Fast Infeasibility Detection Algorithms for Dial-A-Ride Transit Problems

by
Ravindra K. Ahuja
Robert Dial

SWP# 4005 February 1998

Fast Infeasibility Detection Algorithms for Dial-A-Ride Transit Problems

Ravindra K. Ahuja[*]
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Robert Dial
The Volpe National Transportation Systems Center
Kendall Square,
Cambridge, MA 02142, USA

James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

(Revised February 14, 1998)

[*] On leave from Indian Institute of Technology, Kanpur 208 016, INDIA.

Fast Infeasibility Detection Algorithms for Dial-A-Ride Transit Problems

Ravindra K. Ahuja[1], Robert Dial[2], and James B. Orlin[3]

ABSTRACT

Dial-a-ride transit (DART) is a shared taxicab system that typically serves areas of low travel demand and/or populations with special needs. Dial-a-ride transit problems are very difficult combinatorial problems and can be solved exactly only when the problem size is very small. Practical problems of larger size are mostly solved heuristically either by construction-based algorithms or improvement-based algorithms. Several of these heuristic algorithms proceed by starting with a vehicle route and performing insertions either to improve the route or to add a customer to the route. Faster methods for determining the feasibility or infeasibility of such insertions can have a dramatic effect on the efficiency of such algorithms and we focus on such methods in this paper. The schedule of a vehicle typically consists of picking up and dropping off of some customers in a specific sequence and at any point in time several customers can be on-board the vehicle. Existing literature on DART makes a simplifying assumption that a vehicle is not allowed to wait while customers are on board the vehicle. In this paper, we relax this assumption. We show that given a vehicle route, determining its schedule (that is, when the vehicle should pick-up and drop-off customers so as to honor customers pickup and delivery time windows, transit times restrictions between stops, maximum waiting time constraints, and ride time constraints) can be formulated as a shortest path problem with possibly negative arc lengths. We next develop several tests which allow us to detect the infeasibility of many possible insertions in a specified vehicle route in O(1) time after suitable preprocessing has been done.

---

[1] Sloan School of Management, MIT, Cambridge, MA 02139, USA; On leave from Indian Institute of Technology, Kanpur 208 016, INDIA.

[2] The Volpe National Transportation Systems Center, Kendall Square, Cambridge, MA 02142, USA.

[3] Sloan School of Management, MIT, Cambridge, MA 02139, USA.

1. INTRODUCTION

Dial-A-Ride Transit (DART) is a form of mass transit system that typically serves areas of low travel demand and/or a population with special needs. It is a shared taxicab system and is run with a fleet of vehicles operating on flexible routes without fixed schedules. It can also serve as an alternate means of transportation for handicapped people and senior citizens. Customers call the dial-a-ride agency requesting to be carried from some specified point to another specified point. The agency in turn is responsible for dispatching a fleet of vehicles to meet such demands during a typical day of operation. The aim of the dial-a-ride system to provide a high quality of service at a reasonable cost.

Dial-a-ride transit problems are extensively studied in the literature, and we refer the reader to the papers by Jaw et al. [1986], Kontoravdis and Bard [1994], Desrosiers et al. [1995] for further pointers to the literature devoted to this topic. Researchers have developed exact as well as heuristic algorithms for dial-a-ride transit problems. Since exact algorithms can solve only small sized problems, heuristic algorithms have been more extensively studied. A heuristic algorithm typically performs two functions: routing and scheduling. The routing part determines the route of each vehicle – it assigns customers to a vehicle and the order in which the vehicle will visit the customers assigned to it. The scheduling part assigns a time schedule to the route - the times at which customers will be picked up and delivered. In this paper, we will consider the scheduling part and focus on determining the feasibility of a specified route; that is, given a route can we determine customer pickup and drop-off times so that the route satisfies various user-specified constraints. To be more specific, the scheduling problem we study in this paper, can be stated as follows. Given a route of a vehicle through n stops 1-2- … - n (that is, the order in which some customers will be picked up and delivered), does there exist a feasible schedule that honors customers pickup and delivery time windows, transit times restrictions between stops, maximum waiting time constraints, and ride time constraints?

Most of the existing models studied in the literature on DART do not allow the vehicle to wait while customers are on board the vehicle. While this assumption is reasonable in many situations, it is unnecessarily restrictive. This restriction might eliminate many good schedules. In this paper, we relax this requirement but it is at the expense of making the problem more time-consuming to solve. We note that in practice one must permit schedules with passengers waiting for the following reason: there may be last-minute cancellations due to which a vehicle may need to be rescheduled. If the vehicle skips the cancelled pickup and there are customers on board the vehicle, then waiting times are introduced. Not allowing a vehicle to wait with customers on board the vehicle may result in the route being infeasible. It also leads to interesting problem structures that one can exploit.

Researchers have developed relatively straightforward algorithms for scheduling vehicles when passenger waiting times are not permitted (see, for example, Jaw el al. [1986]). In scheduling problems where vehicle is not allowed to wait with customers on board the vehicle is relatively simple. In this case, the route can be decomposed into subroutes where each subroute corresponds to the route of the vehicle during which at least one customer is on board, and there are no customers on board the vehicle in between any two subroutes. For such a subroute, the pickup time of the first customers determines the pickup/delivery time of each stop in the route, since the vehicle moves directly from one stop to another without waiting. In this case, the scheduling problem consists of determining the pickup times of the first customer in each subroute in the specified route.

In this paper, we first describe the mathematical model for the scheduling problem and show that finding a feasible schedule for a given can be transformed into a shortest path problem with possibly negative arc lengths. This model can be solved in $O(n^2)$ time using standard and well known shortest path label-correcting algorithms. We next focus on the problem of determining the feasibility of inserting a new customer trip request into a vehicle's route. A new customer trip requests adds two new stops in the vehicle route - one corresponding to the pickup stop and one corresponding to the delivery stop. Given an existing route consisting of n stops, a customer trip request can be inserted in $O(n^2)$ ways: the customer can be picked up right after stop i and delivered right after stop j, where i and j vary from 1 to n. Among these $O(n^2)$ potential insertions, only a few will be feasible, that is, satisfy all the time windows, maximum waiting time, maximum riding time, and vehicle capacity constraints. Using our shortest path approach, we can determine the feasibility of each potential insertion in $O(n^2)$ time, but using this approach for each possible insertion will require a total of $O(n^4)$ time, which is unnecessarily time-consuming. We prefer fast algorithms that can identify feasible insertions quickly among all potential insertions. In this paper, we develop three tests for determining the infeasibility of an insertion. Each of these tests runs in O(1) time. Though we believe that most infeasible insertions will "flunk" at least one of these tests, our tests do not guarantee that every infeasible insertion will flunk at least one of these tests. Further, each feasible insertion is guaranteed to pass all the three tests.

2. Mathematical Formulation of the Feasibility Problem

In this section, we will discuss the constraint of the dial-a-ride transit problem and formulate the feasibility problem as a shortest path problem. We use some graph notation in this and the rest of the sections, such as paths, cycles, and walks. We refer the reader to the book of Ahuja, Magnanti and Orlin [1993] for these notation.

A vehicle must visit a specified sequence of n pickup and delivery stops 1-2-3- ... -n. Let $\mathcal{P}$ denote the set of pickup stops. For a pickup stop $i \in \mathcal{P}$, b[i] customers are picked up to be taken to the delivery stop m(i). We denote by δ_i, for each i =1, 2, ... , n, the time when the vehicle visits stop i. The δ_i's are the decision variables in our formulation. Clearly, $\delta_1 \le \delta_2 \le \delta_3 \le \dots \le \delta_n$.

1. **Time window constraints**. Each pickup (or delivery) stop has a pickup (or delivery) time window (l_i, u_i), and the customer can be picked up anytime during its time window. In other words, $l_i \le \delta_i \le u_i$, for each i = 1, 2, ... , n.

2. **Minimum transit time constraints**. Let t_i denote the minimum transit time to go from stop i to stop i+1. Minimum transit time constraints require that $\delta_i + t_i \le \delta_{i+1}$ for each i =1, 2, ..., (n-1). Notice that since $t_i \ge 0$, the constraints $\delta_i + t_i \le \delta_{i+1}$ for each i =1, 2, ..., (n-1) subsume the constraints $\delta_1 \le \delta_2 \le \delta_3 \le \dots \le \delta_n$. Further, notice that we allow the vehicle to wait while going from one stop to the next. If we denote by w_i the *waiting time* while going from stop i to stop i + 1, then $w_i = \delta_{i+1} - \delta_i - t_i$, for each i =1, 2, ..., n.

3. **Maximum waiting time constraints**. We require that the waiting times in between the stops are nonnegative and bounded by $\overline{w}$, that is, $0 \le w_i \le \overline{w}$, for each i = 1, 2, ... , n-1.

4. **Maximum ride time constraints**. We require that the actual ride time of a customer does not exceed its maximum permissible ride time. Let τ_i denote the minimum direct time from the pickup stop i to the delivery stop m(i) (that is, the minimum

travel time needed to go directly from stop i to stop m(i)). We denote the excess ride time to go from the stop i to the stop m(i) by r_i, where $r_i = \delta_{m(i)} - \delta_i - \tau_i$, for each $i \in \mathcal{P}$, where $\mathcal{P}$ is the set of pickup stops. We require that r_i is no more than $\bar{r}_i$, which we assume to be a constant factor of τ_i, say $.8\tau_i$. In other words, r_i must satisfy the following ride time constraints: $r_i \le \bar{r}_i$, for each $i \in \mathcal{P}$.

5. **Vehicle capacity constraints**. We also assume that the vehicle has a limited capacity to carry passengers that we are not allowed to exceed. The number of customers on board the vehicle can be determined directly from the order in which customers are picked up and delivered, and do not depend upon the vehicle arrival and pickup times. For a specified route of the vehicle, this constraint can be easily checked in O(n) time by walking through the route and determining the maximum occupancy of the vehicle. If the maximum occupancy is less than or equal to the vehicle capacity, then the route is feasible with respect to the vehicle capacity constraints. We will henceforth assume that the given route satisfies the vehicle capacity constraints. However, we will return to this constraint in Section 3 when we consider the feasibility of inserting new stops.

We summarize the preceding discussion by putting together all the constraints.

$$l_i \le \delta_i \le u_i, \qquad \text{for all } i = 1, 2, \dots, n, \tag{1a}$$

$$\delta_{i+1} - \delta_i = t_i + w_i, \qquad \text{for all } i = 1, 2, \dots, n-1, \tag{1b}$$

$$0 \le w_i \le \bar{w}, \qquad \text{for all } i = 1, 2, \dots, n-1, \tag{1c}$$

$$\delta_{m(i)} - \delta_i = \tau_i + r_i, \qquad \text{for all } i \in \mathcal{P}, \tag{1d}$$

$$r_i \le \bar{r}_i, \qquad \text{for all } i \in \mathcal{P}. \tag{1e}$$

We reformulate the constraints in (1) by (i) substituting inequalities in (1c) in the equations (1b), substituting inequalities in (1e) in the equations (1d), and introduce a dummy variable δ_0 in the inequalities in (1a) whose value is set to zero. These changes yield the following equivalent set of constraints:

$$\delta_0 - \delta_i \le -l_i, \qquad \text{for all } i = 1, 2, \dots, n, \tag{2a}$$

$$\delta_i - \delta_0 \le u_i, \qquad \text{for all } i = 1, 2, \dots, n, \tag{2b}$$

$$\delta_i - \delta_{i+1} \le -t_i, \qquad \text{for all } i = 1, 2, \dots, n-1, \tag{2c}$$

$$\delta_{i+1} - \delta_i \le (t_i + \bar{w}), \qquad \text{for all } i = 1, 2, \dots, n-1, \tag{2d}$$

$$\delta_{m(i)} - \delta_i \le (\tau_i + \bar{r}_i), \qquad \text{for all } i \in \mathcal{P}, \tag{2e}$$

$$\delta_0 = 0. \tag{2f}$$

The constraints in (2) are a special case of *the system of difference constraints* (see, for example, Ahuja, Magnanti, and Orlin [1993], Section 4.2). The system of difference constraints is a collection of inequalities of the form $w_j - w_i \le f_{ij}$, where the RHS is a constant and the LHS is the difference of two variables. To solve a system of difference constraints, such as the one given in (2), we define a *constraint graph* G in the following manner. The constraint graph has a node corresponding to each of the variables (there are n+1 such variables δ_0, δ_1, δ_2, ... , δ_n), and an arc corresponding to each difference constraint in (2) (there are about 4.5n such constraints). For each difference constraint $\delta_j - \delta_i \le c_{ij}$, we associate an arc (i, j) of length c_{ij} in G. We illustrate this formulation using a numerical example shown in Figure 1(a) consisting of six stops. We assume that stops 1, 2, and 4 are pickup stops with m(1) = 5, m(2) = 3, and m(4) = 6. For the route given in Figure 1(a), we show in Figure 1(b) the corresponding constraint graph.

It is well known that a system of difference constraints, such as the one given in (2), is feasible if and only if the corresponding constraint network does not contain any negative cost cycle. We can determine the presence of a negative cycle in a network by using a label-correcting shortest path algorithm, for example, the FIFO label-correcting algorithm given in Ahuja, Magnanti and Orlin [1993]. The label-correcting algorithms typically require that all the nodes are reachable by a directed path from the source node for the shortest path problem. Notice that if we use node 0 as the source node, then it satisfies the reachability requirement and also ensures that $\delta_0 = 0$, which we require for the validity of our formulation. A label-correcting algorithm either indicates the presence of a negative cycle or provides the shortest path distances. In the former case, the system of difference constraints has no solution, and in the latter case the shortest path distances δ_i's constitute a feasible solution of (2). The running time of the FIFO label-correcting algorithm is O(nm), but since for the constraint graph satisfies m = O(n), the algorithm runs in $O(n^2)$ time. In practice, however, label-correcting algorithms run much faster than indicated by their worst-case running times. So we believe that the feasibility of the constraints (2) in practice can be determined very efficiently, perhaps linear in terms of the number of nodes in the constraint graph.

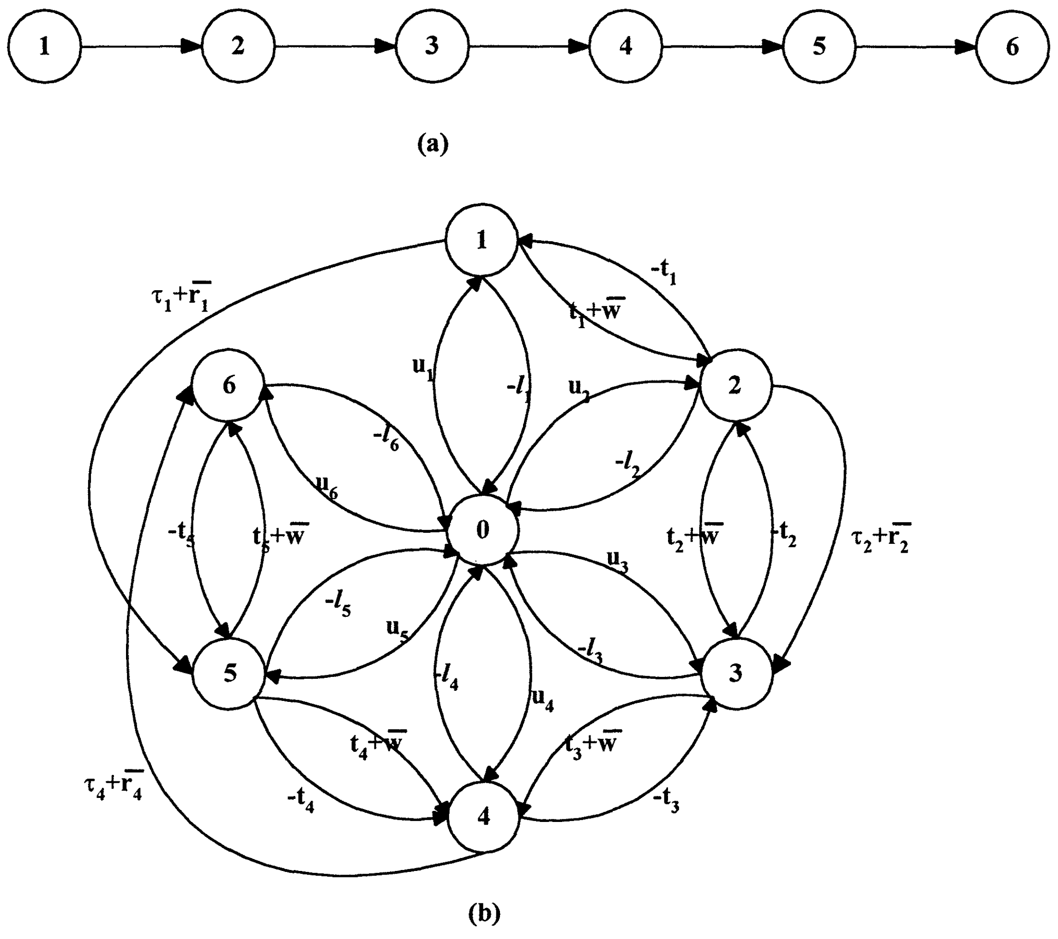

Figure 1. Illustrating the construction of the shortest path network.

If the constraints in (2) are found to be feasible, then a label-correcting algorithm yields the shortest path distances δ^*. These shortest path distances satisfy the property that for each stop i, δ_i^* is the earliest time when the vehicle can visit stop i among all feasible schedules.

3. Determining the Feasibility of an Insertion

In this section, we shall study the problem of determining the infeasibility of inserting a new trip segment into a vehicle's route. Given the route of a vehicle 1-2-3-…- n, we want to know whether a group of h customers can be picked up between stops p and p+1, and delivered between stops q and q+1. Each of these tests runs in O(1) time, and we believe that most infeasible insertions will "flunk" at least one of these tests. Moreover, each feasible insertion is guaranteed to pass all the three tests. All of these tests are sufficient but not necessary. Consequently, if an insertion passes each of these tests, then it does not necessarily mean that the insertion is feasible. We will need to use the shortest path formulation described in Section 2 to determine correctly whether it is feasible or not. However, we believe that the shortest path approach will not be used very often because collectively the three infeasibility tests will identify most of the infeasible insertions.

Our three infeasibility tests focus on different sets of constraints. Our first infeasibility test described in Section 3.1 checks whether the insertion will satisfy the vehicle capacity constraints. Our second infeasibility test described in Section 3.2 checks whether the insertion will satisfy the time window and minimum transit time constraints of all the stops. Our third infeasibility test described in Section 3.3 checks all the constraints except the vehicle capacity constraints and the maximum waiting time constraints.

3.1 Infeasibility Test-I

This infeasibility test determines in O(1) time whether the pickup of h customers between the stops p and p+1 and delivery between the stops q and q+1 will satisfy the vehicle capacity constraints. However, before this infeasibility test can be applied, we need to do some preprocessing that takes $O(n^2)$ time. (Notice that if we average out this preprocessing time over $O(n^2)$ total possible insertions, it also becomes O(1) per insertion.) Let *occupancy[i, j]* for j > i denote the maximum number of persons on board the vehicle as it goes from stop i to stop j prior to dropping off or picking up customers at stop j. We can determine the occupancy values for all valid pairs of stops in $O(n^2)$ time using the algorithm given below.

```
algorithm maximum-occupancy;
begin
    for i : = 1 to n do person[i] : = 0;
    for each i ∈ $\mathcal{P}$ do
        for j := i to (m(i) – 1) do person[j] : = person[j] + b[i];
    for i : = 1 to (n – 1) do
    begin
        occupancy[i, i+1] : = person[i];
        for j : = (i + 2) to n do
            occupancy[i, j] : = max{occupancy[i, j-1], person[j-1]};
    end;
end;
```

The above algorithm first computes person[i], which denotes the number of persons on board the vehicle when it leaves stop i. It then uses the person vector to determine the occupancy values for all valid pairs of stops. Clearly, the algorithm runs in $O(n^2)$ time. Suppose that the occupancy values have been computed and we want to determine whether the vehicle can accommodate h additional customers who need to be picked up between the stops p and p+1 and delivered between the stops q and q+1. To do so, we compare the value (h + occupancy[p, q]) with the vehicle capacity. If (h + occupancy[p, q]) is greater than the vehicle capacity, then the insertion violates the vehicle capacity constraints; otherwise it doesn't.

3.2 Infeasibility Test-II

In this section, we describe an algorithm that identifies insertions violating the time window and the minimum transit time constraints. This algorithm, however, does not identify insertions that violate the maximum waiting time constraints (that is, $w_i \leq \overline{w}$) or the maximum ride time constraints (that is, $r_i \leq \bar{r}_i$). The algorithm requires O(n) time to do some preprocessing of the data for the scheduling problem. After the preprocessing has been done, it can determine infeasible insertions in O(1) time per insertion.

Let δ denote a feasible schedule of a vehicle. Then δ must satisfy the following constraints:

$$l_i \le \delta_i, \qquad \text{for all } i = 1, 2, \dots, n, \tag{3a}$$

$$\delta_i \le u_i, \qquad \text{for all } i = 1, 2, \dots, n, \tag{3b}$$

$$\delta_i - \delta_{i-1} \ge t_{i-1}, \qquad \text{for all } i = 2, 3, \dots, n. \tag{3c}$$

For the given route of the vehicle, the vehicle can visit the stops according to many schedules satisfying (3). We first determine the earliest time the vehicle can visit various stops and still satisfy the constraint in (3). Let α_i, $1 \le i \le n$, denote the earliest time the vehicle can visit stop i in any schedule satisfying (3). We will use an inductive argument to calculate α_i's for increasing values of i. Clearly, $\alpha_1 = l_1$. Now suppose that we know α_{i-1} for some $i > 1$. The condition (3a) implies that $\alpha_i \ge l_i$, and the condition (3c) implies that $\alpha_i \ge \alpha_{i-1} + t_{i-1}$. Hence, $\alpha_i = \max\{l_i, \alpha_{i-1} + t_{i-1}\}$. If $\alpha_i > u_i$, then the vehicle cannot honor the constraint (3b) and there exists no feasible schedule satisfying (3). The preceding discussion suggests the following O(n) method to determine α_i's.

```
algorithm compute-α;
begin
    $\alpha_1 := l_1$;
    for i := 2 to n do
    begin
        $\alpha_i := \max\{l_i, \alpha_{i-1} + t_{i-1}\}$;
        if $\alpha_i > u_i$ then stop as there is no feasible schedule;
    end;
end;
```

We next determine the latest time the vehicle can visit various stops and still satisfy the constraint in (3). Let β_i, $1 \le i \le n$, denote the latest time the vehicle can visit stop i in any schedule satisfying (3). We will use an inductive argument to calculate β_i's for decreasing values of i. Clearly, $\beta_n = l_n$. Now suppose that we know β_{i+1} for some $i \le n-1$. The condition (3b) implies that $\beta_i \le u_i$, and the condition (3c) implies that $\beta_i \le \beta_{i+1} - t_i$. Hence, $\beta_i = \min\{u_i, \beta_{i+1} - t_i\}$. If $\beta_i < l_i$, then the vehicle cannot honor the constraint (3a) and there exists no feasible schedule satisfying (3). The preceding discussion suggests the following O(n) method to determine β_i's.

```
algorithm compute-β;
begin
    $\beta_n$ : = $u_n$;
    for i : = (n-1) downto n do
    begin
        $\beta_i$ : = min{$u_i$, $\beta_{i+1}$ - $t_i$};
        if $\beta_i < l_i$ then stop as there is no feasible schedule;
    end;
end;
```

The above methods determine in O(n) time whether there exists a schedule satisfying (3). We shall henceforth assume there exists a vehicle route satisfying (3) and thus the α_i's and β_i's are well defined. We now address the question whether a new trip request can be feasibly inserted into the vehicle's route. Adding a new customer trip request into a vehicle's route amounts to adding two stops: one corresponding to the pickup of the customer and another corresponding to the delivery of the customer. Suppose that the new trip request adds stop A between stop p and stop p+1, and adds stop B between stop q and stop q+1 (see Figure 2). Let t_{A1} and t_{A2}, respectively, denote the time needed to go from stop p to stop A and stop A to stop p+1. Similarly, let t_{B1} and t_{B2}, respectively, denote the time needed to go from stop q to stop B and stop B to stop q+1.

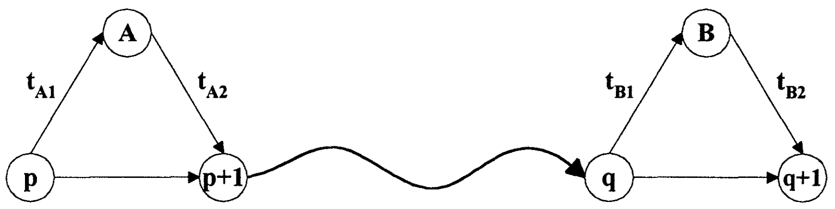

Figure 2. Illustrating insertion of a customer trip request in a route.

Let φ denote the earliest schedule after the insertion of stop A and prior to the insertion of stop B. We will show how to obtain φ from α. It is easy to see that $\varphi_i = \alpha_i$ for all i = 1, 2, ... , p, $\varphi_A = \max\{l_A, \alpha_p + t_{A1}\}$, and $\varphi_{p+1} = \max\{l_{p+1}, \varphi_A + t_{A2}\}$. We claim that φ_i for all i = p+2, p+3, ... , n, is given by the following formula:

$$\varphi_i = \max\{\alpha_i, \varphi_{p+1} + t_{p+1} + t_{p+2} + \dots + t_{i-1}\}. \qquad (4)$$

The correctness of (4) can be easily established by performing induction on the value of i. The equation (4) allows us to compute φ_i for any $i \geq p+2$ in O(1) time if suitable preprocessing has been done. Let $R[i] = t_1 + t_2 + \ldots + t_{i-1}$ denote the total ride time from stop 1 to stop i. We can compute R[i] for all i in O(n) time. In terms of R[i], (4) can be stated as

$$\varphi_i = \max\{\alpha_i, \varphi_{p+1} + R[i] - R[p+2], \tag{5}$$

which permits us to compute any φ_i in O(1) time. The φ_i's computed using (4) or (5) satisfy the constraints (3a) and (3c) because these constraints have been used in deriving (4). But φ_i may violate the constraints in (3b) which may make the route of the vehicle infeasible after the insertion of stop A. If $\varphi_A > u_A$, then clearly the route is infeasible. For other stops, the following lemma gives us a simple condition to determine the feasibility of the schedule φ.

Lemma 1. *The schedule φ is a feasible schedule if and only if $\varphi_{p+1} \leq \beta_{p+1}$.*

Proof. Clearly, if $\varphi_{p+1} > \beta_{p+1}$, then stop p+1 cannot be visited within its specified time window and φ is not a feasible schedule. Note that β_{p+1} is unaffected by the stop A since the algorithm for computing β_{p+1} computes β_i's in the decreasing order of the index i. We now prove the converse result that if φ is not a feasible schedule then $\varphi_{p+1} > \beta_{p+1}$. We prove this result by contradiction. Suppose that φ is not a feasible schedule. Then, $\varphi_r > u_r$ for some $r > p+1$. Since $\alpha_r \leq u_r$, it follows from (4) that

$$\varphi_r = \varphi_{p+1} + t_{p+1} + t_{p+2} + \ldots + t_{r-1}, \tag{6}$$

Further, it follows from the definition of β_i's that

$$\beta_{p+1} \leq u_r - t_{r-1} - t_{r-2} - \ldots - t_{p+1}. \tag{7}$$

The conditions in (6) and (7) imply that

$$\varphi_r - u_r \leq \varphi_{p+1} - \beta_{p+1}. \tag{8}$$

Hence if $\varphi_r > u_r$, then $\varphi_{p+1} > \beta_{p+1}$, which is what we wanted to establish. The lemma now follows. ■

It follows from Lemma 1 that to determine whether stop A can be feasibly inserted or not, we simply check whether $\varphi_A \leq u_A$ and $\varphi_{p+1} \leq \beta_{p+1}$. If both the constraints are satisfied, then there exists a feasible schedule with stop A being inserted; otherwise there is no feasible schedule. We next consider the insertion of stop B. First we need to determine the earliest time to visit stop q after stop A has been inserted. It follows from (4) that $\varphi_q = \max\{\alpha_q, \varphi_{p+1} + R[q] - R[p+2]\}$. Next observe that $\varphi_B = \max\{l_B, \varphi_q + t_{B1}\}$ and $\varphi_{q+1} = \max\{l_{q+1}, \varphi_B + t_{B2}\}$. Using similar arguments as we used in the case of stop A, it can be shown that the insertion of stop B is feasible if and only if $\varphi_B \leq u_B$ and $\varphi_{q+1} \leq \beta_{q+1}$. We summarize our discussion in this section by the algorithm given below. The running time of the algorithm is O(1) per insertion.

```
**algorithm** *infeasibility-test-II(A, B)*;
**begin**
    $\varphi_A := \max\{l_A, \alpha_p + t_{A1}\}$;
    $\varphi_{p+1} := \max\{l_{p+1}, \varphi_A + t_{A2}\}$;
    **if** $\varphi_A > u_A$ **then** the insertion is infeasible;
    **if** $\varphi_{p+1} > \beta_{p+1}$ **then** the insertion is infeasible;
    $\varphi_q := \max\{\alpha_q, \varphi_{p+1} + R[q] - R[p+2]\}$;
    $\varphi_B := \max\{l_B, \varphi_q + t_{B1}\}$;
    $\varphi_{q+1} := \max\{l_{q+1}, \varphi_B + t_{B2}\}$;
    **if** $\varphi_B > u_B$ **then** the insertion is infeasible;
    **if** $\varphi_{q+1} > \beta_{q+1}$ **then** the insertion is infeasible;
**end**;
```

3.3 Infeasibility Test-III

We shall now describe our third infeasibility test. This test is able to identify insertions violating the time window constraints, the minimum transit time constraints, and the maximum ride time constraints for all stops except for the newly added stops: pickup stop A between the stops p and p+1 and the delivery stop B between the stops q an q+1. This algorithm does not identify insertions that violate the maximum waiting time constraints (that is, $w_i \leq \overline{w}$). As in the previous section, we assume that the vehicle takes t_{A1} time to go from stop p to stop A and t_{A2} time to go from stop A to stop p+1, and it takes t_{B1} time to go from stop q to stop B and t_{B2} time to go from stop B to stop q+1. Our algorithm requires $O(n^2 \log n)$ in preprocessing. Subsequently, it tests the infeasibility of an insertion in O(1) time. This algorithm spends more time in preprocessing than the algorithms presented in Section 3.2, but is less restrictive in the types of infeasibilities it will discover.

We have shown in Section 2 that the constraints (2) associated with the sequence of stops 1-2-3- ... -n is associated with a shortest path problem on the constraint graph G = (N, A). In this feasibility test, we do not include the arcs corresponding to the maximum waiting time constraints, that is, the arcs (i, i+1) for all i =1, 2, ... , n-1, of length $t_i + \overline{w}$, and we will ignore the maximum waiting time constraints in this infeasibility test. Let G' denote this network. Recall that this network has n nodes and O(n) arcs. Let $P^*[i, j]$ denote the shortest path from node i to node j in G' and $d^*[i, j]$ denote its length. We can determine $d^*[i, j]$ for all node pairs [i, j] by solving an all-pairs shortest path problem. The all-pairs shortest path problem can be solved in $O(nm + n^2 \log n)$ time by applying the label-correcting algorithm once and then using Dijkstra's algorithm (n-1) times (see, for example, Ahuja, Magnanti, and Orlin [1993], Section 5.6). Since the constraint graph has m = O(n), the running time of this algorithm becomes $O(n^2 \log n)$.

If we insert the pickup of a new trip segment between the stops p and p+1 and the delivery between the stops q and q+1, then the new stop sequence becomes 1 – 2 – 3 - ... - p – A - p+1 - ... – q – B – q+1 - ... n. To determine the impact of this change on the feasibility of the solution, we need to determine the changes in the corresponding constraint graph and ensure that no negative cycles are created due to this change. We test for feasibility under the assumption that the vehicle does not introduce any new waiting time before or after visiting the stops. If this schedule is infeasible, it will also be infeasible when additional waiting time is introduced. Adding the stop A increases the ride time from stop p to stop p+1; earlier it was t_p and after the insertion it becomes t'_p, which equals the time needed to go from stop p to stop A plus the time needed to go from stop A to stop p+1. Clearly, $t'_p \geq t_p$. Similarly, adding the stop B increases the direct ride time from stop q to stop q+1 to t'_q where $t'_q \geq t_q$. In terms of the graph G', it implies that the costs of the arcs (p+1, p) and (q+1, q) decrease. Thus, we need to check whether decreasing the costs of arcs (p+1, p) and (q+1, q) creates any negative cycles or not.

We will now consider the case that the costs of two arcs in the constraint graph, say, of arcs (u, v) and (k, *l*), decrease, and we need to determine whether it creates any negative cycles. We denote by c'_{uv} and c'_{kl} the modified arc costs, and denote by G'' the modified constraint graph. For every pair of nodes i and j, let $d'[p, q]$ denote the length of the shortest path from node i to node j. Observe that $d'[i, j] \leq d^*[i, j]$ for every node pair i and j, since decreasing costs of arcs (u, v) and (k, l) does not increase the cost of any path but might actually decrease it. Also observe that if the shortest path from node i to node j does not contain the arc (u, v) or (k, l), then $d'[i, j] = d^*[i, j]$. The following lemma states the necessary and sufficient conditions for the presence of a negative cycle in G''.

Lemma 2. *The network G'' contains a negative cost cycle if any one of the following three conditions is satisfied: (i) $d^*[v, u] + c'_{uv} < 0$; (ii) $d^*[l, k] + c'_{kl} < 0$; and (iii) $c'_{uv} + d^*[v, k] + c'_{kl} + d^*[l, u] < 0$. If none of these three conditions is satisfied, then the network G'' does not contain any negative cost cycle.*

Proof. We first show that if any of the three conditions (i), (ii) and (iii) are satisfied, then G'' contains a negative cost cycle. If $d^*[v, u] + c'_{uv} < 0$, then the directed cycle $W = P^*[v, u] \cup \{(u, v)\}$ has cost $d'[v, u] + c'_{uv} \leq d^*[v, u] + c'_{uv} < 0$, implying that W is a negative cycle. Similarly, it can be shown that if $d^*[l, k] + c'_{kl} < 0$ then $W = P^*[l, k] \cup \{(k, l)\}$ is a negative cycle. If $c'_{uv} + d^*[v, k] + c'_{kl} + d^*[l, u] < 0$ then $W = \{(u, v)\} \cup P^*[v, k] \cup \{(k, l)\} \cup P^*[l, u]$ is either a negative cycle or a negative cost directed walk. In the latter case, W may be expressed as the sum of the arc-disjoint directed cycles, at least one of which is negative.

We next prove the converse result that if the network G'' contains a negative cycle W, then at least one of the conditions in (i), (ii), and (iii) must be satisfied. The negative cycle W must contain the arc (u, v) or (k, l) or both, because the network did not contain any negative cycle before decreasing the costs of the arcs (u, v) and (k, l). If W contains the arc (u, v) but not arc (k, l), then it is of the form $\{(u, v)\} \cup P[v, u]$, where P[v, u] is a directed path from node v to node u and does not contain any of the arcs (u, v) and (k, l). Hence the length of the path P[v, u] is at least $d'[v, u] = d^*[v, u]$. Consequently, the length of the cycle W is at least $d^*[v, u] + c'_{uv}$. Since W is a negative cycle, we get $d^*[v, u] + c'_{uv} < 0$, establishing that condition (i) is satisfied. Using similar arguments, it can be shown that if W contains the arc (k, l) but not (u, v), then condition (ii) is satisfied. We next prove that if W contains both the arcs (u, v) and (k, l), then condition (iii) is satisfied. In this case, $W = \{(u, v)\} \cup P[v, k] \cup \{(k, l)\} \cup P[l, u]$, where $P[v, k]$ is a directed path from node v to node k and $P[l, u]$ is a directed path from node l to node u. Note that both the paths $P[v, k]$ and $P[l, u]$ do not contain any of the arcs (k, l) and (u, v). Consequently, $d'[v, k] = d^*[v, k]$ and $d'[l, u] = d^*[l, u]$. Hence, the length of the cycle W is at least $c'_{uv} + d^*[v, k] + c'_{kl} + d^*[l, u]$. Since W is a negative cycle, it follows that $c'_{uv} + d^*[v, k] + c'_{kl} + d^*[l, u] < 0$, thereby satisfying condition (iii). This completes the proof of the lemma. ■

It follows from Lemma 2 that to check the presence of negative cycles in G'', we simply check three conditions: (i) $d^*[v, u] + c'_{uv} < 0$; (ii) $d^*[l, k] + c'_{kl} < 0$; and (iii) $c'_{uv} + d^*[v, k] + c'_{kl} + d^*[l, u] < 0$. If any of these three conditions is true, then G'' contains a negative cycle implying that the new insertion is not feasible. If these conditions are not true, then the insertion will satisfy the feasibility constraints with respect to all the stops 1-2-3- ... -n. However, the insertion may lead to infeasibilities for the constraints corresponding to the new stops A and B.

We point out that if we include the maximum waiting time constraints too in our formulation, then adding the two new stops will decrease the costs of two arcs (p+1, p) and (q+1, q), and will increase the cost of two arcs (p, p+1) and (q, q+1). It appears difficult to determine the presence of a negative cycle in O(1) time with respect to these changes. We thus did not consider the maximum waiting time constraints in our infeasibility test.

4. Summary

In this paper, we considered the following problem which is solved repeatedly by heuristic algorithms for dial-a-ride transit problems: Given the route of a vehicle through n stops 1-2- ... - n, does there exist a feasible schedule that honors customers pickup and delivery time windows, transit times restrictions between stops, maximum waiting time constraints, and ride time constraints? Most of the existing models studied in the literature on dial-a-ride transit problems do not allow the vehicle to wait while customers are on board the vehicle. Our model relaxes this assumption. We show that the problem of determining a feasible route through the n stops while honoring all constraints can be formulated as a shortest path problem and solved in $O(n^2)$ time. We next consider the problem of inserting a customer pickup trip request in the vehicle's route. This problem can be solved in $O(n^2)$ time using our shortest path approach by determining the feasibility of the new set of stops; however, faster approaches can be developed that could detect the infeasibilities of possible insertions in O(1) time. In this paper, we developed three such infeasibility tests which consider different sets of constraints. The table shown in Figure 3 summarizes the characteristics of our shortest path model and the three infeasibility tests we developed in this paper.

Infeasibility Test	**Constraints Considered**	**Preprocessing Time**	**Time per Insertion**
Feasibility Test (Section 2)	(i) Time window constraints (ii) Minimum transit time constraints (iii) Maximum waiting time constraints (iv) Maximum ride time constraints	None	$O(n^2)$
Infeasibility Test-I (Section 3.1)	(i) Vehicle capacity constraints	$O(n^2)$	O(1)
Infeasibility Test-II (Section 3.2)	(i) Time window constraints (ii) Minimum transit time constraints	O(n)	O(1)
Infeasibility Test-III (Section 3.3)	(i) Time window constraints (ii) Minimum transit time constraints (iii) Maximum ride time constraints for all stops except the newly added stops	$O(n^2 \log n)$	O(1)

Figure 3. Summary of the various infeasibility tests developed in this paper.

ACKNOWLEDGEMENTS

We gratefully acknowledge support from the Office of Naval Research under contract ONR N00014-96-1-0051 as well as a grant from the United Parcel Service.

REFERENCES

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

Desrosiers, J., Y. Dumas, M. M. Solomon, and F. Soumis. 1995. Time constrained routing and scheduling. *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, North Holland, Amsterdam.

Jaw, J. J., A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. 1986. A Heuristic Algorithm for the Multi-Vehicle Advance request Dial-A-Ride Problem with Time Windows. *Transportation Research B*, Volume 20B, 243-257.

Kontoravdis, K, and J. F. Bard. 1994. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing* 7, 10-23.