DISCRETE AND CONTINUOUS
DYNAMICAL SYSTEMS
SUPPLEMENT 2007

Website: www.AIMSciences.org

pp. **240–249**

SOLVE THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS VIA A GENETIC ALGORITHM

Yaw Chang and Lin Chen

Mathematics and Statistics
University of North Carolina at Wilmington
601 S. College Rd
Wilmington, NC 28403, USA

Abstract. The objective of vehicle routing problem (VRP) is to deliver a set of customers with known demands on minimum-cost routes originating and terminating at the same depot. A vehicle routing problem with time windows (VRPTW) requires the delivery made in a specific time window given by the customers. Prins (2004) proposed a simple and effective genetic algorithm (GA) for VRP. In terms of average solution, it outperforms most published tabu search results. We implement this hybrid GA to handle VRPTW. Both the implementation and computation results will be discussed.

1. **Introduction.** Vehicle routing problem (VRP) is defined on a undirected network $G = (V, E)$ with a node set $V = \{0, 1, \ldots, n\}$ and an edge set E. Node 0 is a depot. Each other node $i > 0$ represents a customer with a known service time d_i and each edge has a non-negative travel cost $c_{ij} = c_{ji}$. The VRP consists of determining a set of m vehicle trips of minimum total cost, such that each vehicle starts and ends at the depot, each client is visited exactly once, and the total demand handled by any vehicle does not exceed the vehicle's capacity W.

The vehicle routing problem is a well-known integer programming problem which falls into the category of NP-Hard problems. For such problems it is often desirable to obtain approximated solutions, so they can be found quickly enough and are sufficiently accurate for the purpose. Usually this task is accomplished by using various heuristic methods, which rely on some insight into the nature of the problem.

The VRP arises naturally as a central problem in the fields of transportation, distribution, and logistics [4]. In some market sectors, transportation means a high percentage of the value is added to goods. Therefore, the utilization of computerized methods for transportation often results in significant savings ranging from 5% to 20% in the total costs, as reported in [11].

The vehicle routing problem with time windows (VRPTW) is the generalization of the vehicle routing problem (VRP) with the additional time constraints. In these problems, the service of a customer can begin within a time window $[e_i, l_i]$ defined by customer i. The vehicle can not arrive earlier than time e_i and no later than time l_i. The regular VRP can be viewed as the situation $e_i = 0$ and $l_i = \infty$ for all $1 \leq i \leq n$. When $0 < e_i < l_i < \infty$, it is also known as a double-sided time window while a single-sided time window is refer to either $e_i = 0$ or $l_i = \infty$, but not both. A

2000 *Mathematics Subject Classification.* Primary: 58F15, 58F17; Secondary: 53C35.

Key words and phrases. vehicle routing problem, vehicle routing problem with time window, genetic algorithm.

vehicle arriving earily than the earliest service time of a customer will incur waiting time. This penalizes the scheduling either in the direct waiting cost or the increase number of vehicles.

Due to its complexities and usefulness in real life, the VRPTW continue to draw attention from researchers and has been a well-known problem in network optimization. Early surveys of solution methods for VRPTW can be founded in Golden and Assad [6, 7], Desrochers *et al.*[5], and Solomon and Desrosiers [10]. The recent developments can be found in Ioannou, Kritikos, and Prastacos [8], Brăysy and Gendreau [2, 3], and Brăysy, Dullaert, and Genderau [1]. The goal of this paper is to adapt and implement Prins' algorithm to handle VRP with time windows. For simplicity, we will only consider a single-sided time window with $e_i > 0$ and $l_i = \infty$ for all i.

2. Problem Formulation.

Before we formally give the mathematical formulation of the vehicle routing problem with time windows, we need to introduce some basic terminologies from graph theory.

Definition 1. An undirected graph G is an ordered pair $G = (V, E)$ where $V = \{0, 1, 2, ..., n\}$ is the set of vertices and $E = \{(i, j)|i, j \in V\}$ is the set of unordered pairs of distinct vertices, called edges.

Definition 2. Let $G = (V, E)$ be a graph, a path P is a sequence of vertices $i_1, i_2, ..., i_k$, such that $(i_j, i_{j+1}) \in E$, for all $1 \leq j \leq k - 1$.

Definition 3. Let $G = (V, E)$ be a graph. G is called a connected graph if there is a path between every pair of vertices .

Now let G be a given undirected graph corresponding to a vehicle routing problem where vertex 0 corresponds to the depot and vertex i corresponds to customer i, for $1 \leq i \leq n$. For convenience, let us define the following known constants and variables:

- c_{ij} : the cost associated with edge (i, j), the cost could be the travel distance, the travel time, or the travel cost from customer i to customer j. For simplicity, we assume $c_{ij} = c_{ji}$. If there is no edge between vertex i and j, $c_{ij} = \infty$.
- b_i : the vehicle arrival time at customer i.
- d_i : the service time for customer i.
- q_i : the demand for customer i.
- e_i : the earliest arrival time for customer i.
- L: maximal operation time for each vehicle.
- Q: the vehicle capacity.
- R_i: a sequence $(i_1, i_2, ...i_k)$ corresponding to a route where $i_j \in V$, such that a particular vehicle will serve customers $(i_1, i_2, ...i_k)$ according to the order of the sequence.
- W_{i_j}: the waiting time for vehicle i at customer j.

Consider a given route $R_i = (i_1, i_2, ..., i_k)$, the actual arriving time b_{i_j} can be defined by

$$b_{i_j} = b_{i_{j-1}} + d_{j-1} + c_{i_{j-1}, i_j},$$

i.e., the arrival time at customer i_{j-1}, plus the service time at customer i_{j-1} and the travel time from customer i_{j-1} to customer i_j. To ensure the earliest arrival

time constraint, it can be modified as

$$b_{i_j} = \max\{e_{i_j}, b_{i_{j-1}} + d_{i_{j-1}} + c_{i_{j-1},i_j}\}.$$

The waiting time for vehicle i at customer j can be defined as

$$W_{i_j} = \max\{0, e_{i_j} - b_{i_{j-1}} - d_{i_{j-1}} - c_{i_{j-1},j}\}.$$

An additional constraint

$$b_{i_k} + d_{i_k} + c_{i_k,0} \leq L,$$

will ensure a vehicle will not operate over the operation time allowed, ie. R_i is a feasible route. The total travel time for route R_i can be defined as

$$C(R_i) = c_{0,i_1} + \sum_{j=2}^{k}(c_{i_{j-1},i_j}) + \sum_{j=1}^{k} W_{i_j} + \sum_{j=1}^{k} d_{i_j} + c_{i_k,0}.$$

A solution S is a collection of routes $R_1, R_2, ..., R_l$, such that each customer will be covered by exactly one route R_i. The Vehicle Routing with time windows can be formulated as the following optimization problem

$$\min_S C(S) = \sum_{i=1}^{l} C(R_i).$$

3. **Chromosomes and Evaluation.** Similar to most GA that a chromosome S is a permutation of n positive integers, such that each integer is corresponding to a customer *without trip delimiters*. A chromosome may be broken into several different routes. For example, a chromosome with 5 customers, $S = (1, 2, 3, 4, 5)$ may be broken into $R_1 = (1, 2)$ and $R_2 = (3, 4, 5)$, or $R_1 = (1, 2, 3)$ and $R_2 = (4, 5)$; etc.. Christian Prins [9] proposed an *optimal splitting procedure* to get the best solution (among all the possible routes) respecting to a given chromosome. We will review the main idea of this splitting procedure in Section 3.1 and discuss our implementation in Section 3.2.

3.1. **A Splitting Algorithm by Prins.** Because a chromosome can be split into many different routes, Prins [9] proposed a splitting procedure which can find an optimal splitting i.e a trip delimiter so that the total cost is minimized. The main idea can be described as follows. Without loss of generality, let $S = (1, 2, 3, ..., n)$ be a given chromosome. Consider an auxiliary graph $H = (\bar{V}, \bar{E})$ where $\bar{V} = \{0, 1, 2, ..., n\}$. An arc $(i, j) \in \bar{E}$ if

$$\bar{E_{ij}} = c_{0,i+1} + \sum_{k=i+1}^{j-1} (d_k + C_{k,k+1}) + d_j + c_{j,0} \leq L; \sum_{k=i+1}^{j} q_k \leq Q. \qquad (1)$$

Then $\bar{E_{ij}}$ is the total travel cost(time) for the route $(i + 1, i + 2, ..., j)$. An optimal split for S corresponds to shortest path P from vertex 0 to vertex n in H.

The following example (Prins, [9]) demonstrates the construction. The first graph of Fig. 1 shows a sequence S=(a, b, c, d, e). The number associates with each edge is the travel time. Let us assume the service time (delivery time) is 0 and the demands are 5, 4, 4, 2 and 7, respectively. The capacity of the vehicle is 10 and the maximum operation time is 90. Then the auxiliary graph H is the second graph in Fig. 1. The edge in H with weight 55 corresponds to the travel time of the trip $(0, a, b, 0)$. The weight associated with the other edges are similarly defined. The shortest path

from 0 to e is $(0, b, c, e)$ with minimal total travel time 205. The last graph gives the optimal result with three trips.

The auxiliary graph helps us understand the idea how to split a given chromosome S into optimal trips. But in practice, we do not have to construct such graph H. It can be done by a labeling algorithm and a splitting procedure [9]. Let $S = (1, 2, ..., n)$ be a given chromosome. Two labels V_j and P_j for each vertex j in S are computed. V_j is the cost of the shortest path from node 0 to node j in H, and P_j is the predecessor of j on this path. The minimal cost is given at the end by V_n. For any given i, note that the increment of j stops when L or Q are exceeded. The labeling algorithm can be described as follow:

$V_0 := 0$
for $i := 1$ to n do $V_i := +\infty$ endfor
for $i := 1$ to n do
    cost $:= 0$; load $:= 0$; $j := i$
    repeat
        load $:=$ load $+ q_{S_j}$
        if $i = j$ then
            cost $:= c_{0,S_j}, +d_{S_j} + c_{S_{j,0}}$
        else
            cost $:=$ cost $- c_{S_{j-1},0} + c_{S_{j-1},S_j} + d_{S_j} + c_{S_{j,0}}$
        endif
        if (cost $\le L$) and (load $< Q$) then
            if $V_{i-1} +$ cost $< V_j$ then
                $V_j := V_{i-1} +$ cost
                $P_j := i - 1$
            endif
            $j := j + 1$
        endif
    until $(j > n)$ or (cost $> L$) or (load $> Q$)
endfor

3.2. **Implementation for Time Windows.** Recall $\bar{E_{ij}}$ in (1) represents the total travel time for route $(i + 1, i + 2, ..., j)$. To incorporate the earliest arrival time e_i, the total travel time $\bar{E_{ij}}$ for the same route can be calculated by

$\bar{E_{ij}} = c_{0,i+1} + \sum_{k=i+1}^{j-1}(W_i + d_k + c_{k,k+1}) + W_j + d_j + c_{j,0}$,

where W_i is the waiting time defined earlier. If $\bar{E_{ij}} \le L$, then the arc (i, j) exists in the auxiliary graph H, and the shortest path from 0 to n in H corresponding to an optimal split for S.

To accommodate this modification, the splitting procedure can be modified as follows:

$V_0 := 0$
for $i := 1$ to n do $V_i := +\infty$ endfor
for $i := 1$ to n do
    cost := 0; load := 0; $j := i$
    repeat
        load := load + q_{S_j}
        if $i = j$ then
            cost := $\max[c_{0,S_j}, e_{S_j}] + d_{S_j} + c_{S_{j,0}}$
        else
            cost := $\max[\text{cost} - c_{S_{j-1},0} + c_{S_{j-1},S_j}, e_{S_j}] + d_{S_j} + c_{S_{j,0}}$
        endif
        if (cost $\leq L$) and (load $< Q$) then
            if $V_{i-1} + \text{cost} < V_j$ then
                $V_j := V_{i-1} + \text{cost}$
                $P_j := i - 1$
            endif
            $j := j + 1$
        endif
    until $(j > n)$ or (cost $> L$) or (load $> Q$)
endfor

Here c_{0,S_j} is the regular arriving time at customer S_j, and e_{S_j} is the earliest arrival time at customer S_j. If $e_{S_j} < c_{0,S_j}$, we will pick e_{S_j}. Otherwise, we will pick the regular arrival time c_{0,S_j}.

4. **CROSSOVER.** As mentioned earlier, the reproductive process in GA can be done either by crossover or by mutation procedures. A crossover (or recombination) operation is performed upon the selected chromosomes. We will use the *order crossover* which can be explained as follow: First, two chromosomes are randomly selected from the initial population and the least-cost one becomes the first parent P_1. Two cutting sites i and j are randomly selected in P_1, here i=4 and j=6. Then, the substring $P_1(i)...P_1(j)$ is copied into $C(i)...C(j)$. Finally, P_2 is swept circularly from $j+1$ onward to complete C with the missing nodes. C is also filled circularly from $j+1$. The other child maybe obtained by exchanging the roles of P_1 and P_2.

Table 1 demonstrates the process. Let $i = 4$ and $j = 6$, so $C(4) = P_1(4)$, $C(5) = P_1(5)$, and $C(6) = P_1(6)$. Now $C(7)$ should equal to $P_2(7)$. Because $C(6) = 3$, we shift to $P_2(8)$, so $C(7) = P_2(8) = 2$. Now $C(8)$ should equal to $P_2(9)$. Again $C(5) = 10 = P_2(9)$, we will skip $P_2(9)$ and let $C(8) = P_2(10) = 1$. Repeat this process until C is constructed.

5. **MUTATION.** The classical genetic algorithm framework must be hybridized with some kind of mutation procedure. In this paper, we simply adopt the local search procedure in [9].

A child C produced by crossover could be improved by local search with a fixed mutation rate p_m which stands for the rate or the probability of mutation and is a fundamental parameter in genetics and evolution. Here p_m is the measure of the impact of mutations on the child C. The mutation procedure can be described as follows. A chromosome C is converted into a VRPTW solution. Then for all possible pairs of distinct vertexes (u, v), the following simple moves are tested. Let

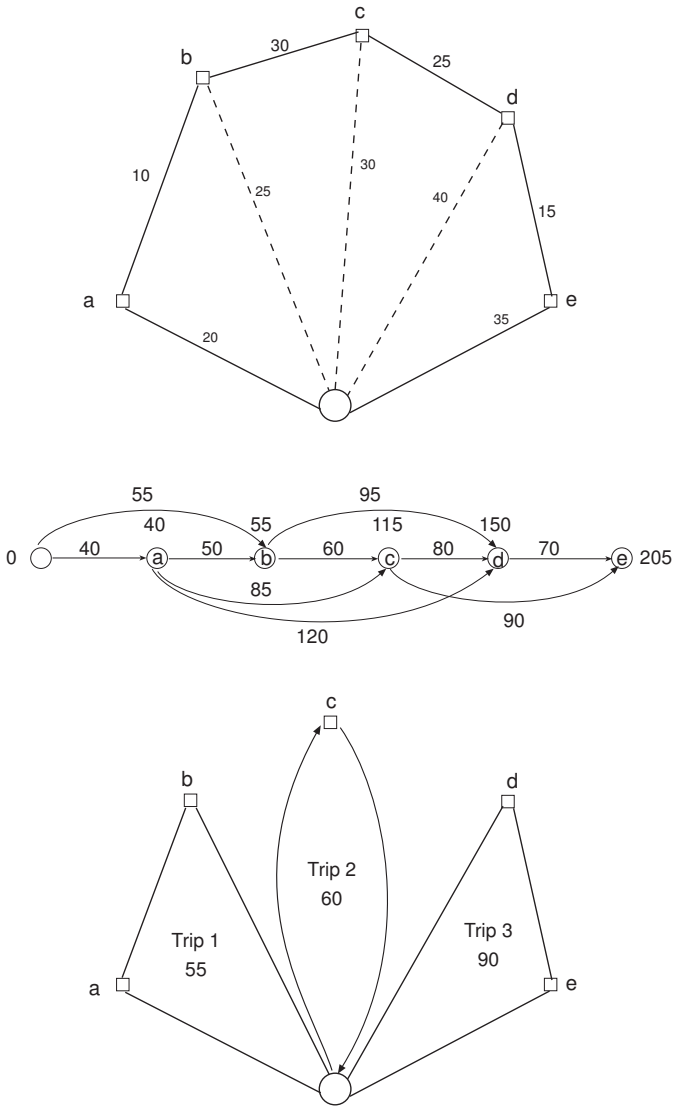

FIGURE 1. An example of chromosome evaluation

x and y be the successors of u and v in their respective trips. The following rules are used to create new chromosomes.

- M1. Remove u from C and insert it after v, ie, the new sequence is $(...v, u, ...)$.
- M2. If x is a client, remove (u, x) from C then insert (u, x) after v, ie, the new sequence is $(...v, u, x, ...)$.
- M3. If x is a client, remove (u, x) from C then insert (x, u) after v, for example, the new sequence is $(...v, x, u, ...)$.
- M4. Swap u and v, for example, the new sequence is $(...v, ..., u, ...)$.
- M5. If x is a client, swap (u, x) and v, ie, the new sequence is $(...v, ...u, x, ...)$.
- M6. If x and y are clients, swap (u, x) and (v, y), ie, the new sequence is $(...v, y, ...u, x, ...)$.

Rank	1	2	3	4	5	6	7	8	9	10
				i=4		j=6				
				↓		↓				
P1	9	8	7	5	10	3	6	2	1	4
P2	9	8	7	6	5	4	3	2	10	1
C	7	6	4	5	10	3	2	1	9	8

TABLE 1. Example of crossover

- M7. If u and v are in the same trip, replace (u, x) and (v, y) by (u, v) and (x, y), ie, the new sequence is $(...u, v, ...x, y, ...)$.
- M8. If u and v are not in the same trip, replace (u, x) and (v, y) by (u, v) and (x, y), ie, the new sequence is $(...u, v, ...x, y, ...)$.
- M9. If u and v are not in the same trip, replace (u, x) and (v, y) by (u, y) and (x, v), ie, the new sequence is $(...u, y, ...x, v, ...)$.

Let $\bar{C}$ be a new chromosome produced by the mutation procedure. If $\bar{C}$ is better than the worst in the current population, $\bar{C}$ will be added into the current population, and the worst one will be moved. If $\bar{C}$ is better than the current best solution, we consider the current mutation process as a productive mutation.

6. **THE IMPLEMENTED ALGORITHM FOR VRPTW.** The population is implemented as an array Π of σ (σ is the population size) chromosomes, always sorted in increasing order of cost to ease the basic genetic algorithm iteration. Thus, the best solution is Π_1.

Clones (identical solutions) are forbidden in Π to ensure a better dispersal of solutions. This also allows a higher mutation rate p_m by local search, giving a more aggressive genetic algorithm. To avoid comparing chromosomes in details and to speed-up clone detection, we impose a stricter condition: the costs of any two solutions generated by crossover or mutation must be spaced at least by a constant $\triangle > 0$.

A population satisfying the following condition will be said to be well-spaced. The simplest form with $\triangle = 1$ (solutions with distinct integer costs) was already used in an efficient Genetic Algorithm. That is,

$$\forall P_1, P_2 \in \Pi : P_1 \neq P_2 \Rightarrow |F(P_1) - F(P_2)| \geq \triangle$$

where P_1 and P_2 are the parents we select from the initial population, and $F(P_1)$ and $F(P_2)$ are the costs of P_1 and P_2

The stopping criterion for GA is either after reaching a maximum number of general iterations, α_{max} (number of crossovers that do not yield a clone) or a maximal number of iterations without improving the best solution, β_{max}. Let σ be the population size, p_m be the mutation rate, and space $\Delta = 5$. The GA can be described as follows:

Step 1: (Initial population)

Randomly generate σ chromosomes.
Step 2: (Sort Initial population)
Sort the initial population according to the total travel time of each chromosome.
Step 3: (setup counter)
Set α, β=0
Step 4: (Crossover)
Let $\alpha = \alpha + 1$. Select two parents (P_1) and (P_2) by binary tournament. Apply Crossover to (P_1, P_2) and select one child C at random.
Step 5: (Mutation)
Compare r (randomly generated) and p_m. If $r > p_m$ then go to Step 6. Otherwise, mutation is applied to the child and C' is generated. The split procedure is then used to get the solution of C'.
Step 6: (Productive iteration)
Since Π_1 is the best solution of the population, if $(F(C') - F(\Pi_k)) \geq \Delta, 1 \leq k \leq \sigma$, then let $\beta = 0$ and shift C' to re-sort population. Otherwise, $\beta = \beta + 1$.
Step 7: (Check stopping criterion)
If $\alpha = \alpha_{max}$ or $\beta = \beta_{max}$, stop. Otherwise, go to Step 4.

7. **COMPUTATIONAL RESULTS.** In general, some good solutions must be computed using heuristics to build initial populations. Because the testing problems size are relative small, we just build the initial populations randomly without using any heuristics. This will be implement when we test our algorithm on standard VRPTW instances.

The Genetic Algorithm was implemented in the Visual Basic script, on a Pentium-4 PC clocked at 2.7 GHz under the operating system Windows XP. We ran two different randomly generated data sets with n=10. For each data set we ran 3 different population sizes with 3 different mutation rates. For all the cases, we set α_{max}=1000 and β_{max}=100. The best solution was calculated by complete enumeration for both date sets.

In the following tables, the first column is the population size, the second column is the mutation rate, and the third column is the best solution in the initial population with the number of trips in the parentheses. Columns 4 and 5 give the final values of α and β, respectively. Column 6 gives the final solution produced by the GA with number of trips in the parentheses. Column 7 is the best solution with the number of trips in the bracket. The last column is the relative error.

Table 2, sorted in increasing order of population size, shows very encouraging results. When the population size equals 30 and 100, the mutation rate equals to 0.3, and our solutions reach the best solution with β reaching the maximal number of iterations without improving the best solution, which means that the mutation procedure was applied to the child C at least 100 times.

In general, when the mutation rate increases, the relative errors are decreasing, and the number of trips is decreasing. Only two relative errors are greater than 5%.

One exception that we notice is that when the population size equals 50 and the mutation rate equals 0.1, our solution also reaches the best solution that has two trips instead of one.

Table 3 is based on a second data set, sorted in increasing order of population size. It shows results similar to those in table 1. When population size equals 50 and 100, mutation rate equals to 0.3. Our solutions also reach the best solution with β reaching the maximal number of iterations without improving the best solution.

Population size n	Mutation Rate r	Original Solution	α	β	Final Solution	Runtime t(seconds)	Best Solution	Relative Error
30	0.1	1380(3)	928	100	520(1)	90	490(1)	6.12%
	0.2	1550(2)	990	100	510(1)	91	490(1)	4.08%
	0.3	1280(3)	551	100	490(1)	98	490(1)	0%
50	0.1	1050(2)	1000	87	490(2)	90	490(1)	0%
	0.2	990(2)	1000	98	510(1)	90	490(1)	4.08%
	0.3	1360(1)	996	100	500(1)	98	490(1)	2.04%
100	0.1	1270(3)	1000	98	520(2)	90	490(1)	6.12%
	0.2	1823(4)	864	100	500(1)	93	490(1)	4.08%
	0.3	1900(4)	980	100	490(1)	106	490(1)	0%

TABLE 2. Computational results for date set 1

Population size n	Mutation Rate r	Original Solution	α	β	Final Solution	Runtime t(seconds)	Best Solution	Relative Error
30	0.1	1787(4)	506	100	1022(2)	85	962(2)	6.3%
	0.2	2629(5)	1000	75	997(2)	88	962(2)	3.6%
	0.3	1860(4)	579	100	987(2)	96	962(2)	2.6%
50	0.1	1640(4)	583	100	987(2)	90	962(2)	2.6%
	0.2	1744(4)	573	100	987(2)	89	962(2)	2.6%
	0.3	1702(4)	516	100	962(2)	97	962(2)	0%
100	0.1	1787(4)	506	100	1022(2)	89	962(2)	6.3%
	0.2	1441(3)	1000	83	970(2)	93	962(2)	0.83%
	0.3	1518(3)	1000	71	962(2)	97	962(2)	0%

TABLE 3. Computational results for date set 2

Also, as population size and mutation rate increase, the solutions are getting better, the relative errors are decreasing, and the number of trips is decreasing. Only two relative errors are greater than 5%.

8. **CONCLUSION.** We implement a hybrid GA Vehicle Routing Problem to solve The Vehicle Routing Problem with Time Windows. The final results are encouraging when the customer size is small. We will have more comparable experiments with large number of customers in the future.

For simplicity, we only consider the earliest arrival time and assume the capacity of the vehicle is unlimited. We will re-examine the module to handle the capacity cases in the future. We would also like to implement the current algorithm to handle a real time window.

Acknowledgements. The authors are grateful to editor and referees for suggestion to improve the paper.

REFERENCES

[1] O. Bräysa, W. Dullaert and M. Gendreau, *Evolutionary Algorithms for the Vehicle Routing Problem with Time Windows*, J. of Heuristics, **10** (2004), 587–611.

[2] O. Bräysa and M. Gendreau, *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms*, Transportation Science, **39** (2005), 104–118.

[3] O. Bräysa and M. Gendreau, *Vehicle Routing Problem with Time Windows, Part I: Metaheuristics*, Transportation Science, **39** (2005), 119–139.

[4] G. B. Dantzig and R. H. Ramser, *The Truck Dispatching Problem*, Management Science, **6** (1959), 80–81.

[5] M. Desrochers, J. K. Lenstra, M. W. S. Savelsbergh and F. Soumis, *Vehicle Routing with Time Windows: Optimization and Approxinmation*, In "B. Golden and A. Assad (eds.) Vehicle Routing: Methods and Studies, Amsterdam: Elsevier Science Publishers," 1988.

[6] B. L. Golden and A. A. Assad, *Perspectives on Vehicle Routing: Exciting New Developments*, Operations Research, **34** (1986), 803–809.

[7] B. L. Golden and A. A. Assad, *Vehicle Routing: Methods and Studies*, Amsterdam: Elsevier Science Publishers, 1988.

[8] G. Ioannou, M. Kritikos and G. Prastacos, *A Greedy Look-Ahead Heuristic for the Vehicle Routing Problem with Time Windows*, J. of Operational Research Society, **52** (2001), 523–537.

[9] Christian Prins, *A simple and effective evolutionary algorithm for the vehicle routing problem*, Computer and Operations Research, **31** (2004), 1985–2002.

[10] M. M. Soloman and J. Desrosiers, *Time Window Constrained Routing and Scheduling Problems*, Transportation Science, **22** (1988), 1–13.

[11] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2001.

Received September 2006; revised June 2007.

E-mail address: changy@uncw.edu