



INTRACTABILITY OF THE DIAL-A-RIDE PROBLEM AND A MULTIOBJECTIVE SOLUTION USING SIMULATED ANNEALING

JOHN W. BAUGH JR. , GOPALA KRISHNA REDDY KAKIVAYA & JOHN R. STONE

To cite this article: JOHN W. BAUGH JR. , GOPALA KRISHNA REDDY KAKIVAYA & JOHN R. STONE (1998) INTRACTABILITY OF THE DIAL-A-RIDE PROBLEM AND A MULTIOBJECTIVE SOLUTION USING SIMULATED ANNEALING, *Engineering Optimization*, 30:2, 91-123, DOI: [10.1080/03052159808941240](https://doi.org/10.1080/03052159808941240)

To link to this article: <https://doi.org/10.1080/03052159808941240>



Published online: 27 Apr 2007.



Submit your article to this journal [↗](#)



Article views: 201



View related articles [↗](#)



Citing articles: 44 View citing articles [↗](#)

INTRACTABILITY OF THE DIAL-A-RIDE PROBLEM AND A MULTIOBJECTIVE SOLUTION USING SIMULATED ANNEALING

**JOHN W. BAUGH JR., GOPALA KRISHNA REDDY
KAKIVAYA and JOHN R. STONE**

*Department of Civil Engineering, North Carolina State
University, Raleigh, NC 27695-7908, U.S.A.*

(Received 10 January 1996; In final form 17 January 1997)

Numerous techniques for generating approximate solutions have been proposed in the last decade for routing and scheduling in multi-vehicle dial-a-ride problems. While some of these techniques have mathematical foundations, it is often difficult to assess the global optimality of the generated solution due to the use of pure local improvement methods. In addition, most of these methods are based on a single objective, such as minimization of the number of vehicles used, and cannot account for different or competing objectives that characterize the problem. This paper proves the intractability of the dial-a-ride problem, and then describes a new approximate method based on simulated annealing that is used to solve these problems in the presence of multiple objectives.

Keywords: Dial-a-ride problem; simulated annealing; multiobjective programming; decision-support systems

1. INTRODUCTION

Vehicle routing and scheduling are key activities in many transit operations, including the demand-responsive systems that are sometimes referred to as “dial-a-ride”. The basic elements of the dial-a-ride routing and scheduling problem (DARP) include the following:

- *Service requirements*—customers place requests in advance to be picked up and dropped off at particular locations within time windows.

- *Provider capabilities*—a fleet of shared-ride vehicles is located at a central depot. Schedules are generated in advance.

The objective of the problem is to design a complete tour for each vehicle that services all the customers within their time windows. A description of a tour includes the 'route', or sequence of locations visited by each vehicle, and 'schedule', or times at which the vehicle should arrive at those locations. The problem is complicated by the presence of competing objectives such as minimization of the total distance traveled by the vehicles, customer inconvenience, number of vehicles used, etc.

This paper gives an overview of related work and a concise description of the dial-a-ride problem that includes computational and tractability issues. A multiobjective approach using simulated annealing is then presented, followed by results obtained on a set of real-world data. It concludes with some observations about the multiobjective nature of the dial-a-ride problem.

2. RELATED WORK

Most of the early research efforts in vehicle routing and scheduling concentrated on the development of approximate methods for solution of the problem. The approximate approaches can be broadly classified into two types: *insertion heuristics* and *cluster-first and route-second heuristics*. In the first approach additional customers are inserted into an initial route in an optimal way. Thus, the quality of the solution depends on the order in which the customers are selected. In the second approach the entire set of customers is partitioned into as many subsets as there are available vehicles, and routes are then developed for the individual vehicles. Thus, the quality of the solution depends on the way in which partitioning is performed.

A solution to the dial-a-ride problem based on insertion heuristics has been proposed by many researchers [1–3]. The work of Jaw *et al.* [1] was extended to include vehicle capacity constraints and was applied to a small-scale demand-response operation in Winnipeg [4].

The single-vehicle dial-a-ride problem, although seldom occurring in practice, has been studied by a number of researchers because it occurs

in the second part of cluster-first and route-second algorithms. An exact dynamic programming approach has been used to solve the single-vehicle dial-a-ride problem [5]. This work was extended by incorporating additional state elimination rules to reduce the computational requirements [6]. Other approaches to the single-vehicle problem, including Benders decomposition [7], have also been used.

Other research [8] provides some insight into the worst case performance of well-known heuristics. An excellent overview of the algorithms and techniques used in vehicle routing and scheduling can be found in an article by Bodin *et al.* [9].

3. MATHEMATICAL MODEL

The static dial-a-ride problem is characterized by the following:

- *Routing constraints*: each customer's origin and destination need to be visited exactly once.
- *Assignment constraints*: each customer must be assigned to a trip, and the number of available trips cannot be exceeded.
- *Timing constraints*: each customer's origin and destination can be visited only within the prescribed time windows.
- *Precedence constraints*: the same vehicle that picks up a customer at an origin should drop off the customer at the destination.
- *Capacity constraints*: the capacity of the vehicle cannot be exceeded at any time.
- *Competing objectives*: various objectives should be minimized, including the distance traveled by all vehicles, customer inconvenience, and the number of vehicles used.

In this statement of the problem, customers are assigned to trips, or depot-to-depot components of a tour. Because only a subset of trips overlap in time, the number of vehicles required to service the customers will often be much less than the total number of scheduled trips.

A mathematical model for the problem is given in Figures 1 and 2, in which the following parameters are used:

- N number of customers
- C capacity of the vehicle

$$\text{Minimize } z_1 = \sum_{k=1}^V \sum_{i=0}^{2N} \sum_{j=0}^{2N} t_{ij} x_{ij}^k, \quad z_2 = \sum_{i=1}^{2N} s_i, \quad z_3 = v$$

Subject to

Trips leaving the depot

$$\sum_{k=1}^V \sum_{j=1}^N x_{0j}^k = v$$

Trips returning to the depot

$$\sum_{k=1}^V \sum_{i=N+1}^{2N} x_{i0}^k = v$$

Every origin should be visited

$$\sum_{k=1}^V \sum_{j=1}^{2N} x_{ij}^k = 1 \quad i = 1, \dots, N$$

A node is in only one trip

$$\sum_{j=0}^{2N} x_{ji}^k - \sum_{j=0}^{2N} x_{ij}^k = 0 \quad i = 1, \dots, 2N, k = 1, \dots, V$$

Origin and destination of a customer are in the same trip

$$\sum_{j=0}^{2N} x_{ij}^k - \sum_{j=0}^{2N} x_{(i+N)j}^k = 0 \quad i = 1, \dots, N, k = 1, \dots, V$$

Time-window and capacity constraints...

FIGURE 1 Multiobjective dial-a-ride model.

- V maximum number of trips
 - S maximum time violation permitted at a node
 - t_{ij} travel time between nodes i and j
 - ET_i earliest allowed departure time at node i
 - LT_i latest allowed departure time at node i
 - T a number larger than the latest drop-off time
- where a node represents either an origin or a destination.

Start time at depot

$$y_0 = 0$$

A trip should depart from a node after arrival

$$y_j \geq y_i + t_{ij} \sum_{k=1}^V x_{ij}^k - \left(1 - \sum_{k=1}^V x_{ij}^k\right) T \quad i = 1, \dots, 2N, j = 1, \dots, 2N$$

Time-window violation at a node

$$ET_j - s_j \leq y_j \leq LT_j + s_j \quad j = 1, \dots, 2N$$

Capacity of the vehicle at the start of the trip

$$c_0 = 0$$

Capacity of the vehicle after visiting an origin node

$$c_j \geq c_i + \sum_{k=1}^V x_{ij}^k - \left(1 - \sum_{k=1}^V x_{ij}^k\right) T \quad i = 1, \dots, 2N, j = 1, \dots, N$$

Capacity of the vehicle after visiting a destination node

$$c_j \geq c_i - \sum_{k=1}^V x_{ij}^k - \left(1 - \sum_{k=1}^V x_{ij}^k\right) T \quad i = 1, \dots, 2N, j = N+1, \dots, 2N$$

Capacity of the vehicle should not be exceeded

$$0 \leq c_j \leq C \quad j = 1, \dots, 2N$$

FIGURE 2 Time-window and capacity constraints.

The primary decision variables in the model are as follows:

- x_{ij}^k $\begin{cases} 1 & \text{if travel from node } i \text{ to node } j, \text{ denoted as arc } ij, \text{ is on trip } k \\ 0 & \text{otherwise} \end{cases}$
- v number of trips
- y_i departure time at node i
- s_i amount of time-window violation at node i
- c_i number of customers in the vehicle at node i

The following arcs are infeasible and hence can be eliminated from the model:

$$x_{ij}^k \text{ is infeasible for all } k \text{ when } \begin{cases} i = j \\ i = j + N \\ i \leq N \text{ and } j = 0 \\ i = 0 \text{ and } j > N \\ ET_j > LT_i + t_{ij} + 2S \\ LT_j < ET_i + t_{ij} - 2S \end{cases}$$

The objective function contains three objectives: cumulative travel time, degree of violation of time windows, and number of trips. The total travel time does not include the time spent by the vehicles idling to adhere to time windows: it includes only the time spent by vehicles in transit.

4. DIAL-A-RIDE PROBLEM IS NP-HARD

Before proposing a general algorithmic solution, it is important to understand the tractability of a problem. This section shows that the dial-a-ride problem is NP-hard, thereby justifying the development of approximate techniques for the solution of large data sets.

To prove that DARP is NP-hard, two other NP-hard problems are used, namely, the Hamiltonian cycle problem and the traveling salesman problem with time windows (TSPTW). Hard time windows for both DARP and TSPTW are assumed for clarity in presentation, but note that the results are equally applicable to DARP with soft time windows. Intuitively, hard time windows are a special case of soft time windows in the sense that soft windows can be made hard by imposing arbitrarily large penalties for their violation. Since the special case is NP-hard, the general case cannot be solved more easily.

DEFINITION 1 Hamiltonian cycle problem.

Given a graph G , is there a cycle that passes through every node in the graph exactly once?

DEFINITION 2 Traveling salesman problem with time windows (TSPTW).

Given a weighted graph G , time windows at every node of G , and a start node, find the lowest cost cycle that starts and ends at the origin, and that enters and leaves each node within its time window, with the leaving time being later than the entering time.

DEFINITION 3 Dial-a-ride problem.

Given a weighted graph G consisting of origin/destination nodes, time windows at every node of G , and a start node, find the cycles that minimize the summation of arc weights. Each cycle should start and end at the start node, contain both the origin and destination nodes for requests serviced by the cycle, visit an origin before the destination, and do so within the prescribed time windows.

LEMMA 1 *The traveling salesman problem with time windows is NP-hard.*

The decision version of TSPTW can be stated as follows:

Given a weighted graph G , time windows at every node of G , and a start node, is there a cycle not exceeding K in cost that starts and ends at the start node and that enters and leaves each node within its time window, with the leaving time being later than the entering time?

The decision version of TSPTW is shown to be NP-complete by showing that

- TSPTW is in NP.
The guesser guesses some random permutation of nodes as comprising a cycle.
The checker checks to see that it is a valid cycle. This checking can be done in polynomial time.
- The Hamiltonian cycle problem can be reduced to TSPTW.

Let $G=(V, E)$ be the input graph for the Hamiltonian cycle problem, which is transformed to a weighted graph G' that is given as input to TSPTW using the following rules:

1. For every node in G , add a corresponding node in G' .
2. For every arc in G , add an arc of weight 1 to G' .
3. Let the time window at each node in G' be $(0, n)$ where n is the number of nodes in G .
4. Designate any one of the nodes in G' as the start node.

It can be seen that G' may be constructed from G in polynomial time. G' is given as the input to TSPTW with K set to n . If G' has a TSPTW solution, then G has a Hamiltonian cycle, and conversely, if G has a Hamiltonian cycle, then G' has a TSPTW solution. Hence, the decision version of TSPTW is NP-complete.

THEOREM 1 *The dial-a-ride problem is NP-hard.*

The decision version of DARP can be stated as follows:

Given a weighted graph G consisting of origin/destination nodes, time windows at every node of G , and a start node, is it possible to visit every node exactly once within its time window in n cycles at a cost not exceeding C ? The cycles should satisfy the following: the leaving time at a node is later than the entering time, the origin/destination pairs are in the same cycle, the origin is visited before the corresponding destination, and each cycle starts and ends at the start node.

The decision version of DARP is shown to be NP-complete by showing that

- DARP is in NP.

The guesser guesses n cycles.

The checker checks to see that each cycle is valid and that the summation of arc weights in the cycles does not exceed C . This checking can be done in polynomial time.

- TSPTW can be reduced to DARP.

Let $G = (V, E)$ be the input graph for TSPTW, which is transformed to a graph G' that is given as the input to DARP using the following rules:

1. For every node in G , add a source/destination pair of nodes in G' with the same time windows as the node in G .
2. For every arc in G , add an arc of the same weight to G' . Also add arcs of zero weight between every source/destination pair of nodes in G' .

G' can be constructed from G in polynomial time. G' is given as the input to DARP with n set to 1 and C set to K . If G' has a DARP solution, then G has a TSPTW solution, and conversely, if

G has a TSPTW cycle, then G' has a DARP solution. Hence, the decision version of DARP is NP-complete.

5. MATHEMATICAL PROGRAMMING TECHNIQUES

Even though a problem is shown to be NP-hard, the problem sizes encountered in practice may be sufficiently small that they can be solved in a reasonable amount of time. Also, in the case of integer programming, different models representing the same problem may present different degrees of difficulty in solving the problem. In other words, it is worth exploring alternative mathematical programming approaches for solving a practical problem that has been shown to be NP-hard. Several mixed-integer linear programming (MILP) models have been developed, and are described below along with their computational behaviors.

- *Discretization of time windows and vehicle paths*

Each arc between two nodes is split into a number of arcs for the various trips and time-window steps that are feasible. Thus, if there are v trips and the time window at each node is split into t steps, then each arc between two nodes results in vt binary variables. The difficulty in solving a model based on this approach is the presence of a large number of binary variables. A variant of this approach, described in Section 3, discretizes only the trips and has continuous variables that represent time.

- *Separation of routing and assignment variables*

Each arc between two nodes has associated with it a binary variable indicating whether it is used in the route or not. In addition, each node has associated with it V binary variables, where V is the number of available trips, indicating which trip has visited that node. Since the number of nodes is much less than the number of arcs for most real-world problems that have overlapping time windows, there is consequently a reduction in the number of binary variables when compared with the model based on the first approach. Nevertheless, the model remains difficult to solve because of an overall increase in the number of constraints, particularly assignment constraints.

- *Introduction of general integer variables*

This is a variant of the model based on the second approach. Instead of associating v binary variables with each node, a single general integer variable is associated with each node to represent the vehicle that visits the node. This general integer variable can take any integral value between 1 and V , where V is the number of available trips. The advantage of this model is the significant reduction in the number of constraints compared to the other models. However, models based on this approach are inherently flexible, and require fathoming more branches in the branch-and-bound process to prove optimality.

Each of the above models has been examined for ease in solvability with a commercial optimization package. The largest problem size that could be solved was about 10 customers, which is small compared to the practical size of around 300 customers. To enable fine-tuning of the branch-and-bound solution process, a custom branch-and-bound solver was developed for the MILP formulation of the dial-a-ride problem. The algorithm used in the custom branch-and-bound solver is described below.

1. Solve the problem as an LP problem with the integrality requirements relaxed.
2. Examine the depot-origin arcs of the n customers. If any of these arcs has a nonzero value, select the depot-origin arc with the largest nonzero value. Assign a new vehicle to that depot-origin arc and branch on that arc by assigning it a unit value. Go to Step 1.
3. For each vehicle, follow its path from the depot along the arcs that assume a unit value. Stop at the first occurrence of a fractional value for an arc along the path, i.e., where the vehicle splits and goes in more than one direction. Note the arc with the largest fractional value. Repeat the process for all the vehicles. Select the arc that has the largest fractional value among all the vehicles and let it be arc ij . The summation of the product of arc value (obtained in the relaxed LP) and arc length for all the arcs converging at node j gives a reasonable lower bound on the length of the arc that can enter node j (call that length a). If the length of arc ij is less than a , branch on arc ij by assigning it a value of zero; otherwise, branch on the same arc ij by assigning it a value of one.

A commercial package is used for solving the relaxed LP at each node in the branch-and-bound tree. The custom branch-and-bound solver has been tested on small, randomly generated data sets and the initial results looked promising because the custom solver found initial feasible and optimal solutions much faster than the commercial package. However, for problem sizes beyond 15 customers, it faced difficulties in finding any feasible integer solutions. Thus, the custom branch-and-bound solver improved only marginally upon the commercial package by solving problem sizes of 15 customers.

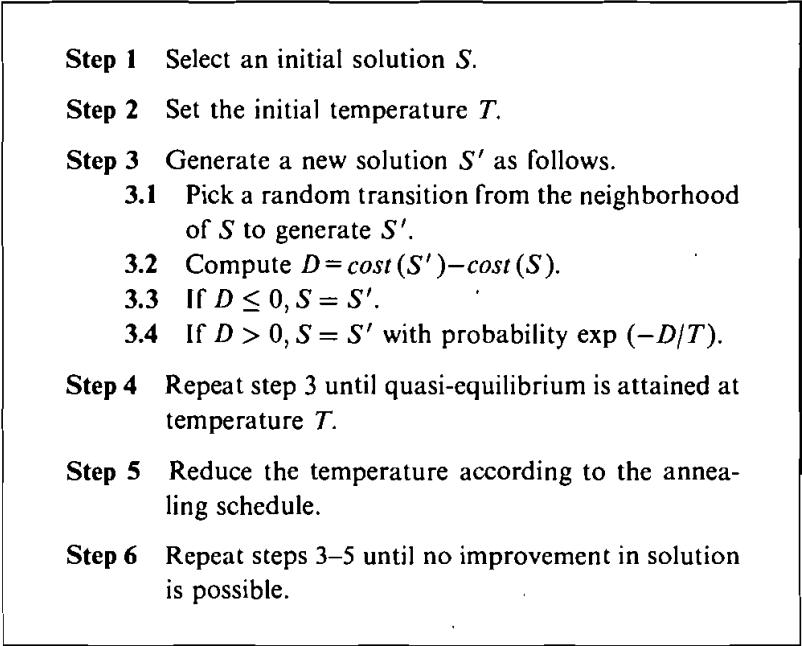
6. SIMULATED ANNEALING

Since the dial-a-ride problem is NP-hard, one cannot expect to find globally optimal solutions in polynomial time. Various heuristic methods mentioned in the literature are successful in solving realistic problems with reasonable computational requirements but suffer from limitations concerning global optimality of the solutions and the ability to account for different or competing objectives. Modern heuristic techniques, such as simulated annealing, tabu search, and genetic algorithms, are capable of efficiently searching the solution space for near-globally optimal solutions with respect to an appropriately defined objective function.

The approach described below is based on simulated annealing, a straightforward local improvement procedure with systematic randomness incorporated. As shown in Figure 3, the algorithm starts with an initial solution and temperature. At each step it generates a new solution that is accepted if it improves the objective; otherwise it is accepted with a probability $\exp(-\Delta C/t)$, where ΔC is the increase in the cost and t is the current annealing temperature. After a number of steps the annealing temperature is reduced and the process is repeated until the system “freezes” with a solution.

Of the modern heuristic techniques, simulated annealing appears to be most suitable for the dial-a-ride problem for the following reasons:

- The technique can easily be adapted for problems with a well-defined neighborhood structure.



Step 1 Select an initial solution S .

Step 2 Set the initial temperature T .

Step 3 Generate a new solution S' as follows.

3.1 Pick a random transition from the neighborhood of S to generate S' .

3.2 Compute $D = \text{cost}(S') - \text{cost}(S)$.

3.3 If $D \leq 0$, $S = S'$.

3.4 If $D > 0$, $S = S'$ with probability $\exp(-D/T)$.

Step 4 Repeat step 3 until quasi-equilibrium is attained at temperature T .

Step 5 Reduce the temperature according to the annealing schedule.

Step 6 Repeat steps 3–5 until no improvement in solution is possible.

FIGURE 3 Simulated annealing algorithm.

- The technique has desirable theoretical convergence properties. Even though in the worst case it can take infinite amount of time to find the global optimum, given a suitable annealing schedule the algorithm has been shown to find near-globally optimal solutions in a reasonable amount of time.
- Other heuristics, especially tabu search, can easily be integrated into simulated annealing.

Given simulated annealing as the approach for solving a problem, the following must be defined: the neighborhood of a solution, transitions on the neighborhood, and an annealing schedule for utilizing the approach. These characteristics are described in the next few sections.

6.1. Cluster-First Route-Second Strategy

The cluster-first and route-second strategy partitions the entire set of customers into clusters so that customers belonging to the same cluster

are serviced together. Routes are then developed for the individual clusters. The problem is approached by using simulated annealing for clustering and a modified space-time nearest neighbor heuristic for developing the routes for the clusters. The reasons for this choice are given below.

- The most crucial decision for DARP is the assignment of customers to available vehicles.
- Once customers have been clustered, very good heuristic algorithms, such as the Lin-Kernighan algorithm, are available for routing. Further, since the number of customers in a cluster is of the order of 10 or 20, even approximate heuristics like the space-time nearest neighbor heuristic give very good solutions with minimal computational requirements.

6.1.1. Clustering Strategy

Clustering is performed by the simulated annealing algorithm. The initial clustering is determined by a random assignment of customers to clusters. Routes for the clusters are developed using the modified space-time heuristic algorithm, and the cost of clustering assessed.

Two operations are used to alter the clustering of customers at each stage of the simulated annealing algorithm.

- *Exchange*—Two customers are chosen randomly and their current clusters are identified; if both are in the same cluster, the customers are discarded and a new pair is generated. The cluster assignment of the customers is then exchanged, leaving the total number of clusters unchanged.
- *Swap*—A random customer and a random cluster are selected. The cluster of the customer is identified, and the customer is then swapped to the randomly generated cluster. Again, if the customer's cluster is the same as the randomly generated one, a new customer and cluster are randomly generated. The swap move has the potential to increase or decrease the total number of clusters by creating a new cluster for the chosen customer or by swapping out the only customer of a given cluster.

The neighborhood structure that results from these operations has the following desirable properties:

- It is simple.
- It is possible to generate any cluster from any other cluster by a series of transitions. This property is very important for the proper functioning of the simulated annealing algorithm.
- The exchange operation results in a smoother objective space, while the swap operation allows for the dynamic increase or decrease of the number of clusters.

At each iteration of the simulated annealing algorithm, the routing algorithm is invoked on the selected clusters. After developing routes for these clusters, the number of vehicles needed to service the developed routes is reassessed. Finally, the objective function is evaluated by appropriately penalizing the following parameters of the solution:

- the total distance traveled by all the vehicles
- the total disutility caused to the customers
- the number of vehicles used

If the new partitioning results in improvement, it is accepted; otherwise it is accepted with probability $\exp(-\Delta C/t)$, where ΔC is the increase in cost and t is the current temperature. The simulated annealing algorithm uses one type of transition operation (either an exchange or a swap) until a transition is rejected, at which point the alternate operation is used to generate new transitions.

6.1.2. Routing Strategy

After clusters have been generated, routing is performed by a greedy algorithm based on a space-time nearest neighbor heuristic. It starts a route by visiting the customer with the earliest pickup time. At each location the cost of visiting other locations is determined by estimating the cost of the next three succeeding moves if the location under consideration is visited. The algorithm identifies the succeeding moves by considering the space-time separation between locations and selecting the shortest move. The space-time separation between two locations is quantified by a weighted summation of travel time between

the locations and the time-window violation at the destination. The time-window violation is positive if the latest time at which the destination can be visited precedes its time window; conversely, the violation is negative if the earliest time at which the destination can be visited exceeds its time window. Thus, distance is affected by separation in both space and time. The cost of a move between two locations is the weighted summation of travel time and the absolute amount by which the time window is violated at the visited location. Of course, the costs are normalized by their associated penalties.

Implementation of the routing strategy is realized by converting hard time windows to soft ones. A hard time window cannot be violated: in a given schedule a vehicle must reach a destination within its time window, otherwise the solution is not feasible. In contrast, a soft window can be violated at a cost, and therefore may be regarded as a generalization of a hard time window. The advantages of converting hard windows to soft are as follows:

- In a real-world situation, windows are usually soft in the sense that there is a limit up to which a service provider will adhere to the time windows.
- Soft time windows are useful in evaluating the tradeoffs between service requirements and cost requirements. In addition, hard time windows can be modeled with soft ones by imposing large penalties on violated service requirements.
- Algorithms based on soft time windows are capable of finding solutions in cases where hard time windows would have failed. Thus, solutions obtained with soft windows indicate the degree of violation, allowing penalty methods to distinguish between a given pair of infeasible solutions (in attempting to find a feasible region).

Further details of the routing strategy are given in Appendix A.

6.2. Annealing Schedule

The success of the simulated annealing algorithm depends on the proper choice of the annealing parameters, viz., initial temperature, number of transitions carried out at each temperature, rate of cooling, and final temperature.

6.2.1. Number of Transitions at a Given Temperature

The number of transitions N_t carried out at any given temperature should be sufficient to establish quasi-equilibrium (stationary probability distribution over possible states) at that temperature. Since accepted transitions are the only ones that contribute to the attainment of quasi-equilibrium, N_t should be such that the number of accepted transitions is equal to some number η_{\min} . However, as the temperature is reduced, transitions are accepted with decreasing probability, implying that $\lim_{t \rightarrow 0} N_t = \infty$. Consequently, a ceiling \bar{N} is put on the number of transitions generated. The constant \bar{N} should be based on the neighborhood size so that the algorithm can search a major part of the neighborhood of the current solution at low temperatures.

In the current implementation of the simulated annealing algorithm, \bar{N} is fixed as $N(N-1)/2$ and η_{\min} is fixed as N , where N is the number of customers in the problem. However, for large problems, the above parameter values result in a large number of transitions to be generated at each temperature. Since evaluating a proposed transition is a time consuming process, \bar{N} is reduced to 1/5 of the neighborhood size for the WSTA problem described in Section 7.

6.2.2. Initial Temperature

The initial temperature should be such that virtually all transitions are accepted, i.e.

$$\exp(-\Delta C/T_0) \approx 1$$

For a certain temperature T , let m_1 be the number of cost-decreasing transitions and m_2 be the number of cost-increasing transitions generated, and let $\overline{\Delta C^+}$ be the average increase in cost over the m_2 transitions. Then, the acceptance ratio χ is obtained by the following:

$$\chi = \frac{m_1 + m_2 \exp(-\overline{\Delta C^+}/t)}{m_1 + m_2}$$

The above formula can be rewritten as

$$t = \frac{\overline{\Delta C^+}}{\ln\left(\frac{m_2}{m_2\chi - m_1(1-\chi)}\right)}$$

In the current implementation of the simulated annealing algorithm, the initial temperature is chosen to accept 90% of the transitions. First, T is fixed at 100, and the algorithm is executed for N_t transitions. The values of m_1 , m_2 , and $\overline{\Delta C^+}$ are calculated, and the above formula is used to update the value of T . The above process is repeated until T reaches a stable value.

6.2.3. Rate of Cooling

The rate of cooling should be slow enough so that a relatively small number of transitions is sufficient to re-establish quasi-equilibrium at the new temperature. Most of the implementations of simulated annealing reported in the literature use geometric cooling rates, e.g.

$$T_{k+1} = \alpha T_k$$

where α is a constant smaller than 1, typically in the range 0.8–0.99.

Another approach to temperature reduction is based on the notion that the cooling rate should be such that relatively few transitions are needed to re-establish quasi-equilibrium at the new reduced temperature (The assumption is that quasi-equilibrium is achieved at the initial temperature). The reduction in temperature in such a case should be proportional to the entropy of the system. The higher the entropy, the larger the possible temperature reduction. This phenomenon is also observed in the metal annealing process. At high temperatures, the metal has high entropy, and consequently, the rate of cooling is faster than at lower temperatures. The standard deviation of the cost distribution at any given temperature is a good quantifier of the entropy of the system at that temperature. The rate of cooling in the current implementation is based on the above concept and uses the following two equations, the first of which is due to Laarhoven [10]:

$$T_{k+1} = \frac{T_k}{1 + \frac{T_k \ln(1+\delta)}{3\sigma_k}}$$

$$T_{k+1}/T_k \geq 0.8$$

where δ is a constant chosen to be 1 in the current implementation and σ_k is the standard deviation of the cost distribution at temperature T_k .

6.2.4. Final Temperature

Theoretically, the temperature should be allowed to become zero. However, since at very low temperatures very few transitions are accepted, the system freezes with a solution at some temperature which is slightly higher than zero. Therefore, the algorithm is stopped if 3 successive temperatures pass without a single acceptance.

6.3. Tabu List

To improve performance, a tabu list is used to give short-term memory to the annealing algorithm so that accepted transitions are not immediately reversed. That is, an accepted transition is retained for a fixed number of new state transitions before it is allowed to be reversed. This approach has the effect of allowing the newly accepted transitions some time to show their effectiveness.

In the current implementation, the length of the tabu list is $N/10$, where N is the number of customers in the problem. The list contains customers involved in the most recently accepted transitions, keeping the customers in FIFO order. When a generated transition contains a tabu customer, it is rejected, and a new transition is generated. When a proposed transition is accepted, the customers involved in the transition are inserted into the tabu list.

7. RESULTS

Although simulated annealing has been shown to converge to the global optimum, an infinite amount of computation time is required to prove optimality because of the need for an extremely slow annealing schedule. However, less conservative (and less time-consuming) schedules have been shown to produce near-globally optimal solutions. A side-effect of any practical annealing schedule is the variability of solutions in different runs. For a properly selected annealing schedule, however, this variability should be small.

The robustness of the present algorithm in consistently finding good optimal solutions is ascertained by running it on a problem with 25 customers with a different random seed for each run. The results are

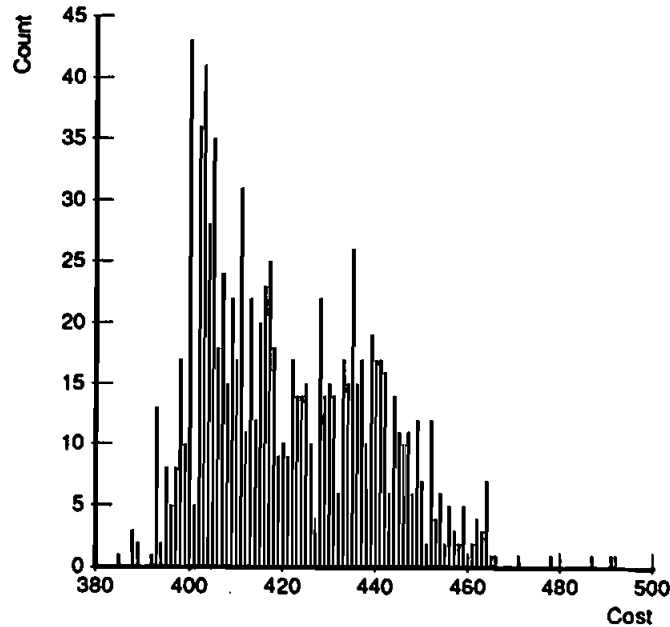


FIGURE 4 Histogram of SA results.

plotted as a histogram in Figure 4. The mean and standard deviation of the solutions are 421.46 and 18.38, respectively.

The algorithm has been tested on a data set provided by the Winston Salem Transit Authority (WSTA), which services over 300 customers a day. A customer may request either a desired pick-up time or a desired drop-off time. The travel-time distribution, spatial distribution, and type of requests for March 14, 1995, at WSTA are shown in Figures 5, 6 and 7. The travel-time distribution shows that a significant number of WSTA customers travel very short distances. These are mostly senior citizens who live around the hospital and day care centers. The spatial distribution indicates the distance in miles from WSTA to the requested service locations. The distribution of request types shows a clustering of drop-off requests in the morning and pick-up requests in the evening — an expected pattern. The drop-off requests peak around 11:30 a.m. and most of these requests are for hospital appointments. The pick-up requests have two peaks: the first

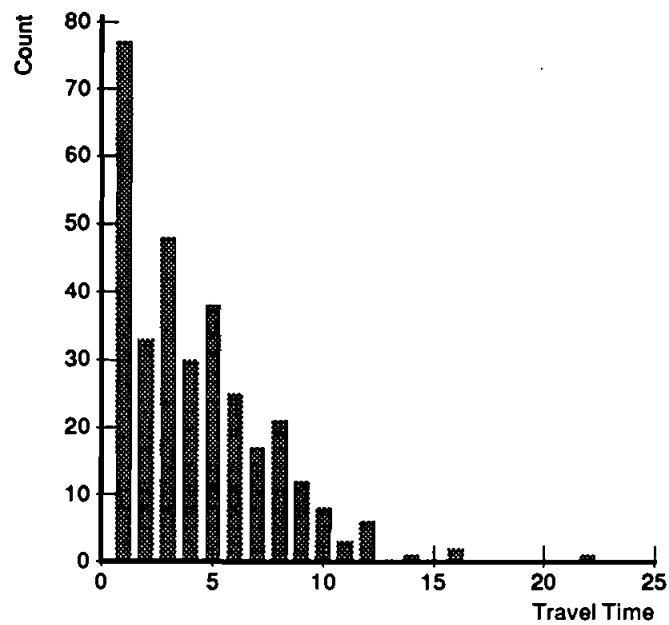


FIGURE 5 Travel-time distribution for WSTA data.

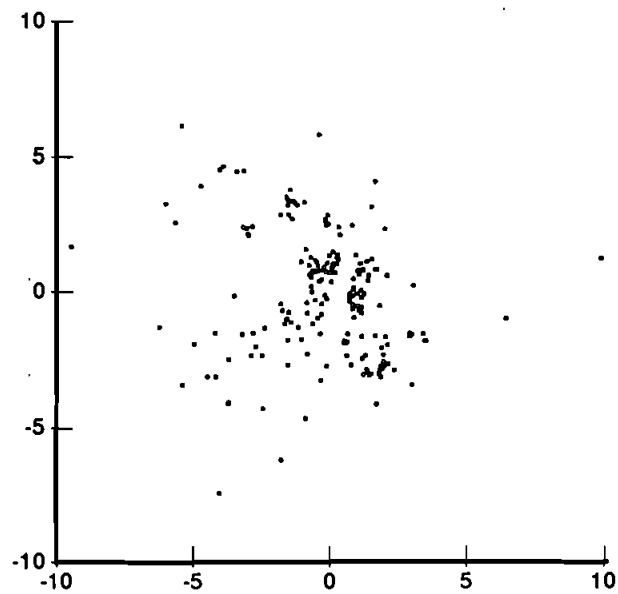


FIGURE 6 Spatial distribution of WSTA data (in miles).

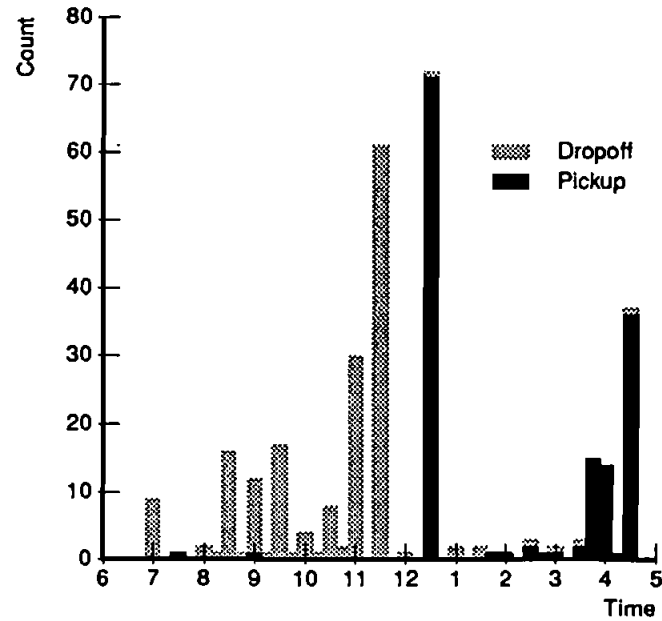


FIGURE 7 Request-type distribution for WSTA data.

around 12:30 p.m. and the second around 4:30 p.m. The first peak follows the peak in the drop-off requests, possibly indicating the requests of customers returning from hospital visits.

By varying weights on objectives, the tradeoff curve shown in Figure 8 is generated, where each circle represents a different schedule with an associated customer disutility and operation cost. The filled circles designate non-dominated solutions, which are summarized in Table I. The table also includes characteristics of the schedule actually used at WSTA, which, while being non-optimal, incorporates several policies that were not modeled.

For the results presented, customer disutility is quantified by the amount of violation of time windows, and cost is quantified by the total travel time of all vehicles and the number of vehicles used. The tradeoff curve is interesting because it shows that no single-objective approach is adequate for these real-world data. For example, the minimum cost solution of 1533 minutes has a disutility 2123 minutes, but if one is willing to increase vehicle travel time by only 61

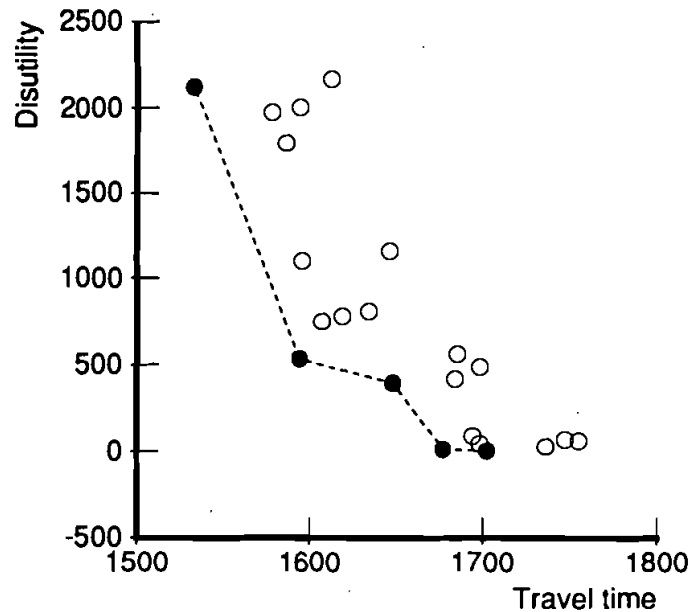


FIGURE 8 Cost vs. disutility for WSTA data.

TABLE I Non-dominated solutions for WSTA data

<i>Method</i>	<i>Travel time</i>	<i>Disutility</i>	<i>Trip time</i>	<i>Vehicles</i>	<i>Trips</i>
SA	1702	8	3917	10	26
SA	1677	17	3653	9	27
SA	1648	394	4381	11	26
SA	1594	534	4669	11	23
SA	1533	2123	6002	15	25
WSTA	2270	5697	13259	14	31

minutes (30.5 miles), the customer disutility is reduced by almost 75%. A similar argument can be made against the minimum disutility solution. Such tradeoff curves should help decision-makers identify desirable operating strategies based on a realistic assessment of the various competing objectives.

Because the WSTA data contain a significant fraction of short trips, it was felt that this particular aspect of the data might favor customer satisfaction over other competing objectives. To change this property

of the data, the WSTA data set was altered by removing short trips and scaling the service region by a factor of two, leaving over a hundred customer requests. The resulting tradeoff curve is shown in Figure 9. Again, the tradeoff curves shows that neither minimum cost nor minimum disutility schedules are reasonable operating strategies, since the vertical and horizontal legs of the curve are nearly parallel with their respective axes.

8. SOFTWARE ENVIRONMENT

The key characteristics of the dial-a-ride problem are the *number of customers* to be serviced, *size of the service area*, *distribution of customers in space and time*, and *time-window size at each location*. The following software components were developed as part of this study:

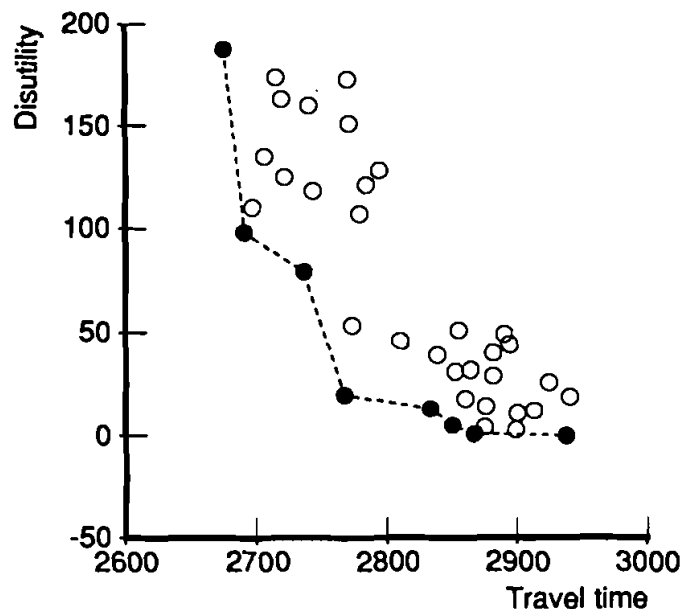


FIGURE 9 Cost vs. disutility for modified WSTA data.

- *Random data set generators*

1. Generate locations with random x and y coordinates within the specified service area. The locations are uniformly distributed across the service area.
2. Generate a prescribed number of customers with random origin/destination pairs and pick-up/drop-off times. The customers are uniformly distributed in space and time.

- *Space-time nearest neighbor heuristic algorithm*

Generate the route for a given assignment of customers to a vehicle. The program is used at each stage of the simulated annealing algorithm to evaluate the cost of the generated partitions.

- *Simulated annealing algorithm*

The simulated annealing algorithm should be run multiple times on the same input data for the following reasons.

1. Because simulated annealing is a stochastic method, multiple runs using different random seeds can be used to build confidence in the quality of the solutions obtained.
2. Tradeoffs between various objectives may be determined by changing the weights associated with them.

Because each run of the simulated annealing algorithm takes time to generate a solution, it is helpful to provide capabilities that allow new solutions to be generated while examining previous ones. The simulated annealing solver is implemented as a concurrent server using a client-server approach. The server may be simultaneously run on multiple machines, including the one on which the client is running. The client is implemented as an X-windows-based GUI, as shown in Figure 10. The user interacts only with the client in solving the problem, which maintains a user-defined list of servers that are available for remote processing. The client is also responsible for balancing loads among the servers. Once a server list has been defined, communication takes place between client and server using TCP/IP stream sockets. After the input data is read, new runs can be started by changing various parameters, including the random seed and the weights associated with objectives. Solutions are displayed graphically

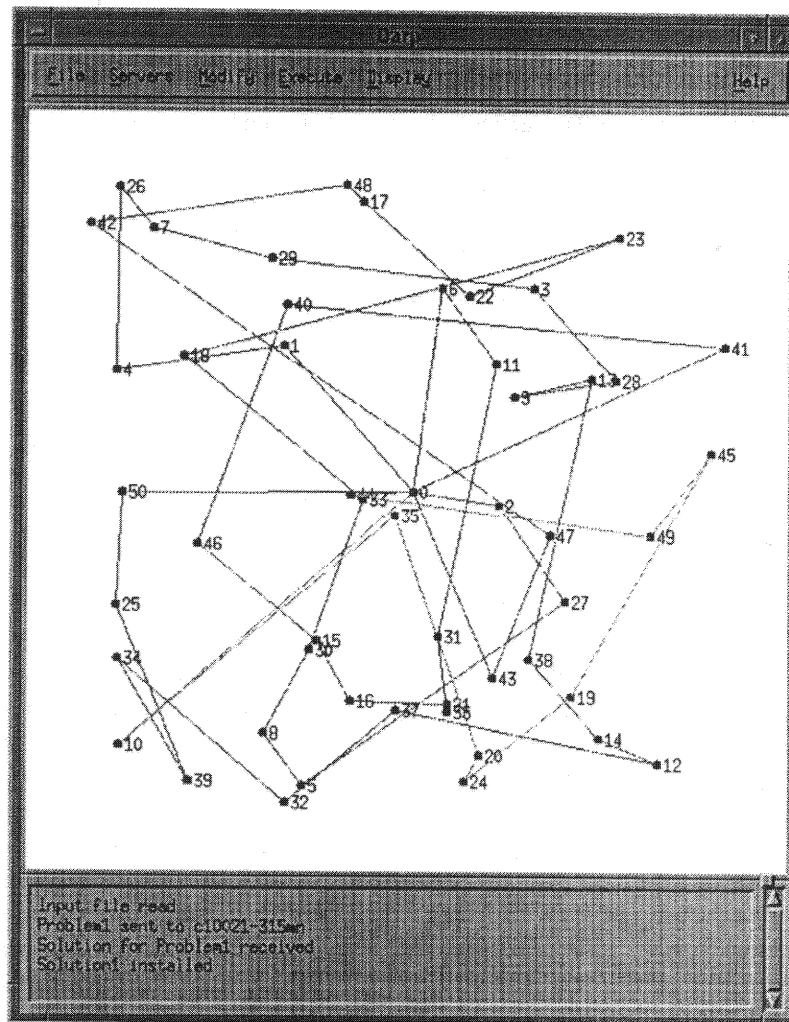


FIGURE 10 Graphical interface of client.

and interactively: users can query the features of a solution, e.g., when a customer is serviced, how many trips are made, and the number of vehicle miles traveled.

These software tools collectively represent a highly automated software environment for the solution of dial-a-ride problems.

9. CONCLUSION AND FURTHER RESEARCH

A multiobjective approach to the dial-a-ride problem allows us to investigate tradeoffs, particularly those between various measures of customer satisfaction and cost. Customer satisfaction is characterized by the degree of violation of desired time windows, and cost is characterized by the total travel time of all vehicles and the number of vehicles required.

Because the dial-a-ride problem is shown to be NP-hard, any practical approach to the problem must resort to heuristic techniques. The proposed approach, based on simulated annealing, gives near-globally optimal solutions, is robust, requires minimal user input in fine tuning the annealing parameters, and runs in time polynomial in input size. The approach is useful for generating tradeoff curves that help decision makers identify desirable operating strategies based on a realistic assessment of the various competing objectives.

An issue left for future work is a comprehensive validation of the approach. Generally speaking, there are several ways in which validation might be done. First, if an approach is straightforward, it may be mathematically tractable to determine a worst-case bound on the solutions generated, as is done for some heuristic methods; it seems unlikely that the simulated annealing algorithm is amenable to that type of analysis, however. Another approach would be compare solutions with either known optimal solutions or the best solutions that have been generated by other approximate techniques. Because neither of these appear to be available in the literature, one might make comparisons with respect to simplified cases, e.g., single objective models and small data sets, where optimal or provably near-optimal solutions can be obtained. For example, MILP solutions were found to be comparable to the simulated annealing solutions for 10- and 15-customer problems. Finally, by generating data sets with a known probability distribution (in both space and time), the quality of actual solutions might be compared with the "expected" quality of a solution.

In addition to validation, there are many potential directions for subsequent study, including experimentation with the generated schedules in practice, extensions to combined demand-response and fixed-route systems, and an analysis of the effect of "no-shows" on

schedules generated with various objectives. Because some paratransit operations have regular subscription trips, and there is some cost in running what may be completely different schedules each day, an interesting follow-on study would examine the reduction in cost obtained by rescheduling subscription trips daily as opposed to fixing subscription routes and adding non-subscription trips.

Acknowledgments

This research is supported in part by the Federal Transit Administration, the North Carolina Department of Transportation, and the Department of Civil Engineering at North Carolina State University. The comments of Dr. Matthias Stallmann on the NP-hardness proof are appreciated.

References

- [1] Jaw, J.-J., Odoni, A. R., Psaraftis, H. N. and Wilson N. H. M. (1996). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research-B*, **20B**(3), 243–257.
- [2] Kikuchi, S. and Rhee J.-H. (1989). Scheduling method for a demand responsive transportation system. *Journal of Transportation Science*, **115**(6), 630–645.
- [3] Kikuchi, S. and Donnelly, R. A. (1992). Scheduling demand-responsive transportation vehicles using fuzzy-set theory. *Journal of Transportation Science*, **118**(3), 391–409.
- [4] Alfa, A. S. (1986). Scheduling of vehicles for the transportation of the elderly. *Transportation Planning and Technology*, **11**, 203–212.
- [5] Psaraftis, H. N. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, **17**(3), 351–357.
- [6] Desrosiers, J. Dumas, Y. and Soumis, F. (1986). A dynamic programming solution of the large scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, **6**(3/4), 301–325.
- [7] Sexton, T. R. and Bodin, L. D. (1985). Optimizing single vehicle many-to-many operations with desired delivery times: Scheduling and routing. *Transportation Science*, **19**(4), 378–435.
- [8] Solomon, T. R. (1986). On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time windows. *Networks*, **16**, 161–174.
- [9] Bodin, L., Golden, B., Assad, A. and Ball, M. (1983). The state of the art in the routing and scheduling of vehicles and crews. *Computers and Operations Research*, **10**, 63–211.
- [10] Van Laarhoven, P. J. M. and Aarts E. H. (1987). *Simulated annealing: Theory and applications*. D. Reidel Publishing Company, Holland.

10. APPENDIX A

Space-time Nearest Neighbor Algorithm

C: set of customers;
CS: set of customers serviced;
CV: set of customers in the vehicle;
CN: set of customers yet to receive service;
w1: weight on travel time;
w2: weight on violation of time windows;

separation (*cnode*, *nnode*: node, *ctime*: time)
 returns space-time separation between *cnode* and *nnode*
ttime, *etime*, *ltime*: time;
twviol: integer;
 begin
 ttime = distance from *n1* to *n2* divided by speed;
 etime = *ctime* + *ttime*;
 if *ctime* > latest time window at *cnode* then
 ltime = *etime*;
 else
 ltime = latest time window at *cnode* + *ttime*;
 endif

 twviol = 0;
 if *ltime* < earliest time window at *nnode* then
 twviol = earliest time window at *nnode* - *ltime*;
 else if *etime* > latest time window at *nnode*
 twviol = latest time window at *nnode* - *etime*;
 endif
 return (*w1* * *ttime* + *w2* * *twviol*)

end separation

movecost (*cnode*, *nnode*: node, *ctime*: time)
 returns cost of the move from *cnode* to *nnode*
ttime, *etime*, *ltime*: time;
twviol: integer;
 begin
 ttime = distance from *n1* to *n2* divided by speed;

```

etime = ctime + ttime;
if ctime > latest time window at cnode then
    ltime = etime;
else
    ltime = latest time window at cnode + ttime;
endif

twviol = 0;
if ltime < earliest time window at nnode then
    twviol = earliest time window at nnode - ltime;
else if etimelatest time window at nnode
    twviol = etime - latest time window at nnode;
endif
return (w1 * ttime + w2 * twviol)
end movecost

closest (cnode: node, ctime: time, N: set of nodes)
    returns node closest to cnode
minldis, newldis: integer;
cstnode: node;
begin cstnode = -1;
    minldis = ∞;
    for each c in CV do
        if destination of c ∉ N then
            newldis = separation (cnode, destination of c, ctime);
            if minldis < newldis then
                minldis = newldis;
                cstnode = destination of c;
            endif
        endif
    endfor
    if |C V| > capacity of vehicle then
        return (cstnode)
    endif

    for each c in CN do
        if origin of c ∉ N then
            newldis = separation (cnode, origin of c, ctime);

```

```

        if minldis < newldis;
            minldis = newldis;
            cstnode = origin of c;
        endif
    endif
endfor
return (cstnode)
end closest

visit (gnode: node, ctime: time)
    marks the gnode as visited by updating global data
    and returns the updated time
ntime: time
begin
    for each c in C do
        if origin of c = gnode then
            CN = CN − {gnode};
            CV = CV ∪ {gnode};
            break
        endif
        if destination of c = gnode then
            CV = CV − {gnode};
            CS = CS ∪ {gnode};
            break
        endif
    endfor
    return;
end visit

unvisit (gnode: node)
    marks the gnode as not visited by updating global data
begin
    for each c in C do
        if origin of c = gnode then
            CN = CN ∪ {gnode};
            CV = CV − {gnode};
            break
        endif
        if destination of c = gnode then

```

```

     $CV = CV \cup \{gnode\};$ 
     $CS = CS - \{gnode\};$ 
    break
  endif
endfor
return;
end unvisit

cost (newnode, curnode: node, curtime: time)
  returns cost of visiting nnode
VN: set of nodes
nnode, cnode: node
ttime, ntime, ctime: time;
totcost: integer;
begin
  cnode = curnode;
  ctime = curtime;
  nnode = newnode;
  totcost = 0;
  for each  $i$  in 1 ... 4 do
    totcost = totcost + movecost (cnode, nnode, ctime);
    visit(nnode);
     $VN = VN \cup \{nnode\};$ 
    ttime = distance from cnode to nnode divided by speed;
    ctime = ctime + ttime;
    if ctime < earliest time window at nnode then
      ctime = earliest time window at nnode;
    endif
    cnode = nnode;
    nnode = closest (cnode, ctime,  $\emptyset$ );
    if nnode = -1 then
      break;
    endif
  endfor

  for each  $n$  in VN do
    unvisit (n);
  end for

```



```

return (totcost);
end cost

```

```

route ( )

```

```

    generates route for the set of customers C

```

```

    N4: set of nodes;

```

```

    firstcus: customer;

```

```

    cnode, nnode: node;

```

```

    ctime: time;

```

```

    mincost: integer;

```

```

begin

```

```

    time =  $\infty$ 

```

```

    for each c in C do

```

```

        if earliest pick-up time of c < time do

```

```

            time = earliest pick-up time of c;

```

```

            firstcus = c;

```

```

        endif

```

```

    endfor

```

```

    CS =  $\emptyset$ ;

```

```

    CV = { firstcus };

```

```

    CN = C - CV;

```

```

    while CN  $\neq \emptyset$  do

```

```

        N4 =  $\emptyset$ ; for each i in 1 ... 4 do

```

```

            nnode = closest (cnode, ctime, N4);

```

```

            if nnode = -1 then

```

```

                break;

```

```

            else

```

```

                N4 = N4  $\cup$  { nnode };

```

```

            endif

```

```

    mincost =  $\infty$ 

```

```

    for each n in N4 do

```

```

        if mincost < cost (n, cnode, ctime) then

```

```

            mincost = cost (n, cnode, ctime);

```

```

            nnode = n

```

```

        endif

```

```

    endfor

```

```

    visit (nnode);

```

```
ttime = distance from cnode to nnode divided by speed;  
ctime = ctime + ttime;  
if ctime < earliest time window at nnode then  
    ctime = earliest time window at nnode;  
endif  
cnode = nnode;  
endwhile  
endroute
```