Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

The Vehicle Scheduling Problem for Fleets with Alternative-Fuel Vehicles

Jonathan D. Adler, Pitu B. Mirchandani

To cite this article:
Jonathan D. Adler, Pitu B. Mirchandani (2017) The Vehicle Scheduling Problem for Fleets with Alternative-Fuel Vehicles. Transportation Science 51(2):441-456. https://doi.org/10.1287/trsc.2015.0615

http://pubsonline.informs.org/journal/trsc/

TRANSPORTATION SCIENCE
Vol. 51, No. 2, May 2017, pp. 441–456
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

The Vehicle Scheduling Problem for Fleets with Alternative-Fuel Vehicles

Jonathan D. Adler,[a] **Pitu B. Mirchandani**[a]

[a] School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, Arizona 85281
Contact: jonadler@asu.edu (JDA); pitu@asu.edu (PBM)

Received: March 14, 2013
Revised: February 22, 2014; July 15, 2014
Accepted: January 2, 2015
Published Online in Articles in Advance: March 14, 2016

https://doi.org/10.1287/trsc.2015.0615

Abstract. Alternative-fuel vehicles are gaining popularity as a mode of transit, and research is being done into how current infrastructure can accommodate them. The problem of vehicle scheduling consists of assigning a fleet of vehicles to service a given set of trips with start and end times. Vehicle scheduling changes when alternative-fuel vehicles are used since the vehicles can carry only a limited amount of fuel and can refuel only at fixed locations. This paper presents the alternative-fuel multiple depot vehicle scheduling problem, a modification of the standard multiple depot vehicle scheduling problem where there is a given set of fueling stations and a fuel capacity for the vehicles. The problem is formally defined and formulated as an integer program, and a branch-and-price algorithm is proposed to solve the problem. A heuristic solution is also presented, and both are tested on randomly generated data and data on the Valley Metro bus network in the Phoenix, Arizona, metropolitan area.

Funding: 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.
Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2015.0615.

Keywords: vehicle scheduling problem • alternative-fuel vehicles • fuel constraint • column generation • branch-and-price

1. Introduction and Background

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 has 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 has been defined and are referred to as vehicle scheduling problems (VSPs). 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, Pollice, and Selkow 2008).

The multiple depot vehicle scheduling problem (MDVSP) is conceptually straightforward: 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 of the service trips with the vehicles? Here the service trips are predefined in terms of start and end times and locations, and the services being performed during the trips could be something like picking up and dropping off passengers. The MDVSP 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 biofuel 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 biofuel vehicles, they can refill only at special fueling stations (Ogden, Steinbugler, and Kreutz 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 vehicle 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 paper focuses on the problem where, for environmental sustainability or government regulation reasons, a fleet of vehicles is changed to a fleet of alternative-fuel vehicles 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 paper 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 paper 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 MDVSP, especially when there are additional constraints due to time, often are solved using column generation techniques as first shown by Ribeiro and Soumis (1994). Other researchers have extended the column generation approach by adding time constraints (Desaulniers, Lavigne, and Soumis 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 as 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 that 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; the combination of column generation and branch-and-bound techniques is known as *branch-and-price*. For an in-depth introduction to column generation, see Desrosiers and Lübbecke (2005). Regarding VSPs, 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 paper, we will consider how the limited refueling locations and limited range of alternative fuel vehicles change the MDVSP. 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 the 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 nontrivial and is in fact an instance of the weight constrained shortest path problem with replenishment arcs (Smith, Boland, and Waterer 2012). If the column generation algorithm produces a noninteger solution, a branch-and-bound approach is used to repeatedly solve column generation subproblems until the optimal integer solution is found.

This paper is organized as follows: Section 2 formally defines the problem and discusses how to expand the problem to handle cases that violate the assumptions made in the definition. Section 3.1 formulates the linear relaxation of the problem as a master problem and subproblem to be solved using the column generation approach. Section 3.2 discusses how to solve the subproblem for the column generation technique, and Section 3.3 discusses the branch-and-price approach to get the integer solution. In Section 4 an additional heuristic approach is presented and discussed. In Section 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.

2. Formal Definition of the Problem

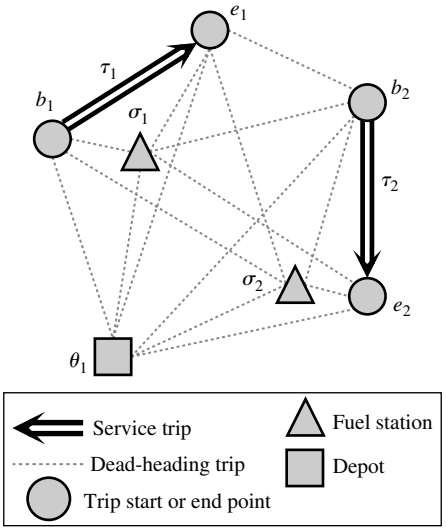
We define the alternative-fuel multiple depot vehicle scheduling problem (AF-VSP) as follows. The problem is defined for a given n service trips $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$,

s fuel stations $S = \{\sigma_1, \sigma_2, \ldots, \sigma_s\}$, and d depots $D = \{\theta_1, \ldots, \theta_d\}$. For each service trip τ_i, there is a start location b_i, end location e_i, start and end times bt_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 $z_1, z_2 \in S \cup T \cup D$ 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 w. 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 w.

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 at 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 w fuel, but the depots themselves are not 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 $bt_i \leq bt_j$ for $1 \leq i \leq j \leq n$. A visual representation of a sample AF-VSP with a single depot can be seen in Figure 1.

Two service trips $\tau_i, \tau_j \in T$ are *compatible* if $et_i + t(\tau_i, \tau_j) \leq bt_j$, 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* $\sigma_k \in S$ if $et_i + t(\tau_i, \sigma_k) + t(\sigma_k, \tau_j) \leq bt_j$, 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 for when τ_i and τ_j are compatible with fuel station σ_k as $\text{compf}(\tau_i, \sigma_k, \tau_j)$. 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, service trips, and fuel stations as an ordered sequence of elements of $S \cup T \cup D$. 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.

Figure 1. An Example AF-VSP Where $n = 2$, $s = 2$, and $d = 1$



The AF-VSP is a generalization of the multiple depot vehicle scheduling problem since VSP is a special case of AF-VSP where $w = \infty$ and $s = 0$. Without the fuel constraint the MDVSP 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., $S = \emptyset$ 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 a 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 alternative 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 s since a vehicle would never want to visit the same fuel station twice between two service trips. In fact, for any sequence of k refueling stations $\{m_1, m_2, \ldots, m_k\}$ for $m_1, \ldots, m_k \in S$, $2 \le k \le s$, and $f(m_j, m_{j+1}) \le w$ for all $j < k$, the amount of fuel required for a vehicle to dead-head 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 $\sigma_{\{m_1, m_2, \ldots, m_k\}}$ can be created representing this sequence of refueling stations, where the dead-heading trips from each service trip to $\sigma_{\{m_1, m_2, \ldots, m_k\}}$ have the same costs, travel times, and fuel requirements as the dead-heading trips to fuel station m_1. The dead-heading trips from $\sigma_{\{m_1, m_2, \ldots, m_k\}}$ to each service trip $\tau_i \in N$ have travel times $t(\sigma_{\{m_1, m_2, \ldots, m_k\}}, \tau_i) = t(m_k, \tau_i) + \sum_{x=2}^{k} t(m_{x-1}, m_x)$, the cost $c(\sigma_{\{m_1, m_2, \ldots, m_k\}}, \tau_i) = c(m_k, \tau_i) + \sum_{j=2}^{k} c(m_{j-1}, m_j)$, and the fuel requirement of $f(\sigma_{\{m_1, m_2, \ldots, m_k\}}, \tau_i) = f(m_k, \tau_i)$. The proxy station $\sigma_{\{m_1, m_2, \ldots, m_k\}}$ thus 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.

Although there are at most $s!$ 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 $M_1 = \{\bar{m}, m_1^1, m_2^1, \ldots, m_{k_1-1}^1, \bar{\bar{m}}\}$ and $M_2 = \{\bar{m}, m_1^2, m_2^2, \ldots, m_{k_2-1}^2, \bar{\bar{m}}\}$. For the two sequences, $f(\sigma_{M_1}, z) = f(\sigma_{M_2}, z)$ and $f(z, \sigma_{M_1}) = f(z, \sigma_{M_2})$ for any $z \in T \cup S \cup D$ since they share the same start and end stations. Similarly, $c(z, \sigma_{M_1}) = c(z, \sigma_{M_2})$ and either $c(\sigma_{M_1}, z) \le c(\sigma_{M_2}, z)$ or $c(z, \sigma_{M_1}) \ge c(z, \sigma_{M_2})$ holds for all x. That is to say, either the costs of the dead-heading trips from the 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 that are not Pareto optimal with respect to cost and time can be removed; this corresponds to sequences of fuel stations between $\bar{m}$ and $\bar{\bar{m}}$ that 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 θ_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. Similar 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.

3. 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-price 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 $G = (V, A)$ be a directed graph, let c', f': $A \to \mathbb{R}^+$ be cost and fuel requirement functions, and let w be a 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 $i \in \{1, \ldots, n\}$ and $k \in s$. 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

$$H = \{h(\tau_i, \sigma_k): i \in \{1, \ldots, n\}; k \in \{1, \ldots, s\}\} \cup \{h(\theta_j, \sigma_k): j \in \{1, \ldots, n\}, k \in \{1, \ldots, s\}\},$$

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 A is defined as the union of arc sets $A_1, \ldots, A_7$ given in Table 1.

Arc set A_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 A_2 and A_3 and between a service trip and a fuel station and depot, respectively; again we add the fuel cost for the origin service trip. Arc sets A_4 and A_5 represent traveling from fuel stations to service trips and depots, and arc sets A_6 and A_7 represent traveling from depots to fuel stations and service trips. An example G

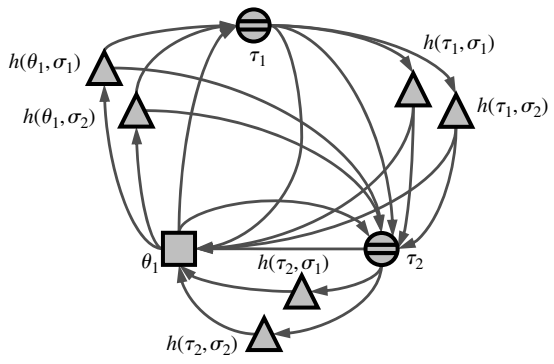
Table 1. The Arcs in Graph $G = (V, A)$, Where $A = \bigcup_{i=1}^{7} A_i$

Set	Definition	Fuel f'	Cost c'	Description
A_1	$\{(\tau_i, \tau_j)\colon \tau_i, \tau_j \in T, \text{comp}(\tau_i, \tau_j)\}$	$f_\tau(\tau_i) + f(\tau_i, \tau_j)$	$c(\tau_i, \tau_j)$	Service trip to service trip
A_2	$\{(\tau_i, \theta_j)\colon \tau_i \in T, \theta_j \in D\}$	$f_\tau(\tau_i) + f(\tau_i, \theta_j)$	$c(\tau_i, \theta_j)$	Service trip to depot
A_3	$\{(\tau_i, h(\tau_i, \sigma_j))\colon \tau_i \in T, \sigma_j \in S\}$	$f_\tau(\tau_i) + f(\tau_i, \sigma_j)$	$c(\tau_i, \sigma_j)$	Service trip to fuel station
A_4	$\{(h(z, \sigma_j), \tau_k)\colon z \in T \cup D, \sigma_j \in S, \tau_k \in T, \text{compf}(z, \sigma_j, \tau_k)\}$	$f(\sigma_j, \tau_k)$	$c(\sigma_j, \tau_k)$	Fuel station to service trip
A_5	$\{(h(z, \sigma_j), \theta_k)\colon z \in T \cup D, \sigma_j \in S, \theta_k \in D\}$	$f(\sigma_j, \theta_k)$	$c(\sigma_j, \theta_k)$	Fuel station to depot
A_6	$\{(\theta_i, \tau_j)\colon \theta_i \in D, \tau_j \in T\}$	$f(\theta_i, \tau_j)$	$c(\theta_i, \tau_j)$	Depot to service trip
A_7	$\{(\theta_i, h(\theta_i, \sigma_j))\colon \theta_i \in D, \sigma_j \in S\}$	$f(\theta_i, \sigma_j)$	$c(\theta_i, \sigma_j)$	Depot to fuel station

is given in Figure 2 representing the AF-VSP in Figure 1. Note that in Figure 2 it is assumed that service trips τ_1 and τ_2 are compatible with each other and with station σ_1.

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 w, 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; 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 $\text{compf}(\tau_i, \sigma_j, \tau_m)$. 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. Furthermore, all fuel requirements are associated with arcs despite the fact that service trips are represented by vertices. Finally, every cycle 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.

Figure 2. A Graph G Corresponding to the AF-VSP in Figure 1



3.1. 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 one if schedule p contains service trip τ_i and zero otherwise. Similarly, let variable u_p^j be one if schedule p starts and ends at depot θ_j and zero 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 one if we include schedule p in the solution and zero otherwise. The integer program is then the following:

$$\text{Minimize} \quad \sum_{p \in \Omega} q_p x_p \tag{1}$$

$$\text{subject to} \quad \sum_{p \in \Omega} v_p^i x_p = 1 \quad i = 1, \ldots, n, \tag{2}$$

$$\sum_{p \in \Omega} u_p^j x_p \le r_j \quad j = 1, \ldots, d, \tag{3}$$

$$x_p \in \{0, 1\} \quad \forall p \in \Omega. \tag{4}$$

Here, the objective (1) is to minimize the sum of all of the costs of the schedules used in the solution. Constraint (2) ensures that each service trip is traversed exactly once. Constraint (3) 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 the one used in 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 of feasible schedules is intractable for problems 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; 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 dual 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 P to add to the restricted master problem.

Let π_i for $i = 1, \ldots, n$ be the solutions for the dual variables associated with constraint (2) of the restricted master problem, and let ρ_j for $j = 1, \ldots, d$ be the solutions associated with constraint (3). 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 the 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 $v_{\bar{p}_i}^i = 1$ and $v_{\bar{p}_i}^{i'} = 0$ for $i' \neq i$, and $u_{\bar{p}_i}^j = 0$ for all j.

3.2. The 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, Boland, and Waterer 2012) on graph G with weights f'' and costs c''. Here 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 $c''(v_1, v_2)$ and

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

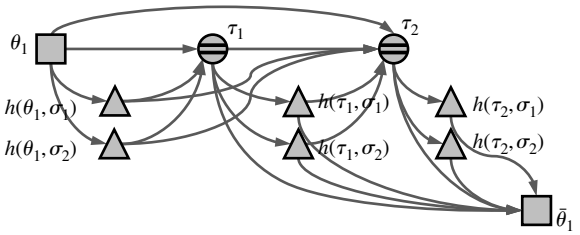
For simplicity, we add a copy of the depot θ_j and label it $\bar{\theta}_j$ and label the original as the start node and the new one as the end node. We also remove all vertices $\theta_{j'} \in D \backslash \{\theta_j\}$. All of the outgoing edges of θ_j are associated with the start node, and all 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 in the paper by Smith, Boland, and Waterer (2012). Note that although 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. Label 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 alternative partial path are removed. For an in-depth discussion of label correcting algorithms for the weight constrained shortest path algorithm, see Desrosiers et al. (1995). Meta-network methods rely on creating a new network that 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, Boland, and Waterer (2012) 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 $s+2$ 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 Figure 2 can be seen in Figure 3. 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

Figure 3. The Graph G from Figure 2 with Replicated Node θ_1, Ordered from Left to Right



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 the 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 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 one of the following: a depot, visiting a station after a depot, an earlier service trip, or visiting a station after an earlier service 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

$$\{\theta_j, h(\theta_j, \sigma_1), \ldots, h(\theta_j, \sigma_s), \tau_1, h(\tau_1, \sigma_1), \ldots, h(\tau_1, \sigma_s), \tau_2, h(\tau_2, \sigma_1), \ldots, h(\tau_2, \sigma_s), \tau_3, \ldots, h(\tau_n, \sigma_s)\}.$$

After labeling each vertex in $T \cup H$, we can find the length of the paths associated with each label and immediately return 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.

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 $T \cup S \cup D$ 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 zero. Thus there can be a single label that dominates all other labels (in the case that both the cost and fuel remaining are equal to several labels, one can be chosen arbitrarily). Each fueling station label set will contain a single element. Furthermore, 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 only be 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 zero 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 Figure 4 with a subfunction in Figure 5. 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

Figure 4. A Function to Solve the Weight Constrained Shortest Path Problem for a Fixed Depot

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 $best_{sch} \leftarrow \{\}$, $best_{cost} \leftarrow \infty$ \\ Store the best route and the cost of the route
Initialize $L(v) \leftarrow \emptyset$ for each $v \in H \cup T$ \\ Store the labels for each vertex
For $k = 1$ to s \\ Initialize the labels of the stations following the depot.
  If $f''(\theta_j, h(\theta_j, \sigma_k)) \le w$
    $L(h(\theta_j, \sigma_k)) \leftarrow \{(c''(\theta_j, h(\theta_j, \sigma_k)), 0, \{\theta_j, \sigma_k\})\}$
  End if
End for
For $i = 1$ to n \\ For each trip in order
  $L(\tau_i) \leftarrow L(\tau_i) \cup (c''(\theta_j, \tau_i), f''(\theta_j, \tau_i), \{\theta_j, \tau_i\})$
    For $k = 1$ to s \\ Find the best label for each station.
      \\ Note that each label set for a vertex in H has a single label.
    $a = \arg\min_a \{l_c(r) + c''(h(a, \sigma_k), \tau_i) \mid \{r\} = L(h(a, \sigma_k)), a \in \{\theta_j, \tau_1, \ldots, \tau_{i-1}\}, \text{compf}(a, \sigma_k, \tau_i)\}$
    $L(\tau_i) \leftarrow L(\tau_i) \cup \{(l_c(r) + c''(h(a, \sigma_k), \tau_i), l_f(r) + f''(h(a, \sigma_k), \tau_i), \{l_{sch}(r), \tau_i\}) \mid \{r\} = L(h(a, \sigma_k))\}$
  End for
  For $i' = 1$ to $i - 1$ \\ Join the labels from the previous trips.
    If $\text{comp}(\tau_{i'}, \tau_i)$
      $L(\tau_i) \leftarrow L(\tau_i) \cup \{(l_c(r) + c''(\tau_{i'}, \tau_i), l_f(r) + f''(\tau_{i'}, \tau_i), \{l_{sch}(r), \tau_i\}) \mid r \in L(t_{i'})\}$
    End if
  End for
  $L(\tau_i) \leftarrow \text{DeleteLabels}(L(\tau_i))$
  For $k = 1$ to s \\ Find the best labels for each station following the trip, and check if the returning
      to the depot after visiting each station generates a new shortest path.
    $r = \arg\min_{r' \in L(\tau_i)} \{l_c(r') \mid l_f(r') + f''(\tau_i, h(\tau_i, \sigma_k)) \le w\}$
    $L(h(\tau_i, \sigma_k)) \leftarrow \{(l_c(r) + c''(\tau_i, h(\tau_i, \sigma_k)), 0, \{l_{sch}(r), \sigma_k\})\}$
    If for $\{r\} = L(h(\tau_i, \sigma_k))$, $l_f(r) + f''(h(\tau_i, \sigma_k), \theta_j) \le w$ and $l_c(r) + c''(h(\tau_i, \sigma_k), \theta_j) < best_{cost}$
      $best_{sch} \leftarrow \{l_{sch}(r), \theta_j\}$
      $best_{cost} \leftarrow l_c(r) + c''(h(\tau_i, \sigma_k), \theta_j)$
    End if
  End for
  $r = \arg\min_{r' \in L(\tau_i)} \{l_c(r') \mid l_f(r') + f''(\tau_t, \theta_j) \le w\}$
  If $l_c(r) + c(\tau_i, \theta_j) < best_{cost}$ \\ Check if returning to the depot after the trip generates a new shortest path.
    $best_{sch} \leftarrow \{l_{sch}(r), \theta_j\}$
    $best_{cost} \leftarrow l_c(r) + c''(\tau_i, \theta_j)$
  End if
End for
Return $(best_{sch}, best_{cost})$

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 w 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 nondominated labels is returned.

Figure 5. A Subfunction of the Function in Figure 4 That Deletes Dominated and Invalid Labels

Function DeleteLabels
On input set L
$L \leftarrow L \backslash \{r \mid r \in L, L_f(r) > w\}$
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 $L_c(r) \ge \bar{c}$
    $L' \leftarrow L' \cup \{r\}$
  Else
    $\bar{c} \leftarrow L_c(r)$
  End if
End for
$L \leftarrow L \backslash L'$
Return L

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. Although 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.

3.3. The Branch-and-Price 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-price approach when the solution is noninteger. Branch-and-bound algorithms are a class of algorithms that find an integer solution to a problem by repeatedly solving linear relaxations of the problem, and if the solution to the linear relaxation is not integer, then the problem branches into multiple subproblems that are also linear relaxations. Branch-and-price algorithms are a special case of branch-and-bound algorithms where column generation is used to solve the subproblem of each branch. In branch-and-price algorithms the columns from the solutions to the linear relaxations are used as starting columns in the branched problems. For an introduction to branch-and-bound techniques see Winston and Venkataraman (2003), and for an introduction to branch-and-price techniques see Barnhart et al. (1998).

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, and then serve trip τ_i would be acceptable, but if a schedule 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-price algorithm progresses, the problems to be solved will have a longer and longer list of pairs that must be included or excluded. These 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., $((T \cup D)\backslash\{a\}) \times \{b\} \subseteq \Gamma$. 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 $\Gamma_1 = \Gamma \cup ((PT \cup D)\backslash\{a\}) \times \{b\}$, 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 3.1, the columns from the previous problem can be used. The only exception is that any of the columns in the previous problem that violated the excluded pair sets must be removed. It is likely that only a small amount of additional columns must 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 3.2 must be adjusted. This can be done simply when aggregating the sets of labels for each service trip by 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 $L(h(\tau_j, \sigma_k))$ for any k can be used.

4. A Heuristic Algorithm for AF-VSP

Because 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, Rosenfield, and Kydes 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 the 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 at which stations to refuel 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.

Although 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. Furthermore, 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.

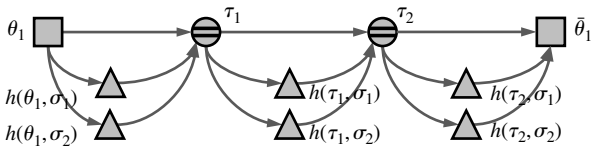
The concurrent scheduler algorithm requires that we have the fuel scheduling algorithm: a method of taking a vehicle assigned to a sequence of service trips and determining where and when the vehicle should stop to refuel with minimal cost. Such a method would be useful for other algorithms as well since 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 is also 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, Gertsch, and Russell 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, 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, similar to the column generation subproblem from Section 3.2. 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, whereas 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 is Figure 6. 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 3.2 where each label is of the form (c, f, sch). Now the labeling algorithm defined above can be performed after changing step 1 for each service trip to the following:

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

Figure 6. Graph G for an Example AF-VSP Where $d = 1$ and $r_r = 1$ for Finding Where on a Sequence of Service Trips to Stop and Refuel



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 returns the minimum cost path that 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.

This algorithm solves the fuel station sequencing problem in $\mathcal{O}(sn^2 \log(sn) + s^2 n)$ time. 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 $|L(\tau_{i-1})| + s$ for $i > 1$ and when $i = 1$ $|L(\tau_1)| = s + 1$ since the only labels are from visiting the fuel station or arriving directly from the depot. This bounds the label set size to $|L(\tau_i)| \leq s(i-1) + 1$, 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 $\mathcal{O}(sn^2 \log(sn))$ computations. Since the labels associated with refueling stations do not need to sort and instead only need to find the minimum cost label, a label for a station following service trip τ_i will require $\mathcal{O}(si)$ computations. Since there are $s \cdot (n+1)$ vertices in H, when all of the calculations attributed to the refueling station vertices are combined, the computations will require $\mathcal{O}(s^2 n)$ time. Therefore, the total complexity of the algorithm is $\mathcal{O}(sn^2 \log(sn) + s^2 n)$.

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 $\mathcal{O}(s^2 n)$ time. Since the fuel requirement is bounded, a vehicle cannot serve more than $\lfloor w/\gamma \rfloor$ trips before refueling, and thus any schedule that has q trips between two stations is infeasible where $q = \lfloor w/\gamma \rfloor + 1$. 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 $\mathcal{O}(sn)$. 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 $\mathcal{O}(s^2 n)$ time. The complexity $\mathcal{O}(s^2 n)$ dominates the overall computational complexity in this case.

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 performed on a number of differently 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. Although 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.

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, Matteo, and Paolo (1993). Their method was altered so that 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, s 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 $\{\lceil n/3\rceil, \lceil n/3+1\rceil, \ldots, \lceil n/2\rceil\}$. 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 $\{\lceil 3 + d/(2.5d)\rceil, \ldots, \lfloor 3 + d/(3.5d)\rfloor\}$. 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 five units of time with a cost of 150. The fuel capacity of a vehicle w is set as $w = 150$.

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 15% probability of being chosen from [420, 480), a 70% probability of being uniformly chosen from [480, 1,019), and a 15% probability of being uniformly chosen from [1,020, 1,080). The ending time et_j was chosen as a uniformly random integer from $[bt_j + \theta(a, b) + 5, \ldots, bt_j + \theta(a, b) + 40]$. 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 2. The costs were different from those in Dell'Amico, Matteo, and Paolo (1993) 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 2,000 for each bus to be used.

All of the results were computed on a personal computer having an Intel Core 2 Duo 2.6 GHz dual core processor and 4 GB RAM that was running Windows 7 Ultimate. All 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 3 shows the results for the column generation algorithm where each row corresponds to the 10 randomly generated instances of a particular problem size. The mean runtime and mean cost columns denote the average time in seconds the algorithm took to compute the result and the average cost of that result. The mean number of vehicles shows the average number of vehicles used in the optimal solution. The mean relaxation gap denotes the average ratio of the cost of the optimal solution over the cost of the relaxed LP problem (minus 1). The mean and max number of nodes shows the average and max number of nodes in the branch-and-price tree, and the mean and max level show the average and maximum number of levels deep the branch-and-price tree went. Finally, the mean number of columns denotes on average how many columns were generated in finding the solution.

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, two stations, and two 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 more than 2.5 hours. Table 4 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: whereas the heuristic never took more than a second to run, the runtimes for the

Table 2. 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
$\tau_i \in T$	$\sigma_j \in S$	$\theta(\tau_i, \sigma_j)$	$10\theta(\tau_i, \sigma_j)$	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
$\sigma_i \in S$	$\tau_j \in T$	$\theta(\sigma_i, \tau_j)$	$150 + 10\theta(\sigma_i, \tau_j)$	Fuel station to service trip
$\sigma_i \in S$	$\theta_j \in D$	$\theta(\sigma_i, \theta_j)$	$150 + 10\theta(\sigma_i, \theta_j)$	Fuel station to depot
$\theta_i \in D$	$\tau_j \in T$	$\theta(\theta_i, \tau_j)$	$2{,}000 + 10\theta(\theta_i, \tau_j)$	Depot to service trip
$\theta_i \in D$	$\sigma_j \in S$	$\theta(\theta_i, \sigma_j)$	$2{,}000 + 10\theta(\theta_i, \sigma_j)$	Depot to fuel station

Table 3. Results from the Exact Column Generation Solution of the AF-VSP from Section 3 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 vehicles	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	1,091	10.52	9.5	0.29	1,100	7,001	12.7	20	7,677
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	3,769	13.43	12.0	0.07	3,577	11,925	16.3	25	16,516
50	4	2	1,616	13.44	12.1	0.05	730	5,835	9.0	22	5,523
50	4	4	1,159	13.49	12.4	0.28	583	4,931	13.0	37	6,188
50	8	4	273	12.77	11.8	0.23	61	141	8.4	15	1,000

exact algorithm went into minutes on the larger sample problems. Out of the 200 (20×10) 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.

5.2. Data from the Valley Metro Bus Service

All of 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,

Table 4. 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	1,091	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	3,769	13.43	12.0	0.21	14.11	12.2	5.0	1.7
50	4	2	1,616	13.44	12.1	0.22	13.93	12.2	3.6	0.8
50	4	4	1,159	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

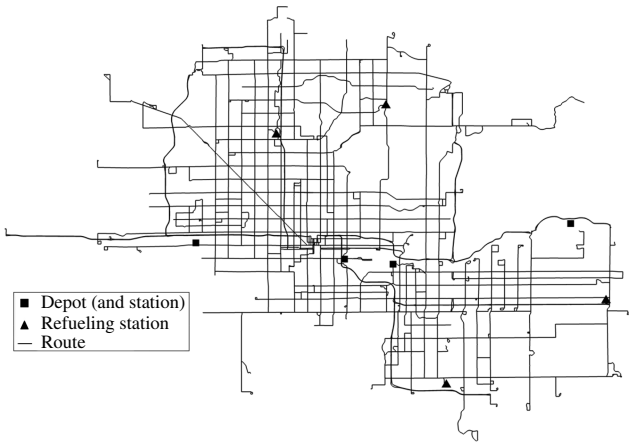
and the trip paths themselves (of which the length of the service trip can be calculated). Four depot locations were found based on publicly available information on where the Valley Metro contractors First Transit and Veolia have facilities located. The four depots were chosen to be fueling stations, 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 because of their locations relative to the depots.

The Valley Metro trips are organized into routes, where a route was a set of service trips that all shared origin and destination points (although they could be traveled in either direction). For example, Valley Metro route 30 traverses University Drive in Tempe and Mesa in both directions, east and west. 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 these data we used Bing Maps API (Microsoft 2013). A map of the 72 routes and the stations and depots can be seen in Figure 7.

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 2. 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.

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, eight stations, and four depots. This random selection was done five times to create five distinct problems on which to test. Each depot was assumed to have 10 buses available at it. The results of the algorithm can be seen in Table 5, 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 two and 12 hours. Interestingly, both algorithms used the same number of buses for each solution in all but one case.

Figure 7. The Valley Metro Network and Station and Depot Locations



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

Table 5. The Results of 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	9,971	8,964	13	1.77	1,941	29	28,153	0.63	9,883	13	10.3
2	14,901	8,077	11	3.03	1,707	32	33,408	0.60	9,591	12	18.7
3	24,134	10,239	16	1.59	5,385	58	61,578	0.74	11,333	16	10.7
4	32,851	10,803	17	3.52	7,467	45	81,609	0.85	11,998	17	11.1
5	37,183	10,614	17	3.66	7,153	46	80,948	0.79	11,468	17	8.1

Table 6. Results from Applying the Concurrent Schedule Algorithm to the 4,373 Service Trips from the Valley Metro Data

Number of service trips	4,373
Number of stations	8
Number of depots	4
Cost	359,827
Number of available buses	600
Number of buses used	477
Runtime (sec)	290

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 6. Of the 600 buses, only 477 of them were assigned, and it took 290 seconds to run the algorithm.

6. Conclusion and Future Work

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 exact column generation algorithm, and a heuristic algorithm to solve the problem we 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, solving the decision problem of deciding where to place a limited number of fuel stations, and modifying the vehicle assignments to a particular problem when there is a sudden change in the network because of breakdowns or accidents. More research could also be done into how to apply heuristic algorithms for the MDVSP to the AF-VSP; in the online appendix we give an overview on how this conversion can be done for several MDVSP heuristics.

Acknowledgments

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

References

Bakker J (2011) Contesting range anxiety: The role of electric vehicle charging infrastructure in the transportation transition. Unpublished doctoral dissertation, Eindhoven University of Technology, Eindhoven, Netherlands.

Ball M (1980) A comparison of relaxations and heuristics for certain crew and vehicle scheduling problems. *ORSA/TIMS Meeting, Washington, DC*.

Barnhart C, Johnson E, Nemhauser G, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Bodin L, Rosenfield D, Kydes A (1978) UCOST: A micro approach to a transportation planning problem. *J. Urban Anal.* 5(1):47–69.

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(2):63–211.

Botsford C, Szczepanek A (2009) Fast charging vs. slow charging: Pros and cons for the new age of electric vehicles. *EVS24 Internat. Battery Hybrid Fuel Cell Electr. Vehicle Sympos., Stavanger, Norway*, 1–9.

Bunte S, Kliewer N (2009) An overview on vehicle scheduling models. *Public Transportation* 1:299–317.

Ceder A (2002) Urban transit scheduling: Framework, review and examples. *J. Urban Planning Development* 128(4):225–244.

City of Phoenix (2013) City of Phoenix public transit developers portal. Accessed October 2013, http://phoenix.gov/publictransit/developers/index.html.

Daduna JR, Paixão JMP (1995) Vehicle scheduling for public mass transit—An overview. Daduna JR, Branco I, Paixão JMP, eds. *Computer-Aided Transit Scheduling*, Lecture Notes Econom. Math. Systems, Vol. 430 (Springer-Verlag, Berlin Heidelberg), 76–90.

Dell'Amico M, Matteo A, Paolo F (1993) Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Sci.* 39(1):115–125.

Desaulniers G, Hickman MD (2007) Public transit. Barnhart C, Laporte G, eds. *Transportation*, Handbooks Oper. Res. Management Sci., Vol. 14 (North-Holland, Amsterdam), 69–127.

Desaulniers G, Lavigne J, Soumis F (1998) Multi-depot vehicle scheduling problems with time windows and waiting costs. *Eur. J. Oper. Res.* 111(3):479–494.

Desrosiers J, Lübbecke ME (2005) A primer in column generation. Desaulniers G, Desrosiers J, Solomon MM, eds. *Column Generation* (Springer, New York), 1–32.

Desrosiers J, Dumas Y, Solomon MM, Soumis F (1995) Time constrained routing and scheduling. Ball MO, Magnanti TL, Monma CL, Nemhauser GL, eds. *Network Routing*, Handbooks Oper. Res. Management Sci., Vol. 8 (North-Holland, Amsterdam), 35–139.

Diestel R (2000) *Graph Theory* (Springer-Verlag, Heidelberg, Germany).

Dror M (2000) *Arc Routing: Theory, Solutions and Applications* (Kluwer Academic Publishers, Norwell, MA).

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(1–2):349–357.

Freling R, Paixão JMP (1995) Vehicle scheduling with time constraint. Daduna JR, Branco I, Paixão JMP, eds. *Computer-Aided Transit Scheduling*, Lecture Notes Econom. Math. Systems, Vol. 430 (Springer-Verlag, Berlin Heidelberg), 130–144.

Golden BL, Wong RT (1981) Capacitated arc routing problems. *Networks* 11(3):305–315.

Haghani A, Banihashemi M (2002) Heuristic approaches for solving large-scale bus transit vehicle scheduling problem with route time constraints. *Transportation Res. Part A* 36(4):309–333.

Heineman GT, Pollice G, Selkow S (2008) *Algorithms in a Nutshell* (O'Reilly Media, Inc., Sebastopol, CA).

Lin SH, Gertsch N, Russell JR (2007) A linear-time algorithm for finding optimal vehicle refueling policies. *Oper. Res. Lett.* 35(3):290–296.

Microsoft (2013) Bing maps. http://www.microsoft.com/maps/.

Ogden JM, Steinbugler MM, Kreutz TG (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(2):143–168.

Oukil A, Amor HB, Desrosiers J, El Gueddari H (2007) Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. *Comput. Oper. Res.* 34(3):817–834.

Ribeiro CC, Soumis F (1994) A column generation approach to the multiple-depot vehicle scheduling problem. *Oper. Res.* 44(1): 41–52.

Smith OJ, Boland N, Waterer H (2012) Solving shortest path problems with a weight constraint and replenishment arcs. *Comput. Oper. Res.* 39(5):964–984.

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. *Transportation Sci.* 44(3):367–382.

Valley Metro (2013) Valley Metro fact sheets. http://www.valleymetro.org/publications_reports/fact_sheets.

Wang H, Shen J (2007) Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints. *Appl. Math. Comput.* 190(2):1237–1249.

Wei M, Jin W, Fu W, Hao X (2010) Improved ant colony algorithm for multi-depot bus scheduling problem with route time constraints. *8th World Congress Intelligent Control Automation, Jinan, China*, 4050–4053.

Winston WL, Venkataraman M (2003) *An Introduction to Mathematical Programming*, 4th ed. (Brooks/Cole, Pacific Grove, CA).

Zhu C, Chen X, Wu J, Ye J (2012) Assessment of battery electric bus operation in Shanghai Expo. *91st Annual Meeting Transportation Board, Washington, DC*, 3182.