INFORMS Journal on Computing

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

A Two-Stage Heuristic with Ejection Pools and Generalized Ejection Chains for the Vehicle Routing Problem with Time Windows

Andrew Lim, Xingwen Zhang,

To cite this article:
Andrew Lim, Xingwen Zhang, (2007) A Two-Stage Heuristic with Ejection Pools and Generalized Ejection Chains for the Vehicle Routing Problem with Time Windows. INFORMS Journal on Computing 19(3):443-457. https://doi.org/10.1287/ijoc.1060.0186

INFORMS Journal on Computing
Vol. 19, No. 3, Summer 2007, pp. 443–457
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1903 | 0443

DOI 10.1287/ijoc.1060.0186

A Two-Stage Heuristic with Ejection Pools and Generalized Ejection Chains for the Vehicle Routing Problem with Time Windows

Andrew Lim
Department of Industrial Engineering and Logistics Management, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, iealim@ust.hk

Xingwen Zhang
Graduate School of Business, Stanford University, Stanford, California 94305-5015, xingwenz@stanford.edu

The vehicle routing problem with time windows (VRPTW) is an important problem in logistics. The problem is to serve a number of customers at minimum cost without violating the customers' time-window constraints or the vehicle-capacity constraint. In this paper, we propose a two-stage algorithm for the VRPTW. The algorithm first minimizes the number of vehicles with an ejection pool to hold temporarily unserved customers, which enables the algorithm to go through the infeasible solution space. Then it minimizes the total travel distance using a multi-start iterated hill-climbing algorithm with classical and new operators including generalized ejection chains, which enable the algorithm to search a larger neighborhood. We applied the algorithm to Solomon's 56 VRPTW instances and Gehring and Homberger's 300 extended instances. The experimental results showed that the algorithm is effective and efficient in reducing the number of vehicles and is also very competitive in terms of distance minimization. The *m*-VRPTW is a variant of the VRPTW in which a limited number of vehicles is available. A feasible solution to *m*-VRPTW may contain some unserved customers due to the insufficiency of vehicles. The primary objective of *m*-VRPTW is to maximize the number of customers served. We extended our VRPTW algorithm to solve *m*-VRPTW and the experimental results showed consistently good performance of the algorithm when compared with other methods.

Key words: transportation; vehicle routing; scheduling
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received May 2004; revised September 2004, August 2005; accepted January 2006. Published online in *Articles in Advance* July 20, 2007.

1. Introduction

The vehicle routing problem with time windows (VRPTW) is an important problem in logistics. In the VRPTW, let $G = (V, E)$ be a directed graph, where $V = \{0, 1, \ldots, N\}$ is the node set and $E = \{(i, j)\colon 0 \le i, j \le N,\ i \ne j\}$ is the edge set. Node 0 is the depot and $C = \{1, \ldots, N\}$ denotes the set of customers. All the routes must start from the depot, go to a number of customers and end at the depot. Each node i $(0 \le i \le N)$ has a specified time window, $[e_i, l_i]$. All vehicles must leave the depot after time e_0 and return to the depot before time l_0. For customer i, a vehicle may arrive before e_i and wait until e_i to start the service, but it may not arrive after l_i. Each customer i has a service demand q_i to be delivered from the depot and a required service time s_i. Each edge $E(i, j)$ has a travel distance d_{ij} and a travel time t_{ij}. In the following, we assume that $d_{ij} = t_{ij}$ for all edges. A set of identical vehicles with capacity Q is given. The total service demand of the customers in a route may not exceed the vehicle capacity Q. The primary objective of the VRPTW is to find a minimal number of routes to serve all customers exactly once while satisfying the capacity and time-window constraints. For the same number of routes, the secondary objective is to minimize the total distance traveled. The VRPTW is NP-complete (Lenstra and Rinnooy Kan 1981), and a large number of heuristics, including route-construction methods, route-improvement methods, and meta-heuristics, have been applied to the problem due to its great importance in practice. Bräysy and Gendreau (2005a, b) provide excellent surveys of work done on the VRPTW.

In this paper, we present a two-stage algorithm for the VRPTW. The algorithm focuses on minimizing the number of vehicles in the first stage, and on minimizing the total travel distance in the second stage. The algorithm for minimizing the number of vehicles uses a data structure called the *ejection pool* (EP) to hold temporarily unserved customers. Unserved customers in the EP are inserted into existing routes iteratively, and customers already in

existing routes may be kicked out and added to the EP. The algorithm for minimizing the total distance is a multi-start hill-climbing algorithm using classical and newly-defined operators including generalized ejection chains (GEC). In the GEC, a sequence of customers is ejected from an existing route and re-inserted into another route. The ejection and insertion procedures are repeated to reach a new solution with less total distance.

In the VRPTW, the number of vehicles is unlimited. Recently, Lau et al. (2003) proposed a new variant of the VRPTW, the m-VRPTW, to address real-world constraints such as a fixed fleet. Given m and a VRPTW instance, the primary objective function of m-VRPTW is to maximize the total number of customers served with m or fewer routes, and the secondary objective function is to minimize the total distance traveled. Different from a solution of the VRPTW, a feasible solution of m-VRPTW may contain some unserved customers because only a limited number of vehicles is available. This variant of the problem has also been studied by other authors. Bent and Van Hentenryck (2004a) considered "the partially dynamic vehicle routing problem with time windows, where some customers are known at planning time, while others are dynamic." Stochastic information is assumed to be available on the dynamic customers. Campbell and Savelsbergh (2005) defined a dynamic routing and scheduling problem as the home delivery problem (HDP). In the HDP, requests from a known set of customers for a delivery arrive in real time, and must be accepted or rejected as they arrive. Stochastic information is available on all customers. A brief survey on static stochastic problems and dynamic problems is given in Bent and Van Hentenryck (2004a). We extend our VRPTW algorithm to solve the m-VRPTW, in which those customers that are difficult to serve are left in the EP.

2. Algorithm for the VRPTW

In the literature, separating route reduction from distance minimization in the VRPTW is not a new idea, e.g., this separation is suggested in Homberger and Gehring (1999), Bent and Van Hentenryck (2004b) and Bräysy et al. (2002). Two-stage approaches have been shown to be useful by emphasizing the primary objective. Following the two-stage principle, our algorithm consists of three main components: a procedure to generate the initial solution, a procedure to reduce the number of vehicles, and a procedure to minimize the total distance. The initial solution is generated with squeaky-wheel optimization (SWO) (Joslin and Clements 1999), where Solomon's I1 (Solomon 1987) heuristic is used to construct a feasible solution at each iteration of SWO. Then, the algorithm focuses on reducing the number of vehicles of the initial solution with the EP data structure. Finally, the algorithm attempts to reduce the distance by iterated local search with the GEC procedure.

2.1. Initial Solution

The initial solution is constructed by iteratively applying Solomon's well-known I1 heuristic within an SWO framework. With Solomon's I1 heuristic, an unserved customer is initially selected as the seed customer of a route. The seed customer is chosen to be the farthest one from the depot, the one with the earliest allowed starting time for service, or simply a random one. The remaining unserved customers are inserted into this route one after another until no further insertion is possible due to the capacity or time-window constraints. The process is repeated with a new empty route until all customers are served.

The I1 heuristic defines two criteria, c_1 and c_2, to select the customer to be inserted into the route currently under construction. Let $\{v_0 = 0, v_1, v_2, \ldots, v_m, v_{m+1} = 0\}$ be the current route. For each unserved customer u, the lowest insertion cost is computed as $c_1(u) = \min_{q=0,\ldots,m} c_1(v_q, u, v_{q+1})$, where the insertion of u between v_q and v_{q+1} is feasible. The best customer, u^*, to be inserted into the current route is the one with $c_2(u^*) = \max_u c_2(u)$. The definitions of $c_1(i, u, j)$ and $c_2(u)$ are

$$c_1(i,u,j) = \alpha_1 \times (d_{iu} + d_{uj} - \mu \times d_{ij}) + \alpha_2 \times (t'_j - t_j), \quad (1)$$

where

$$\alpha_1 + \alpha_2 = 1,\ \alpha_1 \geq 0,\ \alpha_2 \geq 0,\ \mu \geq 0,$$
$$c_2(u) = \lambda \times d_{0u} - c_1(u), \quad \lambda \geq 0, \quad (2)$$

where α_1, α_2, μ, and λ are control parameters, d_{iu}, d_{uj}, d_{ij}, and d_{0u} are distances between i and u, u and j, i and j, and 0 and u, respectively, t'_j denotes the new time of beginning service at j after u is inserted between i and j, while t_j is the time of beginning service before the insertion. For more details, we refer to Solomon (1987) and Bräysy and Gendreau (2005a). The selected u^* is then inserted into the current route at the position that leads to the lowest cost, $c_1(i, u^*, j)$.

To improve the quality of the initial solution, Solomon's I1 heuristic is invoked for a certain number of times within an SWO framework. In SWO, a greedy algorithm is used to construct a solution. The solution found by the greedy heuristic is analyzed, and the priority factors of some problem elements are updated with the knowledge derived from the analysis. The priorities of those elements that work poorly are increased. The new priorities are then used to guide the greedy algorithm in the construction of the next solution. This construct/analyze/prioritize

(Joslin and Clements 1999) cycle continues until the maximal number of iterations has been reached or the time limit is exceeded. As Solomon's I1 heuristic is invoked several times by the SWO, multiple solutions are generated. The best one is chosen as the initial solution for the route-reduction procedure.

In our SWO framework for the VRPTW, each customer is assigned a priority factor, which is initially one and is dynamically updated from one iteration to the next during the execution of the SWO algorithm. The priorities are used to guide Solomon's I1 heuristic in the selection of the next customer to insert. More precisely, for customer u with priority factor p_u, the new criteria $c_1'(u)$ and $c_2'(u)$ are

$$c_1'(u) = \min_{q=0,\ldots,m} \frac{c_1(v_q, u, v_{q+1})}{p_u}$$

$$c_2'(u) = \begin{cases} c_2(u) \times p_u & \text{if } c_2(u) \geq 0 \\ \dfrac{c_2(u)}{p_u} & \text{otherwise,} \end{cases}$$

where the definitions of $c_1(v_q, u, v_{q+1})$ and $c_2(u)$ are given in (1) and (2). The best customer, u^*, to be inserted into the current route is the one with $c_2'(u^*) = \max_u c_2'(u)$. With the new criteria, $c_1'(u)$ and $c_2'(u)$, those customers with higher priorities will be inserted earlier by the modified Solomon's I1 heuristic. After all customers have been routed, the solution generated is analyzed and the customers in the routes with fewer customers or less total service demand are penalized more. More specifically, let $size(r)$ and $load(r)$ be the number of customers in a route, r, and their total service demand. Thus, $lost(r) = 1 - load(r)/Q$ indicates the percentage of unused vehicle capacity in r. The priority of each customer in r is then increased by the amount $lost(r)/size(r)$. Thus, the increase is directly proportional to the percentage of unused vehicle capacity, while inversely proportional to the number of customers served in the route. When those elements that work poorly are handled sooner, they also tend to be handled better and thus this is more likely to improve the objective function.

2.2. Procedure to Reduce the Number of Vehicles

2.2.1. Lower Bounds on the Number of Vehicles. In the procedure to reduce the number of vehicles, two types of lower bounds are used to check the optimality of the heuristic solutions and to avoid unnecessary computations. The first lower bound, LB_1 in (3), is calculated from the ratio of the total demand of all customers to the vehicle capacity:

$$LB_1 = \left\lceil \frac{\sum_{i=1}^{N} q_i}{Q} \right\rceil. \tag{3}$$

The second lower bound, LB_2, originally proposed in Kontoravdis and Bard (1995), is computed by finding the maximum clique of the associated incompatibility graph, $G' = (V', E')$. $V' = \{1, \ldots, N\}$ is the customer set and $E' = \{(i, j)$: i and j are incompatible$\}$. Customers i and j are incompatible if they cannot be served on the same route due to the time-window or capacity constraints. For the time-window constraints, i and j are incompatible if $e_i + s_i + d_{ij} > l_j$ and $e_j + s_j + d_{ji} > l_i$. For the capacity constraint, they are incompatible if $q_i + q_j > Q$. Two vertices in V' are adjacent if they are connected by an edge in E'. A clique of the graph is a set of vertices, any two of which are adjacent. In the maximum-clique problem, one desires to find a maximum clique, which is the largest among all cliques in the graph. It is clear that the size of the maximum clique is a lower bound on the number of routes, since any two customers in the maximum clique cannot be served in the same route.

For the maximum-clique problem, we used the exact algorithm in Östergård (2002), which is a branch-and-bound algorithm with the vertex order taken from a coloring of the vertices and with a new pruning strategy. In a vertex-coloring, adjacent vertices must be assigned different colors. For good performance of the algorithm, the vertices are ordered so that those belonging to the same color class are grouped. To get such a coloring, we used the same method as in Östergård (2002). The color classes are determined one at a time. When determining a new color class, the graph induced by the uncolored vertices is first constructed. Then, as long as there exists a vertex that can be added to the color class, such a vertex with largest degree is added. The vertices are labeled v_N, v_{N-1}, ... in the order that they are added to a color class. Note that the coloring is an upper bound on the maximum clique, because a graph cannot contain a clique of size $c+1$ if the graph can be colored with c colors. Therefore, if the number of colors used is less than or equal to LB_1, we do not need to compute LB_2 as $LB_2 \leq LB_1$.

2.2.2. Route-Eliminating Procedure. Given an initial solution generated by SWO, the route-eliminating procedure always tries to reduce the number of vehicles by one. If the reduction is successful before some terminating conditions are reached, the procedure is repeated to reduce the number of vehicles further. The procedure terminates when the number of routes reaches the lower bound, $\max(LB_1, LB_2)$ or some time limit has been exceeded. The pseudo-code of the algorithm is

procedure reduce_route (*routes*)
  while (terminating conditions are not reached) **do**
    delete the route with the fewest customers from *routes*

```
add the customers in the deleted route
  into the EP
while (EP is not empty) and (terminating
    conditions are not reached) do
  select a customer u from the EP
  determine an insertion position of u in a
    remaining route of routes
  remove u from the EP and insert it into the
    target route
  while (the target route is not feasible) do
    remove a customer from the route
      and add it into the EP
  end while
  apply Exchange, Relocate, 2-opt, Or-opt, 2-opt*,
    Cross to routes
 end while
end while
```

The central part of the procedure is the ejection pool (EP), a data structure to hold unserved customers temporarily. Initially, the route with the fewest customers is removed from the solution and the customers served in this route are added into the EP. At each iteration, the procedure selects an unserved customer in the EP and inserts it into an existing route. Customers who can be feasibly inserted into an existing route are always selected for insertion before those who cannot be feasibly inserted into any route. If the insertions of two customers are both feasible or both infeasible, the one that was added into the EP earlier is favored to avoid some customers "sinking deeply" in the pool.

After the candidate customer, denoted u, is selected, the algorithm needs to determine the target route and the position to insert u. The cost $c(i, u, j)$ of inserting u between two consecutive customers i and j is defined as

$$c(i, u, j) = \beta_1 \times (d_{iu} + d_{uj} - d_{ij}) + \beta_2 \times (dl_u + dl_j),$$

where

$$\beta_1 + \beta_2 = 1,\ \beta_1 \geq 0,\ \beta_2 \geq 0,$$

and d_{iu}, d_{uj}, and d_{ij} are the distances between i and u, u and j, and i and j, respectively. Parameters β_1 and β_2 control the relative weights of the increased distance and increased service delay. Because inserting u between i and j may violate the time-window constraints of u, j and the customers served after j in the target route, their service delay should be taken into account when calculating the insertion cost $c(i, u, j)$. For this purpose, dl_u measures the delay of u, while dl_j measures the delay of j and those customers following j. In (4) and (5), t_u is the earliest time to service u after u is inserted between i and j by temporarily relaxing the late time-window constraint of u, i.e., by temporarily allowing $t_u > l_u$. In (5), s_u is the given service time of u and la_j is the latest arrival time of j such that the time-window constraints of j and those customers after j will not be violated:

$$dl_u = \max(t_u - l_u, 0), \tag{4}$$

$$dl_j = \max(t_u + s_u + d_{uj} - la_j, 0). \tag{5}$$

If there exist some feasible insertion positions for u, these feasible positions make up the candidate position list. Otherwise, the candidate list consists of all possible positions. Elements in the candidate list are sorted according to their corresponding insertion cost $c(i, u, j)$ in nondecreasing order. The final insertion position is chosen to be the pth element in the candidate list, and $p = random(0, 1)^{e_1} \times s$ where $e_1 > 1$ and s is the size of the candidate list. Thus, the algorithm favors the positions with lower insertion cost and allows them to be selected with higher probabilities.

After the insertion position is chosen, the customer is inserted into the target route accordingly. When the insertion is not feasible, the algorithm continues to select a customer, except u, to be taken off (or kicked out of) the route and added into the EP until the target route becomes feasible. At each iteration of the kick procedure, one of those customers (denoted as *relevant customers*) whose removal will potentially benefit the feasibility of the route is removed. Let $\{0, v_1, v_2, \ldots, v_{i-1}, v_i = u, v_{i+1}, \ldots, v_m, v_{m+1} = 0\}$ be the resulting route after the insertion of u and some possible kicks, and t_{v_x} be the earliest time to service v_x in the route. Two different situations should be considered. If some customers are not feasible with regard to their time-window constraints, let v_j $(j \geq i)$ be the first such customer in the route. Then, those customers, except u, whose removals will reduce t_{v_j} are *relevant customers*. In addition, v_j itself is also in the set of *relevant customers* if $v_j \neq u$. If all remaining customers in the route are feasible with regard to their time-window constraints, but the route is still infeasible due to the vehicle-capacity constraint, then all the customers except u are *relevant*.

If some of the relevant customers can be feasibly inserted into another route without violating any constraints, then those customers that cannot be feasibly inserted into any other route are no longer relevant. Thus, we prefer to remove those "feasible relevant" customers as long as there exist such customers, because "feasible relevant" customers are generally easier to be re-inserted. Let v_x be any of the remaining relevant customers. This customer's kick saving, $ks(v_x)$, is

$$ks(v_x) = \theta_1 \times (d_{v_{x-1}v_x} + d_{v_x v_{x+1}} - d_{v_{x-1}v_{x+1}}) - \theta_2 \times mdl(v_x, r, \sigma), \tag{6}$$

where

$$\theta_1 + \theta_2 = 1,\ \theta_1 \geq 0,\ \theta_2 \geq 0,$$

$$mdl(v, r, \sigma) = \min_{r' \in \sigma \wedge r' \neq r} mdl(v, r') \tag{7}$$

$$mdl(v, r') = \max(\eta \times mdl_{load}(v, r'), mdl_{time}(v, r')) \tag{8}$$

$$mdl_{load}(v, r') = \begin{cases} 0 & \text{if } load(r') + q_v \leq Q \\ \dfrac{(l_0 - e_0) \times (load(r') + q_v - Q)}{load(r') + q_v} & \text{otherwise} \end{cases} \tag{9}$$

$$\begin{aligned} load(r') &= \sum_{v' \in r'} q_{v'} \\ mdl_{time}(v, r') &= \min_{0 \leq p \leq n} c_t(v, v'_p, r') \\ c_t(v, v'_p, r') &= p_l(v, v'_p, r') + p_e(v, v'_p, r') \\ &\quad + p_b(v, v'_p, r') \\ p_l(v, v'_p, r') &= \max\left(ed_{v'_p} + d_{v'_p v} - l_v, 0\right), \\ p_e(v, v'_p, r') &= \max\left(e_v - \left(la_{v'_{p+1}} - d_{v v'_{p+1}} - s_v\right), 0\right), \\ p_b(v, v'_p, r') &= \max\left(\max\left(ed_{v'_p} + d_{v'_p v}, e_v\right)\right. \\ &\quad \left. + s_v + d_{v v'_{p+1}} - la_{v'_{p+1}}, 0\right). \end{aligned} \tag{10}$$

The one with maximal kick saving is removed from the route and added into the EP and then the above procedure is repeated until the route becomes feasible.

Kick saving considers the distance reduction by kicking v_x out of the current route and the cost needed to re-insert v_x into another route. In the following, σ is the set of routes in the partial solution, r is the current route, and $r' = \{0, v'_1, v'_2, \ldots, v'_n, v'_{n+1} = 0\}$ represents any other route in σ. The cost of inserting a customer v into r', $mdl(v, r')$, is an extension of the *minimal delay* (*mdl*) defined in Homberger and Gehring (1999) and Bent and Van Hentenryck (2004b). The new definition takes into account the penalty of violating the vehicle-capacity constraint, in addition to that of violating time-window constraints. The penalty of overload, mdl_{load}, is approximated in (9) by the additional amount of time needed to accommodate the excessive load. If $load(r') + q_v \leq Q$, v can be inserted into r' without violating the vehicle-capacity constraint and thus no overload penalty is incurred. Otherwise, the total available time resource, $l_0 - e_0$ (as mentioned previously, the depot operates within the time window $[e_0, l_0]$), is equally allocated to the total demand $load(r') + q_v$ after the insertion of v. Then the penalty of overload is taken to be the time allocated to the overloaded portion, $load(r') + q_v - Q$. The penalty of time-window violations, mdl_{time}, is given by the minimal c_t value over all insertion positions. c_t is in turn defined to be the summation of the violation of l_v induced by relocating v after $(0, v'_1, \ldots, v'_p)$ (p_l), the violation of e_v induced by relocating v before $(v'_{p+1}, v'_{p+2}, \ldots, v'_n, 0)$ (p_e) and the violation of inserting v between the above two customer sequences (p_b). $ed_{v'_p}$ is the earliest departure time of v'_p when $(0, v'_1, \ldots, v'_p)$ is feasible and $la_{v'_{p+1}}$ is the latest arrival time of v'_{p+1} to make $(v'_{p+1}, v'_{p+2}, \ldots, v'_n, 0)$ feasible. The cost of inserting v into r', $mdl(v, r')$, takes into account both $mdl_{load}(v, r')$ and $mdl_{time}(v, r')$, because the insertion may violate the capacity constraint or time-window constraints (or both). As mdl_{load} is estimated by the additional amount of time needed to accommodate the excessive load, we are able to compare the penalty of violating the capacity constraint with that of violating time-window constraints. In addition, the parameter η in (8) is introduced to control the relative weights of mdl_{load} and mdl_{time}.

After each insert/kicks cycle, hill climbing is applied to improve the quality of the routes in the partial solution. The procedure applies *Exchange*, *Relocate*, *2-opt*, and *Or-opt* (Or 1976) sequentially to improve a single route, and uses *2-opt** and *Cross* (Taillard et al. 1997) for a pair of routes, with a hierarchical overall objective function. The primary objective of the local search is to minimize the total *mdl* of all ejected customers in the EP, i.e., $\sum_{v \in EP} mdl(v, \sigma)$. The *mdl* of an unserved customer v in the EP is

$$mdl(v, \sigma) = \min_{r' \in \sigma} mdl(v, r'), \tag{11}$$

where $mdl(v, r')$ is defined in (8). The *mdl* value estimates how easily the customer can be inserted back into a route in the partial solution. Minimizing the total *mdl* of the customers in the EP would possibly make it easier to re-insert them.

Only when the primary objective value of a solution is the same as that of a neighboring solution in the hill-climbing process will their secondary objective values be compared. The secondary objective of the local search is to maximize the maximal free time (*mft*) of the route if only a single route is concerned, or the sum of *mft* for a pair of routes. The *mft* of route $r' = \{0, v'_1, v'_2, \ldots, v'_n, v'_{n+1} = 0\}$ is

$$\begin{aligned} mft(r') &= \max_{0 \leq p \leq n} ft(v'_p, v'_{p+1}, r'), \\ ft(v'_p, v'_{p+1}, r') &= la_{v'_{p+1}} - ed_{v'_p} - d_{v'_p v'_{p+1}}, \end{aligned} \tag{12}$$

where $ed_{v'_p}$ and $la_{v'_{p+1}}$ are as described above. The *mft* value estimates the maximal usable time of a route for possible future insertions. Therefore, maximizing the *mft* may help make more space for some customer to be inserted into the route in the future.

When the primary and secondary objective values of two solutions are both equal, the algorithm reduces the total distance of the routes. After the hill climbing,

the algorithm continues selecting another unserved customer in the EP and inserting it back into an existing route until the EP is empty or the terminating conditions are reached.

The ideas of *ejection pool* and *minimal delay* are not new for the VRPTW (for example, in Lau et al. 2003, a *holding list* that contains unserved customers is treated as a "phantom" route, which participates in the regular local search like a normal route). But in our algorithm, the EP is the key part of the procedure to reduce the number of vehicles, and the algorithm fully focuses on inserting the unserved customers in the EP back to the routes. The insert/kicks operation to transfer customers between the EP and the routes allows the procedure to search for better solutions by going through the infeasible solution space. In Koskosidis et al. (1992) and Taillard et al. (1997), a customer is allowed to be served after the upper bound of its time window and a penalty for the violation is added to the objective value. By relaxing the late time-window constraints, their algorithms are also able to go through the infeasible solution space. As opposed to their approaches, our algorithm separates infeasible customers from feasible ones by storing those unserved customers in the EP. Thus, it is easy for the algorithm to focus explicitly on re-inserting the unserved customers back to the routes. The integration of the generalized minimal delay with the EP considers the time-window constraints and the vehicle-capacity constraint for those unserved customers in the EP. The local search to minimize the total *mdl* of the customers in the EP generally reduces constraint violations of an infeasible solution and drives the search towards a feasible one. The introduction of the maximal free time of routes maximizes the estimated usable time of routes for possible future insertions of unserved customers. The experimental results in the following sections will show that the approach is highly effective in reducing the number of vehicles.

2.3. Procedure to Reduce the Total Distance

Given the solution after the route-reduction process, the algorithm uses a local search with classical operators, an enumeration-based operator (E-opt) and a generalized-ejection-chains (GEC)-based operator to minimize the total distance traveled. The pseudocode of the algorithm is given by the procedure *reduce_distance*. In the distance-minimization process, the number of routes is fixed and the cost function of the local search is simply the total distance. In the hill-climbing algorithm, only those local moves that lead to solutions with lower total distance are accepted.

procedure reduce_distance (*routes*)
  while (terminating conditions are not reached) **do**
  while (the total distance of *routes* is reduced) **do**
    for (each route, r, of *routes*) **do**
      while (the total distance of r is reduced) **do**
        apply hill-climbing with *Exchange*, *Relocate*, *2-opt*, *Or-opt* to reduce $dis(r)$
        apply hill-climbing with *E-opt* to reduce $dis(r)$
      end while
    end for
    for (each pair of routes, $(r1, r2)$, of *routes*) **do**
      while (the total distance of $r1$ and $r2$ is reduced) **do**
        apply hill-climbing with *2-opt** to reduce $dis(r1) + dis(r2)$
        apply hill-climbing with *Cross* to reduce $dis(r1) + dis(r2)$
      end while
    end for
    if (the total distance of *routes* is **not** reduced) **then**
      apply generalized ejection chains (GEC) to reduce $dis(routes)$
    end if
  end while
  perturb *routes* with *tabu search*
  end while

The classical operators, *Exchange*, *Relocate*, *2-opt*, and *Or-opt*, are applied sequentially to reduce the distance of a single route. When the search procedure reaches a local optimum in which these four operators fail to reduce the distance further, the enumeration-based *E-opt* is applied to the route. E-opt selects a consecutive sequence of customers of the route and exhaustively computes the best ordering of these customers within a branch-and-bound framework, while other parts of the route are fixed. Each time, E-opt optimizes the ordering for a segment at most $L_{E\text{-}opt}$ ($L_{E\text{-}opt} = 10$ in our computational study) in length and thus limits the computation time. The hill-climbing procedure with the E-opt operator terminates when it is not able to improve any customer sequence of at most $L_{E\text{-}opt}$ in length, and then the local search with classical operators is applied again. The optimization of a single route terminates when none of the operators is able to improve the route further. For a pair of routes, *2-opt** and *Cross* are applied sequentially to reduce the total distance in a similar hill-climbing manner.

If the hill-climbing algorithm fails to improve any route or any pair of routes further with the above operators, the *GEC* operator, which is able to search a larger neighborhood, is applied. If the solution is improved by the GEC operator, the local search goes back to optimize each single route and each pair of routes as previously described.

The GEC operator is a generalization of Glover's ejection chains (EC) (Glover 1991, 1992), which have

been applied to the VRPTW (e.g. Rousseau et al. 2002 and Bräysy et al. 2002). Ejection chains combine sequences of single moves into compound moves. In the VRPTW context, a single move removes a customer from its route and inserts it in another route. The insertion of the customer may require removal of another customer from the target route. The removal and insertion procedures are repeated until the customer can be inserted into another route without the removal of any other customer.

The ejection chains based on the removal and insertion of a single customer are generalized by allowing the removal and insertion of a sequence of consecutive customers (a segment). In each phase of the GEC, a segment is unrouted. The removal and insertion procedures are repeated until the segment can be inserted into another route without the need to remove any other segment. To limit the computational complexity, the length of the ejected segment is at most L_{GEC} ($L_{GEC} = 4$ in our computational study). To improve further the efficiency of searching the GEC neighborhood, we implemented a very-large-scale-neighborhood algorithm (VLSN) (Ahuja et al. 2001, 2004) to identify heuristically a profitable GEC that will lead to distance reduction. As in Ahuja et al. (2001) and Ibaraki et al. (2005), the VLSN algorithm transforms a profitable GEC into a negative-cost subset-disjoint cycle in a graph, the improvement graph (IG). In the IG, a regular node $n_{v_i,j}$ represents a segment of length j, which starts with customer v_i. Let $r_1 = \{0, v_1, \ldots, v_{i-1}, v_i, \ldots, v_{i+j-1}, v_{i+j}, \ldots, v_m, 0\}$ be the route containing the segment $(v_i, \ldots, v_{i+j-1})$, and $r_2 = \{0, v'_1, \ldots, v'_{k-1}, v'_k, \ldots, v'_{k+l-1}, v'_{k+l}, \ldots, v'_n, 0\}$ be the route containing the segment $(v'_k, \ldots, v'_{k+l-1})$ represented by the node $n_{v'_k,l}$. Thus, $r'_2 = \{0, v'_1, \ldots, v'_{k-1}, v'_{k+l}, \ldots, v'_n, 0\}$ represents the resulting route after $(v'_k, \ldots, v'_{k+l-1})$ is ejected from r_2. An arc from $n_{v_i,j}$ to $n_{v'_k,l}$ in the IG indicates that the segment $(v_i, \ldots, v_{i+j-1})$ is removed from r_1 and inserted into r'_2 at the best position. The segment $(v'_k, \ldots, v'_{k+l-1})$ has been ejected from r_2 before insertion of $(v_i, \ldots, v_{i+j-1})$ in r'_2 (as stated above, r'_2 is the resulting route after the ejection of $(v'_k, \ldots, v'_{k+l-1})$ from r_2). The cost of removing $(v_i, \ldots, v_{i+j-1})$ from r_1 is $\Delta_1 = d_{v_{i-1}v_{i+j}} - d_{v_{i-1}v_i} - d_{v_{i+j-1}v_{i+j}}$. After its ejection from r_1, $(v_i, \ldots, v_{i+j-1})$ is inserted between two consecutive customers, v'_p and v'_q, in r'_2 so that the distance increase, $\Delta_2 = d_{v'_p v_i} + d_{v_{i+j-1}v'_q} - d_{v'_p v'_q}$, is minimized. Thus, the cost of the arc from $n_{v_i,j}$ to $n_{v'_k,l}$ is taken to be $\Delta_1 + \Delta_2$, which exactly measures the distance change caused by the removal of $(v_i, \ldots, v_{i+j-1})$ from r_1 and its insertion in r'_2. Similarly, the distance change caused by the ejection and re-insertion of $(v'_k, \ldots, v'_{k+l-1})$ is captured by arcs from $n_{v'_k,l}$ to other nodes. In addition, a pseudo-node for each route and an extra node called the *origin node* are added to the IG as in Ahuja et al. (2001) and Ibaraki et al. (2005). After constructing the IG as described above, there is a one-to-one cost-preserving correspondence between GECs and subset-disjoint cycles in the IG. To search for profitable GECs, the algorithm simply identifies the negative cycles in the IG using a variant of shortest-path label-correcting algorithms (Ahuja et al. 2001). Compared to the approach of Ibaraki et al. (2005), the VLSN implementation of the GEC neighborhood is able to search a larger neighborhood. More specifically, in Ibaraki et al. (2005), $(v_i, \ldots, v_{i+j-1})$ is always inserted between v'_{k-1} and v'_{k+l}, i.e., at the original position of $(v'_k, \ldots, v'_{k+l-1})$ in r_2 before its ejection. With the GEC, the segment $(v_i, \ldots, v_{i+j-1})$ is inserted at the best position in r'_2 so that the distance increase is minimized.

If the GEC fails to find a better solution and the terminating conditions are not reached, the best solution found so far is perturbed by applying *tabu search* (Glover 1986) with *2-opt** and *Cross* operators for k_1 iterations. The tabu list stores edges that have been created within the preceding number of tabu iterations defined by the tabu length, k_2. In each tabu iteration, a move is tabu, i.e., forbidden, if and only if the edges to be removed are in the tabu list. Thus, the edges created in previous k_2 iterations are not allowed to be removed. All nontabu moves in the neighborhoods of *2-opt** and *Cross* are stored in a candidate list and sorted according to their costs, i.e., the incurred distance changes, in a nondecreasing order. The final move is chosen to be the p-th one in the candidate list, $p = random(0, 1)^{e_2} \times s$, where $e_2 > 1$ and s is the size of the candidate list. Thus, the algorithm favors the moves with lower cost and allows them to be selected with higher probabilities. The *tabu-search* procedure then applies the selected move to the current solution and proceeds to the next iteration. This process continues for k_1 iterations and then the hill-climbing algorithm restarts with the new initial solution.

2.4. The Integrated Algorithm for the VRPTW

In our distance-minimization procedure, the solutions generated by the local search are always feasible and the number of vehicles is fixed. This approach has the advantage that the primary objective value is always guaranteed, but the disadvantage is that the solution space may be tightly restricted by the fixed number of vehicles and the capacity and time-window constraints. This restriction introduces some difficulties for the distance-minimization process. It may be even impossible to reach some other solutions that might be better from a given initial solution. Although the *tabu-search* procedure is able to perturb the solutions, it does not have the ability to go through the infeasible solution space.

To overcome this problem without worsening the primary objective value, our overall algorithm for the

VRPTW randomly generates a certain number (20 in our final computational study) of initial solutions for each problem instance as described in the previous section and minimizes the number of vehicles for each of them, taking advantage of the effectiveness and efficiency of the route-reduction procedure. Then, the algorithm applies the distance-minimization procedure to these solutions after the route-reduction process only for a small number of iterations (ten in our final computational study). The best solution (or solutions in case the algorithm intends to search multiple solution regions) found during the preliminary distance-minimization process is taken as the seed solution, and the algorithm applies the distance-minimization procedure to it again for at most 1,000 iterations before the time limit is exceeded. The rationale behind the approach is to sample better solution regions in the solution space and to intensify the search only within good regions. This sample/intensify approach is shown to be effective in our experiments.

3. Algorithm for the m-VRPTW

Our algorithm for the VRPTW will now be extended to handle m-VRPTW instances. Similarly, the extended algorithm consists of three main components: a procedure to generate the initial solution, a procedure to maximize the number of customers served, and a procedure to minimize the total distance. The initial solution is generated by almost the same SWO algorithm with the primary objective being changed to maximize the number of customers served. The distance-minimization procedure uses exactly the same algorithm as the one for the VRPTW. But the procedure to maximize the number of customers served is different from its counterpart for the VRPTW in a number of ways, due to insufficient vehicles.

Given an initial solution generated by SWO, only the m routes with the largest numbers of customers are kept. All other routes are removed from the solution and those customers served in these routes are added to the EP. At each iteration, the procedure selects an unserved customer in the EP with the smallest $mdl(v, \sigma)$ value as defined in (11), and inserts it into an existing route of solution σ. The $mdl(v, \sigma)$ value is a heuristic measure of the difficulty of inserting customer v. Generally, a m-VRPTW solution contains some unserved customers because only a limited number of vehicles is available. By selecting the customer with minimal $mdl(v, \sigma)$, the algorithm inserts those customers that are more easily served by the existing m vehicles, while leaving the customers that are difficult to serve in the EP.

After the candidate customer u is selected, it is inserted into the target route r as described in Section 2.2. If insertion of u is not feasible, the algorithm continues selecting a customer to be kicked out of the route and adding it into the EP until the target route becomes feasible. The potential customers to be kicked out are *relevant customers* as in Section 2.2. Let v_x be any of the relevant customers. Its new kick saving, $ks'(v_x)$, is defined in (13). The one with maximal new kick saving is kicked out of the route and added to the EP and the above procedure is repeated until the route becomes feasible.

The new kick saving ks' considers the distance reduction by kicking v_x out of the route and the cost needed to re-insert v_x into another route. In (13), $d_{v_{x-1}v_x} + d_{v_x v_{x+1}} - d_{v_{x-1}v_{x+1}}$ is the distance reduction by kicking v_x out, and $mdl(v_x, r, \sigma)$ is the cost needed to re-insert v_x as defined in (7).

$$ks'(v_x) = \begin{cases} ks(v_x) & \text{if size of the EP} \le S \\ \theta_1 \times \left(d_{v_{x-1}v_x} + d_{v_x v_{x+1}} - d_{v_{x-1}v_{x+1}}\right) + \theta_2 & \\ \quad \times mdl(v_x, r, \sigma) & \text{otherwise,} \end{cases} \tag{13}$$

where

$$\theta_1 + \theta_2 = 1,\ \theta_1 \ge 0,\ \theta_2 \ge 0,\ S > 1.$$

ks' is similar to ks in (6), the kick saving for the VRPTW. The rationale behind the definition of ks is to remove the customers that are easier to insert into other routes so that it is more likely to reach a solution with all customers served. For m-VRPTW, the situation is somehow different because generally not all customers could be served with m or fewer vehicles. When the number of unserved customers in the EP is over a control parameter S, the algorithm estimates that it is not possible to serve all customers. Since it is inevitable that some customers are unserved, the algorithm intends not to serve those difficult customers and removes them from the route. On the other hand, if the number of unserved customers is not more than S, the algorithm removes the easy customers so that they may be served in other routes.

After each insert/kicks cycle, hill climbing is applied to improve the quality of the routes in the partial solution. The procedure applies *Exchange*, *Relocate*, *2-opt*, and *Or-opt* sequentially to improve a single route, and uses *2-opt** and *Cross* for a pair of routes. After the hill climbing, the algorithm continues selecting another unserved customer in the EP and inserting it back into an existing route until the EP is empty or the terminating conditions are reached.

As opposed to the local search in Section 2.2, the primary objective of the local search for m-VRPTW is to reduce the *weighted sum* of mdl for the customers in the EP. The mdl of an unserved customer v is defined in (11). The mdl values of all unserved

customers are sorted in nondecreasing order. Let $\{m_1, m_2, \ldots, m_n\}$ be the sorted array of the *mdl* values. We use $ws = \sum_{i=1}^{n} m_i/i$ to calculate the weighted sum in our computational study. A desired property of the weighting function is that the weight decreases as *mdl* increases. This comes from the fact that the unserved customers are inserted into the routes sequentially, and at each iteration the procedure selects the unserved customer with the smallest *mdl* value for insertion. Therefore, customers with smaller *mdl* are more likely to be re-inserted and should contribute more to the weighted sum, while customers with larger *mdl* have less chance of being served and should contribute less.

In our *m*-VRPTW algorithm, the EP is still the key part of the procedure to maximize the number of customers served. The algorithm inserts the unserved customers that are easy to serve back into the routes, and kicks out customers that are hard to serve back to the EP. The local search to minimize the weighted sum of *mdl* generally favors easy customers. These modifications are made to address the issue that not all customers are served due to the limited number of vehicles.

4. Experimental Results

For the VRPTW, Solomon's 56 benchmark problems (Solomon 1987) are widely used in the literature for comparison of different heuristics. Each instance contains 101 nodes, i.e., a central depot and 100 customers. The 56 instances are categorized into six classes, R1, C1, RC1, R2, C2, and RC2. The customers are located in clusters in C1 and C2, and uniformly distributed in R1 and R2, while the RC1 and RC2 classes contain clustered and randomly distributed customers. The C1, R1, and RC1 classes have narrower time windows for nodes and smaller capacities for vehicles, while the C2, R2, and RC2 classes have wider time windows and larger capacities. Gehring and Homberger (1999) provided a large set of 300 new instances with 200, 400, 600, 800, and 1,000 customers for the VRPTW. The new instances were designed in the same way as Solomon's 100-customer problems. For *m*-VRPTW, test cases were generated from Solomon's VRPTW instances together with an integer for *m*.

Our algorithm was implemented in Java and run on 2.8 GHz Pentium 4 machines. According to the VRPTW survey paper (Bräysy and Gendreau 2005b), "Java is approximately 5–10 times slower than C, and there are not significant differences between C and FORTRAN regarding speed." Java is also considerably slower than C++ (an exact ratio is not possible since the execution speed of C/C++ is greatly affected by the compiler). According to the hardware factors used in Bräysy and Gendreau (2005b), our machines are more powerful than those reported in the literature. For example, a 2.8 GHz P4 machine is approximately 6.44 ($2.8 \times 168/73$ Bräysy and Gendreau 2005b) times faster than a Sun Ultra 10 machine, if we assume that a 2.8 GHz P4 is 2.8 times faster than a 1.0 GHz P3 (the actual ratio should be higher, but 2.8 is nevertheless a reasonable estimation). Overall, the superiority of our machines is compromised by the relative slowness of Java when comparing the computational efforts. All numbers used were floating-point numbers with an error tolerance of 10^{-6}.

4.1. Analyses of Algorithm Components and Parameter Choices

As described above, a number of elements, such as algorithmic components and parameter choices, may affect the performance of our VRPTW algorithm. Thus, a series of experiments was designed to investigate their impact. The various elements were examined sequentially from most to least important. When testing a certain algorithmic component or parameter choice for route minimization, the algorithm was run on Solomon's instances ten times with identical settings for the other components and parameters. The performance measure was taken to be the cumulative running time to find the best solution over all 56 test problems, denoted by CRTB. We chose CRTB, instead of the cumulative total running time over all instances, because CRTB is not affected by pre-defined terminating conditions.

The algorithm was tested with three different approaches for generating the initial solution, i.e., applying SWO for 50 iterations, applying Solomon's I1 heuristic only once, and assigning a separate vehicle to each customer to construct the initial solution. For the three approaches, the average CRTBs over ten runs are 357.61, 374.59, and 390.38 seconds, respectively. The results show that multiple iterations of SWO reduced the running time of the overall algorithm when compared to the other approaches, possibly because SWO provided good starting points for the route-minimization procedure, which is relatively slower than the SWO procedure. However, the differences, at most 9.16%, between the average CRTBs are not significantly large, which indicates that the strength of the algorithm mainly comes from the route-minimization procedure, instead of the initial-solution-construction procedure. Accordingly, the following initial solutions are always constructed by 50 iterations of SWO, although other approaches such as a parallel version of Solomon's I1 heuristic (Potvin and Rousseau 1993) may also be useful. The parameters in (1) and (2) are set to $\alpha_1 = 0.83$, $\alpha_2 = 0.17$,

$\mu = 0.93$, and $\lambda = 0.90$, by simply taking those values that produced the best result for Solomon's I1 heuristic.

In addition, we investigated a variant of the SWO approach, which incorporates the selection of seed customers into the SWO process. Before Solomon's I1 heuristic begins to construct a new route, all unserved customers are stored in a candidate list and sorted according to their priority factors in nonincreasing order. The seed customer is chosen to be the pth one in the candidate list, $p = random(0,1)^{e_3} \times s$, where s is the size of the candidate list. We experimented with e_3 in $\{5, 10, 20, 50\}$ (note that if $e_3 = 1$ the variant is essentially the same as the original SWO approach that selects the seed customer randomly among those unserved ones), and the average CRTBs over ten runs are 382.70, 357.33, 364.26, and 375.64 seconds, respectively. Although the result with $e_3 = 10$ is slightly better than 357.61 seconds, as achieved with random selection, the overall results are insufficient to support the superiority of the variant. Therefore, due to its simplicity, we kept the original SWO approach that selects the seed customer randomly.

To investigate the relative weights of mdl_{load} and mdl_{time}, the algorithm was run with η equal to 0.0, 0.5, 1.0, and 2.0. When $\eta = 0.0$, the penalty for violating the vehicle-capacity constraint is not considered. When $\eta = 1.0$, mdl_{load} is weighted the same as mdl_{time}. The average CRTBs for these η values are 436.36, 396.19, 357.61, and 490.78 seconds, respectively. The results show that the algorithm was improved by taking mdl_{load} into account, and that 1.0 is an appropriate value for η. (More experiments were conducted on the values of η. It appears that when $\eta > 1.0$ the performance deteriorates as η increases. However, no obvious pattern was observed when η was taken from $\{1/8, 2/8, \ldots, 7/8, 1\}$.) Thus, in view of these results, we simply fixed η at 1.0, for which the performance of the algorithm was among the best. After that, experiments were carried out with the p_e component removed from (10), and with the mft component in (12) removed. The average CRTBs are 362.58 and 384.69 seconds respectively, which are 1.39% and 7.57% worse than when these components were included. Thus, p_e and mft were kept in our algorithm due to their usefulness and simplicity in terms of both computational complexity ($O(1)$ for p_e and $O(N)$ for mft) and implementation effort.

To analyze the impact of the local search operators used in the route-minimization procedure, six sets of experiments were run. In the first set, experiments were carried out without the *Cross* operator, while *2-opt** was not applied in the second set. The average CRTBs are 694.89 and 256.92 seconds, respectively. In comparison with 357.61 seconds when using both *Cross* and *2-opt**, the results, as we expected, clearly show that *Cross* is a necessary component of the algorithm. However, it appears that performance was improved dramatically by removing *2-opt**. After analysis, we realized that *2-opt** is dominated by *Cross* when the number of customers in the two routes that are modified is not greater than $2L_{Cross}$. That is, the neighborhood of *Cross* contains that of *2-opt** if the routes are short. Thus, we applied *2-opt** only if $N/|\sigma| > 2L_{Cross}$, where $|\sigma|$ represents the number of routes in solution σ. Moreover, experiments were carried out without *Exchange*, *Relocate*, *2-opt* and *Or-opt* in the other four sets respectively, while *2-opt** was removed. The average CRTBs are 412.67, 428.38, 397.26, and 272.65 seconds, which are not superior to 256.92 seconds when the operators were included. Therefore, we kept all the single-route operators.

Similarly, when testing a certain algorithmic component or parameter for distance minimization, the algorithm was run on Solomon's instances once (not ten times as for route minimization, due to the relative slowness of distance minimization) with other components and parameters fixed. The performance criteria took into account the cumulative travel distance (CTD) and the cumulative running time (CRT) in minutes over all 39 test instances, excluding the 17 easy ones in C1 and C2. To investigate the impact of using multiple initial solutions on the overall solution quality, experiments were carried out with 1, 2, 5, 10, 20, 50, and 100 initial solutions. The results, given as $\langle \text{CTD}, \text{CRT} \rangle$ pairs, are $\langle 46, 313.88, 386.64 \rangle$, $\langle 45, 950.32, 399.63 \rangle$, $\langle 45, 552.44, 446.64 \rangle$, $\langle 45, 424.36, 495.24 \rangle$, $\langle 45, 329.86, 567.26 \rangle$, $\langle 45, 229.57, 827.21 \rangle$, and $\langle 45, 179.58, 1, 266.45 \rangle$, respectively. It is clear that solution quality was improved by using more initial solutions, although with more computation effort. In addition, the results with multiple initial solutions were generally better than those with a single initial solution, when given comparable running times. For example, without searching multiple solution regions, we obtained results such as $\langle 46, 250.10, 762.33 \rangle$ and $\langle 46, 236.25, 829.72 \rangle$, which were inferior to those achieved with the sample/intensify approach. Moreover, it appears that a value between 10 and 20 provided a good tradeoff point between solution quality and computation times. In view of these results, we chose to use 20 initial solutions for further experiments. After that, the result deteriorated to $\langle 45, 613.60, 468.64 \rangle$ when the GEC operator was removed. Compared to the previous result $\langle 45, 329.86, 567.26 \rangle$, it is clear that solution quality was improved by using the GEC, although at higher computation cost. For the classical operators, it appears that the results were somehow consistent with those obtained during route-minimization testing, and we achieved $\langle 45, 259.28, 781.98 \rangle$ with similar neighborhood choices except that *2-opt* was removed

Table 1 Route-Minimization Results for Solomon's VRPTW Instances

Class	Min.	Avg. (CNV)	Max.	Min.	Avg. (CRT)	Max.	Min.	Avg. (CRTB)	Max.
R1	143	143	143	6,000.08	6,001.81	6,034.06	38.20	131.04	471.19
C1	90	90	90	0.02	0.09	0.19	0.02	0.09	0.19
RC1	92	92	92	4,800.00	4,800.00	4,800.00	14.38	58.44	205.67
R2	30	30	30	4,204.60	4,240.59	4,372.94	5.55	43.06	175.37
C2	24	24	24	0.39	0.63	1.03	0.39	0.63	1.03
RC2	26	26	26	4,800.00	4,800.00	4,800.00	1.38	3.71	16.44
Overall	405	405	405	19,805.59	19,843.12	19,973.52	73.79	236.99	548.20

due to its uselessness. Meanwhile, the usefulness of E-opt was shown. An inferior result, $\langle 45, 416.45, 574.35\rangle$, was obtained due to its removal.

The final values of all other parameters, for which no particular insight was observed, are $\beta_1 = 0.50$, $\beta_2 = 0.50$, $e_1 = 50$, $\theta_1 = 0.50$, $\theta_2 = 0.50$, $e_2 = 10$, $k_1 = 20$, $k_2 = 5$, and $S = 16$. Lastly, the same algorithmic settings are used for Solomon's instances, Gehring and Homberger's instances, and m-VRPTW instances.

4.2. Experimental Results for Solomon's VRPTW Instances

To test the performance of our route-reduction algorithm on Solomon's benchmark problems, the algorithm was run on the instances 100 times with the time limit of 600 seconds. The algorithm terminated when the number of routes was reduced to the lower bound or the time limit was exceeded. The results and computational times for these runs are in Table 1, where CNV and CRT indicate the cumulative number of vehicles and cumulative running time over a class of test problems. CRTB indicates the cumulative running time to find the best solution over the instances. The results in Table 1 demonstrate that our route-reduction algorithm is highly effective with regard to the primary objective of the VRPTW. Each of the 100 runs was able to reduce the CNV to 405, while all, except one, of the previous methods listed in Table 6 of Bräysy and Gendreau (2005b) failed to do so with limited computational effort. Meanwhile, our algorithm is also highly efficient. The maximal CRTB of the 100 runs is 548.20 seconds, indicating an average of 9.79 (548.20/56) seconds. From Table 1, the CRT values are much larger than the CRTB values, indicating the need of a better lower bound. For 23 instances the algorithm was able to reduce the number of vehicles to the lower bound, while for the remaining 33 instances the algorithm terminated only when the time limit was exceeded. To illustrate further the effect of the time limit, we reported the CNV values with different cut-off times. If the time limits had been taken to be 1, 10, 60, 180, and 300 seconds, the average CNV over 100 runs would have been 419.61, 409.76, 405.90, 405.07, and 405.01, respectively. These values demonstrate the efficiency of the route-minimization algorithm, which was able to produce good results with only 10 or 60 seconds for each instance. Generally, 300 seconds (half of the ten-minute limit) is enough for an instance with 100 customers, since only one out of 5,600 (56×100) tests took more than 300 (specifically, 320.19) seconds to achieve the best solution. As far as we know, our algorithm is the most effective and efficient one in the literature to reduce the number of vehicles.

In the final algorithm, the time limit is determined by the problem size. More specifically, given $100n$ customers, the route-minimization procedure is applied n times with the time limit of $300n$ seconds. Then $20 - n$ more solutions, to be used in the distance-minimization process, are generated by applying the route-minimization procedure with the time limit of $60n$ seconds. For the distance-minimization process, the time limit is set to be $2{,}000n$ seconds. (The same settings are used for m-VRPTW instances, except that customer maximization is performed $10n$ times with the time limit of $300n$ seconds and $20 - 10n$ solutions are duplicated with the time limit of $60n$ seconds.) In Table 2, our experimental results are compared with earlier approaches listed in Table 6 of Bräysy and Gendreau (2005b) and those in recent papers including Bräysy et al. (2003a, b). Our overall algorithm was run once for each instance, and the average running time (ART) and the average running time to find the best solution (ARTB) are reported. The given running time includes the time to generate 20 initial solutions, to minimize the number of vehicles for these solutions, to reduce their distances and to minimize the distance of the best solution. In Table 2, the average number of vehicles and the average total distance are given for every test class, and CTD indicates the cumulative travel distance over all 56 test problems. According to Tables 1 and 2, our algorithm is one of the most successful approaches for the VRPTW. The CTD is 57,368 units with a gap of 0.31% between the best results achieved by earlier approaches in the literature. The comparison of distances is meaningful only for the same number of vehicles, which is the primary objective. Our algorithm is one of the two methods in Table 2 that are able to reach 405 for the CNV, and achieved a lower CTD. For the six problem classes,

Table 2 Comparison of Results Obtained with Limited Computational Effort

Author	R1	R2	C1	C2	RC1	RC2	CNV/CTD
Kontoravdis and Bard (1995)	12.58	3.09	10.00	3.00	12.63	3.50	427
	1,325.44	1,164.27	827.3	589.65	1,500.94	1,414.21	64,196
Rochat and Taillard (1995)	12.58	3.09	10.00	3.00	12.38	3.62	427
	1,197.42	954.36	828.45	590.32	1,369.48	1,139.79	57,120
Potvin and Bengio (1996)	12.58	3.09	10.00	3.00	12.63	3.38	427
	11,294.7	1,185.9	861.0	602.5	1,465.0	1,476.1	64,679
Taillard et al. (1997)	12.33	3.00	10.00	3.00	11.90	3.38	417
	1,220.35	1,013.35	828.45	590.91	1,381.31	1,198.63	58,614
Homberger and Gehring (1999)	11.92	2.73	10.00	3.00	11.63	3.25	406
	1,228.06	969.95	828.38	589.86	1,392.57	1,144.43	57,876
Gehring and Homberger (1999)	12.42	2.82	10.00	3.00	11.88	3.25	415
	1,198	947	829	590	1,356	1,144	56,946
Brandão (1999)	12.58	3.18	10.00	3.00	12.13	3.50	425
	1,205	995	829	591	1,371	1,250	58,562
Schulze and Fahle (1999)	12.50	3.09	10.00	3.00	12.25	3.38	423
	1,268.42	1,055.90	828.94	589.93	1,396.07	1,308.31	60,651
Gambardella et al. (1999)	12.38	3.00	10.00	3.00	11.92	3.33	418
	1,210.83	960.31	828.38	591.85	1,388.13	1,149.28	57,583
Kilby et al. (1999)	12.67	3.00	10.00	3.00	12.13	3.38	423
	1,200.33	966.56	830.75	592.24	1,388.15	1,133.42	57,423
Liu and Shen (1999)	12.17	2.82	10.00	3.00	11.88	3.25	412
	1,249.57	1,016.58	830.06	591.03	1,412.87	1,204.87	59,318
Bräysy (2003)	11.92	2.73	10.00	3.00	11.50	3.25	405
	1,222.12	975.12	828.38	589.86	1,389.58	1,128.38	57,710
Gehring and Homberger (2001)	12.00	2.73	10.00	3.00	11.50	3.25	406
	1,217.57	961.29	828.63	590.33	1,395.13	1,139.37	57,641
Li and Lim (2001)	12.08	2.91	10.00	3.00	11.75	3.25	411
	1,215.14	953.43	828.38	589.86	1,385.47	1,142.48	57,467
Bent and Van Hentenryck (2004b)	12.17	2.73	10.00	3.00	11.63	3.25	409
	1,203.84	980.31	828.38	589.86	1,379.03	1,158.91	57,707
Berger et al. (2003)	12.17	2.73	10.00	3.00	11.88	3.25	411
	1,251.40	1,056.59	828.50	590.06	1,414.86	1,258.15	60,200
Rousseau et al. (2002)	12.08	3.00	10.00	3.00	11.63	3.38	412
	1,210.21	941.08	828.38	589.86	1,382.78	1,105.22	56,953
Homberger and Gehring (2005)	12.08	2.82	10.00	3.00	11.50	3.25	408
	1,211.67	950.72	828.45	589.96	1,395.93	1,135.09	57,422
Bräysy et al. (2002)	12.00	2.73	10.00	3.00	11.50	3.25	406
	1,220.20	970.38	828.38	589.86	1,398.76	1,139.37	57,796
Ibaraki et al. (2005)	11.92	2.73	10.00	3.00	11.63	3.25	406
	1,220.02	961.64	828.38	589.86	1,378.72	1,132.17	57,480
Bräysy et al. (2003b)	12.00	2.73	10.00	3.00	11.50	3.25	406
	1,212.83	**956.43**	828.38	589.86	1,387.22	1,126.40	57,361
Bräysy et al. (2003a)	12.00	2.73	10.00	3.00	11.50	3.25	406
	1,220.20	970.38	828.38	589.86	1,398.76	1,139.37	57,796
Our algorithm	11.92	2.73	10.00	3.00	11.50	3.25	405
	1,213.61	961.05	828.38	589.86	**1,385.56**	**1,121.82**	**57,368**

Configuration: Pentium IV 2.8 GHz, Java, 1 run, ART = 26.28 min, ARTB = 12.18 min.

Note. Improvements over the previous best results are indicated in bold.

our algorithm generated the best average results for R1, RC1, and RC2, and matched the best published results for the easy instances in C1 and C2. The consistently good performance on problem instances with different characteristics demonstrates the robustness of our algorithm.

To investigate further the efficiency of the algorithm, the gradual improvements obtained after each phase are in Figure 1. A single run of Solomon's I1 heuristic produced $\langle 495, 80, 484\rangle$, which was improved to $\langle 437, 72, 011\rangle$ after 50 iterations of SWO. ART for the initial solution stage was only 0.31 second. With a nearly negligible running time, the SWO approach was shown to be very efficient in generating good initial solutions for the next stage. The route-minimization procedure took an ART of 177.24 seconds to reach $\langle 405, 69, 575\rangle$. The efficiency of the route-minimization algorithm becomes more obvious when considering that the minimal number of vehicles was achieved with no more than 46.30 seconds for each instance, although the algorithm was allowed to run up to 300 seconds. After route

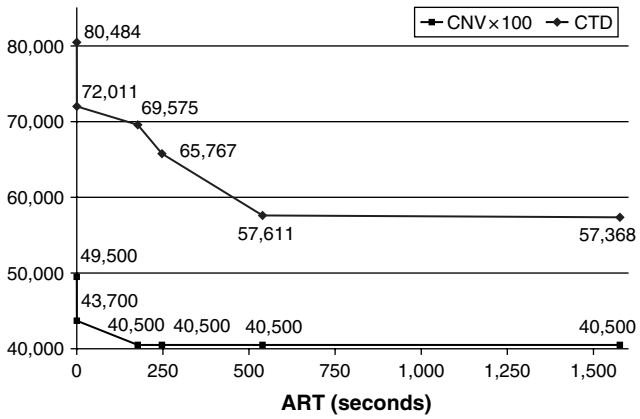

Figure 1 **Gradual Improvements Obtained After Each Algorithm Phase**

minimization, the algorithm further reduced the distance and achieved the final result ⟨405, 57,368⟩ with an ART of 1,576.75 seconds. As illustrated in Figure 1, the distance-minimization algorithm achieved good results with short computation time and was able to improve the solution quality further with longer running time.

In Table 3, the best results achieved by our algorithm are compared with the best published results (available at http://web.cba.neu.edu/~msolomon/problems.htm) at the time of writing (March 20, 2004). New best known results for two of Solomon's instances, R203 and RC202, were found by our algorithm and are given in the appendix. According to Table 3, our algorithm generated equivalent or better solutions for 38 instances. For the other 18 instances, nine are very close (within 0.01%) to the best published results, eight are at most 1% worse, and one is more than 1% worse. Meanwhile, the CTD for the 56 instances is 57,233, which is the least we know among the approaches that reached 405 for CNV. Therefore, our algorithm is at least comparable to previous methods in terms of travel distance.

Lastly, the experimental results also showed that the algorithm performed very well on Gehring and Homberger's extended VRPTW instances and *m*-VRPTW instances. These results are reported and analyzed in the Online Supplement to this paper on the journal's website.

5. Conclusion

In this paper, we proposed a two-stage algorithm for the well-known vehicle routing problem with time windows. The algorithm first minimizes the number of vehicles with an ejection pool to hold temporarily unserved customers, which enables the algorithm to go through the infeasible solution space. Then it minimizes the total travel distance using a multi-start iterated hill-climbing algorithm with classical and new operators including generalized ejection chains,

Table 3 **Comparison of Best Results**

Instance	Previous best results		Our best results		Gap	
R101	19	1,645.79	19	1,650.80	5.01	0.30%
R102	17	1,486.12	17	1,486.12	0	0%
R103	13	1,292.68	13	1,292.68	0	0%
R104	9	1,007.24	9	1,007.31	0.07	0.01%
R105	14	1,377.11	14	1,377.11	0	0%
R106	12	1,251.98	12	1,252.03	0.05	0%
R107	10	1,104.66	10	1,104.66	0	0%
R108	9	960.88	9	960.88	0	0%
R109	11	1,194.73	11	1,194.73	0	0%
R110	10	1,118.59	10	1,118.84	0.25	0.02%
R111	10	1,096.72	10	1,096.73	0.01	0%
R112	9	982.14	9	987.24	5.1	0.52%
C101	10	828.94	10	828.94	0	0%
C102	10	828.94	10	828.94	0	0%
C103	10	828.06	10	828.06	0	0%
C104	10	824.78	10	824.78	0	0%
C105	10	828.94	10	828.94	0	0%
C106	10	828.94	10	828.94	0	0%
C107	10	828.94	10	828.94	0	0%
C108	10	828.94	10	828.94	0	0%
C109	10	828.94	10	828.94	0	0%
RC101	14	1,696.94	14	1,696.95	0.01	0%
RC102	12	1,554.75	12	1,554.75	0	0%
RC103	11	1,261.67	11	1,261.67	0	0%
RC104	10	1,135.48	10	1,135.48	0	0%
RC105	13	1,629.44	13	1,629.44	0	0%
RC106	11	1,424.73	11	1,424.73	0	0%
RC107	11	1,230.48	11	1,230.48	0	0%
RC108	10	1,139.82	10	1,139.82	0	0%
R201	4	1,252.37	4	1,252.37	0	0%
R202	3	1,191.70	3	1,191.70	0	0%
R203	3	939.54	3	**939.50**	**−0.04**	0%
R204	2	825.52	2	832.14	6.62	0.80%
R205	3	994.42	3	994.43	0.01	0%
R206	3	906.14	3	906.14	0	0%
R207	2	893.33	2	896.88	3.55	0.40%
R208	2	726.75	2	726.82	0.07	0.01%
R209	3	909.16	3	909.16	0	0%
R210	3	939.34	3	939.37	0.03	0%
R211	2	892.71	2	904.78	12.07	1.35%
C201	3	591.56	3	591.56	0	0%
C202	3	591.56	3	591.56	0	0%
C203	3	591.17	3	591.17	0	0%
C204	3	590.60	3	590.60	0	0%
C205	3	588.88	3	588.88	0	0%
C206	3	588.49	3	588.49	0	0%
C207	3	588.29	3	588.29	0	0%
C208	3	588.32	3	588.32	0	0%
RC201	4	1,406.91	4	1,406.94	0.03	0%
RC202	3	1,367.09	3	**1,365.65**	**−1.44**	−0.11%
RC203	3	1,049.62	3	1,058.33	8.71	0.83%
RC204	3	798.41	3	798.46	0.05	0.01%
RC205	4	1,297.19	4	1,297.65	0.46	0.04%
RC206	3	1,146.32	3	1,146.32	0	0%
RC207	3	1,061.14	3	1,061.14	0	0%
RC208	3	828.14	3	828.71	0.57	0.07%
Total	405	57,192.04	405	57,233.23	41.19	

Note. Improvements over the previous best results are indicated in bold.

which have the ability to search a larger neighborhood. We applied the algorithm to Solomon's 56 VRPTW instances and Gehring and Homberger's 300 extended instances. Experimental results show that the algorithm with integration of the generalized minimal delay definition and the EP is highly effective and efficient in reducing the number of vehicles. It is also very competitive in term of distance minimization and generated some new best known results for the Solomon benchmark instances.

The algorithm for the VRPTW was extended to solve the *m*-VRPTW. The modifications were made to address the impossibility of serving all customers due to the limited number of vehicles. The extended algorithm intends to serve more easy customers, while leaving the customers that are hard to serve in the EP. The experimental results show that the extended algorithm greatly improved solution quality in term of the number of served customers when compared with previous approaches.

Acknowledgments

The authors thank the referees and editors for their invaluable comments.

Appendix

Below are the new best known results to Solomon's instances R203 and RC202 at the time of writing (March 20, 2004):

Vehicle	Customers	Routes	Capacity	Distance
1	25	27 52 18 45 46 36 64 11 62 88 31 30 76 3 79 78 9 35 34 29 68 12 26 13 58	368	256.442941
2	44	94 95 97 92 98 37 42 57 15 43 14 44 38 86 16 85 59 99 96 6 84 83 8 48 47 49 19 10 63 90 32 66 20 70 7 82 17 61 91 100 93 5 60 89	654	358.397777
3	31	50 33 81 65 71 51 1 69 39 67 23 72 73 21 40 53 87 2 41 22 75 56 74 4 55 25 54 24 80 77 28	436	324.662602
3	100		1,458	939.503320
1	39	91 92 95 85 63 33 28 26 27 29 31 34 30 62 67 71 61 41 38 40 81 90 84 49 20 83 66 56 50 32 89 48 21 24 25 77 75 58 52	560	526.865370
2	32	65 82 12 14 47 15 11 98 69 88 73 16 87 99 53 78 79 8 6 46 2 55 68 54 43 35 37 72 96 93 94 80	541	388.619962
3	29	45 5 3 1 42 39 36 44 64 23 19 18 76 51 22 57 86 9 10 97 59 74 13 17 7 4 60 100 70	623	450.159699
3	100		1,724	1,365.645031

References

Ahuja, R. K., J. B. Orlin, D. Sharma. 2001. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Math. Programming* **91** 71–97.

Ahuja, R. K., J. B. Orlin, S. Pallottino, M. P. Scaparra, M. G. Scutellà. 2004. A multi-exchange heuristic for the single-source capacitated facility location problem. *Management Sci.* **50** 749–760.

Bent, R., P. Van Hentenryck. 2004a. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Oper. Res.* **52** 977–987.

Bent, R., P. Van Hentenryck. 2004b. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Sci.* **38** 515–530.

Berger, J., M. Barkaoui, O. Bräysy. 2003. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR* **41** 179–194.

Brandão, J. 1999. Metaheuristic for the vehicle routing problem with time windows. S. Voss, S. Martello, I. H. Osman, C. Roucairol, eds. *Meta-Heuristics—Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Boston, MA, 19–36.

Bräysy, O. 2003. A reactive variable neighborhood search algorithm for the vehicle routing problem with time windows. *INFORMS J. Comput.* **15** 347–368.

Bräysy, O., M. Gendreau. 2005a. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Sci.* **39** 104–118.

Bräysy, O., M. Gendreau. 2005b. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Sci.* **39** 119–139.

Bräysy, O., G. Hasle, W. Dullaert. 2002. A fast local search algorithm for the vehicle routing problem with time windows. Working paper, SINTEF Applied Mathematics, Department of Optimization, Trondheim, Norway.

Bräysy, O., J. Berger, M. Barkaoui, W. Dullaert. 2003a. A threshold accepting metaheuristic for the vehicle routing problem with time windows. *Central Eur. J. Oper. Res.* **11** 369–387.

Bräysy, O., G. Hasle, J. Berger, M. Barkaoui. 2003b. Systematic diversification metaheuristic for the vehicle routing problem with time windows. Working paper, SINTEF Applied Mathematics, Department of Optimization, Trondheim, Norway.

Campbell, A., M. Savelsbergh. 2005. Decision support for consumer direct grocery initiatives. *Transportation Sci.* **39** 313–327.

Gambardella, L. M., E. Taillard, G. Agazzi. 1999. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization*. McGraw-Hill, London, 63–76.

Gehring, H., J. Homberger. 1999. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. *Proc. of EUROGEN'99*. University of Jyväskylä, Jyväskylä, Finland, 57–64.

Gehring, H., J. Homberger. 2001. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Asia-Pacific J. Oper. Res.* **18** 35–47.

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13** 533–549.

Glover, F. 1991. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Working paper, College of Business and Administration, University of Colorado, Boulder, CO.

Glover, F. 1992. New ejection chain and alternating path methods for traveling salesman problems. O. Balci, R. Sharda, S. Zenios, eds. *Computer Science and Operations Research: New Developments in Their Interfaces*. Pergamon Press, Oxford, UK, 449–509.

Homberger, J., H. Gehring. 1999. Two evolutionary meta-heuristics for the vehicle routing problem with time windows. *INFOR* **37** 297–318.

Homberger, J., H. Gehring. 2005. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *Eur. J. Oper. Res.* **162** 220–238.

Ibaraki, T., S. Imahori, M. Kubo, T. Masuda, T. Uno, M. Yagiura. 2005. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Sci.* **39** 206–232.

Joslin, D. E., D. P. Clements. 1999. "Squeaky wheel" optimization. *J. Artificial Intelligence Res.* **10** 353–373.

Kilby, P., P. Prosser, P. Shaw. 1999. Guided local search for the vehicle routing problem with time windows. S. Voss, S. Martello, I. H. Osman, C. Roucairol, eds. *Meta-Heuristics—Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Boston, MA, 473–486.

Kontoravdis, G. A., J. F. Bard. 1995. Grasp for the vehicle routing problem with time windows. *ORSA J. Comput.* **7** 10–23.

Koskosidis, Y. A., W. B. Powell, M. M. Solomon. 1992. An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Sci.* **26** 69–85.

Lau, H. C., M. Sim, K. M. Teo. 2003. Vehicle routing problem with time windows and a limited number of vehicles. *Eur. J. Oper. Res.* **148** 559–569.

Lenstra, J. K., A. H. G. Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* **11** 221–227.

Li, H., A. Lim. 2001. Large scale time-constrained vehicle routing problems: A general metaheuristic framework with extensive experimental results. Technical report, School of Computing, National University of Singapore, Singapore.

Liu, F. H., S. Y. Shen. 1999. A route-neighborhood-based metaheuristic for vehicle routing problem with time windows. *Eur. J. Oper. Res.* **118** 485–504.

Or, I. 1976. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Ph.D. thesis, Northwestern University, Evanston, IL.

Östergård, P. R. J. 2002. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.* **120** 197–207.

Potvin, J. Y., S. Bengio. 1996. The vehicle routing problem with time windows part ii: Genetic search. *INFORMS J. Comput.* **8** 165–172.

Potvin, J. Y., J. M. Rousseau. 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Eur. J. Oper. Res.* **66** 331–340.

Rochat, Y., E. Taillard. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1** 147–167.

Rousseau, L. M., M. Gendreau, G. Pesant. 2002. Using constraint-based operators to solve the vehicle routing problem with time windows. *J. Heuristics* **8** 43–58.

Schulze, J., T. Fahle. 1999. A parallel algorithm for the vehicle routing problem with time window constraints. *Ann. Oper. Res.* **86** 585–607.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35** 254–265.

Taillard, E., P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Sci.* **31** 170–186.