever at 0 units of time when the vehicle is at a location without a charging station, then the vehicle is stranded. Upon arriving at a refueling station $v \in B$, the vehicle has the option to recharge which takes g_v time and refills the car to having a full battery. Upon arriving at a vertex, the vehicle has the option of traveling along any of the edges, however it may not alter its path in the middle of an edge.

What road a vehicle drives down alters the probability that the vehicle will run out of battery. Certain drivers have different amounts of risk tolerance: they may be willing to take routes that have a higher probability of having the vehicle run out of charge in exchange for shorter travel times. While there are many different possible methods for formulating risk, we define a driver's *risk profile* as a variable $p \in [0,1]$ which represents the minimum probability of reaching t a driver is willing to tolerate when making each decision. This includes not just the chance of running out of battery during the next edge, but along any edge between the current location of the vehicle and the destination. For example if $p = 0.95$ then the driver would take an action that could have up to a 5% chance of having the vehicle be stranded before reaching the end point, but nothing higher. Notice that each time the vehicle arrives at a vertex it has a new action to take, and thus the value of p is enforced at each action during the trip. In the event that all possible actions have an associated probability of running out of fuel greater than p, the driver will take the action with the lowest probability of running out of fuel. A p-*feasible policy* is a policy that ensures the driver only takes actions which respect their risk profile.

We will construct a three step process for finding an optimal policy for routing an electric vehicle in a network with stochastic edge lengths. The policy will assume that the driver has a specific risk profile for how willing he or she is to have the vehicle be stranded. Actions that are too risky for the driver will be avoided unless no other actions are available. The optimal policy will minimize the expected travel time of the vehicle, under the assumption that if the vehicle does get stranded a second vehicle will deliver a new battery to it with a fixed time penalty. The three step process is:

1. Compute for each location in the network and each possible battery charge level, what is the probability that the vehicle gets stranded assuming the driver is minimizing the probability of being stranded and not the travel time. This is done by formulating and finding the optimal policy for a Markov decision process.
2. Using the results from the previous step, determine which actions should be avoided by the driver to have a p-feasible policy.
3. Construct a new Markov decision process that does not have actions forbidden by the results from the previous step, and accounts for the vehicle getting a

replacement battery delivered if the vehicle is stranded. Find the optimal policy to this MDP to get the solution to the problem.

5.1.2. Step 1: finding the probability of being stranded

Because the knowledge of an edge length is reset after each traversal, how a vehicle has travels to a vertex does not impact future decisions, only where the vehicle is currently and how much battery it has remaining influence future decisions. Since a vehicle cannot alter its course while traversing an edge, when the vehicle is not stranded the state of the system is entirely encoded in what vertex the vehicle is along the graph, and how much battery it has remaining. When the vehicle is stranded, its location no longer matters since it will never reach the end point. Therefore, we can define a Markov decision process to represent the vehicle moving through the network. Let $\mathcal{M} = (\mathcal{S}, A, T, R, S_0, D)$ be a Markov decision process with state set $\mathcal{S}$, action set A, transition function $T: \mathcal{S} \times A \times \mathcal{S} \to [0,1]$, reward function $R: \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$, initial state $S_0 \in \mathcal{S}$ and terminal state set $D \subseteq \mathcal{S}$. The set of possible states $\mathcal{S}$ that the vehicle can be in are:

$$\mathcal{S} = \{(v, x): v \in V, x \in \{0, \dots, w\}\} \cup \{X\},$$

where state X represents the vehicle having run out of battery power while traversing an edge. Let $D = \{(t, x): x \in [0, \omega]\} \cup \{X\}$ be the set of terminal states. The initial state is $S_0 = (s, \omega)$.

In the deterministic case of this problem where all of the edge lengths were fixed, the optimal solution is a walk for the vehicle to take from s to t. The exact route the vehicle would take could be determined before it left the starting point. In this problem, the vehicle may have to alter its course depending on how much fuel it has remaining.

Therefore the optimal solution is a policy for the vehicle to execute. For each $S = (v, x) \in \mathcal{S} \setminus \{X\}$ where $v \in V$ and $x \in \{0, \dots, \omega\}$, the set of actions A_S the vehicle can take at that state is:

$$A_S = \begin{cases} \{z: (v,z) \in E\} \cup \rho & v \in B \\ \{z: (v,z) \in E\} & \text{otherwise.} \end{cases}$$

Here action $z \in V$ represents the vehicle travelling to vertex x, and the action ρ represents refueling if available. Since once the vehicle is stranded it cannot move to any other state, define $A_X = \emptyset$. The set $A = \bigcup_{S \in \mathcal{S}} A_S$ represents all possible actions the vehicle can take.

Let $T: \mathcal{S} \times A \times \mathcal{S} \to [0,1]$ be a transition function representing the probability of the vehicle entering a new state the vehicle after executing an action from its current state. When $S = (v, x) \in \mathcal{S} \setminus \{X\}$ where $v \in V$ and $x \in [0, w]$:

$$T(S, a, S')$$
$$= \begin{cases} P(L_{(v,v')} = x - x') & \text{for } S' = (v', x') \in \mathcal{S}, a = v', (v, v') \in E, c - c' \geq 0, \\ P(L_{(v,v')} > x) & \text{for } s' = X, a = v', (v, v') \in E, \\ 1 & \text{for } s' = (v, \omega)\ a = \rho, v \in B \\ 0 & \text{otherwise,} \end{cases}$$

The transitions from non-stranded states fall into four cases. In the first case the driver takes an action of traversing an edge to an adjacent vertex and arriving with a specified amount of battery. The transition probability in this case depends on the probability that the edge length is the one that leaves the vehicle with the specified battery level. The second case is for the action when the driver tries to traverse an edge to an adjacent vertex but gets stranded while traversing the edge. This transition has a probability depending on how likely the vehicle is to run out of battery. The third case is when a vehicle is at a vertex that is a charging station, and the vehicle takes an action of recharging. This action has a guaranteed result so the transition probability is 1. Finally, any other transition between states will not occur and has probability 0. If the vehicle is in the stranded state then since it is a terminal state there are no possible actions.

Finally, a policy is a function $\pi: \mathcal{S} \to A$ such that $\pi(S) \in A_s$ for all $S \in \mathcal{S}$. What the vehicle does upon arriving at a vertex depends on how much fuel is left in the vehicle's tank. We are interested in finding a policy that gets the vehicle to its destination, which is represented by any of the states in $D \setminus \{X\}$, with maximal probability

Theorem: There exists a policy π^* that minimizes the probability that the vehicle reaches the terminal state B instead of terminating states $\{(\tau, x): x \in \{0, \dots, \omega\}\}$.

Proof: Construct a reward $R: \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}^+$ function defined as follows:

$$R(S, a, S') = \begin{cases} 1 & \text{when } s' \in \{(\tau, x): x \in \{0, \dots, \omega\}\} \\ 0 & \text{otherwise.} \end{cases}$$

Since the only nonzero reward is incurred at a terminal state, a dynamic programming algorithm to find a policy π^* that maximizes the expected sum of the rewards will converge to the utilities $U(S)$ for each $S \in \mathcal{S}$. The utility $U(\mathcal{S})$ corresponds with the probability that the vehicle will reach the end point starting from state s and applying policy π^*. By the construction of the dynamic programming algorithm, policy π^* minimizes the probability that the vehicle will be stranded starting from any state. ■

5.1.3. Step 2: finding the forbidden actions

For action $a \in A_v$ taken from state s, let $Q(S, a)$ be the probability that the vehicle will be stranded at any point after taking that action, assuming the vehicle follows policy π^*. The value $Q(S, a)$ is then

$$Q(S, a) = \sum_{S' \in \mathcal{S}} U(S')T(S, a, S')$$

The willingness for a driver to take a route that could cause the vehicle to be stranded depends on his or her tolerance for risk. Each time the driver is in state s chooses to take an action a, there is an associated maximum probability that the vehicle will reach the destination over all possible policies. That probability is precisely $Q(S, a)$. Given that the driver is in state s and takes action a, if the driver is only interested in arriving at the destination and not how long it takes to get there then the driver will reach the destination with probability $Q(S, a)$. It may be possible that the driver implements some policy $\pi' \neq \pi^*$, in which case the probability that the vehicle will reach the destination instead of being stranded after applying action a from state s is less than or equal to $Q(S, a)$.

For each non-terminal state $S = (v, x) \in \mathcal{S} \setminus D$, the *p-feasible actions* $\bar{A}_S \subseteq A_S$ are the actions that can be used in a p-feasible policy. Action set $\bar{A}_S$ can fall into three cases:

1. There exists a nonempty maximal set $A'_S \subseteq A_S$ such that $Q(a, s) \geq p$ for all $a \in A'_S$. In this case the feasible actions from S are $\bar{A}_S = A'_S$. This is the case where there are some actions that have a probability at most p of leaving the vehicle

stranded in the future. Any of these actions with probability at most p are feasible.

2. There does not exist a nonempty set $A'_S \subseteq A_S$ such that $Q(a, S) \geq p$ for all $a \in A'_S$, but there does exist $a \in A_S$ such that $Q(a, S) > 0$. This corresponds to the situation where no action ensures the vehicle arrives with the desired probability, but there is a least one action that is not guaranteed to strand the vehicle. This situation is broken into two subcases depending on whether or not the vertex v of the state is a refueling station. If it is a refueling station and $x < \omega$, then $\bar{A}_S = \{\rho\}$, i.e. the vehicle must refuel. If $x = \omega$ then $\bar{A}_S = \left\{\operatorname{argmax}_a\{Q(a, S): a \in A_S, a \neq \rho\}\right\}$, so the vehicle must traverse the edge with the highest probability of getting the vehicle to the destination. If the vertex v is not a refueling station then $\bar{A}_S = \left\{\operatorname{argmax}_a\{Q(a, S): a \in A_S\}\right\}$, so the vehicle must also take the edge with the highest probability of reaching the destination.
3. $Q(a, S) = 0$ for all actions $a \in A_S$. When the vehicle is in this state, regardless of what actions are taken the vehicle is going to be stranded. Here $\bar{A}_S = A_S$, since regardless of what action is taken the vehicle will run out of battery and be stranded, and thus the driver is allowed to choose any of them.

To find a p-feasible policy, we can use standard techniques to find policies such as value iteration, with the adjustment that at no point do we allow the agent to take actions that are not p-feasible.

5.1.4. Step 3: computing the value of a p-feasible policy

The classic method of measuring the value of a policy is the one that minimizes the expected cost, or in this case the expected travel time. Unfortunately, just because a policy is p-feasible does not guarantee that by using the policy the vehicle will reach the destination. Therefore when computing the expected travel times the situations where the vehicle does not reach the destination have to be considered. There any many different possible methods for computing the value of a policy. One possible approach as discussed in Kolobov et al. (Kolobov et al., 2012) is to associate a penalty d for reaching the fail state. Additionally, any other state will have a value of at most d, since if an agent gets to that state they would simply give up. Another possible approach would be to have the value of a state represent the expected value given that the agent never reaches the fail state. We will take the approach of removing the failure state entirely and avoiding this problem. We will assume that if a vehicle runs out of battery charge while traversing an edge, the vehicle will pay a large penalty for a new battery to be delivered to it, and the vehicle will then continue. Thus the MDP will have to be reconstructed so that instead of being stranded and ending the trip the vehicle still arrives at a state with nonzero battery charge but with a large negative reward.

In this situation we construct a new Markov decision process to represent allowing the vehicle to have its battery replaced for a fixed cost ρ after it is stranded. Thus, we no longer have a state for the vehicle being broken down, since any time it runs out of battery mid-trip it will get a new battery and continue. Thus we define a new Markov decision process $\mathcal{M}' = (\mathcal{S}', A', T', R', S_0', D')$. Here $S' = \{(v, x): v \in V, x \in \{0, \dots, \omega\}\}$, which is the state of states from $\mathcal{M}$ excluding the state for being stranded. Similarly as before, the initial state $S_0' = (s, w)$ and $D' = \{(t, x) \colon x \in \{0, \dots, w\}\}$. The set of actions A' is the same set of actions from $\mathcal{M}$, so $A' = A = \bigcup_{S \in \mathcal{S}'} A_S$. The transition probability function needs to be altered from the original one in $\mathcal{M}$, since now when the vehicle runs out of battery charge mid-trip it needs to still reach the next vertex rather than being stranded. The amount of battery it will have remaining when it reaches the destination will relate to the fact that during the trip the battery was swapped out.

Define the transition probability function $T' \colon \mathcal{S}' \times A' \times \mathcal{S}'$ as, for $(v, x), (v', x') \in \mathcal{S}'$ and $a \in A'$:

$$T'\big((v,x), a, (v',x')\big) = \begin{cases} P(L_{(v,v')} = x - x') & \text{for } a = v', (v,v') \in E, c - c' > 0, \\ P(L_{(v,v')} = x + \omega - x') & \text{for } a = v', (v,v') \in E, c - c' < 0, \\ P\left(L_{(v,v')} \in \{0, \omega\}\right) & \text{for } a = v', (v,v') \in E, x = x', \\ 1 & \text{for } s' = (v, \omega)\ a = \rho, v \in B, \\ 0 & \text{otherwise.} \end{cases}$$

In this function, the probability of transitioning to the new state by traversing an edge will yield a lower amount of battery charge if the edge was traversed without battery swap, which is the first case. If the battery needed to be swapped, the vehicle will arrive with more battery charge remaining, which is the second case. If the vehicle arrives with the same amount of battery charge then either the battery was not swapped or it was swapped and the length of the edge was precisely the amount that requires a full battery, which is the third case.

The reward function needs to account for the length of the edge being traversed plus the cost of swapping a battery mid-trip. The function R' is defined below, where the cases correspond to those of the transition function T':

$$R'\big((v,x),a,(v',x')\big)$$

$$= \begin{cases} -(x-x') & \text{for } a = v', (v,v') \in E, x - x' > 0, \\ -(x+\omega-x') - \rho & \text{for } a = v', (v,v') \in E, x - x' < 0, \\ -P\left(L_{(v,v')} = \omega\right)(\omega+\rho) & \text{for } a = v', (v,v') \in E, x = x', \\ -g_v & \text{for } s' = (v,\omega)\ a = \rho, v \in B, \\ -\infty & \text{otherwise.} \end{cases}$$

This gives a full description of a Markov decision process. The optimal policy can be found using standard techniques such as value iteration. While there is no discount factor, the value iteration algorithm should still converge for any state that could reach a terminal state due to the construction of the reward function, assuming a feasible path exists from the initial state to a terminal state.

5.1.5. Empirical results

We randomly generated networks to find the expected values of the optimal policies. We used MATLAB 2012b on an Intel Core 2 Duo 2.4Ghz (x2) processor machine with 4GB of RAM running Windows 7 Ultimate. The random network was generated in the following manner for input parameters: number of intersections n, number of stations m, and grid size g. First, n points were randomly selected with uniform distribution from $\{1, \ldots, g\}^2$, and the car was set to be able to drive at most distance $\omega = 35$. Of those, m were selected to be refueling stations, and of those that were not refueling stations one was selected to be the start vertex and one was selected to be the end vertex. The edges were selected by taking the Delaunay triangulation of the vertices. For edge $e = (v_1, v_2)$ the distribution X_e was uniform between $\min(d, \lfloor 0.75\|v_1 - v_2\|_2 \rfloor)$ and $\min(d, \lceil 1.25\|v_1 - v_2\|_2 \rceil)$. The penalty for a vehicle requiring a battery to be delivered was a $\rho = 70$ and for all stations $v \in B$ the charge time was $g_v = 10$. For both finding the allowed actions and computing the optimal policy, a value iteration algorithm was used with 150 iterations. Table 9 gives the expected travel times for the vehicles following the optimal policy from the initial state at different levels of p. In most cases the difference between the risk free and high risk policies were small or non-existent, however for some networks such as the one with a grid size of 100, 200 intersections, and 20 stations there was over a 25% difference in expected value between the risk free and high risk policies. In this network allowing for risk would dramatically lower the amount of time a driver would need. The average runtime for the algorithm was 76.0 seconds when there were 100 intersections and 161.0 when there were 200 intersections.

Table 9
Expected travel times for the electric vehicle stochastic shortest path problem

Grid size	Intersections	Stations	Expected value of initial state					
			$p =$ 75%	$p =$ 80%	$p =$ 85%	$p =$ 90%	$p =$ 95%	$p =$ 100%
100	100							
		10	70.8	70.8	70.8	70.8	70.8	70.8
		20	101.9	101.9	104.4	105.6	114.3	126.9
		30	94.6	94.6	94.6	95.1	96.1	97.4
		40	100.9	100.9	100.9	101.0	101.1	101.4
100	200							
		10	107.3	107.3	107.3	107.3	170.3	184.2
		20	95.9	95.9	96.4	96.4	97.6	125.9
		30	75.5	75.5	75.5	75.5	76.2	81.0
		40	70.7	70.7	70.7	70.7	70.7	70.8
200	100							
		10	122.3	122.3	122.6	122.8	122.8	298.3
		20	143.5	143.5	143.5	143.5	144.1	144.1
		30	141.2	141.2	141.2	141.2	141.2	141.2
		40	232.4	232.4	232.4	232.4	232.7	232.7
200	200							
		10	127.8	127.8	127.8	127.8	127.8	127.8
		20	116.2	116.2	116.2	116.2	116.2	116.2
		30	244.3	244.3	244.3	244.3	244.4	244.4
		40	90.8	90.8	90.8	90.8	90.8	90.8

5.2. Online routing and battery reservations model extensions

The routing and reservation system from Chapter 4 made several major simplifying assumptions about the system. The list of assumptions include:

1. the rate at which vehicles arrive is constant throughout the day,
2. vehicles will always start with a full battery and all vehicles have the same battery capacity, and
3. every time a vehicle drops off a battery it will have zero charge.

In the following subsections we will show how the model can be modified to relax these assumptions. Each of the assumptions will be treated independently of the others but constructed in such a way that several of the generalizations can be used at the same time, or even all of them can be used to obtain a more realistic model.

5.2.1. Vehicle arrival rates vary during the day

The model assumed that the rate at which vehicles arrive into the system is constant throughout the day. In addition the distribution of the origins and destinations is constant as well. This is unrealistic since the rate at which vehicles arrive will vary as time progresses, due to factors such as rush hour. The distribution of OD pairs will change as well: in the morning cars will be more likely to begin their work trips in suburban areas heading to work, and similarly in the evening cars will likely be heading home. To improve the model it would be best to allow variable arrival rates to vary over the day.

To do this, we partition the time interval $\{0, \dots, T\}$ into a set of κ intervals $\{\{T_0, \dots, T_1\}, \{T_1, \dots, T_2\}, \dots, \{T_{\kappa-1}, T_\kappa\}\}$ where $T_0 = 0$ and $T_\kappa = T$. During each of these intervals we will have a new rate for vehicles requesting a route for OD pair j. Thus the value p_j^k will represent a the probability that during a unit of time in interval $\{T_{k-1}, T_k\}$, a vehicle will arrive at origin o_j wanting to travel to destination d_j. Similar to before $p_0^k = 1 - \sum_{j=0}^{\sigma} p_j^k$ for $k = 1, \dots, \kappa$. With this modification, the rest of the problem formulation stays the same. We still are interested in finding the optimal policy to route a vehicle that arrives at a particular time with the state of batteries at particular charge levels at the stations. By using different arrival rates for each OD pair in each interval, this captures the need to adjust total demand as a function of the time of day.

The method of using a Markov chance-decision problem to model the system and approximate dynamic programming still apply as well. However we need to adjust the basis functions to reflect the fact that the time period is split into intervals. Recall that the basis functions for the original problem were ϕ_0, representing the number of time periods until the end of the day, and $\phi_{(i,j)}$ for $i = 1, \dots, \beta$ and $j = 1, \dots, n_i$, representing the number of time periods in which the station i has at least j batteries unavailable. Now that the time interval is partitioned, one set of these functions is needed for each interval. Thus the function ϕ_0^k will represent the number of time units remaining in time interval k. In the case that the interval has not yet start then $\phi_0^k = T_k - T_{k-1}$ and if the interval has already been completed then $\phi_0^k = 0$. Function $\phi_{(i,j)}^k$ represents the number of remaining time periods during time interval k at which station i has at least j unavailable batteries. Again if the interval of time has been completed then by definition $\phi_{(i,j)}^k = 0$.

In the linear function to represent the state value approximation, there is a coefficient $\theta_{f,k}^{m,1}$ and $\theta_{f,k}^{m,2}$ for each $f \in \mathcal{F}$ and each $k = 1, \dots, \kappa$. This allows the state approximation to treat the number of batteries available in each interval separately. This is justifiable since during each of the intervals the rate at which vehicles will arrive is different. Further, the number of intervals can be taken to be sufficiently small such that complex demand functions can be sufficiently approximated by discrete time periods. Also in situations where there is a time interval with a dramatically varying demand rate, having small time intervals will allow for better fidelity in having the policies change before of after peak demand times. If the intervals were to be taken such that each interval is only one time period long, then this would equivalent to not assuming that $\theta_{tf}^{m} = \theta_{f}^{m}$ for all $t \in \{0, \dots, T\}$ and thus treating each time period independently.

Having different coefficients for each time interval has little impact on the runtime of the approximate dynamic programming algorithm. The only changes in runtime occur in calculating the value of the states that could be transitioned to when simulating the system to approximate the state values, however that time increase is small compared to the total runtime. The nonnegative least squares component of the optimization is most heavily effected by the increase in the number of coefficients, since the number of coefficients is a multiple of the number of intervals.

5.2.2. Vehicles start with different battery charge levels

Suppose we wanted to relax the assumption that vehicles will always begin trips with full batteries. While the vehicles may often start with a full charge since they can usually be plugged in at the origin, this may not always be the case. Further, the distribution may vary as a function of the origin location. An origin that is a residential location may be more likely to start with a fully charged battery than a vehicle starting at a parking structure that does not have a charging station.

For OD pair j assume that the vehicle will start with a battery charged to $\chi_j \in \{0, \dots, \omega\}$. Thus if there are to be k possible battery levels for a vehicle to start with at a origin, then the origin pair should be replicated into k pairs where each has a different starting battery level. Now for each OD pair j the route set $\mathcal{R}_j$ should be generated to reflect the amount of starting battery power in the vehicle; no OD pair should have a route assigned to it that would be infeasible for it to serve. For the k OD pairs that share the same origin and destination vertices but have different starting battery levels, the route sets can be generated simultaneously by first assuming that the battery has full charge and generating the route set, then for each pair with a lower battery level pruning all of the infeasible routes.

5.2.3. Batteries have charge left in them when they are dropped off

Allowing vehicles to drop off partially charged batteries is a simple modification to make. Notice that the charging time of a battery does not change the time required to traverse a route given the current state of the system, nor does it change whether or not a vehicle can traverse a route. When routing a vehicle, the only thing that changes due to this modification is the state that the system will transition to. When a battery is dropped off at station it may require less time to charge since it may not me empty, and thus the state the system transitions to will have more time periods where the battery is available than in the original model.

Suppose a vehicle at time t requests a route wanting to travel OD pair j. The algorithm in Fig. 20 determines the lengths of each route in $\mathcal{R}_j$ given current state of the system S_t, and thus this is the only part of the process that needs to be altered. For route $R =$

$\left(b_1^R, b_2^R, \dots, b_{|R|}^R\right) \in \mathcal{R}_j$, suppose $\delta_1^R . \delta_2^R, \dots, \delta_{|R|}^R$ are the battery levels remaining when the vehicle arrives at stations $b_1^R, b_2^R, \dots, b_{|R|}^R$ respectively. The new algorithm can be seen in Fig. 29. Notice that the only change from Fig. 20 is that the occurrences of h have been replaced with δ_x^R. Now this algorithm will provide an updated state depending on how long the battery will need before being fully charged.

```
On input $S_t, j_t$
Initialize $Y \leftarrow \emptyset, a \leftarrow 0, w \leftarrow 0, i \leftarrow 0,$
Initialize $S_t^{'}$ as a $(T-t)$-tuple of zeros
Initialize $Z$ as a $(\beta+1) \times |\mathcal{R}_{j_t}|$-matrix
For each $R = \left(b_1^R, b_2^R, \dots, b_{|R|}^R\right) \in \mathcal{R}_{j_t}$
        Set $i \leftarrow i+1$
    Set $a \leftarrow l^{'}\left(o_{j_t}, b_1^R\right)$
    Set $S_t^{'} \leftarrow S_t$
    Set $Z(\beta+1, i) \leftarrow l^{'}\left(o_{j_t}, b_1^R\right)$
    For $x \leftarrow 1, \dots |R|$
        Set $w \leftarrow 0$
        While $S_t(b_x^R, y) \geq n_x^R$ for any $y \in \{a+w, \dots, \min(a+w+g+\delta_x^R-1, T)\}$
            Set $w \leftarrow w + w'$ where $w'$ is the count of times $S_t(b_x^R, y) \geq n_x^R$ for $y \in$
            $\{a+w, \dots, \min(a+w+g+\delta_x^R-1, T)\}$
        End while
          Set $a \leftarrow a+w+g$
        Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}(b_x^R, b_{x+1}^R) + \rho_1 g$
        Set $Z(b_x^R, i) \leftarrow \rho_2 w$
          For each $y \in \{a+w, \dots, \min(a+w+g+\delta_x^R-1, T)\}$
            Set $S_t^{'}(b_x^R, y) \leftarrow S_t^{'}(b_x^R, y) + 1$
        End for
        If $x < |R|$ then
            Set $a \leftarrow a + l^{'}(b_x^R, b_{x+1}^R)$
            Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}(b_x^R, b_{x+1}^R)$
                End if
          End for
    Set $Z(\beta+1, i) \leftarrow Z(\beta+1, i) + l^{'}\left(b_x^R, d_{j_t}\right)$
    Replace column $i$ of $L$ with $z$
        Set $Y \leftarrow Y \cup \Lambda(S_t^{'})$
End for
Return $Y, Z$
```

Fig. 29. The algorithm from Fig. 18 with variable battery remaining amounts.

5.2.4. Empirical results

We took the Arizona road network from Section 3.3 and adjusted it to allow for the new relaxed assumptions. The course of the day was split into nine time periods, and the arrival probability increased linearly from 20% of the full value at time period 1 to the full value at time period 5, and then decreased back until it was 20% at the last time period. Instead of the vehicles arriving with a full battery, the vehicles had a 50% chance of starting at full and a 50% chance of starting at half full. The time interval was reduced to having $T = 3240$ time units due to the additional memory requirements for the runs. We assumed that the amount of time it takes to charge a battery is proportional travel time since the last battery swap. We used the parameters $M = 48$, $U = 8$, $\alpha_m = \frac{20}{20+m}$, $\lambda_m = 0.99$, which were similar to those from Section 3.3 except M and λ were adjusted since the algorithm would likely take more iterations to converge. The results can be seen in Table 10. While the algorithm still decreased delays as compared to the greedy policy, the reductions were less significant than the simpler model. It is likely that with further adjustments to the parameters a higher amount of reduction could be observed.

Table 10
The empirical results for the adjusted online routing and reservation system

Measure	Maximum arrival probability in each time period					
	0.025	0.05	0.075	0.1	0.125	0.15
Network						
Number of batteries across stations	1500	1500	1500	1500	1500	1500
Average number of vehicles in day	50.3	94.7	135.0	170.7	235.0	260.5
Greedy policy						
Average total route value of all vehicles	21519	41428	58267	72008	100611	112088
Average route value of each vehicle	8	13	19	23	33	36
Average total delay for all vehicles	2046	4007	5920	7284	10351	11471
Average number of cars with any delays	2877	3078	3047	3088	3072	3085
Algorithm policy						
Average total route value of all vehicles	21416	41214	58204	71757	100592	111892
Average route value of each vehicle	7	13	19	23	33	36
Average total delay for all vehicles	1943	3793	5857	7033	10331	11275
Average number of cars with any delays	2908	3087	3062	3088	3089	3085
Comparison						
Average times policies differed	9.2	22.7	24.2	29.7	33.5	38.0
Percent decrease in delay time	5.03%	5.34%	1.06%	3.44%	0.19%	1.70%
Runtimes						
Routing per vehicle (10^{-2} sec)	0.27	0.14	0.10	0.08	0.06	0.06
To generate coefficients (10^{3} sec)	18161	19543	16475	17365	17290	17681

5.3. Alternative-fuel vehicle scheduling problem extensions

In addition to determining how to schedule a fleet of new alternative-fuel vehicles, organizations implementing alternative-fuel fleets may also need to determine where the refueling stations are to be placed. For instance, when gasoline buses are replaced with those that have swappable batteries, the locations at which the batteries can be swapped may also need to be decided. Rather than first placing the stations then scheduling the vehicles, it may make sense to determine the station locations while building the schedule. This may improve the cost compared to treating the problem independently. This changes the problem from a scheduling problem to a location/scheduling problem.

A similar problem to the vehicle scheduling problem is the vehicle routing problem. In this problem vehicles are stored at a depot and there are a set of customers that each must be visited by a vehicle exactly once, and the vehicles have a limit to the capacity they can serve. There has been some work into locating the depot while simultaneously solving the vehicle routing problem, see Nagy and Salhi (Nagy and Salhi, 2007) for a review of the research. Unfortunately, we were unaware of any research done into solving both the vehicle scheduling problem with the additional decision on where to locate the depot or depots.

For the Alternative-Fuel Vehicle Scheduling and Location Problem (AF-VSLP), we take the AF-VSP and add the ability to determine where to place the stations. There is now a set of b *candidate locations* $B = \{\sigma_1, \dots, \sigma_s\}$ of which up to ζ may be included in the solution. Each candidate station σ_k has a cost of η_k associated with placing a station at that location. The objective is now to find a minimum cost set of schedules and facility locations to serve a given set of trips T.

5.3.1. Problem formulation and column generation algorithm

The integer program representing the AF-VSP location problem is presented below. Again x_p is the decision to have a vehicle serve schedule p in the solution, and Boolean variables v_p^i and u_p^j represent if trip τ_i and depot θ_j are in schedule. We add integer variable z_p^k to represent number of times the station σ_k is in schedule p, and Boolean decision variable s_k represents whether or not to include station σ_k in the solution.

$$\text{Minimize} \quad \sum_{p\in\Omega} q_p x_p + \sum_{k=1}^{s} \eta_k \omega_k \tag{17}$$

$$\text{subject to} \quad \sum_{p\in\Omega} v_p^i x_p = 1 \quad i = 1, \dots, n \tag{18}$$

$$\sum_{p\in\Omega} u_p^j x_p \le r_j \quad j = 1, \dots, d \tag{19}$$

$$\sum_{p\in\Omega} z_p^k x_p \le M s_k \quad k = 1, \dots, b \tag{20}$$

$$\sum_{k=1}^{s} \omega_k \le \zeta \tag{21}$$

$$x_p \in \{0,1\} \quad \forall p \in \Omega \tag{22}$$

$$s_k \in \{0,1\} \quad k = 1, \dots, b \tag{23}$$

Here the value $M = n + \sum_{j=1}^{d} r_j$ is an upper bound on the number of times a refueling station can be visited in a day. The upper bound comes from the fact that a bus can stop at most once after leaving the depot and after every trip. This is the same integer programming problem as the one for the AF-VSP by with a small number of modifications. The objective function (17) in this problem now includes the cost of placing the stations. Constraint (20) ensures that vehicles only refuel at stations that are included in the solution, since M is an upper bound on the number of times a station could be visited in a day by any vehicles and the sum of $z_p^k x_p$ is the number of times a station is used in a day. Constraint (21) ensures that at most ζ refueling stations are selected for the solution.

This problem can be solved using a column generation approach. We begin by allowing all s_k decision variables to be included in the solution, as well as any other dummy trips as described in Chapter 4. Let the dual variables associated with the constraints be π_i for $i = 1, \dots, t$, ρ_j for $j = 1, \dots, d$, ϕ_k for $k = 1, \dots, b$, and ψ for constraints (18) through (21) respectively. In the AF-VSP column generation algorithm, new columns representing possible schedules in the solution were added to the restricted master problem that satisfied $c^* = \min_{p \in \Omega}\{c_p - \sum_{i=1}^{n} v_p^i \pi_i + \sum_{j=1}^{d} u_p^i \rho_j\}$. Since all of the decision variables that represent adding stations to the problem are included for the first restricted master problem, like before the only columns that need to be generated are those for schedules to add. In the AF-VSLP, the schedules should be selected to minimize:

$$c^* = \min_{p \in \Omega}\left\{c_p - \sum_{i=1}^{n} v_p^i \pi_i + \sum_{j=1}^{d} u_p^i \rho_j + \sum_{k=1}^{b} z_p^k \phi_k\right\}.$$

This cost function modifies the one from the AF-VSP by adding an additional penalty for visiting refueling stations. That penalty is found using the dual variables of the solution for the restricted master problem, similar to before. Like before, to find the new columns to add we need to solve a weight constrained shortest path problem with replenishment. Only now we have added an additional cost of ϕ_k to the route each time a schedule uses station σ_k. The algorithms in Fig. 25 and Fig. 26 do not need to be changed for the AF-VSLP, since the underlying weight constrained shortest path problem with replenishment stays the same. Again the algorithm must be run separately for each depot, although they can be run in parallel, and the lowest cost schedules for each depot can be added to the restricted master problem provided the value for c^* is negative. Thus the cost function c'' to use in the column generation subproblem needs to include the penalty for using a station, and is:

$$c''(v_1, v_2) = \begin{cases} c'(v_1, v_2) - \pi_i & \text{for } v_1 \in T, v_2 \in V \\ c'(v_1, v_2) - \phi_k & \text{for } v_1 = h(x, \sigma_k) \in H; x, v_2 \in V \\ c'(v_1, v_2) + \rho_j & \text{for } v_1 \in D, v_2 \in V. \end{cases}$$

This cost function has the similar bonuses and penalties for visiting trips and starting a depots from the original AF-VSP. It also includes a penalty ϕ_k that in incurred from any edge that leaves a vertex representing visiting fuel station σ_k.

Since there are now decision variables based on using stations in the solution, the branch and bound algorithm needs to be adjusted. Recall that before the branching was based on including and excluding a pair from $(T \cup D) \times T$. For the AF-VSLP we first branch by observing noninteger variables in the solutions that represent stations. If s_k is noninteger for station σ_k, then the problem should be branched into two. In one problem, station σ_k must be in the solution, in which case decision variable ω_k is removed from the master problem, the number of stations allowed in the solution is reduced by one, and the cost of the included station is added to the value of the solution. When solving the column generation subproblem, any possible schedule may include station σ_k for no additional cost; i.e. $\phi_k = 0$. In the other branch, station σ_k cannot be included in the solution, in which case no schedule may include station σ_k and decision variable ω_k is removed from the problem.

5.3.2. Empirical results

We tested the algorithm on randomly generated data. The data was generated in the same method as Section 4.5.1, only now the instead of s being the number of used stations it is the number of candidate stations. We tested several different combinations of trips, candidate stations, and depots, and for each set we varied the number of allowed stations between 2 and 6, while keeping the rest of the problem the same. We also compared the results to placing stations randomly out of the possible candidates. The results can be seen in Table 11. Notice that compared to the AF-VSP results in Table 5, the relaxation gap is several order of magnitudes higher. This is due to the station constraints; because the M value in constraint (20) has to be sufficiently high to allow for every vehicle to stop at a station between trips, the decision variables corresponding to using a station can be values fairly close to zero. Therefore in the linear relaxation of the problem the solution will only have a small component of the station costs allocated to it. This can be migrated by having the station number constraint (21) be strict. The solutions with optimal station placement have dramatically lower costs than randomly selecting from possible sites. This is due to the vehicles not having to travel as far, fewer stations being needed since they are better placed, and cheaper stations being selected as appropriate.

Table 11
Results of applying a column generation algorithm to randomly generated data for the AF-VSLP

Number of trips	Number of stations	Number of depots	Maximum allowed stations	Mean runtime (sec)	Mean cost (10^4)	Mean number of buses	Mean number of stations	Mean relaxation gap (10^{-3})	Mean number of nodes	Max number of nodes	Mean max level	Max level	Mean number of columns	Percent cost of solution compared to randomly selecting location
10	4	8												
			2	11	4.7	3.9	1.0	127	33	71	10.0	15	23.0	82.1%
			4	13	4.7	3.9	1.0	127	51	193	9.5	15	22.4	66.1%
			6	13	4.7	3.9	1.0	127	53	205	9.5	15	21.9	54.8%
20	4	8												
			2	90	7.4	6.0	1.1	127	71	129	11.9	18	62.9	84.9%
			4	83	7.4	6.0	1.1	126	123	277	13.6	23	61.0	75.5%
			6	94	7.4	6.0	1.1	127	147	411	13.1	21	59.6	66.0%
30	4	8												
			2	352	10.1	8.4	1.4	104	122	221	15.8	21	94.6	89.6%
			4	326	10.1	8.4	1.4	104	286	811	16.1	23	93.8	83.5%
			6	446	10.1	8.4	1.4	104	495	2201	16.5	23	94.0	74.9%

Chapter 6

CONCLUSION

In this dissertation several decision problems relating to systems operations and infrastructure design that arise when switching from standard gasoline vehicles to ones that use alternative-fuels or electricity were formulated, solved, and analyzed.

In Chapter 2 we presented a method for routing a single electric vehicle through a network with charging or battery exchange stations. The vehicle was routed without consideration for other vehicles, and the stations had unlimited capacity. The case where the number of battery exchange or recharging stops the vehicle could make was restricted was considered, as well as the case when the driver wanted to minimize range anxiety rather than driving distance. In all of these instances a fast polynomial-time algorithm was developed based on generating a meta-network representation of the original road network. The algorithm was tested on randomly generated data to empirically validate its speed and functionality.

Chapter 3 considered the case where many electric vehicles were to be routed, and each routing decision for an arriving vehicle was made knowing only the rate of future arrivals in the system. We assumed the vehicles would use battery swapping technology, however the capacity at each station was limited to the number of batteries available. Vehicles when routed would also have batteries reserved at the stations, however if no batteries were available at the time the vehicle arrived the driver would have to wait until the reserved battery was fully charged.

We represented the system using the state of the batteries at each system, and the problem became one of finding the optimal policy for a Markov chance-decision process. We used approximate dynamic programming with linear temporal differencing to find a good routing policies. The values of each state of the system were approximated using linear functions to describe the available batteries at each system. Further the policy generated by the algorithm was improved by associating the delays to particular stations rather than the system as a whole. The algorithm was tested on the Arizona highway network using the gravity model for origin-destination demand from Upchurch et al. (Upchurch et al., 2009) and it was found that the delays in the system could be reduced by up to 23% as compared to routing the vehicles in a greedy manner.

Scheduling fleets of alternative-fuel or electric vehicles was considered in Chapter 4. Here the classic multiple depot vehicle scheduling problem was modified to give the vehicles a limited range they could travel before needing to refuel at set locations. This new problem was defined to be the alternative-fuel multiple depot vehicle scheduling problem, and an exact column generation algorithm was proposed to solve it. A concurrent scheduler heuristic was developed, and other possible heuristics were discussed. The column generation and concurrent scheduler algorithms were tested on randomly generated data with up to 50 trips as well as on real world data from the Valley Metro transit group.

Finally, in Chapter 5 we extended the models of each of the previous chapters with additional constraints and considerations. In Section 5.1 we added stochastic edge lengths to the problem of routing electric vehicles. This turn the problem from finding a shortest walk to finding a policy to route the vehicle depending on its location and battery level. The problem proved to be complicated due to the possibility that the vehicle may run out of battery and be stranded before reaching the destination. Using the idea of imposing a large penalty if the vehicle ran out of fuel, representing the cost of a service vehicle bringing a fully charged battery to the stranded vehicle, we solved the problem using a value iteration scheme for the corresponding Markov Decision Process.

In Section 5.2 we extend the problem of online routing and making battery reservations for electric vehicles. We extend the problem in several ways, including having the batteries be dropped off without being fully empty, having vehicles arrive in the system with different charge levels, and having multiple rates of arrivals throughout the day. The new extensions were also tested on the Arizona road network.

In Section 5.3 we modified the alternative-fuel vehicle scheduling problem to also be a refueling station location problem. In this case instead of just scheduling the vehicles we also need to decide the locations of the refueling stations. This problem was solved using an alteration to the column generation algorithm for the original scheduling problem. The solutions generated by the algorithm were compared to solutions with randomly placed refueling stations for randomly generated sets of trips.

There are many possible areas for future work on these topics. For the single electric vehicle routing problem, while we added stochastic edge lengths to the problem, we assumed that the edge lengths were independent, that the driver had full information on the distribution of the edge lengths, and that the edge lengths reset every time the driver traversed them. This problem could be altered so that the edge lengths are dependent and based on certain scenarios, such as low or high traffic. In this version of the problem the driver then needs to discover which scenario they are in. This would likely convert the problem from being a Markov decision process to a partially observable Markov decision process.

Future work for the online routing and battery reservations for electric vehicles problem could revolve around uncertainty in the parameters the model is based on. We assumed that the rate of arrivals for each OD pair was fully known, however this is unlikely to be the case in the real world. The problem could also be improved by formulating better basis functions for the system states, and by finding better weighting functions for the θ updates. One could also allow for vehicles to pick up non-full batteries, which would likely require the problem to be adjusted so that different types of inventory are considered (full batteries, mostly full batteries, etc). Finally, the decision problem of when to charge the batteries could be considered; given the cost of electricity throughout the day it may make sense to avoid having a battery charge that won't be needed during the day.

For the alternative-fuel vehicle scheduling problem, future work should be based on finding more effective heuristic techniques. Basing a heuristic on one from the multiple depot vehicle scheduling problem has much potential, however it is non-trivial to add the fuel constraints to these heuristics. A related problem could be defined to model rescheduling: in this situation part of the way through the day the schedule needs to be revised due to a stranded vehicle or some external situation. In Section 5.3 we gave an exact solution for the refueling location and scheduling problem, however we proposed no heuristic. Future work could consider developing a heuristic for the location problem that effectively chooses the locations of the refueling stations as well as schedules the vehicles.

REFERENCES

Adler, J.D., Mirchandani, P.B., Xue, G., Xia, M., 2014. The electric vehicle shortest-walk problem with battery exchanges. Networks Spat. Econ. Sustain. SI Online Fir.

Ahuja, R.K., Magnanati, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, NJ.

Bakker, J., 2011. Contesting range anxiety: The role of electric vehicle charging infrastructure in the transportation transition. alexandria.tue.nl. Eindhoven University of Technology.

Ball, M., 1980. A comparison of relaxations and heuristics for certain crew and vehicle scheduling problems, in: National ORSA/TIMS Meeting. Washington, D.C.

Beasley, J.E., Christofides, N., 1989. An algorithm for the resource constrained shortest path problem. Networks 19, 379–394. doi:10.1002/net.3230190402

Berman, O., Larson, R.C., Nikoletta, F., 1992. Optimal location of discretionary service facilities. Transp. Sci. 26, 201–211.

Bertsekas, D.P., 2007. Dynamic Programming and Optimal Control, 4th ed. Athena Scientific, Belmont.

Better Place, 2013. Better Place: global progress [WWW Document]. URL http://www.betterplace.com/global/progress

Bodin, L., Golden, B., Assad, A., Ball, M., 1983. Routing and scheduling of vehicles and crews: The state of the art. Comput. Oper. Res. 10, 63–211.

Bodin, L., Rosenfield, D., Kydes, A., 1978. UCOST: a micro approach to a transportation planning problem. J. Urban Anal. 5, 47–69.

Botsford, C., Szczepanek, A., 2009. Fast charging vs. slow charging: pros and cons for the new age of electric vehicles, in: EVS24 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium. pp. 1–9.

Bunte, S., Kliewer, N., 2009. An overview on vehicle scheduling models. Public Transp. 1, 299–317. doi:10.1007/s12469-010-0018-5

Capar, I., Kuby, M., 2012. An efficient formulation of the flow refueling location model for alternative-fuel stations. IIE Trans. 44, 622–636. doi:10.1080/0740817X.2011.635175

Ceder, A., 2002. Urban transit scheduling: framework, review and examples. J. urban Plan. Dev. 128, 225–244.

City of Phoenix, 2013. City of Phoenix public transit developers portal [WWW Document]. URL http://phoenix.gov/publictransit/developers/index.html

Daduna, J.R., Paixão, J.M.P., 1995. Vehicle scheduling for public mass transit - an overview, in: Lecture Notes in Economics and Mathematical Systems. pp. 76–90.

De Weerdt, M.M., Gerding, E.H., Stein, S., Robu, V., Jennings, N.R., 2013. Intention-aware routing to minimise delays at electric vehicle charging stations, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. AAAI Press.

Dell'Amico, M., Matteo, A., Paolo, F., 1993. Heuristic algorithms for the multiple depot vehicle scheduling problem. Manage. Sci. 39, 115–125.

Desaulniers, G., Hickman, M.D., 2007. Public Transit, in: Laporte, G., Barnhart, C. (Eds.), Handbooks in Operations Research and Management Science: Vol. 14. Elsevier Science, Amsterdam, pp. 69–127. doi:10.1016/S0927-0507(06)14002-5

Desaulniers, G., Lavigne, J., Soumis, F., 1998. Multi-depot vehicle scheduling problems with time windows and waiting costs. Eur. J. Oper. Res. 111, 479–494. doi:10.1016/S0377-2217(97)00363-9

Desrochers, M., Soumis, F.C., 1988. A generalized permanent labelling algorithm for the shortest path problem with time windows. INFOR 26, 191–212.

Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F., 1995. Time constrained routing and scheduling, in: Handbooks in OR & MS Vol. 8. pp. 35–139.

Desrosiers, J., Lübbecke, M.E., 2005. A primer in column generation, in: Desrosiers, J., Solomon, M.M., Desaulniers, G. (Eds.), Column Generation. Springer.

Diestel, R., 2000. Graph Theory, Online Edi. ed. Springer-Verlag, Heidelberg.

Dror, M. ed., 2000. Arc Routing: Theory, Solutions and Applications. Kluwer Academic Publishers, Norwell, Massachusetts.

Edmunds.com, 2013. 2013 Nissan Leaf Hatchback Review [WWW Document]. URL http://www.edmunds.com/nissan/leaf/2013/?sub=hatchback

Eiger, A., Mirchandani, P.B., Soroush, H., 1985. Path preferences and optimal paths in probabilistic networks. Transp. Sci. 19, 75–84.

Fan, Y.Y., Kalaba, R.E., Moore, J.E., 2005. Arriving on time. J. Optim. Theory Appl. 127, 497–513. doi:10.1007/s10957-005-7498-5

Folkesson, A., Andersson, C., Alvfors, P., Alaküla, M., Overgaard, L., 2003. Real life testing of a hybrid PEM fuel cell bus. J. Power Sources 118, 349–357. doi:10.1016/S0378-7753(03)00086-7

Frank, H., 1969. Shortest paths in probabilistic graphs. Oper. Res. 17, 583–599.

Freling, R., Paixão, J.M.P., 1995. Vehicle scheduling with time constraint, in: Lecture Notes in Economics and Mathematical Systems. pp. 130–144.

Freling, R., Wagelmans, A.P.M., Paixão, J.M.P., 2001. Models and algorithms for vehicle scheduling. Transp. Sci. 35, 165–180.

Gallagher, K.S., Muehlegger, E., 2011. Giving green to get green? Incentives and consumer adoption of hybrid vehicle technology. J. Environ. Econ. Manage. 61, 1–15. doi:10.1016/j.jeem.2010.05.004

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. F. Freeman, New York, NY.

Glover, F., 1990. Tabu search: a tutorial. Interfaces (Providence). 20, 74–94.

Golden, B.L., 1988. Vehicle routing: methods and studies. North-Holland.

Golden, B.L., Wong, R.T., 1981. Capacitated arc routing problems. Networks 11, 305–315.

Hacker, F., Harthan, R., Matthes, F., Zimmer, W., 2009. Environmental impacts and impact on the electricity market of a large scale introduction of electric cars in Europe, European Topic Centre on Air and Climate Change.

Haghani, A., Banihashemi, M., 2002. Heuristic approaches for solving large-scale bus transit vehicle scheduling problem with route time constraints. Transp. Res. Part A Policy Pract. 36, 309–333. doi:10.1016/S0965-8564(01)00004-0

Handler, G.Y., Zang, I., 1980. A dual algorithm for the constrained shortest path problem. Networks 10, 293–309.

Heineman, G.T., Pollice, G., Selkow, S., 2008. Algorithms in a Nutshell. O'Reilley Media, Inc., Sebastopol, CA.

Hentenryck, P. Van, Mercier, L., Upfal, E., 2009. Markov chance-decision processes, in: Online Stochastic Combinatorial Optimization. The MIT Press, pp. 194–218.

Hodgson, M.J., 1990. A flow-capturing location-allocation model. Geogr. Anal. 22, 270–279.

Ichimori, T., Ishii, H., Nishida, T., 1981. Routing a vehicle with the limitation of fuel. J. Oper. Res. Soc. Japan 24, 277–281.

Jeeninga, H., Arkel, W.G. Van, Volkers, C.H., 2002. Performance and acceptance of electric and hybrid vehicles: determination of attitude shifts and energy consumption of electric and hybrid vehicles used in the ELCIDIS project. Muncipality of Rotterdam, Rotterdam, Netherlands.

Jiang, N., Xie, C., Waller, S.T., 2012. Path-Constrained Traffic Assignment: Model and Algorithm, in: 91st Annual Meeting of the Transportation Board.

Kaibel, V., Peinhardt, M.A.F., 2006. On the Bottleneck Shortest Path Problem. Berlin, Germany.

Kershner, I., 2013. Israel venture meant to serve electric vehicles is ending its run [WWW Document]. New York Times. URL http://www.nytimes.com/2013/05/27/business/global/israeli-electric-car-company-files-for-liquidation.html

Kolobov, A., Mausam, Weld, D.S., 2012. Stochastic shortest path MDPs with dead ends, in: HSDIP 2012.

Kuby, M., 2005. The flow-refueling location problem for alternative-fuel vehicles. Socioecon. Plann. Sci. 39, 125–145. doi:10.1016/j.seps.2004.03.001

Kuby, M., Lim, S., 2007. Location of alternative-fuel stations using the flow-refueling location model and dispersion of candidate sites on arcs. Networks Spat. Econ. 7, 129–152. doi:10.1007/s11067-006-9003-6

Kuby, M.J., Kelley, S.B., Schoenemann, J., 2013. Spatial refueling patterns of alternative-fuel and gasoline vehicle drivers in Los Angeles. Transp. Res. Part D Transp. Environ. 25, 84–92. doi:10.1016/j.trd.2013.08.004

Kurani, K.S., Heffner, R.R., Turrentine, T., 2008. Driving plug-in hybrid electric vehicles: reports from U.S. drivers of HEVs converted to PHEVs, circa 2006-07. Transp. Res. Rec. J. Transp. Res. Board 38–45.

Laporte, G., 2009. Fifty years of vehicle routing. Transp. Sci. 43, 408–416. doi:10.1287/trsc.1090.0301

Laporte, G., Pascoal, M.M.B., 2011. Minimum cost path problems with relays. Comput. Oper. Res. 38, 165–173. doi:10.1016/j.cor.2010.04.010

Lavrinc, D., 2014. Toyota will sell you a hydrogen-powered car next year [WWW Document]. Wired. URL http://www.wired.com/autopia/2014/01/toyota-hydrogen/ (accessed 8.1.14).

Lawler, E.L., 2001. Combinatorial Optimization: Networks and Matroids. Courier Dover Publications.

Lawson, C.L., Hanson, R.J., 1974. Solving Least Squares Problems. Prentice-hall, Englewood Cliffs, NJ.

Lim, S., Kuby, M., 2010. Heuristic algorithms for siting alternative-fuel stations using the Flow-Refueling Location Model. Eur. J. Oper. Res. 204, 51–61. doi:10.1016/j.ejor.2009.09.032

Lin, S.H., Gertsch, N., Russell, J.R., 2007. A linear-time algorithm for finding optimal vehicle refueling policies. Oper. Res. Lett. 35, 290–296. doi:10.1016/j.orl.2006.05.003

Mak, H.-Y., Rong, Y., Shen, Z.-J.M., 2013. Infrastructure planning for electric vehicles with battery swapping. Manage. Sci. 59, 1557–1575. doi:10.2139/ssrn.2022651

Microsoft, 2013. Bing maps [WWW Document]. URL http://www.microsoft.com/maps/

Mirchandani, P.B., 1976. Shortest distance and reliability. Comput. Oper. Res. 3, 347–355.

Mirchandani, P.B., Adler, J.D., Madsen, O.B.G., 2014. New Logistical Issues in Using Electric Vehicle Fleets with Battery Exchange Infrastructure. Procedia - Soc. Behav. Sci. 108, 3–14. doi:10.1016/j.sbspro.2013.12.815

Mirchandani, P.B., Francis, R.L., 1990. Discrete Location Theory, Discrete Location Theory, Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons.

Mirchandani, P.B., Veatch, M.H., 1986. "Hot job" routing through a stochastic job-shop network. Large Scale Syst. 11, 131–148.

Motavalli, J., 2013. Tesla fast-tracks battery swapping while fighting a legislative attack [WWW Document]. New York Times. URL http://wheels.blogs.nytimes.com/2013/06/21/tesla-fast-tracks-battery-swapping-while-fighting-a-legislative-attack/

Nagy, G., Salhi, S., 2007. Location-routing: issues, models and methods. Eur. J. Oper. Res. 177, 649–672. doi:10.1016/j.ejor.2006.04.004

Ogden, J.M., Steinbugler, M.M., Kreutz, T.G., 1999. A comparison of hydrogen, methanol and gasoline as fuels for fuel cell vehicles: implications for vehicle design and infrastructure development. J. Power Sources 79, 143–168. doi:10.1016/S0378-7753(99)00057-9

Oukil, A., Amor, H. Ben, Desrosiers, J., El Gueddari, H., 2007. Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. Comput. Oper. Res. 34, 817–834. doi:10.1016/j.cor.2005.05.011

Pepin, A.-S., Desaulniers, G., Hertz, A., Huisman, D., 2009. A comparison of five heuristics for the multiple depot vehicle scheduling problem. J. Sched. 12, 17–30. doi:10.1007/s10951-008-0072-x

Polychronopoulos, G.H., Tsitsiklis, J.N., 1996. Stochastic shortest path problems with recourse. Networks 27, 133–143.

Powell, W.B., 2011. Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd ed. John Wiley & Sons, Inc., Hoboken, NJ.

Rebello, R., Agnetis, A., Mirchandani, P.B., 1995. The inspection station location problem in hazardous material transportation. INFOR 33, 100.

Ribeiro, C.C., Soumis, F., 1994. A column generation approach to the multiple-depot vehicle scheduling problem. Oper. Res. 44, 41–52.

Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall, Upper Saddle River, NJ.

Sachenbacher, M., 2010. The shortest path problem revisited: optimal routing for electric vehicles, in: KI 2010: Advances in Artifical Intelligence, 33rd Annual German Conference on AI. pp. 309–316.

Sachenbacher, M., Leucker, M., Artmeier, A., Haselmayr, J., 2011. Efficient energy-optimal routing for electric vehicles, in: AAAI. pp. 1402–1407.

Senart, A., Kurth, S., Roux, G. Le, Antipolis, S., Street, N.C., 2010. Assessment Framework of Plug-in Electric Vehicles Strategies. Smart Grid Commun. 2010 First IEEE Int. Conf. 155–160.

Shemer, N., 2012. Better Place Unveils Battery-Swap Network. Jerusalem Post.

Sigal, C.E., Pritsker, A.A.B., Solberg, J.J., 1980. The stochastic shortest route problem. Oper. Res. 28, 1122–1129.

Smith, O.J., Boland, N., Waterer, H., 2012. Solving shortest path problems with a weight constraint and replenishment arcs. Comput. Oper. Res. 39, 964–984. doi:10.1016/j.cor.2011.07.017

Sovacool, B.K., Hirsh, R.F., 2009. Beyond batteries : An examination of the benefits and barriers to plug-in hybrid electric vehicles (PHEVs) and a vehicle-to-grid (V2G) transition. Energy Policy 37, 1095–1103. doi:10.1016/j.enpol.2008.10.005

Steinzen, I., Gintner, V., Suhl, L., Kliewer, N., 2010. A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. Transp. Sci. 44, 367–382. doi:10.1287/trsc.1090.0304

Upchurch, C., Kuby, M., Lim, S., 2009. A model for location of capacitated alternative-fuel stations. Geogr. Anal. 41, 85–106. doi:10.1111/j.1538-4632.2009.00744.x

Valley Metro, 2013. Valley Metro fact sheets [WWW Document]. URL http://www.valleymetro.org/publications_reports/fact_sheets

Vaughan, A., 2011. Electric Car Infrastructure Begins to Roll Out Across the UK [WWW Document]. Guard. URL http://www.guardian.co.uk/environment/blog/2011/oct/14/electric-car-infrastructure-uk

Waller, S.T., Ziliaskopoulos, A.K., 2002. On the online shortest path problem with limited arc cost dependencies. Networks 40, 216–227. doi:10.1002/net.10049

Wang, H., Shen, J., 2007. Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints. Appl. Math. Comput. 190, 1237–1249. doi:10.1016/j.amc.2007.02.141

Wei, M., Jin, W., Fu, W., Hao, X., 2010. Improved ant colony algorithm for multi-depot bus scheduling problem with route time constraints, in: 8th World Congress on Intelligent Control and Automation. pp. 4050–4053. doi:10.1109/WCICA.2010.5553800

Whiten, B., 2012. nnls - non negative least squares [WWW Document]. MATLAB Cent. File Exch. URL http://www.mathworks.com/matlabcentral/fileexchange/38003-nnls-non-negative-least-squares

Winston, W.L., Venkataraman, M., 2003. An Introduction to Mathematical Programming, 4th ed. Brooks/Cole, Pacific Grove, CA.

Worley, O., Klabjan, D., 2011. Optimization of battery charging and purchasing at electric vehicle battery swap stations, in: 2011 IEEE Vehicle Power and Propulsion Conference. Ieee, pp. 1–4. doi:10.1109/VPPC.2011.6043182

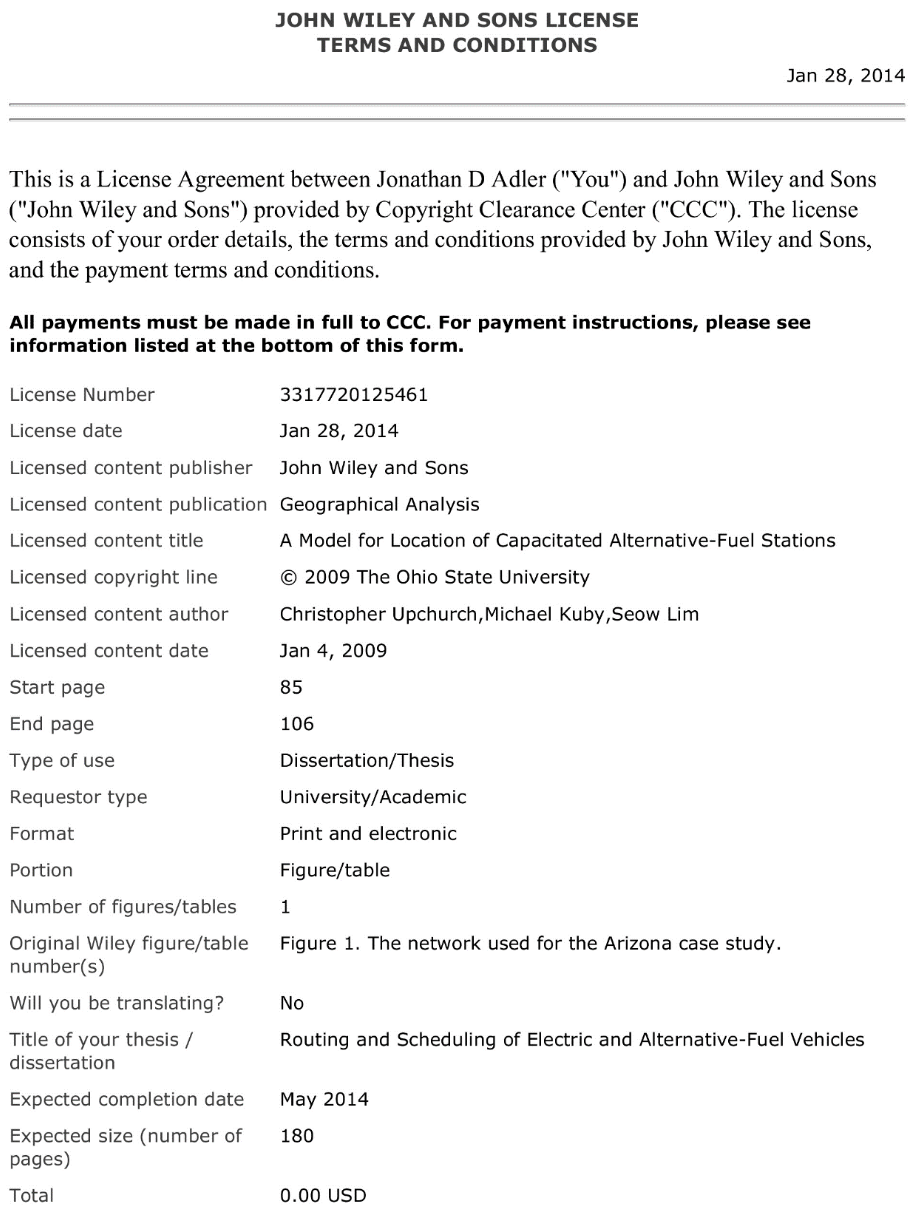
Xiao, Y., Thulasiraman, K., Xue, G., Jüttner, A., 2005. The constrained shortest path problem: algorithmic approaches and an algebraic study with generalization. AKCE Int. J. Graphs Comb. 2, 63–86.

Yu, A.S.O., Silva, L.L.C., Chu, C.L., Nascimento, P.T.S., Camargo, A.S., 2011. Electric vehicles: struggles in creating a market, in: Technology Management in the Energy Smart World. Portland, OR, pp. 1–13.

Zhu, C., Chen, X., Wu, J., Ye, J., 2012. Assessment of Battery Electric Bus Operation in Shanghai EXPO, in: 91st Annual Meeting of the Transportation Board. Washington, D.C.

APPENDIX A

PERMISSION TO USE COPYRIGHTED MATERIAL



JOHN WILEY AND SONS LICENSE
TERMS AND CONDITIONS

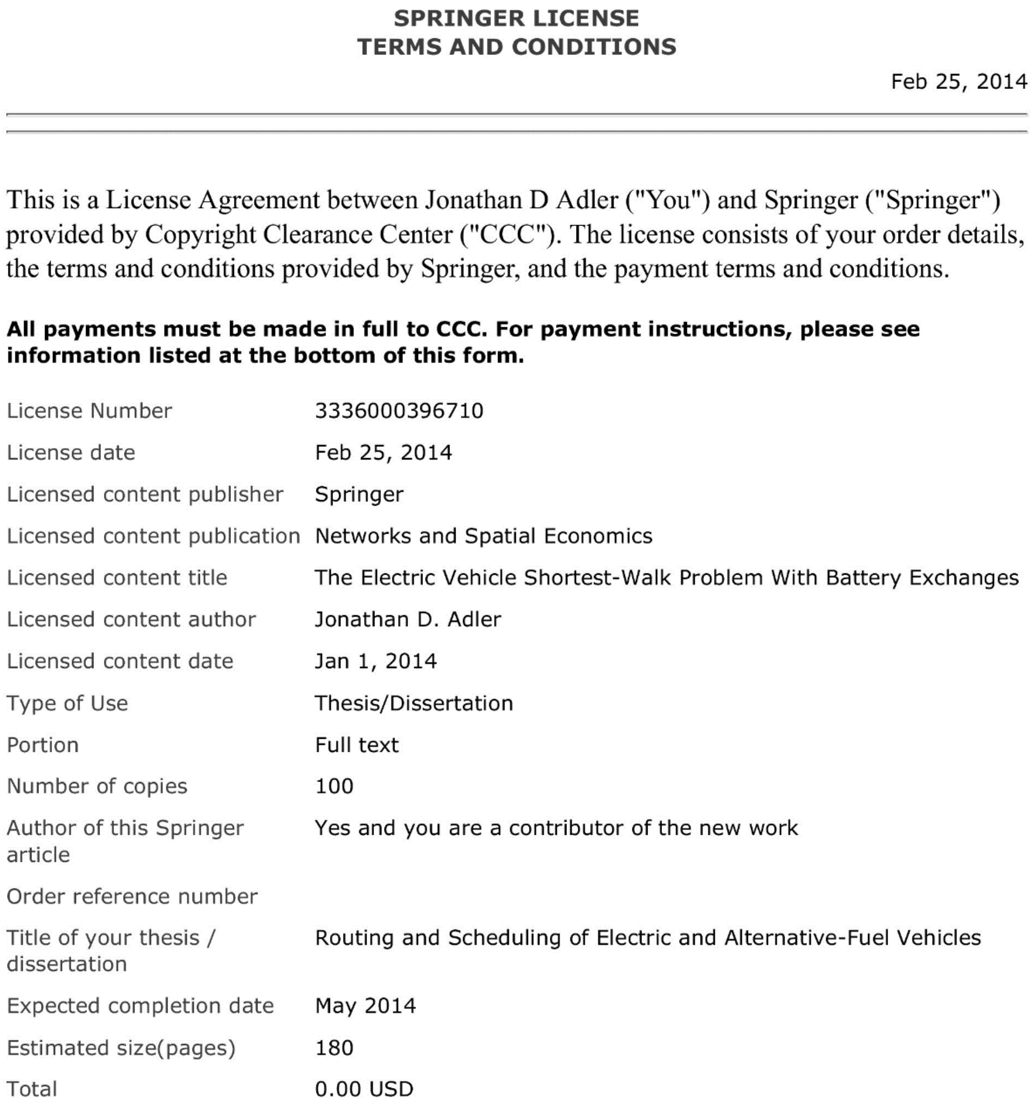
Jan 28, 2014

This is a License Agreement between Jonathan D Adler ("You") and John Wiley and Sons ("John Wiley and Sons") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by John Wiley and Sons, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

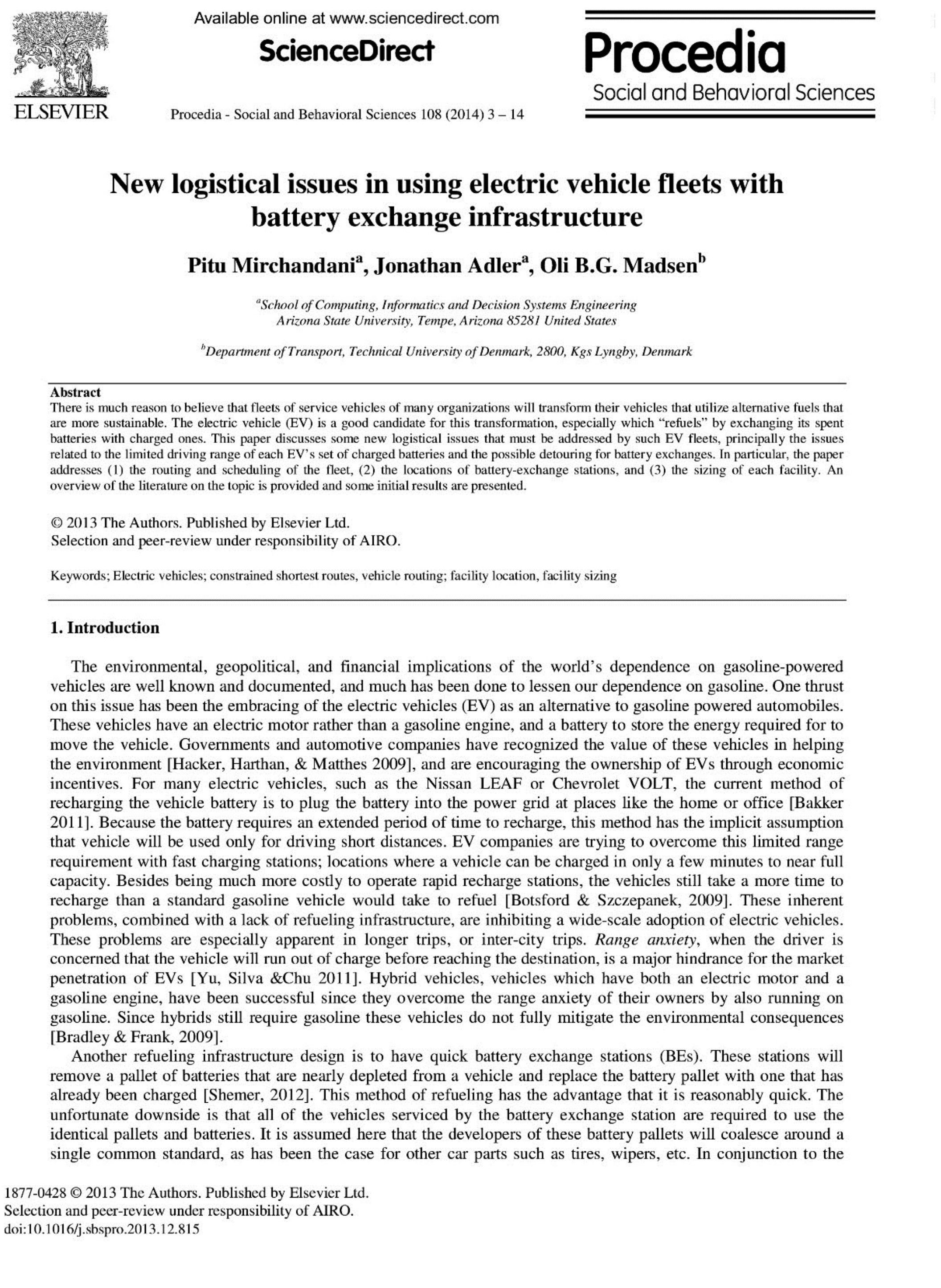
License Number	3317720125461
License date	Jan 28, 2014
Licensed content publisher	John Wiley and Sons
Licensed content publication	Geographical Analysis
Licensed content title	A Model for Location of Capacitated Alternative-Fuel Stations
Licensed copyright line	© 2009 The Ohio State University
Licensed content author	Christopher Upchurch,Michael Kuby,Seow Lim
Licensed content date	Jan 4, 2009
Start page	85
End page	106
Type of use	Dissertation/Thesis
Requestor type	University/Academic
Format	Print and electronic
Portion	Figure/table
Number of figures/tables	1
Original Wiley figure/table number(s)	Figure 1. The network used for the Arizona case study.
Will you be translating?	No
Title of your thesis / dissertation	Routing and Scheduling of Electric and Alternative-Fuel Vehicles
Expected completion date	May 2014
Expected size (number of pages)	180
Total	0.00 USD

Fig. 30. The copyright license for Fig. 20.



License Number	3336000396710
License date	Feb 25, 2014
Licensed content publisher	Springer
Licensed content publication	Networks and Spatial Economics
Licensed content title	The Electric Vehicle Shortest-Walk Problem With Battery Exchanges
Licensed content author	Jonathan D. Adler
Licensed content date	Jan 1, 2014
Type of Use	Thesis/Dissertation
Portion	Full text
Number of copies	100
Author of this Springer article	Yes and you are a contributor of the new work
Order reference number	
Title of your thesis / dissertation	Routing and Scheduling of Electric and Alternative-Fuel Vehicles
Expected completion date	May 2014
Estimated size(pages)	180
Total	0.00 USD

Fig. 31. The copyright license for Chapter 2.



Available online at www.sciencedirect.com

ScienceDirect

Procedia

Social and Behavioral Sciences

ELSEVIER

Procedia - Social and Behavioral Sciences 108 (2014) 3 – 14

New logistical issues in using electric vehicle fleets with battery exchange infrastructure

Pitu Mirchandani[a], Jonathan Adler[a], Oli B.G. Madsen[b]

[a]*School of Computing, Informatics and Decision Systems Engineering Arizona State University, Tempe, Arizona 85281 United States*

[b]*Department of Transport, Technical University of Denmark, 2800, Kgs Lyngby, Denmark*

Abstract

There is much reason to believe that fleets of service vehicles of many organizations will transform their vehicles that utilize alternative fuels that are more sustainable. The electric vehicle (EV) is a good candidate for this transformation, especially which "refuels" by exchanging its spent batteries with charged ones. This paper discusses some new logistical issues that must be addressed by such EV fleets, principally the issues related to the limited driving range of each EV's set of charged batteries and the possible detouring for battery exchanges. In particular, the paper addresses (1) the routing and scheduling of the fleet, (2) the locations of battery-exchange stations, and (3) the sizing of each facility. An overview of the literature on the topic is provided and some initial results are presented.

© 2013 The Authors. Published by Elsevier Ltd.
Selection and peer-review under responsibility of AIRO.

Keywords; Electric vehicles; constrained shortest routes, vehicle routing; facility location, facility sizing

1. Introduction

The environmental, geopolitical, and financial implications of the world's dependence on gasoline-powered vehicles are well known and documented, and much has been done to lessen our dependence on gasoline. One thrust on this issue has been the embracing of the electric vehicles (EV) as an alternative to gasoline powered automobiles. These vehicles have an electric motor rather than a gasoline engine, and a battery to store the energy required for to move the vehicle. Governments and automotive companies have recognized the value of these vehicles in helping the environment [Hacker, Harthan, & Matthes 2009], and are encouraging the ownership of EVs through economic incentives. For many electric vehicles, such as the Nissan LEAF or Chevrolet VOLT, the current method of recharging the vehicle battery is to plug the battery into the power grid at places like the home or office [Bakker 2011]. Because the battery requires an extended period of time to recharge, this method has the implicit assumption that vehicle will be used only for driving short distances. EV companies are trying to overcome this limited range requirement with fast charging stations; locations where a vehicle can be charged in only a few minutes to near full capacity. Besides being much more costly to operate rapid recharge stations, the vehicles still take a more time to recharge than a standard gasoline vehicle would take to refuel [Botsford & Szczepanek, 2009]. These inherent problems, combined with a lack of refueling infrastructure, are inhibiting a wide-scale adoption of electric vehicles. These problems are especially apparent in longer trips, or inter-city trips. *Range anxiety*, when the driver is concerned that the vehicle will run out of charge before reaching the destination, is a major hindrance for the market penetration of EVs [Yu, Silva &Chu 2011]. Hybrid vehicles, vehicles which have both an electric motor and a gasoline engine, have been successful since they overcome the range anxiety of their owners by also running on gasoline. Since hybrids still require gasoline these vehicles do not fully mitigate the environmental consequences [Bradley & Frank, 2009].

Another refueling infrastructure design is to have quick battery exchange stations (BEs). These stations will remove a pallet of batteries that are nearly depleted from a vehicle and replace the battery pallet with one that has already been charged [Shemer, 2012]. This method of refueling has the advantage that it is reasonably quick. The unfortunate downside is that all of the vehicles serviced by the battery exchange station are required to use the identical pallets and batteries. It is assumed here that the developers of these battery pallets will coalesce around a single common standard, as has been the case for other car parts such as tires, wipers, etc. In conjunction to the

1877-0428 © 2013 The Authors. Published by Elsevier Ltd.
Selection and peer-review under responsibility of AIRO.
doi:10.1016/j.sbspro.2013.12.815

Fig. 32. The copyright for reproducing material by Mirchandani et al. (Mirchandani et al., 2014).