ORSA Journal on Computing
Vol. 7, No. 1, Winter 1995

0899-1499/95/0701-0010 $01.25

A GRASP for the Vehicle Routing Problem with Time Windows

GEORGE KONTORAVDIS / *Arrowsmith Technologies, Inc., Austin, TX 78705; Email: condor@acm.org*

JONATHAN F. BARD / *Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063; Email: jbard@utxvm.cc.utexas.edu*

(Received: October 1992; revised October 1993; accepted: September 1994)

This paper addresses the problem of finding the minimum number of vehicles required to visit a set of nodes subject to time window constraints. A secondary objective is to minimize the total distance traveled. Each node requires a predetermined amount of service in the form of pickups and deliveries. The fleet is homogeneous and is located at a common depot. Vehicle capacity is finite and split service is not permitted. A greedy randomized adaptive search procedure (GRASP) is used to obtain feasible solutions. Results are reported for standard 100 node data sets as well as for a number of real-world problems with up to 417 customers. The findings indicate that, in general, the proposed procedure outperforms current techniques and requires only a small fraction of the time taken by exact methods. To gauge the quality of the solutions, three different lower bounding heuristics were developed. The first considers the "bin packing" aspect of the problem with regard to vehicle capacity; the second is based on the maximum clique associated with the customers' incompatibility graph; the third independently exploits the time window constraints. Using these heuristics, it is empirically demonstrated that the optimum is found in almost all cases involving balanced service and tight capacity constraints and a significant proportion of the remainder.

The vehicle routing problem with time windows (VRPTW) can be defined as follows. Let $G = (V, E)$ be a connected digraph consisting of a set of $n + 1$ nodes each of which can be reached only within a specified time interval or time window, and a set E of nonnegatively weighted arcs with associated traveling times. Let one of the nodes be designated as the depot. Each node i from the remaining n imposes a service requirement q_i which can be a delivery from, or a pickup for the depot. The problem is to find the minimum number of tours, K^*, such that each node is reached within its time window and the accumulated service up to any node does not exceed a positive number Q (vehicle capacity). A secondary objective is to minimize the total distance traveled. When the cost of service is not linearly proportional to the distance traveled, an alternative objective might be to minimize total cost.

All problem parameters, such as customer demand and time windows, are assumed to be known with certainty. Moreover, each customer must be served by exactly one vehicle, thus prohibiting split service and multiple visits. The tours, as defined in the problem statement, correspond to feasible routes starting and ending at the depot. Because of the time component and the two different types of service, the routes are directed so that tour orientation is important. Any load that is picked up along a route must be taken to the depot. Similarly, any load to be delivered must be on the vehicle when it leaves the depot. Customers may require both pickups and deliveries. Once the minimum number of tours K^* is determined the number of vehicles required is at most K^* since one vehicle is needed for each route in the worst case. If two routes r_i and r_j are time disjoint then the same vehicle can handle both. In what follows we do not consider this case since it does not arise often in practice; therefore, the terms *routes* and *vehicles* are used interchangeably.

The VRPTW is encountered in many practical situations. Bank deliveries, postal deliveries, dial-a-ride service, school bus routing, and vendor deliveries for just-in-time manufacturing are a few such examples. The importance of the temporal constraints varies depending on the particular application. In some cases, such as regular postal deliveries, violation of the time window constraints may cause only minor customer dissatisfaction, whereas late deliveries in a just-in-time production system may trigger large penalties and jeopardize future contracts.

The VRPTW consists of two subproblems. If the time window constraints are relaxed then it reduces to a bin packing problem. If the capacity constraints are relaxed then it results in a multiprocessor scheduling problem with individual setup costs, release times and deadlines for each job. It is evident that VRPTW is *NP-complete* since both of its subproblems are *NP-complete* (see Garey and Johnson[13]).

This paper presents a greedy randomized adaptive search procedure (GRASP) for solving VRPTW. The results demonstrate measurable improvement over existing techniques, and, for certain instances involving pickups and deliveries, are almost always optimal. Three lower bounding procedures are introduced to assess the quality of the GRASP. Two of them are based on the bin packing problem and can be efficiently calculated. The third is derived from the maximum clique of a generally sparse graph. In the remainder of this section, we outline the general GRASP

Subject classifications: vehicle routing, time windows, heuristics, pickup and delivery, lower bounds.

framework for solving combinatorial optimization problems and state how it is applied to the VRPTW. GRASP has been successfully used to solve a variety of related problems including machine scheduling (Feo et al.[12]) and set covering (Feo and Resende[11]).

Greedy randomized adaptive search procedures combine greedy heuristics, randomization, and local search. The computations are performed in two steps or phases. The first step includes the sequential construction of feasible solutions. At each iteration, all feasible moves are ranked according to an adaptive greedy function that takes into account the present state and one is randomly selected from a restricted candidate list. In the second step, local search is used to arrive at a local optimum. Because the whole procedure is repeated several times, an efficient construction phase is imperative. Moreover, the inherent complexity of local search suggests that it may be preferable to construct a number of feasible solutions and then apply the local search to the most promising.

GRASP is most successful when the construction phase produces feasible solutions that are close to a local optimum. This leads to the fundamental difference in philosophy between GRASP and other metaheuristics such as tabu search and simulating annealing. For GRASP to work well, it is essential that high quality solutions be constructed during the first phase of the algorithm. A large number of feasible solutions are generated, with the most promising carried over to the second phase. By way of contrast, tabu search and simulated annealing do not require high quality initial feasible solutions. They spend virtually all of their time improving the incumbent and trying to surmount local optimality (see Klincewicz[15]; Laguna and Velarde[17] for a comparison of the different methodologies).

Randomization serves as the search diversification mechanism in GRASP. The candidate list of moves is usually taken to be small to preserve the greedy nature of the algorithm. It has been observed that a candidate list with as few as two or three elements is adequate to produce completely different solutions. The main advantage of using GRASP over other heuristics is that it generates many good alternatives. This is particularly important for vehicle routing applications where travel times are often estimates and may vary widely depending on such unpredictable factors as weather and traffic congestion.

The rest of the paper is organized as follows. A brief literature review is presented in Section 1. Notation, feasibility conditions, and the heuristic are presented in Section 2. Section 3 describes our three lower bounding procedures. Computational experience is highlighted in Section 4. We conclude with a discussion on the results and suggestions for future work.

1. Related Work

In recent years there has been a growing interest in the VRPTW. An extensive survey on vehicle routing and scheduling has been done by Bodin et al.[3] A more specific survey on routing problems with temporal constraints was undertaken by Solomon and Desrosiers[28] and has been updated by Desrosiers et al.[10] A concise overview of optimization and heuristic methods for the VRPTW can be found in Desrochers et al.[7] Christofides et al.[5] present a combination of branch-and-bound and dynamic programming state space relaxation to solve the traveling salesman problem with time windows (TSPTW). Baker[1] also studied the TSPTW and presented a compact mathematical programming model containing continuous variables only. However, his formulation includes nonconvex absolute value function constraints that preclude the use of linear programming techniques. Desrosiers et al.[9] modeled the same problem as a minimum cost network flow problem with side time constraints. Their method successfully solved a number of school bus scheduling problems with up to 233 nodes.

Psaraftis[22] studied the single vehicle many-to-many dial-a-ride problem, where many-to-many refers to distinct pickup and delivery points. He used dynamic programming to minimize a measure of customer dissatisfaction accounting for the total time a customer spends in the vehicle. Problems with up to 9 customers were solved optimally. Desrosiers et al.[8] introduced a forward dynamic programming formulation for the same problem and solved real applications with up to 40 customers. They concluded that their method is robust enough to solve much larger problems, since real-life problems tend to have tight time windows which greatly reduce the state space. Sexton and Bodin[24] worked on the many-to-many dial-a-ride problem with time-constrained delivery. They developed a heuristic based on Benders decomposition to find feasible routes.

Nygard et al.[19] addressed the VRPTW with one-sided time windows (deadlines only). Customers are initially clustered by solving a generalized assignment problem. Subsequently, optimal tours are calculated for each cluster and finally a branch exchange procedure is applied to ensure that the temporal constraints are satisfied. Kolen et al.[16] studied the VRPTW with one depot and one service type (pickup or delivery, exclusively). They used an exact branch-and-bound scheme to minimize the total route length, and solved problems with up to 15 customers for a number of different time window settings. Despite the small problem sizes, they point out that the relative width and frequency of the time windows is the most important factor in determining the difficulty of the problem. Neither the vehicle capacity nor a fixed number of vehicles have anywhere near the impact of time windows. Desrochers et al.[6] addressed the same problem and used column generation to solve a linear programming relaxation of a set partitioning formulation. New columns are generated at each iteration by solving a shortest path problem with time windows. They optimally solved problems with up to 100 customers.

Past results indicate that the best available exact methods can solve only relatively small problems that generally are too small to have practical application. Heuristics that find good feasible solutions in reasonable time offer an attractive alternative. Solomon[26] proposed several sequential insertion heuristics for the fleet size minimization of the VRPTW with one type of service. He also showed that the

worst case behavior of those heuristics is $\Omega(n)$, where n is the number of customers. This means that in the worst case the ratio of the heuristic solution to the optimal solution grows proportionally with the number of customers. Thangiah et al.[29] proposed a genetic algorithm for the VRPTW, while Potvin and Rousseau[21] developed a parallel version of Solomon's insertion heuristics. Potvin et al.[20] developed a tabu search scheme based on a specialized exchange algorithm to improve solutions obtained by Solomon's heuristics. Baker and Schaffer[2] studied local search procedures for the VRPTW by modifying 2-opt and 3-opt branch exchanges for the standard VRP to account for vehicle capacity and time windows. Solomon et al.[27] and more recently Savelsbergh[23] worked on reducing the complexity associated with edge exchanges for the VRPTW.

2. Heuristic

Heuristics for solving the basic vehicle routing problem can be divided into two broad categories: sequential and parallel. The former build one tour at a time until all customers are routed; the latter build multiple tours simultaneously. The number of routes is either specified in advance or is determined by the heuristic. Previous work suggests that parallel heuristics are superior, in general, because they are less myopic in deciding customer routing assignments.[21] In this paper we consider parallel heuristics with free fleet size.

2.1. Notation and Assumptions

Let each customer be denoted by a number i ($i = 1, \ldots, n$), with the depot indexed as 0. For easier reference we define the following sets: $I = \{1, \ldots, n\}$ is the set of customers, $I_0 = I \cup \{0\}$ is the set of all customers and the depot; $I_p = \{i \in I: i \text{ requires a pickup}\}$ and $I_d = \{i \in I: i \text{ requires a delivery}\}$. For each customer i, let q_i and $[a_i, b_i]$ be his or her demand and time window, respectively. If a particular customer requires both types of service then it is replaced with two artificial customers, one having demand for pickup and the other for delivery. Let σ_i be the time required to provide customer i with service. The time that a vehicle departs from customer i is denoted by t_i and is a variable to be determined by the heuristic. If a vehicle arrives at a node before the customer's operating time window it has to wait. The travel time between i and j ($i, j \in I_0$) is denoted by τ_{ij}. We assume that customers and the depot are located on the plane and that the distances between them are Euclidean. This implies that the triangle inequality holds between any three locations. We also assume that the vehicle velocity is 1 so travel time is equal to the distance traveled. Vehicle capacity is denoted by Q and all vehicles have the same capacity. For each $i \in I$ we define AC_i as the free capacity of the vehicle when it arrives at i, and DC_i as the free capacity at departure. Hence,

$$DC_i = \begin{cases} AC_i + q_i & \text{if } i \in I_d \\ AC_i - q_i & \text{if } i \in I_p \end{cases}$$

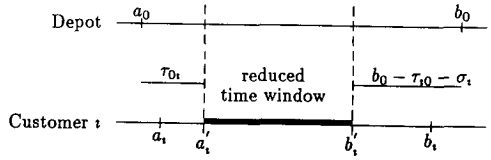


Figure 1. Time window reduction.

Also, let P_i be the set of customers in the same route as i preceding i, and let S_i be the set of customers in the same route as i succeeding i. The immediate predecessor (successor) of i is denoted by $p_i(s_i)$.

To simplify the analysis it is assumed that the depot does not impose a time window and that the service time is 0 for all customers. These assumptions do not restrict the generality of the problem considered, since a VRPTW instance π with depot time window $[a_0, b_0]$ and $\sigma \neq 0$ for $i \in I$ can be transformed into an equivalent instance π' without a depot time window and 0 service time for all customers. A straightforward transformation is all that is necessary. In particular, define the new travel times between i and j ($i, j \in I_0$) as $\tau'_{ij} = \tau ij + \sigma_i$ and the new time window for customer $i \in I$ as $[a'_i, b'_i] = [\max(a_i, a0 + \tau_{0i}), \min(b_i, b_0 - \tau_{i0} - \sigma_i)]$. The corresponding time window reduction is depicted in Figure 1.

2.2. GRASP

In our heuristic, construction of feasible solutions is performed by first initializing a number of routes, r, and then iteratively assigning one customer at a time in accordance with an adaptive set of rules. If at some step no unassigned customer can be inserted in an existing route a new route is created. The number of initial routes can be anywhere from 1 to n. For instance, Potvin and Rousseau use one of Solomon's sequential heuristics in order to get a good estimate. We follow a different approach by setting r to a predetermined lower bound. This is preferable since we never introduce more routes than necessary, thus reducing the effort required in the improvement phase to remove surplus routes.

To initialize the routes, seed customers have to be selected and assigned. Solomon[26] proposes two criteria for doing this: (1) maximum distance from the depot; or (2) earliest deadline. Alternatively, we use an approach that selects seed customers that are either the most geographically dispersed or the most time constrained. The additional effort required in these computations is justified by preliminary results which showed that seed selection is a major determinant in the effectiveness of the GRASP. Two lists, L_{CH} and $L_{\overline{CH}}$ are used in the initialization process. The first, L_{CH}, contains customers that define the convex hull of the customer locations; the second, $L_{\overline{CH}}$ contains the remaining customers. That is, $L_{\overline{CH}} = I \setminus L_{CH}$. The elements in $L_{\overline{CH}}$ are arranged in ascending order of $b_i - \tau_{0i}$, denoted by $slack_i$. Small values of $slack_i$ show that there is little flexibility in assigning customer i since the closing time of his time window is close to the time required to

travel from the depot to i. The complete initialization process is outlined in Figure 2.

In the first two steps, the convex hull of the customer locations is found, L_{CH} and $L_{\overline{CH}}$ are constructed, and the set, S, of seed customers is initialized. The farthest customer from the depot is designated as the first seed customer (Step 3). Subsequent seeds are selected one at a time, so that they are as geographically dispersed as possible. At each iteration the element $j \in L_{CH}$ that maximizes the sum of distances between j and the existing seeds is found and the minimum distance, d_j, is calculated (Step 4c). Similarly, the minimum distance, d_i, between the first element i of $L_{\overline{CH}}$ and the existing seeds is calculated (Step 4d). Given i and j, the one with the largest minimum distance from existing seeds is selected to become the next seed. Once a customer is designated as a seed a new route is constructed and S, and either L_{CH} or $L_{\overline{CH}}$ (depending on which set the customer belongs to), are updated (Step 4a and 4b). The procedure is designed to reduce the possibility of selecting two seed customers that are likely to be in the same route in an optimal solution.

Phase I. After initialization the main routing algorithm is called (see Figure 3). Let η_{ij} denote a vehicle move from i to j in a feasible route and let H_ρ denote the set of vehicle moves (or arcs) in route ρ. Also define Ω as the set of all routes in a partial or complete solution. The cost of inserting customer k between i and j in a partial solution is denoted by $c_{ij,k}$. Its calculation is described in Section 2.3.

During construction, we first find the best feasible insertion location in each route for every unassigned customer k. Then, a penalty (opportunity) cost is calculated which is a measure of the cost that would have to be paid later if the corresponding customer is not assigned to its current best position. Two approaches were tried. The first is the difference between the costs associated with the best and the second best insertions; i.e.,

$$\Pi_k = \min_{\rho \in \Omega \setminus \{\rho^*\}} \{c_{\rho,k}\} - c_{\rho^*,k}.$$

This measure did not perform well because it is shortsighted. It only considers the best two alternatives without taking into account possible future infeasibilities. In the second approach we sum the difference between the least insertion cost for each route and the overall best cost. The corresponding penalty is

$$\Pi_k = \sum_{\rho \in \Omega} (c_{\rho,k} - c_{\rho^*,k}).$$

Example 1. Assume all customers, except k_1, k_2, and k_3, have been assigned to three routes ρ_1, ρ_2, and ρ_3, as depicted in Figure 4. The numbers in parentheses next to each customer represent the insertion costs for the three existing routes. For instance, the minimum costs for inserting k_2 into ρ_1, ρ_2, and ρ_3 are 8, 19, and 40, respectively. An infinite cost means that there is no feasible insertion point for that customer in the corresponding route. Now, due to capacity and/or time constraints assume that if either k_2 or k_3 is assigned to ρ_1 or ρ_2, then k_1 cannot be feasibly assigned to that route.

For simplicity also assume that the best route insertion costs given in Figure 4 do not change after k_1, k_2, or k_3 is assigned to a route. Using the first penalty measure we have $\Pi_{k_1} = 15 - 10 = 5$, $\Pi_{k_2} = 19 - 8 = 11$, and $\Pi_{k_3} = 25 - 13 = 12$. This leads to the following assignments: k_3 is inserted into route ρ_2, k_2 into ρ_1, and finally, k_1 requires

Procedure Seed Selection

Input: Set of unscheduled customers and number r of initial routes

Output: r routes with one seed customer each

Step 1. Construct L_{CH} and $L_{\overline{CH}}$ lists

Step 2. Set $S = \emptyset$

Step 3. Find customer *newseed* in L_{CH} that is the farthest from the depot

Step 4. **WHILE** $|S| < r$ **DO**

Step 4a. Set $S = S \cup \{newseed\}$

Step 4b. Create new route with initial customer *newseed* and designate *newseed* as assigned

Step 4c. Find $j \in L_{CH}$ that maximizes the sum of distances to elements in S
Let $d_j = \min_{k \in S}\{\tau_{jk}\}$

Step 4d. Let i be the first element in $L_{\overline{CH}}$ and let $d_i = \min_{k \in S}\{\tau_{ik}\}$

Step 4e **IF** $d_i \leq d_j$ **THEN**
Set $newseed = j$
ELSE
Set $newseed = i$

Figure 2. Route initialization—seed selection.

Procedure GRASP Routing

Input: Set of unscheduled customers and number r of initialized routes

Output: Feasible schedule for all n customers

Step 1. **FOR** each unassigned customer k and each route ρ **DO**

Find the minimum insertion cost: $c_{\rho,k} = \min_{\eta_{i,j} \in H_\rho} \{c_{ij,k}\}$

Step 2. **FOR** each unassigned customer k **DO**

Find the route ρ^* with the overall minimum insertion cost: $c_{\rho^*,k} = \min_{\rho \in \Omega} \{c_{\rho,k}\}$

Step 3. **FOR** each unassigned customer k **DO**

Find the penalty cost Π_k

Step 4. Find first unassigned customer k, if any, such that $\Pi_k = 0$ and k cannot be feasibly inserted in an existing route and construct a new route with k as seed customer

Go to Step 1

Step 5. Construct a list of unassigned customers with the λ largest Π_k's and randomly select one customer to be routed

IF there exist unassigned customers **THEN**

Go to Step 1

Figure 3. Routing heuristic.

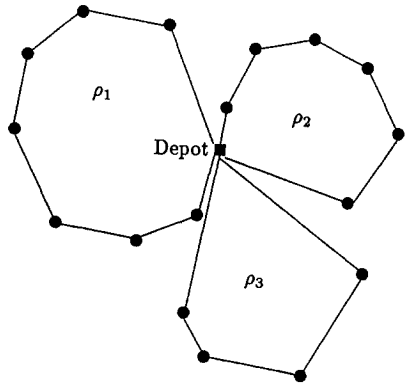


Figure 4. Partial routes for Example 2.

the introduction of a new route. In contrast, using the second penalty measure, $\Pi_{k1} = (15 - 10) + (\infty - 10) = \infty$, $\Pi_{k2} = (19 - 8) + (40 - 8) = 43$, and $\Pi_{k_3} = (25 - 13) + (27 - 13) = 26$. Therefore, the customer assignments are k_1 to ρ_1, k_3 to ρ_2, and k_2 to ρ_1. □

Large penalty values indicate that the associated customers should be considered first to avoid paying a relatively high cost later. Conversely, customers with small penalty values can wait for insertion since the delay is not likely to result in a significant deterioration in the solution. If a customer cannot be inserted in some route ρ, then $c_{p,k}$ is set to infinity. This forces Π_k to assume a large value as well, indicating that k may require the introduction of a new route at a later point. In this way, potential infeasibilities are anticipated and avoided, to the extent possible. Notice that if an unassigned customer k cannot be feasibly inserted into any of the existing routes then, Π_k is zero. This case is handled by introducing a new route.

The number of times that Steps 1 through 5 are repeated is $O(n)$. The computational effort for Step 1 is $O(n^2)$, for Steps 2, 3 and 4 $O(n)$, and for Step 5 $\max\{O(n), O(\lambda \log \lambda)\}$. Therefore, the overall complexity of GRASP is $O(n^3)$. This is the worst case scenario but, in practice, it is much faster. For instance, Step 1 can be accelerated by storing the $c_{p,k}$'s and only recalculating those for the route (call it ρ') containing the most recently inserted customer. This results in $O(|\rho'|n)$ complexity which is almost linear for tightly constrained problems, since each route in this case would only contain a small number of customers.

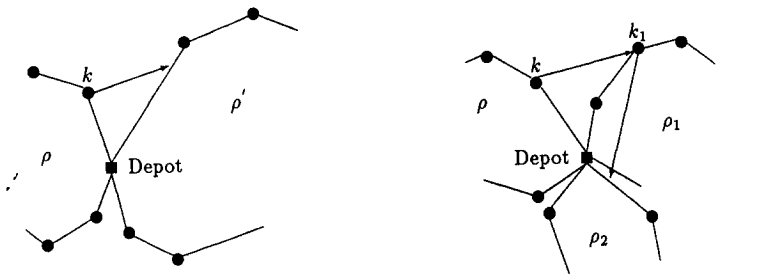
The use of a candidate list (Step 5) of unassigned customers introduces a probabilistic component to tour construction. The length λ of the candidate list is a parameter that needs to be determined. From our computational study, a value of $\lambda = 3$ performed best. It has been observed that as λ increases the number of GRASP iterations needed to obtain quality solutions increases as well, which is no surprise since the larger the value of λ the more the greedy construction is compromised.

Phase II. As described in Section 1, the construction of a feasible solution is followed by a local search in an attempt to find a local optimum. Due to its high computational requirements, local search is applied to the best solution found after every five iterations of *Phase I* rather than to each feasible solution. During *Phase II* each route ρ is considered for elimination, with routes having fewer customers examined first. The algorithm in Figure 5 is employed for this purpose. In the presentation, let ρ be either a route or a set of customers in a route, depending on the context.

In order to eliminate route ρ all of its customers must be assigned to other routes. We consider two cases (see Figure 6). In the first case, we attempt to move k from its current

Procedure Eliminate
While eliminating a route **DO**
  FOR each route ρ **DO**
    FOR each customer $k \in \rho$ **DO**
      Case 1: **IF** k can be inserted in a route ρ' **THEN**
        move k to ρ'
      Case 2: **IF** k can replace $k_1 \in \rho_1 \neq \rho$ and k_1 can be moved to a route ρ_2 **THEN**
        replace k_1 with k in ρ_1 and move k_1 to ρ_2

Figure 5. Route elimination procedure.



Case 1: move k to ρ' Case 2: replace k_1 with k, move k_1 to ρ_2

Figure 6. Illustration of *Procedure Eliminate*.

route ρ to another route ρ'. If more than one alternative exists two possible ways of selecting k's new route are considered. The first and simplest is to select the first route found; the second is to select the route that provides the greatest savings in distance. Although both alternatives have the same worst case complexity, in practice, the former is measurably faster and was used in our implementation. The drawback is that the distance traveled may be increased. To compensate, a 2-opt exchange procedure is applied with the objective of minimizing the total distance traveled for the current number of vehicles.

If it is not possible to insert k into another route, then the second elimination case is considered. Here, k replaces some other customer $k_1 \in \rho_1 \neq \rho$, and k_1 is moved to another route $\rho_2 \neq \rho$. To reduce the effort required to identify such a k_1, we examine the customers in decreasing order of $b_i - a_i/\tau_{0i}$. This ratio gives an estimate of the degree to which a particular customer is time-constrained. If $b_i - a_i/\tau_{0i}$ is large this indicates that customer i is relatively time-unconstrained, so in all likelihood, it can be assigned to another route without too much computational effort.

The complexity of case 1 is $O(n)$ and of case 2 $O(n^2)$ (see Section 2.4). To eliminate route ρ each customer in ρ is examined; therefore, the complexity of eliminating one route is $O(n^3)$. One iteration of the *while* loop in Figure 5 requires that all routes be considered. This loop is repeated as long as there is at least one route reduction. Let e be the number of routes eliminated. Then the overall worst case performance for *Procedure Eliminate* is $O(en^4)$.

Although the theoretical worst case scenario prefigures a very time consuming local search, we noticed during our computational study that the actual running time is very close to $O(n^3)$. This can be explained as follows. Given that the heuristic produces high quality solutions the number of routes, e, that can be eliminated is a small constant (one or two at most) unrelated to the number of customers n. Second, the average number of customers in a route and the number of routes are inversely proportional. This means that the more routes we have the fewer customers are assigned to a route. Therefore, the product between the number of routes and the number of customers in a route is closer to $O(n)$ than to $O(n^2)$.

Procedure Eliminate is totally deterministic. One extension would be to introduce some randomness and accept, with a certain probability, a customer move that leads to a deterioration in the current solution in an attempt to escape from a local optimum. This is a basic principle of simulating annealing. Another extension would be to allow, with a certain probability, some infeasible customer exchanges with the hope that feasibility would be regained at a later stage. This is the concept of strategic oscillation used in tabu search. However, since we would like to use local search on a large number of feasible solutions, and given that both simulated annealing and tabu search are computational intensive, it would be more expedient to apply

them, if so desired, only to the best solution produced by GRASP at the final stage of the optimization.

2.3. Cost Metric

Solomon[26] introduced a number of different metrics to evaluate the cost of inserting an unassigned customer into partially built routes. We outline the one that produced the best results and then discuss our modifications.

Let i and j be two consecutive customers (or the depot) in a partially built tour and let k be an unassigned customer. The cost $c_{ij,k}$ of inserting k between i and j is defined as

$$c_{ij,k} = \gamma_1(\tau_{ik} + \tau_{kj} - \mu\tau_{ij}) + \gamma_2\left(t_j^k - t_j\right) \qquad (1)$$

where γ_1, γ_2 are nonnegative weights such that $\gamma_1 + \gamma_2 = 1$, μ is a route shaping parameter, and t_j^k is the new visit time at j if k is inserted between i and j. The two terms in Eq. (1) measure distance detour and time delay, respectively, occasioned by the insertion of k. Computational experience has shown that this metric is best suited for problems with clustered customers and loose capacity constraints. In fact, the capacity constraint is not considered at all in evaluating a customer insertion. In order to introduce this constraint into the cost metric, we define $c_{ij,k}$ as follows:

$$c_{ij,k} = \delta_1 c^1_{ij,k} + \delta_2 c^2_{ij,k} + \delta_3 c^3_{ij,k}, \qquad (2)$$

where δ_1, δ_2, δ_3 are nonnegative weights such that $\delta_1 + \delta_2 + \delta_3 = 1$.

The first component $c^1_{ij,k}$ accounts for demand and is given by

$$c^1_{ij,k} = \begin{cases} MAC_i - q_k & \text{if } k \in I_d \\ MDC_i - q_k & \text{if } k \in I_p, \end{cases} \qquad (3)$$

where

$$MAC_i = \min_{m \in P_i \cup \{i\}} AC_m$$

$$MDC_i = \min_{m \in S_i \cup \{i\}} DC_m.$$

This function maps large demands into small costs so that customers with large loads are assigned to vehicles first. This is in line with Johnson's[18] best-fit decreasing algorithm for the bin packing problem which has a worst case performance ratio of $^{11}/_{9}$ (heuristic/optimum).

The second cost component $c^2_{ij,k}$ measures the distance increase due to the insertion of k; that is,

$$c^2_{ijk} = \tau_{ik} + \tau_{kj} - \tau_{ij}. \qquad (4)$$

The third cost term $c^3_{ij,k}$ is the difference between t_j^k and t_j, and represents the amount that the schedule has to be pushed forward to accommodate k. A nonnegative $c^3_{ij,k}$ may require that t_m, $m \in S_j$, be increased making future customer insertions less likely. By definition,

$$c^3_{ij,k} = t_j^k - t_j. \qquad (5)$$

Since t_j^k is the new visit time at j, it implicitly takes into account the opening time of k's time window as well as the additional distance introduced by visiting k. That is, $t_k = \max\{t_i + \tau_{ik}, a_k\}$ and $t_j^k = \max\{t_k + \tau_{kj}, a_j\}$.

Some explanation of the complementary behavior of $c^2_{ij,k}$ and $c^3_{ij,k}$ is needed. These terms actually offset one another in an effort to find a good compromise between distance increase versus long route waiting time. Consider the following scenario: k is close to i and j, but his time window is such that $a_k + \tau_{ij} \gg t_j$. Thus, $c^2_{ij,k}$ is small implying that routing k causes a small distance increase. On the other hand, $c^3_{ij,k}$ increases introducing a long waiting time that may prevent more customers to be added later in the route. The combination of $c^2_{ij,k}$ and $c^3_{ij,k}$ results in assigning a small insertion cost to customers that impose small distance increases without introducing long waiting times in the existing schedule.

2.4. Insertion Feasibility Conditions

Let us consider a feasible tour containing customer i. Although the closing time for i is b_i, i may have to be visited earlier than b_i in order to preserve tour feasibility. Let lt_i be the latest time that i can be served given a particular feasible route. Then lt_i can be calculated using the following recursive relation. Recall that s_i is the immediate successor of i.

$$lt_i + \tau_{is_i} \leqslant lt_{s_i} \qquad (6)$$

Note that lt_i depends on the route under consideration and can range between a_i and b_i. Given a feasible route we can determine whether a new customer can be inserted into the route without violating the time window constraints by using (6) to calculate how much the current schedule can be pushed forward or backward. In fact, by starting from the depot at time 0 and by visiting customers as early as possible we do not need to consider the case of pushing a schedule backwards (visiting customers earlier). Using Eq. (6) we define the necessary and sufficient conditions for time feasibility associated with inserting customer k between i and j in a route as follows:

$$t_i + \tau_{ik} \leqslant b_k \qquad (7)$$

$$\max\{a_k, t_i + \tau_{ik}\} + \tau_{kj} \leqslant lt_j. \qquad (8)$$

Condition (7) simply states that the vehicle must arrive at k before the closing time b_k of the operating time window. Condition (8) is also true because $\max\{a_k, t_i + \tau_{ik}\}$ represents the departure time, t_k, from k and $t_k + \tau_{ik}$ is the new arrival time at j which cannot be greater than lt_j if feasibility is to be maintained.

As was noted earlier, we try to visit a customer as early as possible so that we do not need to consider pushing the schedule backwards. The feasibility test introduced by Eq. (7) and Eq. (8) can be performed in constant time given that we maintain the latest times accurately. The algorithm depicted in Figure 7 updates the lt values after inserting customer k between i and j. This procedure starts at a newly inserted customer k and moves toward the begin-

Procedure Update lt

Step 1. Set $z = k$

Step 2. **WHILE** $z \neq 0$ **DO**

Step 2a. Set $lt_z = min\{b_z, lt_{s_z} - \tau_{zs_z}\}$

Step 2b. Set $z = p_z$

Figure 7. Procedure for updating the latest arrival times.

ning of the tour to update the latest allowable times of service. Note that lt_z for all $z \in S_k$ are not affected. The number of times that we may need to go through the *while* loop is equal to the cardinality of P_k, which is $O(n)$. However, this is a conservative upper bound since it depends on how many customers precede k. The procedure may be accelerated by prematurely terminating the *while* loop if for some $z \in P_k$, the new lt_z does not change. In this case, Eq. (6) guarantees that the latest times that the predecessors of z can be served are up-to-date.

The necessary and sufficient condition for capacity feasibility when inserting customer k between i and j in a route is

$$q_k \leq \begin{cases} MAC_j \text{ if } k \in I_d \\ MDC_i \text{ if } k \in I_p \end{cases} \tag{9}$$

If customer k needs a delivery ($k \in I_d$) then condition (9) requires that the minimum vehicle residual capacity from the depot up to j in the current route be at least q_k. Similarly, if $k \in I_p$ then the minimum vehicle residual capacity from i to the end of the route must be at least q_k.

The feasibility test given by Eq. (9) takes constant time to perform in that MAC_i and MDC_i are available for each assigned customer i. The algorithm in Figure 8 presents the general steps needed to update the MAC and MDC values after inserting customer k between i and j. Only the case where customer k has a pickup is presented. The delivery case is similar. To update the MAC values, procedure *Update MAC and MDC* starts at j and moves forward. The MDC values are updated by starting at the end of the route and moving toward the beginning. The number of times that each *while* loop is executed is proportional to the number of customers in the route under consideration, which is $O(n)$ in the worst case.

3. Lower Bounds

In this section, we present three different lower bounds on the number of vehicles. To evaluate the quality of a lower bound we define the *worst case performance ratio* as the minimum value of *lower bound/optimal solution* for any instance of the problem.

The first lower bound is obtained by considering the vehicle capacity constraints only and using any exact or valid lower bounding procedure for the resulting bin packing problem. Since obtaining an exact solution to this problem may be computationally burdensome we derive an alternative lower bound using Martello and Toth's algorithm.[18] Let β_S be a bin packing lower bound for the set

Procedure Update MAC **and** MDC

Step 1. Initialize MAC_k and MDC_k

Step 2. **IF** $k \in I_p$ **THEN**

Step 2a. Set $z = j$

Step 2b. **WHILE** $z \neq 0$ **DO**

Step 2b.1. Set $MAC_z = min\{AC_z, MAC_{p_z}\}$

Step 2b.2. Set $z = s_z$

Step 2c. Set $z =$ last customer in the route

Step 2d. **WHILE** $z \neq 0$ **DO**

Step 2d.1. Set $MDC_z = min\{DC_z, MDC_{s_z}\}$

Step 2d.2. Set $z = p_z$

Figure 8. Procedure for updating the minimum arrival and departure vehicle capacity.

of customers $S \in I$. Then a lower bound on the number of vehicles is

$$LB_1 = \max\{\beta_{I_d}, \beta_{I_p}\} \tag{10}$$

Obviously, LB_1 can be arbitrarily bad. Consider the following instance where $q_i = 1$, $a_i = b_i = \tau_{0i}$ $\forall i \in I$, and $Q \geq n$. Due to the time windows the optimal solution requires n vehicles, one for each customer, whereas $LB_1 = 1$. Therefore, the worst case performance ratio is $1/n$.

Before describing the second lower bound let us first define an incompatible pair of customers. Customers i and j are incompatible if they cannot be in the same route due to capacity or time window constraints. Any one of the following conditions identifies such a situation.

1) $i, j \in I_p$ and $q_i + q_j > Q$, or $i, j \in I_d$ and $q_i + q_j > Q$: Both i and j require the same type of service but the sum of their demands exceeds the vehicle capacity.
2) $a_i + \tau_{ij} > b_j$ and $a_j + \tau_{ji} > b_i$: Due to the time window constraint the vehicle serving i cannot serve j, and vice versa.
3) $i \in I_d$, $j \in I_p$, $a_i + \tau_{ij} > b_j$ and $q_i + q_j > Q$: The time window constraints require that i follow j if both are in the same route. However, this is infeasible since the sum of their demands is more than the vehicle capacity.

Using these conditions we construct the graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and $E = \{(i, j)$: i and j incompatible$\}$. Then any clique of G is a valid lower bound LB_2 on the number of vehicles, since each pair of clique elements corresponds to a pair of incompatible customers. In our computational study, we used an exact method developed by Carraghan and Pardalos[4] to find the maximum clique. For tightly time-constrained problems, LB_2 has proven to be very close, if not equal, to the optimal solution. However, for problems with wide time windows it can be arbitrarily bad. Consider the following instance: $a_i + \tau_{ij} \leq b_j$ and/or $a_j + \tau_{ji} \leq b_i$, $q_i = Q/2$ for all $i, j \in I$, and all customers require pickup service. Then LB_2 is 1 whereas the optimal solution is at least $\lceil n/2 \rceil$.

A third lower bound can be obtained by considering the time that a vehicle has to travel and wait in order to visit each customer i in an optimal schedule. Because this time

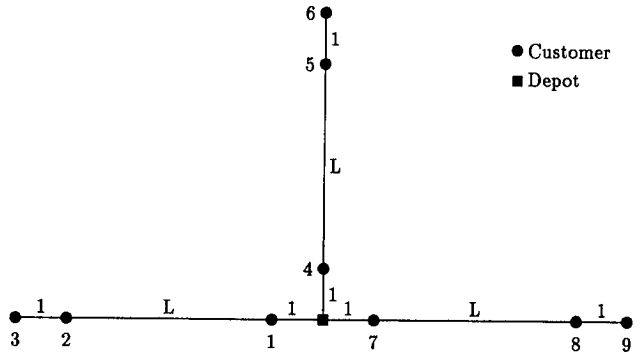


Figure 9. Example of $n = 3\theta$ customer problem ($\theta = 3$).

depends on the actual sequence which is not known, another time ϕ is used instead, where

$$\phi_i = \min_{j \in I_0} \{\max(\tau_{ij}, a_j - b_i - \tau_{ij})\}. \tag{11}$$

The minimization in Eq. (11) determines the amount of time needed to go from i to its closest neighbor or the depot. Therefore ϕ_i is never more than the actual vehicle time assigned to i. To account for the travel time from the depot to the first customer in each route define $lb \equiv \max\{LB_1, LB_2\}$ and let $\phi_{d_1}, \ldots, \phi_{d_{lb}}$ be the floor values of the lb least travel times from the depot to customers. Also define $T \equiv \lceil b_0 \rceil$. Then a lower bound LB_3 on the number of vehicles is the minimum number of bins of capacity T for the following bin packing instance: $n + lb$ items of size $\phi_1, \ldots, \phi_n, \phi_{d_1}, \ldots, \phi_{d_{lb}}$. Note that in order to calculate LB_3 the depot time window is used. In the absence of an explicit depot time window one can be determined by the following equation

$$b_0 = \max_{i \in I} \{b_i + \tau_{i0}\}, \tag{12}$$

where τ_{i0} includes the service time at i, as discussed in Section 2.1.

To determine the worst case behavior of LB_3 consider the instance given in Figure 9. Assume that there are θ triplets of customers forming a star around the depot. The customers $k, k+1, k+2$ $(k = 1, 4, \ldots, 3\theta - 2)$ of each triplet are all located on a line such that $\tau_{0k} = 1, \tau_{k,k+1} = L$ and $\tau_{k+1,k+2} = 1$. The service time and demand is 0 and 1 (pickup), respectively, for each customer. The vehicle capacity is at least n units. The time window is $a_i = b_i = \tau_{0i}$, for all the customers and $a_0 = 0$, $b_0 = 2L + 4$ for the depot. Due to the time window constraints the optimal solution requires $n/3$, or θ vehicles, one for each triplet. The first lower bound yields $LB_1 = 1$ and the second $LB_2 = n/3$. In order to define the associated bin packing problem we use Eq. (11) to obtain $\phi_i = 1$ for all customers. Also $\phi_{d_1} = \cdots = \phi_{d_{n/3}} = 1$. Therefore, LB_3 is the optimum solution to the bin packing problem with $4n/3$ items of size 1 and bin capacity $T = b_0 = 2L + 4$. For $L \geq 2n - 6/3$, LB_3 is 1 while the optimum solution is $n/3$ resulting in a worst case performance ratio of $3/n$.

Table I. Customer Data for Example 2

Customer	Location	Demand	Demand Type	Time Window
0	(0, 0)	0	—	[0, 16]
1	(2, 0)	3	pickup	[3, 5]
2	(0, 1)	6	pickup	[2, 3]
3	(3, 0)	1	pickup	[4, 5]
4	(0, 6)	5	delivery	[6, 8]
5	(4, 0)	2	delivery	[7, 8]

Table II. Travel Time Matrix for Example 2

	0	1	2	3	4	5
0	0	2	1	3	6	4
1	2	0	∞	1	∞	2
2	1	2.37	0	∞	∞	4.12
3	3	1	∞	0	∞	1
4	6	∞	∞	∞	0	7.21
5	4	∞	∞	∞	∞	0

The following example is used to illustrate the proposed lower bounding procedures.

Example 2. Consider the instance with 5 customers given by Table I with customer 0 denoting the depot. Assume that service time (σ_i) is zero for all customers. The vehicle capacity is 10 units. Table II gives the travel times between any pair of customers. If the element (i, j) is ∞ there is no feasible path from i to j due to time window and/or capacity constraints. For instance, entry (2, 3) is ∞ because the earliest time that 3 can be visited after 2 is

$$\begin{aligned}\max\{a_0 + \tau_{02}, a_2\} + \tau_{23} &= \max\{0 + 2, 2\} + \sqrt{(3-0)^2 + (0-1)^2} \\ &= 2 + 3.16 = 5.16\end{aligned}$$

which is past the closing time of customer 3.

From Table II we have $I_d = \{4, 5\}$ and $I_p = \{1, 2, 3\}$. Lower bounds for the corresponding bin packing problems are $\beta_{I_d} = 1$ and $\beta_{I_p} = 1$, therefore $LB_1 = 1$. The distances given in Table II determine the incompatible pairs of customers; the corresponding undirected graph is shown in Figure 10. The maximum clique of this graph is 3 (2, 3, 4), implying that $LB_2 = 3$.

To find the third lower bound we first calculate the value of ϕ_i for all customers using Eq. (11): $\phi_1 = 1$, $\phi_2 = 1$, $\phi_3 = 1$, $\phi_4 = 6$, and $\phi_5 = 4$. To account for travel time from the depot to the customers we calculate $\phi_{d_1} = 2$, $\phi_{d_2} = 1$, and $\phi_{d_3} = 3$. Then the minimum number of bins of size 16 ($= b_0$) required to pack items of size 1, 1, 1, 6, 4, 1, 2, 3 is a lower bound on the number of vehicles. Therefore, $LB_3 = 2$.

4. Computational Experience

The heuristic was tested on Solomon's data sets and four large industry-based problems. The former consist of six data sets (R1, C1, RC1, R2, C2, RC2), each of which contains between eight and twelve 100-node problems over a service area defined on a 100 × 100 grid. For R1 and R2, the customer locations were generated uniformly over the service area. Sets C1 and C2 have clustered customers, and sets RC1 and RC2 have a combination of clustered and randomly placed customers. In addition, R1, C1, and RC1 have tight time windows and a vehicle capacity of 200 units; R2, C2, and RC2 have a long scheduling horizon and vehicle capacity of 1000, 700, and 1000 units, respectively. In all, there are 56 problem instances. For a complete description of the generation procedure refer to the original paper.[26] Time window and the vehicle capacity constraints in problem sets R1, C1, and RC1 allow only a small number of customers to be served by each vehicle. The opposite is true for R2, C2, and RC2.

The industry-based problems were obtained from Robert Russell at the University of Tulsa and contain two instances (T1, T2) with 417 customers and two instances (T3, T4) with 219 customers. Problems T1 and T2 are highly constrained with vehicle capacity equal to 2000 units. In contrast, T3 and T4 have a long scheduling horizon and vehicle capacity of 100 and 50 units, respectively. Customer locations in all four data sets appear to be distributed randomly with a few outliers; the time windows do not depend on the distance from the depot.

All data sets are Euclidean with a single type of service (pickup). All the computations were carried out using real arithmetic without rounding off any distances beforehand. The GRASP and lower bounding procedures were implemented in C and run on a SUN SPARC-10 workstation. Because it is difficult to compare computation times across different hardware platforms, we only report our computational requirements along with those of Potvin et al.[20] who used a similar workstation. The quality of the solution is measured in terms of the minimum number of vehicles and the minimum distance traveled, in that order. This means that a solution with fewer vehicles and longer distance is preferable to one with more vehicles and shorter distance. Waiting time was not considered as a criterion on which to judge a solution, nevertheless, it is reported for the sake of completeness.

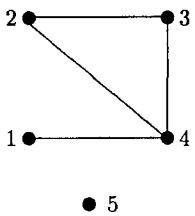


Figure 10. Incompatible graph for Example 2.

4.1. Results for Pickup Problems

To evaluate our heuristic, procedure *GRASP Routing* was applied a maximum of 250 times to each problem instance with a candidate list length of 3. Five different sets of $(\delta_1, \delta_2, \delta_3)$ for the cost metric given by Eq. (2) were tried: (0.0, 0.7, 0.3), (0.2, 0.7, 0.1), (0.4, 0.5, 0.1), (0.6, 0.3, 0.1), (0.8, 0.1, 0.1). The route elimination and distance reduction local search procedures (phase II) were applied to the best solution found after every 5 phase I iterations. If the minimum number of vehicles, as determined by the lower bounding procedures, was obtained at some point in the process the GRASP was terminated without completing all 250 iterations.

Table III summarizes the average results for each of the six data sets. These numbers correspond to an average over all problems in a particular set. The number of problems in each data set is given in column *Instances*. *LB* is the average of the best lower bound for each problem. The number of vehicles is given in column *Vehicles*. *Distance* is the total distance traveled for the solutions produced by *GRASP Routing*, while *Waiting Time* is the corresponding time a vehicle must wait at a service point. The CPU time (sec.) needed to find the best solution, including initialization and calculating the lower bounds, is given in column *CPU Time*. Individual output for all problems can be obtained by writing to the first author.

Table IV presents the number of vehicles and distance traveled obtained by *GRASP Routing* and the corresponding solutions reported by Solomon[26] (SOLO), Potvin and Rousseau[21] (PR), and Potvin et al.[20] (TABU). Solomon, and Potvin and Rousseau used the same objective function as we did; Potvin et al. first minimized the number of vehicles and then, subject to the results, attempted to minimize total route time, where total route time is defined as the sum of total distance and total waiting time.

As can be seen From Table IV *GRASP Routing* clearly outperformed all other heuristics on all data sets. To the

Table III. Average Results for the GRASP Routing Heuristic

Data Set	Instances	LB	Vehicles	Distance	Waiting Time	CPU Time (sec.)
R1	12	10.08	12.6	1325.44	235.83	73.08
R2	11	3	3.1	1164.27	497.10	115.60
RC1	8	9.38	12.6	1500.94	185.58	72.5
RC2	8	2	3.5	1414.21	489.7	127.5
C1	9	10	10	827.3	0.0	9.0
C2	8	3	3	589.65	2.73	13.51

best of our knowledge, the above results are the best reported for the stated objectives of minimizing the number of vehicles and minimizing the total distance.

Table V compares the computational requirements of *GRASP Routing* to those of TABU-2[20] in terms of CPU seconds. GRASP Routing is anywhere from 7 times faster for RC2 to 63 times faster for C1. As a point of reference, the average computational requirement (over all 56 instances) of Solomon's heuristic was approximately 39 seconds on a DEC-10 while Potvin and Rousseau's method required 1177 seconds on a PC.

Table VI compares the GRASP results to the optimal solutions reported by Desrochers et al.[6] For each instance we report the optimal solution and the GRASP solution along with the percentage gap for the total distance traveled. Out of 7 problems that were solved optimally by Desrochers et al., GRASP found the minimum fleet size for 6 of them, while there is a one vehicle-gap for problem R101. For problems C101, C102, C106, C107, and C108, GRASP found optimal solutions both in terms of fleet size and distance traveled. The solution for problem R102 has optimal fleet size and 8.6% distance traveled gap. With regard to waiting time, the GRASP solutions had no waiting time for problems C101, C102, C106, C107, and C108. For problems R101 and R102 the waiting time is 901.7 and 642.2, respectively.

An important element of an iterative randomized search procedure is deciding upon termination criteria. Although we planned to run *GRASP Routing* 250 times in each instance, it seems that a considerably lower number of iterations would have been adequate when the objective is to minimize the fleet size only, as suggested by the data in Table VII. Each row gives the number of problems for which the smallest fleet size was found within the corresponding iteration range. For an overwhelming 98% of the problems the minimal number of vehicles was found in fewer than 100 iterations.

The quality of both the heuristic and the lower bounds can be evaluated by considering their difference (gap). Table VIII provides the relative statistics. For example, the first row indicates that for data set R1, two problems were solved optimally (zero gap between the heuristic and the lower bound). An interesting observation is that for data sets R1 and RC1 the gap is relatively high despite the fact that the solutions obtained were far better than those re-

Table IV. Comparison with Other Heuristics

Data Set	GRASP	SOLO	PR	TABU
R1	12.6	13.6	13.3	12.8
	1325.44	1436.70	1509.04	1294.50
R2	3.1	3.3	3.1	3.2
	1164.27	1402.40	1386.67	1163.50
C1	10.0	10.0	10.7	10.0
	827.30	951.90	1343.69	870.90
C2	3.0	3.1	3.4	3.0
	589.65	692.70	797.59	611.00
RC1	12.6	13.5	13.4	12.8
	1500.94	1596.50	1723.72	1458.70
RC2	3.5	3.9	3.6	3.5
	1414.21	1682.10	1651.05	1448.60

Table V. Comparison of Computational Requirements (CPU sec.)

Heuristic	R1	R2	C1	C2	RC1	RC2
GRASP	73	116	9	14	73	128
TABU	820	1113	569	630	825	997

Table VI. Comparison Between GRASP and Optimal Solutions

Data Set	Optimum	GRASP	Distance Gap
R101	18	19	
	1607.7	1672.7	4.0%
R102	17	17	
	1434.0	1557.0	8.6%
C101	10	10	
	827.3	827.3	0.0%
C102	10	10	
	827.3	827.3	0.0%
C106	10	10	
	827.3	827.3	0.0%
C107	10	10	
	827.3	827.3	0.0%
C108	10	10	
	827.3	827.3	0.0%

Table VII. Best Iteration for *GRASP Routing*

Iterations	Number of Problems
1–50	49
51–100	6
101–150	1
151–200	—
201–250	—

Table VIII. Gap Between Heuristic and Lower Bound

	Heuristic—Lower Bound							
Data Set	0	1	2	3	4	5	6	Avg
R1	2	1	3	2	3	1		2.5
R2	1	8	2					1.1
RC1			3	2	2		1	3.3
RC2		4	4					1.5
C1	9							0.0
C2	8							0.0
Total	20	13	12	4	5	1	1	1.4

ported earlier. Although this is not conclusive we believe that the quality of the lower bounds is not very good in these instances.

The performance of the three lower bounds is presented in Table IX. The average performance of LB_1 is consistently better (except for R1) than the other two bounds, but LB_2 and LB_3 gave superior results for the time-constrained problems in each data set. LB_2 outperformed LB_1 and LB_3 for three problems in R1 and one problem in RC1 by a margin of more than 30%. The poor quality of LB_2 is attributed to the structure of the data sets. This bound addresses capacity and time incompatible pairs of customers but Solomon's data sets do not favor such pairs. First, there are no capacity incompatible pairs because all customer demands are far less than the vehicle capacity. Second, there is a small number of time incompatible pairs. This is because the data sets were created by placing the time window of customer i symmetrically around the time to travel from the depot to i (τ_{0i}). For clustered and semi-clustered problems, customers in the same group are approximately the same distance from the depot. Moreover, they usually are within a small distance from each other and have overlapping time windows. These factors increase the likelihood of incompatible pairs. On the other hand, LB_3 produced consistent results across all data sets. Compared to LB_2 it gave better bounds in 46 instances, it was dominated in 7 instances, and it gave the same lower bound in the remaining 3.

Table IX. Lower Bounds

Data Set	LB_1	LB_2	LB_3
R1	8.0	7.9	18.5
R2	2.0	1.2	2.1
RC1	9.0	6.4	7.6
RC2	2.0	1.1	2.1
C1	10.0	4.9	8.1
C2	3.0	1.4	3.0

Cost Parameters. An important element in fine tuning the performance of an insertion heuristic is determining appropriate parameter values for the cost metric. To better understand how a particular set of δ values affects the solution, consider the data in Table X. For each problem in R1, the δ values that gave the best solution are reported. These settings give a general indication whether a problem is more capacity or time constrained. For instance, the parameter values that gave the best solution for problems R101 and R102 have δ_1 equal to zero, which suggests that the time window constraints dominate the capacity constraints. This can be substantiated, in part, by considering the capacity and time window lower bounds (LB_1 and LB_2), which are 8 and 8, and 18 and 17 for problems R101 and R102, respectively. Similarly, as the time windows become less restrictive (problems R106–R112) δ_3 decreases and δ_1 increases taking the capacity element more into consideration.

Real-World Problems. The rest of this section presents the results for four real-world problems. All parameter settings, such as δ and number of iterations, were the same as those used above. However, local search (phase II) was applied every 30 GRASP iterations only because of the increased computational complexity. Table XI summarizes the results. As can be seen, *GRASP Routing* consistently produced high quality solutions in all four instances. The optimal fleet size was found for problems T2, T3, and T4, while there is a one vehicle gap for T1.

4.2. Results for Mixed Pickup and Delivery Problems

To evaluate our procedure when both pickups and deliveries are required, we considered the problems in R2, C2, and RC2 and constructed three new data sets MR2, MC2, and MRC2 by designating customer demand as either pickup or delivery with equal probability. The decision to restrict this phase of the analysis to derivatives of R2, C2, and RC2 was motivated by the fact that the time window constraints are dominant for R1, C1, and RC1. This means that even if half the customers required deliveries, the solution state space

Table X. Best Parameter Settings for Data Set R1

Delta	R101	R102	R103	R104	R105	R106	R107	R108	R109	R110	R111	R112
δ_1	0.0	0.0	0.0	0.2	0.0	0.2	0.2	0.2	0.2	0.2	0.2	0.4
δ_2	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.5
δ_3	0.3	0.3	0.3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Table XI. *GRASP Routing* Results for Large Data Sets

Data Set	Size	*LB*	Vehicles	Distance	Waiting Time	CPU Time (sec.)
T1	417	54	55	4273.4	27605.4	274
T2	417	55	55	4985.7	26926.1	1021
T3	219	3	3	636.3	97.0	27
T4	219	5	5	914.9	487.0	87

Table XII. Results for Combined Pickup and Delivery Problems

Data Set	Instances	*LB*	Vehicles	Distance	CPU Time (sec.)
MR2	11	4	4	1168.53	122.3
MC2	8	4	4	1094.94	130.1
MRC2	8	4	4.5	1496.91	135.7

would not be greatly affected—the time windows would still determine feasibility.

Nevertheless, the vehicle capacity was reduced from 1000 to 250 units to assure that the capacity constraint had the primary influence on feasibility. The number of GRASP iterations, candidate list length, as well as the δ values were as described in Section 4.1. The results are presented in Table XII.

The first observation is that the minimum number of vehicles was found in all MR2 and MC2 and half of the MRC2 data sets. This is a good indication that *GRASP Routing* is well-suited for capacity constrained problems with pickups and deliveries. The cost metric plays a major role in finding good solutions in these instances. To better understand this, notice that when vehicle capacity dominates, we really have a bin packing problem with loose time windows. For such problems, by setting $\delta_1 > \delta_2$ and $\delta_1 > \delta_3$, the cost metric of Eq. (2) will first route customers with large loads, assigning the others later as the residual capacity and time windows permit. If this can be done successfully, the number of vehicles required will be close, if not equal, to the optimum.

Finally, we note that in order to assess the robustness of our cost metric, the problems in MR2, MC2, and MRC2 were solved for different values of vehicle capacity ranging from 250 to 750 units. In almost all cases, the optimal solutions were found.

5. Discussion and Conclusions

In this paper, we have presented a new randomized heuristic for minimizing the fleet size of temporally constrained VRPs with two types of service. The results are generally stable for a wide range of geographical distributions (random, semi-clustered, and clustered), and indicate an improvement over existing procedures.

A second contribution of this work has been the introduction of lower bounding procedures for assessing the effectiveness of any heuristic that provides feasible routes. Because of their simplicity and low computational requirements, LB_1 and LB_3 can be useful in an implicit enumeration scheme (dynamic programming or branch-and-bound) for finding optimal solutions. In this regard, the overall performance of the lower bounding procedures has been encouraging, especially when compared to some preliminary results obtained from a tentative linear programming relaxation. A challenging area for future research is the development of lower bounds that work well for problems that are equally time window and capacity contrained. Also, a lower bound with constant worst case performance would be helpful in determining a range for the optimal solution.

One of the advantages of our approach is that it can be adapted to a variety of other routing problems with minimal effort. When multiple depots are present, for example, *GRASP Routing* can be applied with only minor modifications. In particular, whenever a decision must be made based on the location or the time window of the depot, the most favorable depot would necessarily be chosen. Dial-a-ride is another application for which *GRASP Routing* is well suited. Because the procedure already accounts for pickups and deliveries, only the subroutine that updates the capacity fields would have to be altered. An additional test would be required, though, to assure that a drop-off location is always preceded by the corresponding pickup point.

In each of these variants, as well as for the basic VRPTW, adding a more sophisticated local search routine to the GRASP would likely improve the results. Whether the accompanying computational burden could be justified, in terms of fleet size or distance reductions, is an issue for future study.

Acknowledgments

This work was supported by grants from Cray Research, Inc. and the Texas Advanced Research Program.

References

1. E. Baker, 1983, An Exact Algorithm for the Time Constrained Traveling Salesman Problem. *Operations Research 31:5*, 938–945.
2. E. Baker and J. Schaffer, 1988. Solution Improvement Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *American Journal of Mathematical and Management Sciences 6:3,4*, 261–300.
3. L. Bodin, B. Golden, A. Assad, and M. Ball, 1983. Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computers & Operations Research 10:2*, 63–211.
4. R. Carraghan and P. Pardalos, 1990. An Exact Algorithm for the Maximum Clique Problem. *Operations Research Letters 9*, 375–382.
5. N. Christofides, A. Mingozzi, and P. Toth, 1981. State-Space Relaxation for the Computation of Bounds to Routing Problems. *Networks 11*, 145–164.
6. M. Desrochers, J. Desrosiers, and M. Solomon, 1992. A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows. *Operations Research 40:2*, 342–354.
7. M. Desrochers, J.K. Lenstra, M. Savelsbergh, and F. Soumis, 1988. Vehicle Routing with Time Windows: Optimization and Approximation, in *Vehicle Routing: Methods and Studies*, B. Golden and A. Assad (ed.), North-Holland, Amsterdam, 65–84.
8. J. Desrosiers, Y. Dumas, and F. Soumis, 1986. A Dynamic Programming Solution of the Large Scale Single Dial-a-Ride Problem with Time Windows. *American Journal of Mathematics and Management Science 6:3,4*, 301–325.
9. J. Desrosiers, M. Sauve, and F. Soumis, 1988. Lagrangian Relaxation Methods for Solving the Minimum Fleet Size Multiple Traveling Salesman Problem with Time Windows. *Management Science 34:8*, 1005–1022.
10. J. Desrosiers, M. Solomon, and F. Soumis, 1994. Time Constrained Routing and Scheduling, in *Handbooks in Operations Research and Management Science: Networks*, M. E. Ball, T.L. Magnati, C. Momma and G.L. Nemhauser (eds.), North-Holland, Amsterdam.

11. T.A. FEO and M. RESENDE, 1988. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters 8*, 67–71.
12. T.A. FEO, J.F. BARD, and K. VENKATRAMAN, 1991. A GRASP for a Difficult Single Machine Scheduling Problem. *Computers & Operations Research 18:8*, 635–643.
13. M.R. GAREY and D.S. JOHNSON, 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York.
14. F. GLOVER, 1989. Tabu Search—Part I. *ORSA Journal on Computing 1:3*, 190–206.
15. J. KLINCEWICZ, 1992. Avoiding Local Optima in the p-Hub Location Problem Using Tabu Search and GRASP. *Annals of Operations Research 40*, 283–302.
16. A. KOLEN, A.H.G. RINNOOY KAN, and H. TRIENEKESN, 1987. Vehicle Routing with Time Windows. *Operations Research 35:2*, 266–273.
17. M. LAGUNA, and J.L. VELARDE, 1990. A Search Heuristic for Just-in-Time Scheduling in Parallel Machines. Technical Report, College of Business Administration, University of Colorado, Boulder.
18. S. MARTELLO and P. TOTH, 1990. Lower Bounds and Reduction Procedures for the Bin Packing Problem. *Discrete Applied Mathematics 28*, 59–70.
19. K. NYGARD, P. GREENBERG, W. BOLKAN, and E. SWENSON, 1988. Generalized Assignment Methods for the Deadline Vehicle Routing Problem, in *Vehicle Routing: Methods and Studies*, B. Golden and A. Assad (eds.), North-Holland, Amsterdam, 107–126.
20. J.Y. POTVIN, T. KERVAHUT, B. GARCIA, and J.M. ROUSSEAU, 1993. A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. Technical Report, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Canada.
21. J.Y. POTVIN and J.M. ROUSSEAU, 1991. A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows. *European Journal of Operational Research 66*, 331–340.
22. H. PSARAFTIS, 1980. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transportation Science 14:2*, 130–1154.
23. M. SAVELSBERGH, 1992. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. *ORSA Journal on Computing 4:2*, 146–154.
24. T. SEXTON and L. BODIN, 1985. Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: I Scheduling and II Routing. *Transportation Science 19*, 378–435.
25. M. SOLOMON, 1986. On the Worst-Case Performance of Some Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Networks 16*, 161–174.
26. M. SOLOMON, 1987. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research 35:2*, 254–265.
27. M. SOLOMON, E. BAKER and J. SCHAFFER, 1988. Vehicle Routing and Scheduling Problems with Time Window Constraints: Efficient Implementations of Solution Improvement Procedures; in *Vehicle Routing: Methods and Studies*, B. Golden and A. Assad (eds.), North-Holland, Amsterdam, 85–105.
28. M. SOLOMON and J. DESROSIERS, 1988. Time Window Constrained Routing and Scheduling Problems. *Transportation Science 22:1*, 1–13.
29. S. THANGIAH, K. NYGARD and P. JUELL, 1991. GIDEON: A Genetic Algorithm System for Vehicle Routing with Time Windows. *Proceedings of the 7th IEEE Conference on Artificial Intelligence Applications*, Miami, 422–425.