Optimizing Ridesharing Services for Airport Access

Lei Feng
Department of Civil & Environmental Engineering
University of Maryland
1173 Glenn L. Martin Hall, College Park, MD 20742
Tel: (240) 475-1723
Email: lfeng@umd.edu

Elise Miller-Hooks
Department of Civil & Environmental Engineering
University of Maryland
1173 Glenn L. Martin Hall, College Park, MD 20742
Tel: (301) 405-2046
Email: elisemh@umd.edu

Paul Schonfeld
Department of Civil & Environmental Engineering
University of Maryland
1173 Glenn L. Martin Hall, College Park, MD 20742
Tel: (301) 405-1954
Email: pschon@umd.edu

and

Matthew Mohebbi
IT Curves (and Supreme Airport Shuttle, Inc.)
Gaithersburg, MD 20879
Email: mmohebbi@itcurves.net

Word Count: 5,971 + (3 Figures + 3 tables)*250 = 7,482 words

Revised from original submission

Submitted for presentation at the 2014 Transportation Research Board 93rd Annual Meeting and for publication in the Transportation Research Record
ABSTRACT
This paper addresses the problem of optimally routing and scheduling airport shuttle vehicles that offer pickup and dropoff services to customers through ridesharing. This problem, which is a version of the Dial-A-Ride Problem (DARP), is formulated as a mixed integer program. An exact solution approach applying constraint programming within a column generation framework is proposed, as well as adaptations of two existing heuristics, for its solution. Implementations of the mathematical program and proposed solution approaches for three different operational policies are presented. Performance of the proposed approaches in terms of computational requirements and solution quality is evaluated on a real-world case study involving service records for one service day out of Washington Dulles International Airport provided by an actual airport shuttle service provider. Results show that the heuristics provide good approximations to the exact solution.
INTRODUCTION

This paper addresses the problem of optimally routing and scheduling airport shuttle vehicles that offer pickup and dropoff services to customers through ridesharing. This work was motivated by a need for tools to support efficient resource management at Supreme Airport Shuttle, Inc. This company provides ridesharing services to customers travelling to/from two major airports in the Washington, D.C. area. In its outbound operations, they have a fleet of vehicles used to pick up customers from the airport’s arrival doors and drop them at customer-chosen destinations. The vehicles also provide inbound services in which they pick up customers at multiple origins outside the airport and drop them at the airport’s departure doors. Customers request services by phone, online or at a kiosk in the airport or hotel. Each request includes information on the number of passengers, pickup location and time, and (or) dropoff location and time. A single request can be for a one-way trip (outbound or inbound) or a round trip (outbound and inbound). Each request results in a trip from the arrival door of the airport to the trip’s destination or from the trip’s origin to the departure door of the airport. Thus, requests can be made in advance or may arise dynamically on the same day of service. One or more trips are served by one vehicle through a route, which is defined by a circuit that is travelled by a vehicle starting from and ending at the holding lot in the airport. Each vehicle may have multiple routes during a shift.

Reduced total passenger-miles traveled resulting from ridesharing and efficiently designed routes can increase profitability of the service provider and aid in diminishing traffic congestion and related negative externalities, including environmental pollution. Thus, an optimization model is proposed for the problem of determining a set of routes and schedules that meet service quality, resource, labor and vehicle capacity constraints while minimizing total cost in terms of vehicular use and total wages in the context of airport ridesharing services. This problem, which is a version of the Dial-A-Ride Problem (DARP), is called the Airport Access Ridesharing Problem (AARP) here.

The AARP is considered under three different operational policies that are illustrated in FIGURE 1. **Policy 1** handles outbound and inbound trips separately, assigning different sets of vehicles to each. This policy is under consideration, because in current operations demand for outbound service far outweighs the demand for inbound service. **Policy 2** handles outbound and inbound trips simultaneously, permitting a single vehicle to drop off outbound and pick up inbound customers at all points along the route. Under **Policy 3**, all outbound trips must be dropped off before the same vehicle starts picking up inbound trips. This last policy gives preference to outbound customers, the majority of the company’s
actual customers. While one can show that Policy 2 will always produce the most efficient routes for the
customer, other policies must be considered under certain contractual agreements or passenger service
policies.

The AARP is a difficult combinatorial optimization problem; the number of possible solutions for
it grows exponentially with increasing problem size. Even obtaining a single feasible solution by hand can
be quite challenging. Yet, efficient use of limited resources is key to providing profitable, quality service
that, by contract, meets service level agreement requirements. Thus, for real-world problem instances,
tools to support the identification of feasible and optimal or near-optimal solutions can be crucial. An
exact solution algorithm and two heuristics are proposed to solve the AARP. The exact solution applies a
Constraint Programming based Column Generation (CPCG) approach (1). The first heuristic is a variant
of the sequential insertion heuristic proposed by Jaw (2) for a related dial-a-ride problem (DARP). The
second is adapted from Solomon’s work on the Vehicle Routing Problem with Time Windows (3).
Performance of the proposed heuristics is compared in a case study involving data from one day’s
operation of the Supreme Airport Shuttle fleet at one airport. The solution approaches were implemented
and adapted to the three operational policies. Results from runs of the algorithms are analyzed and
compared based on a variety of performance measures.

RELATED LITERATURE
The AARP shares several characteristics of a variety of established optimization problems. First, the
AARP is related to the Vehicle Routing Problem with Time Windows (VRPTW) (4). In the VRPTW, a
vehicle must arrive at each customer within a given time window. Where soft time window constraints are
permitted, a penalty for early or late arrival may be incurred. For comprehensive reviews of optimization
algorithms for VRPTW, the reader is referred to (6-9). The AARP similarly has time windows; however,
these constraints are hard. Thus, any solution that violates these constraints is infeasible. The AARP
differs from the VRPTW by also including maximum ride time constraints needed to control the time
spent by each passenger traveling in the vehicle, as well as maximum shift durations. Moreover, customer
stops are paired in the AARP, since each customer has a pair of pickup and dropoff locations, and the
pickup must be completed before the dropoff. This creates additional precedence constraints. Such pairing
and precedence of customers are captured in a generalization of the VRPTW, the Pickup and Delivery
Problem with Time Windows (PDPTW) (10).

As in the AARP, in the PDPTW the origin of each request must precede its destination on each
vehicle tour, and both locations must be visited by the same vehicle. Among the PDPTWs in the literature,
those that address the DARP (2) are most relevant. Comprehensive surveys of optimization algorithms on
the PDPTW are provided in (11) and (12), and on the DARP in (13) and (14). The DARP involves
passenger transportation between paired pickup and delivery points and takes user inconvenience into
account. The AARP can be viewed as a special case of the DARP with one-to-many and many-to-one
operations. See (15) for a discussion of this variant for a single vehicle. The AARP with Policy 1 or 2 can
be treated as a PDPTW; however, operational Policy 3 requires that inbound movements not begin until
outbound movements are complete. This variant does not appear to have been addressed previously.

The Vehicle Routing Problem with Backhauls and Time Windows (VRPBTW) specifically
captures the one-to-many and many-to-one characteristics of the AARP. The VRPBTW involves linehaul
and backhaul operations. In the linehaul operations, the loading of goods onto a vehicle is completed at
one or more depots, and goods are transported to one or more destinations. In backhaul operations, once
linehaul operations are complete (or partially complete), goods are loaded at the linehaul destinations or
other convenient locations, and transported to the depot, the destination of the backhaul deliveries. A
comprehensive survey of algorithms and applications for the VRPB, including the VRPBTW, is given in
(16). Following the classification scheme in (16), the AARP using Policy 1 can be viewed as two
separated DARPs, one addressing the outbound trips and the other the inbound trips. The AARP using Policy 2 can be defined as a Vehicle Routing Problem with Simultaneous Delivery and Pickup and Time Windows (VRPSDPTW). The AARP using Policy 3 can be classified as a Vehicle Routing Problem with Clustered Backhauls and Time Windows (VRPCBTW). The VRPCBTW does not deal with pairing and precedence of service points, a crucial characteristic of the AARP.

Limited works in the literature address the specifics of the VRPBWT and its VRPSBTW and VRPCBTW variants. The VRPBWT has been formulated as a mixed integer linear program, but only relatively small instances can be solved to optimality. An exact solution approach for the VRPSBTW is presented in (17). Specifically, a column generation framework is proposed in which the problem is decomposed into a Master Problem (MP) and Subproblem (SP). The MP is formulated as a set covering problem and branch-and-price is proposed for solution of the SP. Solution of the SP supplies a feasible route for inclusion in the set of possible routes considered in the MP. The largest instance solved to optimality had 20 customers. Yano et al. (18) formulated the VRPCBTW as a set partitioning problem. They proposed an exact solution method based on branch-and-bound to generate optimal tours, each with a maximum of four linehaul and four backhaul customers. For the same problem with possibly more than four customers, Gelinas et al. (19) proposed an exact algorithm based on column generation for solving a similar set partitioning formulation of the VRPCBTW. The algorithm found optimal solutions for problems with up to 100 customers.

While promising, exact solution methods cannot be applied to typical problems of a size seen in real-world operations. Thus, numerous works have proposed heuristics for these problems. The majority of these heuristics include construction and improvement schemes based on classical greedy methods. More powerful methods have been proposed based on met heuristics. For example, Dethloff (20) proposed an extension of the cheapest insertion heuristic for the VRPSBTW. Thangiah et al. (21) proposed a heuristic for solution of the VRPCBTW. In the construction phase, the insertion procedure of Kontoravdis and Bard (22) proposed for the VRPTW is used to obtain initial solutions. Then, the initial solutions are improved through the application of λ-interchanges and 2-opt exchanges in the improvement phase. Duhamel et al. (23) uses an insertion procedure proposed in (3) for initial solution construction, but proposed a tabu search heuristic for the improvement phase. An augmented objective function for the VRPCBTW is presented by Zhong and Cole (24), where violations of time windows, capacity and linehaul-backhaul precedence constraints are penalized. The cluster-first route-second method is used for initial route construction and intra- and inter-route operators are described for use in improving the tours. Metaheuristics, such as Tabu Search (23), genetic algorithms (25) and ant colony optimization (26), have also been proposed.

In the next section, a general formulation is presented that can be used to solve all three variants of the AARP. This formulation combines aspects of previously proposed formulations for the PDPTW and VRPTW. It fills the need for a formulation of a PDPTW with linehaul and backhaul operations or the VRPCBTW with additional pairing and precedence constraints, i.e. AARP with Policy 3. The proposed formulation does not rely on a complete enumeration of the feasible tours, which is a requirement of the set partitioning formulation of the VRPCBTW and set covering approach for the VRPSBTW. The formulation given next includes additional constraints specific to an application involving passengers as opposed to cargo, including maximum passenger ride times and restrictions on idling with passengers onboard.

**PROBLEM FORMULATION**

In this section, the AARP is formulated. The formulation is preceded by the introduction of notation. Additional adaptations needed for different operational policy implementations are given.
1 Notation

\[ n_O \quad \text{number of outbound trips/requests} \]
\[ n_I \quad \text{number of inbound trips/requests} \]
\[ P_O \quad \text{set of outbound pickup nodes located at the arrival door, } P_O = \{1, \ldots, n_O\} \]
\[ P_I \quad \text{set of inbound pickup nodes, } P_I = \{n_O + 1, \ldots, n_O + n_I\} \]
\[ D_O \quad \text{set of outbound dropoff nodes, } D_O = \{n_O + n_I + 1, \ldots, 2n_O + n_I\} \]
\[ D_I \quad \text{set of inbound dropoff nodes located at the departure door } D_I = \{2n_O + n_I + 1, \ldots, 2n_O + 2n_I\} \]
\[ P \quad \text{set of all pickup nodes, } P = P_O \cup P_I \]
\[ D \quad \text{set of all dropoff nodes, } D = D_O \cup D_I \]
\[ V \quad \text{set of available vehicles} \]
\[ q_i \quad \text{demand/supply at node } i; \text{ for pickup nodes, } q_i > 0, \forall i \in P; \text{ for dropoff nodes, } q_i < 0, \forall i \in D; \text{ for the holding lot, } q_0 = q_{2n_O + 2n_I + 1} = 0. \]
\[ e_i \quad \text{earliest service time at node } i, \text{ i.e. the start of the time window} \]
\[ l_i \quad \text{latest service time at node } i, \text{ i.e. the end of the time window} \]
\[ s_i \quad \text{service duration or dwell at node } i \]
\[ c_{ij}^v \quad \text{cost to travel from node } i \text{ to node } j \text{ with vehicle } v \]
\[ t_{ij}^v \quad \text{travel time from node } i \text{ to node } j \text{ with vehicle } v \]
\[ Q^v \quad \text{capacity of vehicle } v \]
\[ T^v \quad \text{shift duration of vehicle/route } v \]
\[ R_i \quad \text{maximum ride time of request } i \]

2 Decision variables:

\[ x_{ij}^v = \begin{cases} 
1, & \text{if arc}(i, j) \text{ is traversed by vehicle } v \\
0, & \text{otherwise} 
\end{cases} \]
\[ L_i^v \quad \text{load of vehicle } v \text{ leave node } i \]
\[ A_i^v \quad \text{arrival time of vehicle } v \text{ at node } i \]
\[ B_i^v \quad \text{time of beginning service of vehicle } v \text{ at node } i \]

With this notation, the AARP problem can be modeled on a digraph \( G = (N, A) \), where \( N \) is the set of all nodes, \( N = P \cup D \cup \{0, 2n_O + 2n_I + 1\} \) and \( A \) is the set of directed arcs, \( A = \{(i, j) : i, j \in N, i \neq 2n_O + 2n_I + 1, j \neq 0, i \neq j\} \).

7 The General AARP Formulation

The general formulation of the AARP builds on the existing formulations for the VRPBTW (16) and
PDPTW (27).

\[
\begin{align*}
\text{min} & \quad \sum_{v \in V} \sum_{(i, j) \in A} c_{ij}^v \cdot x_{ij}^v + \sum_{v \in V} \sum_{j \in P} C_w (B_j^v - A_j^v) \\
\text{s.t.} & \quad \sum_{v \in V} \sum_{j \in (i, j) \in A} x_{ij}^v = 1 & \quad \forall i \in P, \\
& \quad \sum_{j \in (i, j) \in A} x_{ij}^v - \sum_{j \in (n_o + n_i + i, j) \in A} x_{n_o + n_i + i, j}^v = 0 & \quad \forall i \in P, v \in V, \\
& \quad x_{0j}^v = 1 & \quad \forall v \in V, \\
& \quad \sum_{j \in (i, j) \in A} x_{ji}^v - \sum_{j \in (i, j) \in A} x_{ij}^v = 0 & \quad \forall i \in P \cup D, v \in V, \\
& \quad \sum_{i \in (2n_o + 2n_j + 1) \in A} x_{ij}^v = 1 & \quad \forall v \in V, \\
& \quad -M (1 - x_{ij}^v) \leq B_j^v - B_i^v - s_i - t_{ij}^v \leq M (1 - x_{ij}^v) & \quad \forall (i, j) \in A, v \in V, \\
& \quad -M (1 - x_{ij}^v) \leq A_j^v - A_i^v - t_{ij}^v \leq M (1 - x_{ij}^v) & \quad \forall (i, j) \in A, v \in V, \\
& \quad L_v^0 = L_{2n_o + 2n_j + 1}^v = 0 & \quad \forall v \in V, \\
& \quad -M (1 - x_{ij}^v) \leq L_j^v - L_i^v - q_j \leq M (1 - x_{ij}^v) & \quad \forall (i, j) \in A, v \in V, \\
& \quad e_i - A_i^v < M (1 - y) & \quad \forall i \in N, v \in V, \\
& \quad -(L_i^v - q_i) \leq M \cdot y & \quad \forall i \in N, v \in V, \\
& \quad \max(0, q_i) \leq L_i^v \leq \min(Q^v, Q^v + q_i) & \quad \forall i \in N, v \in V, \\
& \quad \max(e_i, A_i^v) \leq B_i^v \leq l_i & \quad \forall i \in N, v \in V, \\
& \quad B_i^v + r_{i, n_o + n_i + i} \leq B_{n_o + n_i + i}^v & \quad \forall i \in P, v \in V, \\
& \quad B_{2n_o + 2n_j + 1}^v - B_0^v \leq T^v & \quad \forall v \in V, \\
& \quad B_{n_o + n_i + i}^v - (B_i^v + s_i) \leq R_i & \quad \forall i \in P, v \in V, \\
& \quad x_{ij}^v \in \{0, 1\} & \quad \forall (i, j) \in A, v \in V.
\end{align*}
\]

The objective function (1) minimizes total routing cost while penalizing waiting times. \(C_w\) is unit cost of vehicle waiting. The cost \(c_{ij}^v\) in the function is expressed in equation (19), which includes costs related to vehicle travel distance and time.

\[
c_{ij}^v = C_d \cdot d_{ij} + C_t \cdot t_{ij}^v, \tag{19}
\]

where \(C_d\), and \(C_t\) are unit costs of vehicle travel distance and travel time, respectively. \(d_{ij}\) is the
distance between node $i$ and $j$.

Constraints (2) and (3) ensure that every node is visited exactly once and pickup and dropoff nodes associated with a particular request are visited by the same vehicle, respectively. Each route starts and ends at a holding lot as required in Constraints (4) and (6), respectively. Constraints (5) enforce flow conservation. Constraints (7)-(8) and (9)-(10) guarantee consistency between time and load variables, respectively. Constraints (11) and (12) ensure that a vehicle does not idle while carrying passengers. Capacity and time window constraints are imposed by inequalities (13) and (14), respectively. Constraints (15) force the pickup node to be visited before the dropoff node for each request. The maximum route duration is restricted in Constraints (16). The passenger maximum ride time constraints are specified in inequalities (17), followed by integrality constraints expressed by Constraints (18). Constraints (2), (4)-(6), (13) and (15) are used in both formulations of (16) and (27). Constraints (7), (9) and (15) are included in the formulation given in (16), while constraints (3) and (14) are included in the formulation of (27). Constraints (8), (10)-(12), (16)-(17) are unique to the AARP. Note that if all vehicles are identical, superscript $v$ in $c_{ij}^v$, $t_{ij}^v$, $Q^v$ and $T^v$ can be eliminated for the AARP formulation.

The formulation is designed to be general and directly applicable for Policy 2. Small adaptations are required for the application of Policies 1 and 3 as described next.

**Adaptation for Policy 1**

To apply the formulation where Policy 1 is implemented, the problem can be posed as two separate DARPs, one for outbound trips and the other for inbound trips. To specify the DARP for outbound trips, the AARP formulation can be applied by presetting certain variables. Specifically, $n_I = 0$, $P_I = D_I = \phi$ for the outbound problem and $n_O = 0$, $P_O = D_O = \phi$ for the inbound problem.

**Adaptation for Policy 3**

For the AARP under Policy 3, additional constraints (20) are required to ensure that each vehicle drops off its outbound passengers before picking up its inbound passengers. This can be implemented by restricting arcs between inbound and outbound customer location nodes. That is, no arc can exist in a route directly connecting any inbound pickup location to an outbound dropoff location. This precludes any tour from allowing a sequence in which an inbound request is served before all outbound dropoffs are completed.

$$x_{ij}^v = 0 \quad \forall i \in P_I, j \in P_O \cup D_O, v \in V. \quad (20)$$

The AARP is difficult to solve directly, since the number of decision variables increases exponentially with increasing problem size (number of nodes to be visited). The proposed formulation was implemented directly in the IBM ILOG CPLEX package on a personal computer with Intel(R) CPU 3.10GHz and 4.0GB RAM. The required computational time was exceptionally long. In a reduced version of the problem instance with only 10 outbound trips and 10 available vehicles, the solution was obtained after more than 6 hours, which is unacceptable in practice. Thus, in the next section, an alternative exact solution method is proposed.

**EXACT SOLUTION METHOD**

A CPCG solution methodology is proposed for exact solution of the AARP. A column generation mechanism is employed wherein the AARP is decomposed into a master problem (MP) and a subproblem (SP). A restricted linear relaxation of the MP (LMP) is solved for a given route set. Optimal dual
variables associated with the requests served are determined in solving the SP. The dual variables are used to calculate the reduced costs associated with the tours (columns). At each iteration, a tour with negative reduced cost is added to the route set of the MP. Solution of the SP is obtained through a constraint programming (CP) methodology. An overview of the proposed CPCG methodology is provided in Figure 2.

The procedure starts by feeding the LMP, modeled as a set covering problem, with a feasible solution consisting of a set of vehicles, each of which serves one request. The SP is a constrained shortest path problem. Solution of the SP produces a route with negative reduced cost. This route is added to the route set (or column set) used in the next iteration in which solution of the LMP is repeated. This process terminates when solution of the SP does not produce a route that is not already included in the column pool with negative reduced cost. With the final column pool (route set), the integer MP, a set partitioning problem, is solved. The route obtained from solution of the integer MP is reassembled through a proposed heuristic procedure to generate the final solution. Details associated with the MP, SP and Heuristic Reassemble procedure are provided next.
Master Problem

Assume that all vehicles are identical and let $\Omega$ denote the set of feasible routes satisfying constraints (3)-(17). For each route, $r \in \Omega$, let $c_r$ be the cost of the route and $a_{ir}$ be a binary constant indicating whether or not a node $i \in P$ is visited by route $r$. Let $y_r$ be a binary variable equal to 1 if route $r \in \Omega$ is selected, and 0 otherwise. The AARP can be reformulated as the following set partitioning problem (MP-SPP):

$$\begin{align*}
(MP-SPP) \quad \min & \sum_{r \in \Omega} c_r \cdot y_r \\
\text{s.t.} & \sum_{r \in \Omega} a_{ir} y_r = 1 \quad \forall i \in P \\
& y_r \in \{0,1\} \quad \forall r \in \Omega
\end{align*}$$

The objective (21) minimizes the cost of the chosen routes. Constraints (22) ensure that every request is served once.

It is impractical to explicitly enumerate all feasible routes in $\Omega$. Instead, as is typical, only a subset $\Omega' \subset \Omega$ is considered. This subset is expanded iteratively by adding a route with negative reduced cost through solution of the SP. The reduced cost of a route is expressed by equation (24)

$$\hat{c}_{ij} = \begin{cases} c_{ij} - \pi_i, & \forall i \in P_s(i,j) \in A \\ c_{ij}, & \forall i \notin P_s(i,j) \in A \end{cases}$$

where $\pi_i$ is the dual value associate with the $i$th constraint (22). The LMP is given next.

$$\begin{align*}
(LMP) \quad \min & \sum_{r \in \Omega'} c_r \cdot y_r \\
\text{s.t.} & \sum_{r \in \Omega'} a_{ir} y_r \geq 1 \quad \forall i \in P \\
& y_r \geq 0 \quad \forall r \in \Omega'
\end{align*}$$

This relaxation allows every request to be served more than once rather than only once. Constraints (27) relax integrality constraints.

Subproblem

The SP formulation is:

$$\begin{align*}
(SP) \quad \min & \sum_{(i,j) \in A} r c_{ij} \cdot x_{ij} + \sum_{v \in F} \sum_{j \in P} C_w \cdot (B_j - A_j) \\
\text{s.t.} & \text{Constraints (3)-(17).}
\end{align*}$$
CP Model for SP

In the CP approach, each decision variable has a domain. For example, in the SP, the domain of each arc, $x_{ij}$, is $\{0, 1\}$. Similarly, the domain of load $L_i$ is $\{0, 1, \ldots, Q\}$. Initially, the search space contains all combinations of the values in the domains of all decision variables. To avoid exploring the entire search space, CP first removes inconsistent values from the domains of the variables involved in each constraint. Then a search strategy (depth first, width first or multi-start) is applied to guide the search for a solution within the reduced search space. The search process can be viewed as traversing a tree, where the root is the starting point, a leaf node is a combination of values in the reduced search space and each branch represents a move (branching) within the search. A solution is a set of value assignments to the decision variables such that each variable is assigned to exactly one value from its domain. Together these values satisfy all constraints and minimize the objective function. Each leaf node is evaluated to determine if it will produce a feasible solution.

Two measures are suggested for speeding up the process of finding a feasible solution: 1) eliminate ineligible decision variable settings from the initial search space, wherein those decisions that include starting from the end depot, ending at the starting depot, selfloops, or that would violate Constraints (13) - (15) are excluded, and 2) set branching limits for the route generation process. As mentioned in (28), in the context of column generation, optimality of the SP is only necessary to prove that no negative reduced cost routes exist in the last iteration, and feasible solutions to the SP are sufficient for preceding iterations. Thus, a lower branching limit ($10^6$) is used for these nonfinal iterations, while higher branching limits ($10^8$) are applied in the last iteration.

Heuristic Reassemble Procedure

The optimal solution to the LMP is obtained when there are no remaining routes with negative reduced cost to the SP. Unfortunately, this solution is not always integer-valued. A branching scheme was proposed in (10) to address this issue through adding additional arc flow constraints to the SP and resolving it. This process is repeated for each branching decision taken in the MP. The following observations are made:

1. The MP starts with a feasible solution in which one vehicle serves one request;
2. Each iteration generates a single unique route;
3. The newly generated routes that are selected by solution of the MP-SPP are always a subset of the newly generated routes that are selected by solution of the set covering problem.
4. The solution to the MP-SPP always includes one or more initial feasible routes.

Since solution from MP-SPP provides useful information, the following heuristic applies:

Step 1. Calculate the value of $V = \text{route cost/number of request served}$ for each route selected by the MP-SPP.

Step 2. Select the route $r$ with maximum $V$. Try to extract the first unvisited request on route $r$ and insert it into the best feasible position on one of the other routes, $r'$. If this decreases the total cost, update $r$ and $r'$, and go to Step 1. Otherwise, mark this request as visited and move to the next request in $r$. If all requests in $r$ have been visited, stop.

Step 3. Repeat Step 1
HEURISTIC SOLUTION APPROACHES

Two heuristics proposed in the literature were modified for solution of the AARP. An overview of each is given first, followed by the modifications required to address the three variants of the AARP.

Jaw’s Heuristic

The first heuristic considered for solution to the AARP is the sequential insertion procedure originally proposed by Jaw (2) for the DARP problem. The algorithm processes each request in an unrouted request list (URL) in sequence, and assigns each request to a vehicle until the URL is exhausted. The main steps of Jaw’s sequential insertion procedure are summarized as follows:

Step 1: Sort URL by the requested pickup times in increasing order. Create a route from and back to the depot. Set \( r = 1 \).

Step 2: Select the first unrouted request \( u \) from URL. Find all feasible insertion positions within all existing routes, 1 to \( r \).

(i) If a feasible insertion position is found, assign the request \( u \) to the route \( r^* \) with minimum insertion cost, and update route \( r^* \).

(ii) If no feasible insertion position exists, create a new route from the depot to request \( u \), and add a return to the depot. Set \( r = r + 1 \).

Delete \( u \) from URL.

Step 3: Repeat step 2 until URL is empty.

The additional insertion cost to route \( r \) of inserting request \( u \) is calculated as the difference between the total cost of route \( r \) after the insertion minus its cost before the insertion. This is expressed in (29)

\[
\sum_{i,j \in \text{new}^r} c_{ij}^r - \sum_{i,j \in \text{old}^r} c_{ij}^r,
\]

where \( \text{new}^r \) denotes route \( r \) after insertion of request \( u \) and \( \text{old}^r \) denotes route \( r \) before insertion of request \( u \).

Solomon’s Insertion I1 Heuristic

The second heuristic considered here, Insertion I1 (I1), was proposed by Solomon (3) for the VRPTW. I1 constructs routes one at a time. For the first created route, a tour is developed from the depot to a “seed” request, which returns to the depot. Remaining requests are considered for insertion in the route. The cost of insertion of all remaining unrouted requests is computed. The request with the minimum insertion cost that can be feasibly inserted is selected. Insertion of additional requests is considered until no remaining unrouted request can be feasibly inserted. A new route is then created. The process is repeated until all requests have been included in a tour. At each iteration in which a new route is created, the remaining unrouted request with the minimum value of \(-\alpha d_{0i} + \beta l_i, 0 \leq \alpha \leq 1, i \in D_0 \) for outbound trips and \( i \in P_I \) for inbound trips is selected as the seed. Trips that are far from the depot and have an earlier deadline are, thus, favored in choosing the request.
The main steps of I1 can now be summarized:

Step 1: Initialize \( r = 0. \)

Step 2: Set \( r = r+1. \) Select the ‘seed’ request \( u^* \) with the minimum value of \(-\alpha d_{0u} + \beta l_i\) from URL for inclusion in route \( r. \) Add \( u^* \) to route \( r \) and delete it from URL. If URL is empty, stop.

Step 3: For each remaining unrouted request \( u \) in URL, find the feasible insertion position in route \( r, \) if a feasible insertion exists, that minimizes the additional insertion cost (equation (30)).

(i) If a feasible insertion exists, select request \( u^* \) with the minimum additional insertion cost (equation (30)), and insert this request at its best feasible insertion position in route \( r. \) Update route \( r, \) and delete \( u^* \) from URL.

(ii) If there is no feasible insertion of any unrouted request in route \( r, \) go to step 2.

Step 4: Repeat step 3 until URL is empty.

The two heuristics are quite similar, but differ in one fundamental aspect relating to the choice of a feasible insertion position for the unrouted requests. In Jaw’s heuristic, for each selected unrouted request \( u, \) its best insertion position within all constructed routes is evaluated and the insertion is made accordingly. When a request cannot be feasibly inserted in any existing route, a new route is constructed. The request is inserted in the new route. The next unrouted request from a list that was not yet tested will be considered for inclusion in this expanded set of constructed routes. In I1 this evaluation is conducted over only the most recently constructed route. The list of unrouted requests must be considered and any possible insertions must be made in that route before considering insertion in another route.

Both heuristics as described can be used directly on the AARP with Policies 1 and 2. For Policy 3, however, feasibility is further restricted by outbound and inbound trip separation requirements. Both heuristics can be adapted to deal with this additional constraint. Specifically, modifications are made when choosing the best feasible insertion position for each unrouted request: if the selected unrouted request \( u \) is an outbound trip, its dropoff location must be inserted before the pickup of the first inbound trip, given that there are inbound trips in the current route. Likewise, if the unrouted request \( u \) is an inbound trip, its pickup location must be inserted after the dropoff location of the last outbound trip, assuming there is an outbound trip in the current route.

Checking Solution Feasibility

Both heuristics ensure that the problem constraints associated with time windows, precedence and pairing, maximum ride time, shift duration for drivers and vehicle capacities are satisfied during the insertion process. An insertion of a request in a route is feasible only if it does not lead to violation of any of these constraints by inclusion of this request. Moreover, its inclusion should not create other violations of these constraints for other requests already included in the route. The implementation of these constraints during this process is important and is described next.

**Time Window Constraints.** Time window feasibility is maintained in a route if the insertion of a new request does not push the vehicle arrival time at any node \( i \) past its latest service time \( l_i. \) While a vehicle without a passenger onboard is permitted to arrive at a pickup node earlier than its earliest service time \( e_i, \) thus incurring an additional waiting cost, no vehicle is permitted to idle while carrying passengers. The procedure proposed in (2) is applied for the calculations of earliest service time, \( e_i, \) and latest service time, \( l_i. \)

To ensure that time window constraints are met, we must check that \( e_i \) and \( l_i \) fall within each request’s time window for each \( i \) in the route and for requests considered for inclusion.

**Precedence and Pairing Constraints.** For any insertion of a new request, precedence and pairing
constraints are ensured by simultaneously inserting both the pickup and dropoff locations associated with a single request within the route. The pickup location must precede the dropoff location.

Maximum Ride Time Constraints. For each considered insertion of a request, the insertion must not cause a violation in constraints (17), whether directly for the request or for other requests already inserted in the route. The maximum ride time $R_i$ is a function of direct (shortest path) ride time $DRT_i$. Herein, a piecewise linear function (30) is applied:

$$R_i = \begin{cases} 
3 \cdot DRT_i, & \text{if } DRT_i \leq 30 \\
2 \cdot DRT_i, & \text{if } 30 \leq DRT_i \leq 60 \\
DRT_i + 30, & \text{if } DRT_i > 60 
\end{cases}$$  \tag{30}

Shift Duration Limit for Drivers. Any insertion of a new request cannot extend the route duration over the shift duration limit for a driver, as expressed by constraints (16). Thus, shift duration must be assessed for each insertion of a request.

Vehicle Capacity Constraints. Any insertion of a new request must adhere to capacity constraints (13). Thus, no insertion is made if its inclusion will cause the vehicle to exceed its capacity. This must be assessed at each potential insertion location, because the number of requests handled at any point in time changes over the route duration.

NUMERICAL EXPERIMENTS

Experiment Design

To investigate the efficiency of the proposed solution approach, the solution methods are tested on a real-world problem instance. The test case involves service records for one service day in January of 2012 out of Washington Dulles International Airport (IAD). It includes 164 outbound requests involving 212 passengers and 22 inbound requests involving 41 passengers. For each request, detailed information, including desired pickup time, number of passengers, latitude and longitude of pickup and dropoff locations, and assigned vehicle index are also included. All requests were served by a fleet of identical vehicles. FIGURE 3 shows the partial distributions of the requested pickup (inbound) and dropoff (outbound) locations. The service area covers Maryland, District of Columbia, Virginia and Pennsylvania. Distances and travel times between pairs of customer locations were calculated through the OD Cost Matrix Tool in the Network Analyst toolbox of ArcGIS. The travel time is computed based on the shortest distance and speed limits.
Parameters of the model and the algorithms are presented in TABLE 1. The solution methods were implemented in Visual C++ 2010 and run on a personal computer with Intel(R) CPU 3.10GHz and 4.0GB RAM. The SP associated with the exact solution method was solved by the C++ Concert Technology of CP (Constraint Programming) solver in the IBM-ILOG CPLEX.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Explanation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_t$</td>
<td>Unit cost of time</td>
<td>0.54 $/min</td>
</tr>
<tr>
<td>$C_d$</td>
<td>Unit cost of distance</td>
<td>0.72 $/mile</td>
</tr>
<tr>
<td>$C_w$</td>
<td>Unit cost of vehicle waiting time</td>
<td>0.23 $/min</td>
</tr>
<tr>
<td>$Q$</td>
<td>Vehicle capacity</td>
<td>7 passengers</td>
</tr>
<tr>
<td>$V$</td>
<td>Maximum fleet size</td>
<td>30</td>
</tr>
<tr>
<td>$s$</td>
<td>Identical service time</td>
<td>3 minutes</td>
</tr>
<tr>
<td>$T$</td>
<td>Shift duration</td>
<td>10 hours</td>
</tr>
<tr>
<td>$\alpha$, $\beta$</td>
<td>Weight parameters</td>
<td>$\alpha=0.8$, $\beta=0.2$</td>
</tr>
<tr>
<td>$TW$</td>
<td>Pre-specified maximum deviation from desired time</td>
<td>45 minutes</td>
</tr>
</tbody>
</table>

Algorithm Performance

The CPCG approach was tested on cases with 10, 20 and 30 requests under the most general policy, Policy 2. Computational time increases exponentially with increasing number of customers. Thus, solution of problem instances with significantly more than 30 customers is precluded. The results are compared with those obtained through the adapted Jaw’s algorithm in TABLE 2. The numbers in parentheses are outbound and inbound requests, respectively. Results show that the maximum gap between the exact solution and the adapted Jaw’s algorithm is approximately 7% (with 20 requests), but
the computational time is around 1/1200 of that of CPCG.

<table>
<thead>
<tr>
<th>TABLE 2 Comparison of Results from CPCG and Adapted Jaw’s Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Requests</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>10 (7+3)</td>
</tr>
<tr>
<td>Total Cost</td>
</tr>
<tr>
<td>Vehicle Use</td>
</tr>
<tr>
<td>Computational Time (s)</td>
</tr>
</tbody>
</table>

Policy Performance

Computational results obtained by applying the two heuristics for each of the three operational policies are shown in TABLE 3. From TABLE 3, two significant conclusions can be reached. First, Jaw’s heuristic outperforms I1. For all three policies, the computation time required by Jaw’s heuristic is only between 14 and 25% of that required by I1. The longer I1 computation time can be explained by the requirement for assessing the insertion of every unrouted request when each route is constructed. For each of the three operational policies, the total cost of the routes built through Jaw’s heuristic is below that developed by I1. This may be due to the ‘seed’ selection process of I1, where the furthest unrouted request is selected for inclusion. The long distance to this request may lead to longer empty vehicle miles, and thus, increased route duration and total cost.

Second, both heuristics reveal that Policy 2 provides the best performance in terms of number of needed vehicles, idle time, empty/loaded driving time or miles traveled, and total cost, Policy 3 the second best performance, and Policy 1 the worst performance. That Policy 2 provides the best performance is unsurprising and can be shown theoretically, because it is the least constrained of the three variants. From run results of Jaw’s heuristic, Policy 2 requires the fewest vehicles, lowest idle time, lowest empty vehicle miles, and lowest total cost of the three policies. Accordingly, Policy 2 has the highest vehicle utilization rate of the three. The vehicle utilization rate of Policy 3 is significantly above that of Policy 1, but below that of Policy 2. This difference in vehicle utilization rate is caused by requirements for ordering outbound and inbound operations with Policies 1 and 3.

<table>
<thead>
<tr>
<th>TABLE 3 Performance Comparisons of Two Heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Measures</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Number of Vehicles</td>
</tr>
<tr>
<td>Total Idle Time¹</td>
</tr>
<tr>
<td>Total DH1Time²</td>
</tr>
<tr>
<td>Total DH1Mile³</td>
</tr>
<tr>
<td>Total DH2Time⁴</td>
</tr>
<tr>
<td>Total DH2Mile⁵</td>
</tr>
</tbody>
</table>
es incurred by a vehicle along its route (min); the driving distance with one or more passengers on board (mile); the empty driving distance from depot to the first pickup (mile); the empty driving distance from last dropoff to the ending depot (mile); the driving time without passengers on board (min); the driving distance without passengers on board (mile); and the driving distance with one or more passengers on board (mile).

To assess the value of this optimization-based approach, solutions obtained from the heuristics were compared against manually derived routes used to deploy the vehicle fleet on the date of the case study. In actual operations on the date of service, Policy 2 was employed. From records maintained for this date, 37 vehicles were employed. Stringing the vehicle routes together where feasible would permit completion by as few as 28 vehicles. Many of the routes did not comply with maximum ride time constraints and several violated constraints that prohibit waiting with a passenger onboard. Of course, violations were addressed during actual operations. In comparison, results from the proposed heuristic for the AARP under Policy 2 required only 17 vehicles to serve the same requests. This is an approximately 60% improvement in vehicle utilization.

### CONCLUSIONS AND FUTURE WORK

The AARP is formulated here as a mixed integer program. Three implementations corresponding to three different operational policies under consideration in practice are investigated. Exact and heuristic solution procedures are proposed. The performance of the proposed solution approaches is compared in a case study involving data from one day’s operation of an actual service provider. No exact solution could be obtained for the full-version of the case study, but an exact solution was obtained for a reduced version with 30 customer requests. In a comparison the results of the adapted Jaw’s algorithm were within 7% of the exact solution and required only 1/1200 the computation time. In the original case study, the adapted Jaw’s algorithm outperformed the second proposed heuristic. Thus, the heuristic is an effective and efficient approach for addressing the AARP, yielding significantly better results than routes and schedules determined manually.

The proposed methodologies are also relevant for other routing and scheduling applications involving ‘one-to-many-to-one’ operations, such as passenger feeder problems involving “subscription services,” school bus routing, carpooling, and express collection and delivery. Other possible applications arise in reverse logistics operations, including the delivery of full and collection of empty bottles between a manufacturer and retailers. However, the reverse logistics problem is simpler than the airport shuttle, because the goods to be transported are identical. Thus, every unit to be picked up can equally satisfy customer demand.

While the heuristics described herein provide good results with low computational effort, more sophisticated heuristics may provide improved solutions. Both described heuristics are construction heuristics. Thus, constructed routes can be improved through the application of improvement operators,

<table>
<thead>
<tr>
<th>Total EDTime</th>
<th>1696</th>
<th>516</th>
<th>2213</th>
<th>2223</th>
<th>548</th>
<th>2771</th>
<th>1537</th>
<th>1802</th>
<th>1861</th>
<th>2141</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total EDMile</td>
<td>1583</td>
<td>482</td>
<td>2065</td>
<td>2075</td>
<td>511</td>
<td>2586</td>
<td>1434</td>
<td>1682</td>
<td>1737</td>
<td>1998</td>
</tr>
<tr>
<td>Total LDTime</td>
<td>3082</td>
<td>586</td>
<td>3668</td>
<td>2798</td>
<td>590</td>
<td>3388</td>
<td>3669</td>
<td>3424</td>
<td>3690</td>
<td>3599</td>
</tr>
<tr>
<td>Total LDMile</td>
<td>2877</td>
<td>547</td>
<td>3424</td>
<td>2611</td>
<td>550</td>
<td>3162</td>
<td>3424</td>
<td>3195</td>
<td>3444</td>
<td>3359</td>
</tr>
<tr>
<td>Route Duration</td>
<td>6883</td>
<td>1414</td>
<td>8298</td>
<td>6947</td>
<td>1557</td>
<td>8505</td>
<td>7319</td>
<td>7877</td>
<td>7953</td>
<td>8615</td>
</tr>
<tr>
<td>Total Cost</td>
<td>6034</td>
<td>1372</td>
<td>7406</td>
<td>6282</td>
<td>1437</td>
<td>7719</td>
<td>6523</td>
<td>6662</td>
<td>7005</td>
<td>7325</td>
</tr>
</tbody>
</table>

1 Sum of all waiting times incurred by a vehicle along its route (min); 2 Empty driving distance from depot to the first pickup (mile); 3 Empty driving distance from depot to the first pickup (mile); 4 Empty driving time from last dropoff to the ending depot (min); 5 Empty driving distance from last dropoff to the ending depot (mile); 6 Driving time without passengers on board (min); 7 Driving distance without passengers on board (mile); 8 Driving time with one or more passengers on board (min); 9 Driving distance with one or more passengers on board (mile)
such as λ-interchange, 2-opt* exchange, trip exchange and trip reinsertion. A cluster-first route-second methodology may also address this myopic nature. Clustering can be based on both temporal and spatial characteristics of the pickup and dropoff locations. The authors are currently investigating these and other improvements.

In a dynamic setting, new requests may be received on short notice while some vehicles are en route. The operator must quickly insert these new requests within previously constructed routes and schedules. In the airport operations of the case study, most inbound requests are known in advance, but almost all outbound trips arise dynamically. A fast algorithm to find a good feasible insertion for the new requests is required. The authors are working to extend the developed model and solution methodologies to operations with uncertainties in travel and service times, as well as dynamic requests.

ACKNOWLEDGEMENTS
This work was funded by the Maryland Industrial Partnership (MIPS) and IT Curves. This support is gratefully acknowledged, but implies no endorsements of the findings.

REFERENCES


