A Two-Stage Heuristic for the Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles

Andrew Lim and Xingwen Zhang
Dept of IEEM, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{iealim,xwzhang}@ust.hk

Abstract

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 and the vehicle capacity constraint. 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. In this paper, we propose a two-stage algorithm for the m-VRPTW. The algorithm first maximizes the number of customers served with an Ejection Pool to hold temporarily unserved customers. Then it minimizes the total travel distance using a multi-start iterated hill-climbing algorithm with classical and new operators including Generalized Ejection Chains. The experimental results showed consistently good performance of the algorithm when compared with other methods.

1. Introduction

The Vehicle Routing Problem with Time Windows is an important problem in logistics. In the VRPTW, let $G = (V, E)$ be a directed graph, where $V = \{0, 1, ..., N\}$ is the node set and $E = \{(i, j) : 0 \leq i, j \leq N, i \neq j\}$ is the edge set. Node 0 is the depot and $C = \{1, ..., N\}$ 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 $(0 \leq i \leq N)$, 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 a 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 cost, c_{ij}, and a travel time, t_{ij}. In the following, we assume that $c_{ij} = t_{ij}$ for all the edges. A set of identical vehicles with capacity Q is given. The total service demand of the customers in a route is not allowed to exceed the vehicle capacity, Q. The primary objective of the VRPTW is to find a minimal number of routes to service all the customers exactly once while satisfying the capacity and time window constraints. For the same number of routes, the secondary objective is often to minimize the total distance traveled or to minimize the total route duration.

In the VRPTW, the number of vehicles is unlimited. Recently, Lau *et al.* [10] 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.

In this paper, we present a two-stage algorithm for the m-VRPTW, in which the algorithm focuses on maximizing the number of customers served in the first step and on minimizing the total travel distance in the second step. The algorithm for maximizing the number of customers served 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 into the EP. Those customers that are difficult to serve are left in the EP. The experimental results show that the approach is highly effective in maximizing the number of customers served. The algorithm for minimizing the total distance is an iterated multi-start hill-climbing algorithm with classical and newly defined operators including Generalized Ejection Chains (GEC) with which not only a single customer but also a sequence of customers is ejected from an existing route and re-inserted into the original route or a differ-

0-7695-2268-8/05/$20.00 (C) 2005 IEEE

ent route. The ejection and insertion procedures are repeated to reach a new solution with less total distance. The experimental results show consistently good performance of the algorithm when compared with other methods.

2. The Algorithm for m-VRPTW

Our algorithm consists of three main components, i.e., the procedure to generate initial solutions, the procedure to maximize the number of customers served and the procedure to minimize the total distance. The initial solutions are generated with Squeaky Wheel Optimization (SWO) [8, 9], where Solomon's I1 [12] heuristic is used to construct a feasible solution in each iteration of SWO. Then, the algorithm focuses on maximizing the number of customers served in 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. The Initial Solutions

The initial solutions are constructed by iteratively applying Solomon's well-known I1 heuristic within an SWO framework. With Solomon's I1 heuristic, initially an unserved customer is selected as the seed customer of a route. The seed customer is chosen to be the geographically farthest one in relation to the depot, the one with the lowest 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 the customers are served.

The I1 heuristic method defines two criteria, c_1 and c_2, to select a customer to insert into the route currently under construction. Let $\{v_0$=0,v_1,v_2,...,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 given in Equations 1 and 2, respectively, 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 for the service to begin at j after u is inserted between i and j, while t_j is the beginning time of service before the insertion. For more details, refer to [12, 2]. The selected u^* is then inserted into the current route at the position that leads to the lowest cost, $c_1(u^*)$.

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

where $\alpha_1 + \alpha_2 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0, \mu \geq 0$.

$$c_2(u) = \lambda \times d_{0u} - c_1(u), \lambda \geq 0. \qquad (2)$$

To achieve initial solutions of better quality, Solomon's I1 heuristic is invoked for a certain number of times within an SWO framework. In SWO [8, 9], a greedy algorithm is used to construct a solution. The solution found by the greedy heuristic is analyzed, and the priority factors of problem elements are updated with knowledge derived from the analysis. Those elements that, if improved, are likely to improve the objective function are assigned more blame, i.e., their priorities are increased more. The new priorities are then used to guide the greedy algorithm to construct the next solution. This Construct/Analyze/Prioritize [8] cycle continues until the maximal number of iterations has been reached or the time limit is exceeded.

In our SWO framework for the VRPTW, each customer is assigned a priority factor. The priority factor of each customer is initially one and is dynamically increased from iteration to iteration during the execution of the SWO algorithm. Therefore, the range of priorities is [1,$+\infty$]. The priorities are used to guide Solomon' I1 heuristic to select the next customer to insert. More precisely, for a customer u with a priority factor, p_u, the new criteria, $c'_1(u)$ and $c'_2(u)$, are defined in Equation 3 and Equation 4, respectively, while the definitions of $c_1(v_q, u, v_{q+1})$ and $c_2(u)$ are given in Equation 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 would be inserted earlier by the modified Solomon's I1 heuristic. After all the 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, i.e., their priorities are increased more. More specifically, the increase is directly proportional to the percentage of unused vehicle capacity, while inversely proportional to the number of customers served by the vehicle. 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. The best solution found by the procedure, which serves the most customers with the least travel distance by its m most densely packed routes, is taken to be the initial solution for later improvement.

$$c'_1(u) = \min_{q=0,\ldots,m} \frac{c_1(v_q, u, v_{q+1})}{p_u} \qquad (3)$$

$$c'_2(u) = \begin{cases} c_2(u) \times p_u & \text{if } c_2(u) \geq 0 \\ \frac{c_2(u)}{p_u} & \text{otherwise.} \end{cases} \qquad (4)$$

2.2. The Procedure to Maximize the Number of Customers Served

Given an initial solution generated by SWO, only the m routes with largest numbers of customers are kept. All other routes are removed from the solution and those customers served in these routes are added into the EP, a data structure to hold temporarily unserved customers. In each iteration, the procedure selects an unserved customer in the EP with the least $mdl(v,\sigma)$ value as defined in Equation 19, and inserts it into an existing route with the solution σ. The $mdl(v,\sigma)$ value, updated whenever σ is modified, is a heuristic measurement of the difficulty of inserting customer v. It is an extension of the *minimal delay* (mdl) defined in [6] and [1]. The new definition takes into account the penalty of violating the vehicle capacity constraint, in addition to that of violating time window constraints. Generally, a solution of m-VRPTW contains some unserved customers because only a limited number of vehicles is available. By selecting the customer with minimal $mdl(v,\sigma)$, the algorithm intends to insert those customers that are more easily served by the existing m vehicles, while leaving the customers that are difficult to serve in the EP. The pseudo-code of the algorithm is given by the following procedure:

procedure maximize_customer ($routes$**)**
  delete extra routes with the fewest customers from $routes$
  add the customers in the deleted routes into the EP
  while (EP is not empty) and (not to terminate) **do**
    select a customer u from the EP
    determine the insertion position of u in $routes$
    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 EP
    end while
    apply $Exchange$, $Relocate$, *2-opt*, *Or-opt*, *2-opt**, $Cross$ to $routes$
  end while

After the candidate customer, denoted as u, is selected, the algorithm needs to determine the target route and the position to insert u. The insertion cost, $c(i,u,j)$, of inserting u between two consecutive customers, i and j, is defined as in Equation 5, and d_{iu}, d_{uj} and d_{ij} are 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 the increased service delay. Because inserting u between i and j would possibly 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 Equation 6 and 7, 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 Equation 7, s_u is the given service time of u and la_j is the latest arrive time of j such that the time window constraints of j and those customers after j will not be violated.

$$c(i,u,j) = \beta_1 \times \delta_d + \beta_2 \times (dl_u + dl_j), \quad (5)$$
where $\beta_1 + \beta_2 = 1, \beta_1 \geq 0, \beta_2 \geq 0$,
$\delta_d = (d_{iu} + d_{uj} - d_{ij})$,
$$dl_u = max(t_u - l_u, 0), \quad (6)$$
$$dl_j = max(t_u + s_u + d_{uj} - la_j, 0). \quad (7)$$

If there exist some feasible insertion positions for u, these feasible positions make up of 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 non-decreasing order. The final insertion position is chosen to be the p-th element in the candidate list, and $p = random(0,1)^e \times s$ where $e > 1$ 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. Meanwhile, the introduction of randomization helps to prevent repeated Insert/Kick cycles for the same set of customers.

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 is feasible. In each iteration of the kick procedure, only those customers whose removals will potentially benefit the feasibility of the route, denoted as *relevant customers*, are considered to be removed. Let $\{0,v_1,v_2,...,v_{i-1},v_i = u,v_{i+1},...,v_m,v_{m+1}=0\}$ 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. There exist three different situations to consider. First, if the earliest time to service u in the route is larger than the latest allowed arrival time of u, i.e., $t_u > l_u$, then those customers whose removals will reduce t_u are *relevant customers*. Secondly, if customer u is served feasibly but some customers after it are not feasible, let $v_j (j > i)$ 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*. Lastly, 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 can not be feasibly inserted into any other route are no longer relevant. Let v_x be any of the re-

maining relevant customers. This customer's kick saving, $ks(v_x)$, is defined in Equation 8. 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.

The definition of kick saving, ks, considers the distance reduction by kicking v_x from the current route and the cost needed to re-insert v_x into another route. In Equation 8, δ_d is the distance reduction by kicking v_x out, and $mdl(v_x, r, \sigma)$ is the cost needed to re-insert v_x. For m-VRPTW, generally not all customers could be served with m or fewer vehicles. When the number of unserved customers in the EP is over a predefined control parameter, S, the algorithm estimates that it is not possible to serve all the customers. Since it is inevitable that some customers are unserved, the algorithm intends not to serve those customers that are hard to serve by removing 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 some other routes. In our computational study, the value of S was fixed to 13 for all benchmark instances.

In the following, σ is the set of routes in the partial solution, r is the current route, and r'=$\{0,v'_1,v'_2,...,v'_m,v'_{m+1}=0\}$ represents any other route in σ. $mdl(v, r')$ is the cost of inserting a customer, v, into r'. The penalty of overload, mdl_{load}, is *approximated* as in Equation 11 by the additional amount of time needed to accommodate the excessive load. 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 $(0, v'_1, ..., v'_p)$ (p_l), the violation of e_v induced by relocating v before $(v'_{p+1}, v'_{p+2}, ..., v'_m, 0)$ (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 $(0, v'_1, ..., v'_p)$ is feasible and $la_{v'_{p+1}}$ is the latest arrival time of v_{p+1} to make $(v'_{p+1}, v'_{p+2}, ..., v'_m, 0)$ feasible.

$$ks(v_x) = \begin{cases} \theta_1 \times \delta_d - \theta_2 \times mdl(v_x, r, \sigma) \text{ if EP} \leq S \\ \theta_1 \times \delta_d + \theta_2 \times mdl(v_x, r, \sigma) \text{ otherwise} \end{cases} \quad (8)$$

$$\text{where } \theta_1 + \theta_2 = 1, \theta_1 \geq 0, \theta_2 \geq 0, S > 1,$$

$$\delta_d = (d_{v_{x-1}v_x} + d_{v_x v_{x+1}} - d_{v_{x-1}v_{x+1}}),$$

$$mdl(v, r, \sigma) = \min_{r' \in \sigma \wedge r' \neq r} mdl(v, r') \quad (9)$$

$$mdl(v, r') = \max(mdl_{load}(v, r'), mdl_{time}(v, r')) \quad (10)$$

$$mdl_{load}(v, r') = \begin{cases} 0 \text{ if } load(r') + q_v \leq Q \\ \frac{(l_0 - e_0) \times (load(r') + q_v - Q)}{load(r') + q_v} \text{ otherwise} \end{cases} \quad (11)$$

$$load(r') = \sum_{v' \in r'} q_{v'} \quad (12)$$

$$mdl_{time}(v, r') = \min_{0 \leq p \leq m'} c_t(v, v'_p, r') \quad (13)$$

$$c_t(v, v'_p, r') = p_l(v, v'_p, r') + p_e(v, v'_p, r') + p_b(v, v'_p, r') \quad (14)$$

$$p_l(v, v'_p, r') = max(ed_{v'_p} + d_{v'_p v} - l_v, 0), \quad (15)$$

$$p_e(v, v'_p, r') = max(e_v - (la_{v_{p+1}} - d_{vv'_{p+1}} - s_v), 0), \quad (16)$$

$$p_b(v, v'_p, r') = max(ed_v + d_{vv'_{p+1}} - la_{v_{p+1}}, 0), \quad (17)$$

$$ed_v = max(ed_{v'_p} + d_{v'_p v}, e_v) + s_v. \quad (18)$$

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* [11] sequentially to improve a single route, and uses *2-opt** and $Cross$ [13] for a pair of routes. These operators are classical and widely used in the VRPTW [2]. Let r=$\{0,v_1,...,v_{i-1},v_i,v_{i+1},..., v_{j-1},v_j,v_{j+1},...,v_m,0\}$ and r'=$\{0,v'_1,...,v'_{p-1},v'_p,v'_{p+1},...,v'_{q-1},v'_q,v'_{q+1},...,v'_n,0\}$ be two different routes. In the following, these operators are informally defined followed by their resulting route or routes.

$Exchange$: Exchange customers i and j. $\{0,v_1,...,v_{i-1},v_j, v_{i+1},...,v_{j-1},v_i,v_{j+1},...,v_m,0\}$.

$Relocate$: Move customer i after j. $\{0,v_1,...,v_{i-1},v_{i+1}, ...,v_{j-1},v_j,v_i,v_{j+1},...,v_m,0\}$.

2-opt: Replace edges (v_i,v_{i+1}) and (v_j,v_{j+1}) by edges (v_i,v_j) and (v_{i+1},v_{j+1}), and reverse the direction of customers between v_{i+1} and v_j. $\{0,v_1,...,v_{i-1},v_i,v_j,v_{j-1}, ..., v_{i+1},v_{j+1},...,v_m,0\}$.

Or-opt: Relocate a chain of l consecutive customers. If $l = 2$, $\{0,v_1,...,v_{i-1},v_{i+2},..., v_{j-1},v_j, v_i,v_{i+1},v_{j+1},...,v_m,0\}$.

*2-opt**: Exchange the end portions of two routes. $\{0,v_1,...,v_{i-1},v_i,v'_{p+1},...,v'_{q-1},v'_q,v'_{q+1},...,v'_n,0\}$ and $\{0,v'_1,..., v'_{p-1}, v'_p,v_{i+1},..., v_{j-1},v_j,v_{j+1},...,v_m,0\}$.

$Cross$: Exchange two consecutive segments of two routes. $\{0,v_1,...,v_{i-1},v'_p,v'_{p+1},...,v'_{q-1},v'_q, \quad v_{j+1},...,v_m,0\}$ and $\{0,v'_1,...,v'_{p-1},v_i,v_{i+1},..., v_{j-1},v_j,v'_{q+1},...,v'_n,0\}$.

The primary objective of the local search for m-VRPTW is to reduce the *weighted sum* of mdl of the customers in the EP. The mdl of an unserved customer, v, in the EP, is defined as in Equation 19 where $mdl(v, r')$ is defined in Equation 10. The mdl value estimates how easily the customer can be inserted back into a route in the partial solution. The $mdl(v, \sigma)$ values of each customer v in the EP are sorted in non-decreasing order. Let $\{m_1, m_2, ..., m_n\}$ be the sorted array of $mdl(v, \sigma)$ values. Equation 20 is the way used to calculate the weighted sum in our computational study. A desired characteristic of the weighting function is that the weight decreases as the $mdl(v, \sigma)$ value increases. It comes from the facts that the unserved customers are inserted into existing routes sequentially, and that in each iteration the procedure selects the unserved customer with the least $mdl(v, \sigma)$ value to insert. Therefore, the customers with less $mdl(v, \sigma)$ are more likely to be re-inserted and they should contribute more to the weighted sum, while the customers with larger $mdl(v, \sigma)$ have less chance to be served and should contribute less.

$$mdl(v, \sigma) = \min_{r' \in \sigma} mdl(v, r') \quad (19)$$

$$ws = \sum_{i=1}^{n} \frac{m_i}{i} \quad (20)$$

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 of two routes for a pair of routes. The mft of route r'={0,v'_1,v'_2,...,v'_m,v'_{m+1}=0} is defined in Equation 21, where $ed_{v'_p}$ and $la_{v'_{p+1}}$ are the same as described above. The mft value estimates the maximal usable time of a route for possible future insertions. Therefore, maximizing the mft would help to make more space for some customer to be inserted into the route in the future.

$$mft(r') = \max_{0 \le p \le m'} ft(v'_p, v'_{p+1}, r') \quad (21)$$

$$ft(v'_p, v'_{p+1}, r') = la_{v'_{p+1}} - ed_{v'_p} - d_{v'_p v'_{p+1}} \quad (22)$$

In case the primary and secondary objective values of a solution are the same as those of a neighboring solution in the hill-climbing process, the algorithm tries to reduce 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 [10], 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 m-VRPTW algorithm, the EP is the key part of the procedure to maximize the number of customers served, and the algorithm fully focuses on inserting the unserved customers in the EP back to the routes. The algorithm inserts the unserved customers in the EP that are easy to serve back into the routes, and kicks customers that are hard to serve back to the EP. Meanwhile, the local search to minimize the weighted sum of mdl of customers in the EP generally favors the customers that are easy to insert. The algorithm greatly improves the solution quality in term of the number of customers served when compared with other methods, which is demonstrated by the experimental results in the following section.

2.3. The 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 Chain (GEC)-based operator to minimize the total distance traveled. The pseudo-code of the algorithm is given by the procedure *reduce_distance*, which is similar to the iterated local search in [7]. In the distance minimization process, the number of customers served is not changed, i.e., 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 less 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** operator to reduce $dis(r1) + dis(r2)$
          apply hill-climbing with $cross$ operator 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. The 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 the E-opt optimizes the ordering for a segment at most L_e in length and thus limits the computational time needed. 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 in length, and then the local search with classical operators are 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) [4, 5]. 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 the 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 of removing and inserting 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. During the process, the algorithm needs to choose the position to insert the unrouted segment and decide which segment is ejected. To limit the computational complexity, only the k best choices of insertion and ejection that lead to partial solutions (with the newly ejected segments unserved) with least total distance are accepted. Then, the algorithm proceeds with the k branches in depth-first order. The GEC stops when it finds a chain that leads to a feasible solution with less total distance or when it reaches the maximal allowed depth, d_{max}. In our computational study, we set $k = 5$ and $d_{max} = 2$.

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* [3] with *2-opt** and *cross* operators for a small number of iterations. Then, the hill-climbing algorithm restarts with the new initial solution. The tabu list stores edges that have been created within the preceding number of tabu iterations defined by the tabu length. A move is tabu, i.e., forbidden, if and only if the edges to be removed are in the tabu list.

To our understanding, two properties are desired for a good method to perturb previous solutions, i.e., the quality of the resulting solutions must be reasonably good and it is not easy for the search procedure to return to previous solutions. Compared to some simple methods like a random $cross$ operation [7], the *tabu search* method to perturb previous solutions has the advantages that it generates solutions of good quality as it is a meta-heuristic method searching for good solutions by itself, and that it is difficult for the hill-climbing algorithm to undo the changes made to the previous solutions.

2.4. The Integrated Algorithm for the m-VRPTW

In our distance minimization procedure, the solutions generated by the local search are always feasible and the numbers of served customers are fixed. This approach has its 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 served customers and the capacity and time window constraints. The restriction would introduce more difficulties to 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 m-VRPTW randomly generates a certain number (20 in our computational study) of initial solutions for each problem instance in the way described in the previous section and maximizes the number of served customers for each of them. Then, the algorithm applies the distance minimization procedure to these solutions only for a small number of iterations or with a short running time. The best solution (or solutions in case the algorithm intends to search multiple solution regions) found during the primitive distance minimization process is taken as the seed solution, and the algorithm applies the distance minimization procedure to it again with a longer running time. The rationale behind the approach is to sample better solution regions in the solution space and to intensify the search only within good regions.

3. The Experimental Results

For the VRPTW, Solomon's 56 benchmark problems [12] are most widely used in the literature for the comparison of different heuristics. For m-VRPTW, test cases were generated from Solomon's VRPTW instances together with an integer for the value of m. Our algorithm was implemented in Java and run on a 2.4G Pentium 4 machine. Meanwhile, all numbers used were floating-point numbers with the error tolerance of 10^{-6}. The control parameters were set as $\alpha_1 = 0.83$, $\mu = 0.93$, $\lambda = 0.90$, $\beta_1 = 0.50$, $e = 50$, $\theta_1 = 0.50$, $S = 13$ and $L_e = 4$.

3.1. The Effectiveness of the Customer Maximization Procedure

For the m-VRPTW, our algorithm is compared with earlier methods including LST [10] and WLLG [14] in Table 1. The average running time (ART) for the 63 instances is 2.86 minutes, while the average running time to find the best solution (ARTB) is 0.35 minutes. The results in Table 1 demonstrate the strength of our algorithm to maximize the number of customers served. For 16, i.e., 25.40%, of the instances, our method achieved new best solutions (highlighted in the table), with the improvement of one or two customers. The overall performance of our algorithm is also superior to previous methods, with 17 or 21 more customers served. Meanwhile, the algorithm was tested on

	Vehicles	10	9	8	7	6	5	4
C101	LST	100	92	84	75	66	57	47
	WLLG	100	92	84	75	66	57	47
	Our Alg.	100	92	84	75	66	57	47
C102	LST	100	99	92	84	72	60	49
	WLLG	100	98	92	83	72	60	49
	Our Alg.	100	99	**93**	84	72	60	49
C103	LST	100	99	93	84	72	60	48
	WLLG	100	99	93	84	72	60	48
	Our Alg.	100	99	**94**	84	**73**	60	**49**
C104	LST	100	99	94	84	72	60	48
	WLLG	100	99	94	84	73	61	49
	Our Alg.	100	99	**95**	84	72	61	48
C105	LST	100	92	84	75	66	57	47
	WLLG	100	92	84	75	66	57	47
	Our Alg.	100	**93**	84	75	66	57	47
C106	LST	100	93	86	77	68	58	47
	WLLG	100	93	86	77	68	58	47
	Our Alg.	100	**94**	86	76	67	58	47
C107	LST	100	93	85	76	66	57	47
	WLLG	100	93	85	76	67	57	47
	Our Alg.	100	**94**	**86**	**77**	67	57	47
C108	LST	100	96	87	78	69	59	48
	WLLG	100	96	87	78	69	59	48
	Our Alg.	100	**98**	**89**	**80**	**70**	59	48
C109	LST	100	99	92	82	72	60	48
	WLLG	100	99	92	82	72	61	49
	Our Alg.	100	99	**93**	**83**	72	61	49
Total	LST=4854, WLLG=4858, Our Alg.=4875							

Table 1. Number of customers served for C1

other Solomon instances, and the best solutions from multiple runs are given in Table 2 with new best solutions highlighted.

3.2. Comparison of Experimental Results

Table 3 shows our computational results for m-VRPTW instances generated from Solomon's benchmark data together with the values of m, taken to be the largest number of vehicles that are insufficient to serve all 100 customers. Since none of previous methods for m-VRPTW has reported their travel distance, no comparison is possible. To further test the strength of our approach, we ran it with standard VRPTW instances. In Table 4, the best results achieved by our algorithm for VRPTW are compared with the best published results (available at http://web.cba.neu.edu/∼msolomon/problems.htm). New best-known results for two of Solomon's instances, R203 and RC202, were found by our algorithm. According to Table 4, our algorithm generated equivalent or better solutions in 36 instances. For the other 20 instances, 9 are very close (within 0.01%) to the best published results, 9 are at most 1% worse, and 2 are more than 1% worse. Meanwhile, the total travel distance for the 56 instances is 57249, which is the least as we know of among the approaches that reached 405 for the total number of vehicles. Therefore, our algorithm is at least comparable to previous methods in terms of total distance traveled.

m	19	18	17	16	15									
R101	100	**99**	**98**	**96**	**93**									
R102	-	-	100	99	98									
m	14	13	12	11	10	9	8	7	6	5	4	3	2	1
R101	**90**	86	**82**	**77**	**71**	65	59	53	46	39	32	25	17	9
R102	**96**	**94**	**92**	**89**	**85**	**81**	**75**	**68**	**61**	53	45	**36**	25	13
R103	-	100	**99**	**98**	**95**	**91**	**85**	**77**	**69**	**59**	**49**	**39**	**27**	14
R104	-	-	-	-	-	100	**93**	**84**	**74**	**64**	52	41	28	14
R105	100	**99**	**97**	92	**87**	**81**	**75**	**67**	59	50	**42**	32	22	12
R106	-	-	100	**99**	**95**	**90**	**83**	**75**	**67**	**58**	**49**	37	**26**	14
R107	-	-	-	-	100	**96**	**90**	81	71	62	**51**	39	27	14
R108	-	-	-	-	-	100	**94**	**86**	**75**	**65**	53	41	**28**	**15**
R109	-	-	-	100	**97**	**90**	**83**	**75**	66	57	47	**37**	25	13
R110	-	-	-	-	100	**95**	**87**	**78**	69	**60**	49	**38**	26	14
R111	-	-	-	-	100	**96**	**89**	80	**71**	61	**51**	39	27	14
R112	-	-	-	-	-	100	92	83	73	62	51	39	27	14
C101	-	-	-	-	100	92	84	75	66	57	47	36	25	13
C102	-	-	-	-	100	99	**93**	84	72	**61**	49	37	25	13
C103	-	-	-	-	100	99	**95**	**85**	**73**	**61**	**49**	**37**	25	12
C104	-	-	-	-	100	99	**95**	84	72	61	49	37	25	12
C105	-	-	-	-	100	**93**	84	75	66	57	47	36	25	13
C106	-	-	-	-	100	**94**	86	77	68	58	47	36	25	13
C107	-	-	-	-	100	**94**	**86**	**77**	67	57	47	36	25	13
C108	-	-	-	-	100	**98**	**89**	**80**	**70**	59	48	36	25	13
C109	-	-	-	-	100	99	**93**	**83**	72	61	49	37	25	13
RC101	100	**98**	**94**	**90**	84	77	71	63	55	48	40	31	21	11
RC102	-	-	100	**97**	**93**	**86**	**79**	71	62	54	**45**	34	23	12
RC103	-	-	-	100	**99**	**93**	**86**	**78**	**68**	**58**	47	36	24	12
RC104	-	-	-	-	100	**97**	**90**	**82**	71	**62**	50	38	26	13
RC105	-	100	98	**96**	**91**	**85**	**78**	70	62	53	**44**	34	23	12
RC106	-	-	-	100	95	**89**	**81**	71	62	53	**44**	33	23	12
RC107	-	-	-	100	**99**	**92**	84	75	**67**	56	46	35	24	12
RC108	-	-	-	-	100	**96**	**88**	**79**	69	58	**48**	36	25	13

Table 2. Number of customers served for C1, R1 and RC1

4. Conclusion

In this paper, we proposed a two-stage algorithm for the Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles. The algorithm first maximizes the number of customers served with an Ejection Pool to hold temporarily unserved customers. The algorithm intends to serve more easy customers, while leaving the customers that are hard to serve in the EP. 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 have the ability to search a larger neighborhood. The experimental results have shown that the algorithm greatly improved the solution quality in term of the number of customers served when compared with previous approaches. It is also very competitive in term of distance minimization.

References

[1] R. Bent and P. V. Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *To Appear in Transportation Science*, 2004.

[2] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *To Appear in Transportation Science*, 2004.

Instance	m	No. of Customers	Distance
R101	18	99	1615.67
R102	16	99	1458.25
R103	12	99	1268.22
R104	8	93	880.15
R105	13	99	1380.84
R106	11	99	1270.06
R107	9	96	1013.11
R108	8	94	826.48
R109	10	97	1147.40
R110	9	95	1035.88
R111	9	96	1050.13
R112	8	92	858.56
C101	9	92	727.05
C102	9	99	1008.16
C103	9	99	899.97
C104	9	99	862.73
C105	9	93	840.56
C106	9	94	942.28
C107	9	94	896.55
C108	9	98	1062.65
C109	9	99	912.88
RC101	13	97	1560.92
RC102	11	97	1425.26
RC103	10	99	1298.93
RC104	9	97	1084.36
RC105	12	98	1538.90
RC106	10	95	1284.78
RC107	10	98	1215.59
RC108	9	95	1055.61
Configuration: Pentium IV 2.4 GHz, Java, 1 run, ART=31.66 min			

Table 3. Results for m-VRPTW

[3] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[4] F. Glover. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. *Working Paper, College of Business and Administration, University of Colorado, Boulder*, 1991.

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

[6] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37:297–318, 1999.

[7] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for the vehicle routing problem with general time window constraints. *To Appear in Transportation Science*, 2004.

[8] D. E. Joslin and D. P. Clements. "squeaky wheel" optimization. In *Proceedings of AAAI-98*, pages 340–346, 1998.

[9] D. E. Joslin and D. P. Clements. "squeaky wheel" optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

[10] H. C. Lau, M. Sim, and K. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148:559–569, 2003.

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

Instance	Previous Best Results		Our Best Results		Gap	
R101	19	1645.79	19	1650.80	5.01	0.30%
R102	17	1486.12	17	1486.12	0	0%
R103	13	1292.68	13	1292.68	0	0%
R104	9	1007.24	9	1007.31	0.07	0.01%
R105	14	1377.11	14	1377.11	0	0%
R106	12	1251.98	12	1252.03	0.05	0%
R107	10	1104.66	10	1104.66	0	0%
R108	9	960.88	9	960.88	0	0%
R109	11	1194.73	11	1197.42	2.69	0.23%
R110	10	1118.59	10	1118.84	0.25	0.02%
R111	10	1096.72	10	1096.73	0.01	0%
R112	9	982.14	9	991.06	8.92	0.91%
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	1696.94	14	1696.95	0.01	0%
RC102	12	1554.75	12	1554.75	0	0%
RC103	11	1261.67	11	1261.67	0	0%
RC104	10	1135.48	10	1135.48	0	0%
RC105	13	1629.44	13	1629.44	0	0%
RC106	11	1424.73	11	1424.73	0	0%
RC107	11	1230.48	11	1230.95	0.47	0.04%
RC108	10	1139.82	10	1139.82	0	0%
R201	4	1252.37	4	1252.37	0	0%
R202	3	1191.70	3	1191.70	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	905.44	12.11	1.36%
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	1406.91	4	1406.94	0.03	0%
RC202	3	1367.09	3	**1365.65**	**-1.44**	-0.11%
RC203	3	1049.62	3	1058.33	8.71	0.83%
RC204	3	798.41	3	798.46	0.05	0.01%
RC205	4	1297.19	4	1297.65	0.46	0.04%
RC206	3	1146.32	3	1146.32	0	0%
RC207	3	1061.14	3	1061.14	0	0%
RC208	3	828.14	3	828.71	0.57	0.07%
Total	405	57192.04	405	57248.77	56.73	

Table 4. Comparison of best results for VRPTW

[12] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, 1987.

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

[14] F. Wang, A. Lim, Z. Li, and S. Guo. Improved grasp with smoothed dynamic tabu search for vehicle routing problem with both time window and limited number of vehicles. *Working paper, Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology, Hong Kong*, 2003.