The Effects of Initial Population in Genetic Search for Time Constrained Traveling Salesman Problems

Cheng-Hong Yang
Department of Electronic Engineering
National Kaohsiung Institute of Technology
Kaohsiung, Taiwan 80782

Kendall E. Nygard
Department of Computer Science and Operations Research
North Dakota State University
Frago, North Dakota 58105

Abstract: We describe the application of Genetic Algorithms to the traveling salesman problem with time windows. A new type of crossover operator, called edge-type crossover, with a heuristically selected initial population, is used in the genetic search. When compared with alternative methods from the literature, experiments indicate that the heuristic initialization speeds the genetic search process.

1. Introduction

Genetic Algorithms (GAs) are stochastic search algorithms that were pioneered by Holland [11]. A GA utilizes an artificial chromosome that represents a solution to the problem of interest, and attempts to successively find better solutions using Darwinian principles of survival of the fittest genes. It has been demonstrated, theoretically and empirically, that GAs are robust and effective in a variety of problems.

GAs use probabilistic transition rules to guide their search. A genetic search is conducted in multiple dimensions by recombining structures and then evaluating the new structures. GAs enable high fitness population members structures to be combined with a structured yet randomized information exchange. New test structures are introduced in every generation, bringing together the fittest pieces of the old with an occasional new part. Although they are randomized, Genetic Algorithms do not follow a simple random walk. They make good use of historical information to discover new search points in order to improve performance [7].

There are many search algorithms that can be used to heuristically find a high performance solution, but if the solution space is large, it is important to use an efficient algorithm. Most of the published methods for solving the time window constrained traveling salesman problem are based on mathematical programming models. In this research, our attempts to solve TSPTW's with GAs led to a new type of crossover operator that consistently provides high performance solutions.

2. Traveling Salesman Problems with Time Windows

The traveling salesman problem (TSP) is a prototypical combinational optimization problem. In the TSP, a salesman must make a complete tour of a given complete graph with N nodes (cities) and find the shortest Hamiltonian circuit through all nodes of the graph. The Hamiltonian circuit must pass through each of the nodes in the graph exactly once.

The traveling salesman problem with time windows (TSPTW) is an extension of the TSP. In the TSPTW, a salesman starts at a specific node (the depot) and must visit each node exactly once and return to the depot. The visit to each node must occur within a time window. For example, if EST_i and LST_i represent the earliest service and latest service times of a specified time window for node i, then the salesman must visit node i between times EST_i and LST_i. When a salesman arrives at a node before the earliest service time, the salesman must wait for that time window to open. Some nodes may have no time windows. The objective of the TSPTW is to find the routing order that minimizes total route distance and visits each node during its associated time window.

Christofides, Mingozzi, and Toth [3] developed a branch and bound algorithm to solve the TSPTW using a dynamic programming state space relaxation procedure to compute bounding information. A branch and bound algorithm for the TSPTW presented by Baker [1] exploits the structure of the dual of a relaxation of the proposed problem, producing lower bounds by solving a longest path problem on an acyclic network. In that study, test problems involving from 8 to 50 customers were produced by adding time windows to some of the nodes in the standard TSP test problems of Eilon *et al.* [5]. Baker and Schaffer [2] produced a heuristic solver for the multiple vehicle routing and scheduling problem with time window constraints effectively by using branch exchange techniques, such as the well-known 2-opt and 3-opt procedures [2]. This procedure can also solve the TSPTW. Nygard and Yang [12] developed

a new crossover operator, called the earliest closing time crossover operator, to solve the TSPTW using Genetic Algorithms (GA). That method consistently provides high performance solutions. In this paper, we solve the same test problems, but using a more effective method.

The TSP is simply stated, but its simplicity is deceptive. It is an NP-complete problem. The incorporation of time window constraints significantly increases the complexity of the TSP. The TSPTW is NP-hard [11]. Like all NP-hard problems, the time required for any known algorithm to compute an exact solution of the TSP increases exponentially with the number of nodes.

3. Experimental Procedures

To apply a GA, candidate solutions to the problem within the space to be searched must be mapped onto an appropriate representation for the artificial chromosome. An adjacency representation was used in this search. In this representation, there is an edge in the tour from node i to node j iff the allele in position i is j [9].

The three components of a basic GA are selection of population members for reproduction, recombination into new members, and evaluation of the fitness of members. The selection function is an evolutionary function that eliminates trials solutions that are relatively ineffective. The quality of the overall result and the computational effort required to achieve convergence critically depend on the selection criteria. Recombination usually involves genetic operators called crossover and mutation. Crossover is the primary genetic recombination operator that was inspired by knowledge of natural genetics. Under the crossover operator, two population members produce offspring by exchanging portions of their artificial chromosomes. The crossover operator we devised, called the edge-type crossover, is discussed in detail in section 3.3. Mutation is used to introduce variety into the population. In mutation, bits in the artificial chromosome are randomly switched with a (typically small) probability. It is a heuristic for avoiding being trapped in a single point of local minimality. In general, we can't hope to search the space globally. Since GAs are direct search algorithms, a performance evaluation function is used to evaluate each point in the search space. The performance evaluation function used in our application is the sum of the total route distance and the the number of tardy nodes multiplied by the average distance between depot and node. In section 3.2, we discuss this function in detail. In our experiments, the algorithm was terminated by reaching a prespecified number of trials.

3.1 Initialization of the Population

The initial population usually consists of randomly generated tours. In this case, the initial population of a tour for the TSPTW is generated by randomly selecting the next visit for each node from a given list of nodes that have not yet been visited until all nodes have been visited, then returning to the node of first departure. As an alternative to the randomly generated initial population, it is likely to be advantageous to begin the GAs with a population of high performance candidate solutions. The heuristically selected initial population of the tour used in this study is generated under restrictions discussed below.

We use the term "time-constrained node" to describe nodes with time windows. Non-time-constrained nodes have no constraints on when they can be visited. Assuming that time windows do not overlap, a feasible TSPTW solution can be viewed as a route constructed from the nodes with the time-constrained nodes sorted by latest service time. Since the time-constrained nodes are sorted by latest service time, there is only one order in which they may be visited. The non-time-constrained nodes are then placed in the route either between time-constrained nodes or at the end of the route. The number of routes depends on the possible combinations of the non-time-constrained nodes.

The initial population of the tour is constructed as follows: first, the nodes are separated into two groups, time-constrained and non-time-constrained. The first group contains the time-constrained nodes sorted by latest service time and forms a partial tour. The second group contains the non-time-constrained nodes. These nodes must be inserted between the nodes of the first group or added to the end of the first group to form a complete route.

Associated with each pair of nodes in the first group is a bucket. The number of buckets is equal to the number of time-constrained nodes. Each bucket contains those non-time-constrained nodes in the second group that can be visited between the associated pair of nodes. A non-time-constrained node is included in a bucket if the sum of the additional distance and the earliest service time of the first node in the pair does not exceed the latest service time of the second node in the pair. The additional distance is the sum of the distance between the first node in the pair and the non-time-constrained node and the distance between the non-time-constrained node and the second node in the pair. When the contents of each bucket have been determined, the complete initial route is formed in the following manner:

1. Randomly select a pair of consecutive nodes in the first group.
2. Repeat steps 4 - 5 a randomly selected number of times. The randomly selected number must be between zero and the number of nodes in the associated bucket.
3. Randomly select a node from the bucket. If the node has already been visited, skip step 4.
4. If the node can be inserted along with the other nodes that have been inserted previously (if any) between the selected pair of nodes without resulting in a tardy arrival time at the second node of the selected pair, then insert the node between the selected pair, and mark the node as having been visited. Otherwise, skip it.
5. Select the next pair and go to step 2. Repeat this

step for each pair of nodes in sequence. When the last pair of nodes is used, continue the sequence with the first pair until all consecutive pairs have been used.

6. If there are non-time-constrained nodes in the second group that have not been visited, then add the node closest to the end of the route to the end of the route, and mark the node as having been visited. Repeat this step until all of the non -time-constrained nodes in the second group have been visited.

For example, consider a 13-node symmetric test problem with the node-to-node distance matrix given in Table 1 [5]. Baker [1] created time windows (earliest service time and latest service time) for the test problem.

Table 1. Node-to-node distance matrix for the example problem

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	9	14	21	23	22	25	32	36	38	42	50	52
2	9	0	5	12	22	21	24	31	35	37	41	49	51
3	14	5	0	7	17	16	23	26	30	36	36	44	46
4	21	12	7	0	10	21	30	27	37	43	31	37	39
5	23	22	17	10	0	19	28	25	35	41	29	31	29
6	22	21	16	21	19	0	9	10	16	22	20	28	30
7	25	24	23	30	28	9	0	7	11	13	17	25	27
8	32	31	26	27	25	10	7	0	10	16	10	18	20
9	36	35	30	37	35	16	11	10	0	6	6	14	16
10	38	37	36	43	41	22	13	16	6	0	12	12	20
11	42	41	36	31	29	20	17	10	6	12	0	8	10
12	50	49	44	37	31	28	25	18	14	12	8	0	10
13	52	51	46	39	29	30	27	20	16	20	10	10	0

Node #	1	2	3	4	5	6	7	8	9	10	11	12	13
EST_i	0	7	12	0	29	48	0	64	0	80	92	100	110
LST_i	*	11	16	*	33	52	*	68	*	84	96	104	114

Legend :
EST_i: Earliest Service Time for Node i
LST_i: Latest Service Time for Node i
*: No Constraint on Service Time

In this problem, the non-time-constrained nodes are 4, 7 and 9. The time-constrained nodes, after sorting by the latest service time, are as follows: 2, 3, 5, 6, 8, 10, 11, 12 and 13. After using the heuristic method to insert the non-time-constrained node in the bucket , we obtain the following result:

2	10 : 7, 9
3	11 : 9
5 : 4	12
6	13
8 : 7	

The right side of the time-constrained node is the non-time-constrained node in the bucket that could be inserted between the previous time-constrained node and the current time-constrained node. For example, the bucket that contains the non-time-constrained nodes 7 and 9 could be inserted between nodes 8 and 10. From this example, we found that, if no non -time-constrained nodes were in the bucket, the node corresponding to the time-constrained node is visited during its associated time window. At least four time-constrained nodes, nodes 3, 6, 12, and 13, are visited during their associated time window.

In a traveling salesman problem, the number of routes and endings at the same node (the depot) corresponds to the permutations of the remaining n - 1 nodes. Hence, there are (n - 1)! such routes, where n is the number of nodes. Let n_{tc} be the number of time-constrained nodes and n_{ntc} be the number of non-time-constrained nodes. The sum of n_{tc} and n_{ntc} is equal to n, where n is the total number of nodes in this study. First, when n_{tc} - 1 nodes which are treated unordered combine with n_{ntc} nodes, giving a total of $n_{tc} + n_{ntc} - 1$ to be permuted, there are $(n_{tc} + n_{ntc} - 1)!$ permutations; in fact, $n_{tc} - 1$ nodes are ordered at this point, and within $(n_{tc} + n_{ntc} - 1)!$ permutations, there are $(n_{tc} - 1)!$ repetitions. So, there are $(n_{tc} + n_{ntc} - 1)! / (n_{tc} - 1)!$ ways to insert those n_{ntc} nodes into n_{tc} -1 ordered nodes.

There are a maximum of $(n_{tc} + n_{ntc} - 1)! / (n_{tc} - 1)!$ routes using this method. In realistic problems, the number of feasible routes is much fewer than the maximum number, since some of the routes would violate time windows. The exhaustive search uses $(n_{tc} - 1)!$ times more routes than our method.

3.2 Evaluation Function

Since GAs are guided by the evaluation function, selecting an appropriate function is important. Any feasible solution for the TSPTW would require that the salesman visit each node during its associated time window. A fitness function can induce time-feasible tours by assessing penalties when time windows are violated. The evaluation function is the sum of the total route distance and the tardiness penalty. It is defined as follows:

$$\text{Fitness} = TD + \text{AvgDist} * NT,$$

where
TD: Total route distance
AvgDist: Average distance between depot and all nodes
NT: Number of tardy nodes
The penalty NT*AvgDist models the expected distance that an additional vehicle would have to travel to visit a tardy node that could not be visited on time by the salesman.

3.3 Edge-Type Crossover

The edge-type crossover employs a class of operations which are modifications of the greedy crossover operator of Grefenstette [9] for solving the TSP. For the TSP, the problem considered is to minimize the total route distance. However, for the TSPTW, the problem considered is not only to minimize the total route distance but also to visit each city during its associated time window.

The edge-type crossover for the TSPTW involves two steps. In the first step, the near-feasible solution (where most nodes are visited during their associated time window in a tour) is found, and in the second step, infeasible nodes (nodes which were not visited

during their associated time windows) are improved, and the traveling distance is minimized. The near-feasible solution is usually easy to obtain after a heuristical method is used to generate the initial population of tours. We sort the nodes with time window constraints by the latest service time, and a route based on the sorted nodes is built. Non-time-constrained nodes are inserted between time constrained nodes into time-feasible locations. Some non-time-constrained nodes may have to added to the end of the route. A significant percentage of the population members are feasible after initialization. Even within the infeasible tours, most of the nodes are serviced during their associated time window. Compared to the results of using the earliest closing time crossover operator with a randomly generated initial population, this saves us much of the time needed to improve time-infeasible nodes.

The edge-type crossover consists of five crossover operators which are the called the shorter edge, longer edge, most nodes, randomly combined, and nearest nodes crossover operators. The edge-type crossover is used to introduce variety (shorter edge, longer edge, most nodes, nearest nodes, etc.) between the time-constrained nodes. The evaluation function guides the search direction. This function penalizes tours with nodes that violate a time constraint; therefore, tours with infeasible nodes will more probably stay in the population after evaluation at each generation. The crossover operator and evaluation function reduce the number of infeasible nodes and minimize the traveled distance. The operator constructs a single offspring using a designated parent in combination with the other parent. The role of the designated parent is reversed to create a second offspring. The edge-type crossover is defined as follows:

Step 1. *Current Node Selection.* The offspring tour begins with a sequence of nodes taken from one parent and called a partial tour. A random node is chosen from the sorted time-constrained group and is specified as the current node. This node serves as the point to begin comparing edges and might or might not be the depot (first node). The partial tour is built from the depot and terminated at the current node.

Step 2. *Best Edge Selection.* The best edge is determined using the distance between the current node and its next time-constrained node or the sum of the distances between the current node, all intervening unvisited non-time-constrained nodes, and the next time-constrained node. Edges leaving the node (as represented in the two parents) are compared. Several methods were available to select the better route among edges from the two parents as follows:

a. *Shorter Edge.* The shorter edge is selected as the better route. If the length of the two edges is the same, then the edge containing the most non-time-constrained nodes is selected.

b. *Longer Edge.* The longer edge is selected as the better route. If the length of the two edges is the same, then the edge containing the most non-time-constrained nodes is selected.

c. *Most Nodes.* The edge with the most nodes that had not yet been visited is selected as the better route between the two time-constrained nodes. If the number of nodes was the same, then the shorter edge is selected as the better route.

d. *Randomly Combined.* The best route for an edge is selected by randomly choosing one of the three previous methods for each edge in the tour.

e. *Nearest Nodes.* Rather than comparing the two edges, a better route is built from the route containing in both edges by using the nearest neighbor method. Any nodes that result in a tardy arrival time at the end of the edges' time-constrained node are ignored.

Step 3. *Tour Extension.* The partial tour is extended using step 2 until the last node of the sorted time-constrained group is reached.

Step 4. *Tour Completion.* If some non-time-constrained nodes exist which are not visited, then the node closest to the end of the partial tour is added to the end of the partial tour until all of the non-time-constrained nodes have been visited.

The newly created offspring replaces the parent if the offspring's performance is better than the parent's. Otherwise, it is discarded.

For example, consider a symmetric test problem with the node-to-node distance matrix given in Table 1 [5]. The following example shows how the shorter edge crossover is processed. In this representation, the integer in the parent field serves as a pointer to the sequence number of the next stop. For example, in Parent 1, the order in which the nodes are visited is 1-2-3-4-5-6-7-8-10-9-11-12-13-1.

Sequence #	1	2	3	4	5	6	7	8	9	10	11	12	13
EST_i	0	7	12	0	29	48	0	64	0	80	92	100	110
LST_i	*	11	16	*	33	52	*	68	*	84	96	104	114

Parent 1	2	3	4	5	6	7	8	10	11	9	12	13	1
AT_i	164	9	14	21	31	50	59	66	88	82	94	102	112
WT_i	0	0	0	0	0	0	0	0	0	0	0	0	0
EAT_i	164	9	14	21	31	50	59	66	76	82	94	102	112
Parent 2	2	3	5	7	6	8	1	9	10	11	12	13	4
AT_i	204	9	14	149	31	50	179	60	74	80	92	100	110
WT_i	0	0	0	0	0	0	0	4	0	0	0	0	0
EAT_i	204	9	14	149	31	50	179	64	74	80	92	100	110
Offspring	2	3	4	5	6	8	1	9	10	11	12	13	7
AT_i	162	9	14	21	31	50	137	60	74	80	92	100	110
WT_i	0	0	0	0	0	0	0	4	0	0	0	0	0
EAT_i	162	9	14	21	31	50	137	64	74	80	92	100	110

Legend:
EST_i: Earliest Service Time for Node i
LST_i: Latest Service Time for Node i
AT_i: Arrival Time at Node i
WT_i: Waiting Time at Node i
EAT_i: Effective Arrival Time at Node i ($AT_i + WT_i$)
*: No Constraint on Service Time

Considering Parent 1 in this example, a node is chosen randomly from the time-constrained group. Assume node 6 is chosen. There are two edges from

node 6 to node 8 for both parents. One edge is from node 6 to node 8; the other edge is from node 6 to node 7 to node 8. Two potential next visit nodes from node 6 is 7 and 8, and 8 for both parents. The arrival time for node 6 is 50 and the potential arrival time at node 8 is the arrival time at node 6 plus the travel time from node 6 to node 8 (50+10=60) or from node 6 to node 7 and from node 7 to node 8 (50+9+7=66) for both parents. Thus, the best edge is selected from node 6 to node 8 directly due to its shorter edge. The process of selecting the shorter edge and extending the tour is continued until reaching node 13. The partial tour at node 13 is 1-2-3-4-5-6-8-10-11-12-13. The potential next node to visit is node 1 for Parent 1 or node 4 for Parent 2. Neither node 1 nor node 4 can be selected as a next visit for node 13 because both of them have been visited. Therefore, the next node to visit for node 13 is selected from the non-time-constrained group which has not been visited. Two non-time-constrained nodes, nodes 7 and 9, have not been visited yet. The distance from node 13 to node 7 and node 9 is 27 and 16, respectively. Hence, node 9 is selected as the next node to visit because it is closest to node 13. After node 7 is added at the end of the partial tour, the offspring complete tour is generated.

The distance of the tour represented by parent 1 is 164, and the distance of the tour represented by parent 2 is 172. The distance of the resulting offspring tour is 162.

4. Experimental Results

The test problems are standard problems from the literature [1]. A population size of 100 was chosen. The crossover rate was set at 60%, and no mutation operator was used. Traditionally, the crossover operator is used immediately after the selection phase. We used the Genesis Code of Grefenstette [8] but modified it slightly. When the selection phase is completed, the elitist policy is introduced: the 60 least fit of the population members are replaced by the best 30 encountered thus far plus 30 randomly generated members.

Table 2 displays the total route time (TT), the total route distance (TD), and the CPU time (CPU) of applying the five types of crossover operators to the ten well-known problems from the literature [1]. Each column in the table represents a separate problem. The number of customers and the percentage of customers that have time window constraints on their service times are identified.

The guaranteed optimal solutions obtained by the disjunctive graph model are shown [1]. The optimal solution of the problems with 90% time windows are obtained at the first generation for all five crossover operators. The optimal solution is easily obtained because the initial population is heuristically selected ; much of the time needed to find a solution by improving infeasible tours is saved because the heuristic method generates a mostly feasible initial tour. Therefore, the more the heuristic method is used to generate the initial populations, the better the result will be and more quickly obtained. The edge-type crossover obtains the optimal solution in nine out of ten cases. The exception is Problem B52. The randomly combined crossover operator is able to reach the optimal solution for all ten problems. For Problem B52, in cases where optimality is not achieved for the other five crossover operators, performance is extremely close to optimal.

Using the heuristic method to generate the initial population of the tour and using the edge-type crossover to manipulate genes, few tardiness nodes are introduced. The traveling salesman problem with time windows is solved in a manner similar to the TSP if we assume that the total route time is equal to the total route distance.

Our experiments involve measuring the tour length for the first-time feasible tour generated by the algorithm. We also track progress toward optimality, which is a function of a number of generations. The experiments showed that about 95% of this range is consistently obtained with 50% or less of the total number of generations. Thus, about half the CPU time is employed to achieve the last 5% of the range between first feasibility and the optimal solution. We found that by using the edge-type crossover with a heursitically selected initial population to solve the problem we can easily obtain about 90% of the optimal solution (the optimal solution for these two problems). When the number of nodes increases, the existing feasible solution seems to increase, and the CPU time is longer.

Compared with the CPU time used by the earliest closing time crossover operator, it appeared that the CPU time used by the edge-type crossover employing the heuristically generated initial population is much shorter, especially with respect to the problem having 90% of the nodes time-constrained. The additional CPU time used by the crossover operator generating an initial population randomly is expected, because the solutions were found by improving infeasible nodes.

5. Conclusions

We experimented with heuristically selected initial population and a new type of crossover operator, the edge-type crossover, for the traveling salesman problem with time windows. The method was applied to 10 well known small and moderate size traveling salesman problems with time windows. The results demonstrate that the proposed method is effective. Particularly, using the heuristic method to generate initial populations saves much time needed to find the first feasible solution. These heuristics restrict consideration to a small portion and possibly feasible domain. All apparently infeasible tours are never generated. This idea should also be applied to the other constrained problems. We expect that the GAs will scale up to relative large problems reasonably well. We conclude that the new method is promising and warrants further investigation on larger and more complex problems.

Table 2. Computational results for the traveling salesman problem with time windows

Problem		B11	B12	B21	B22	B31	B32	B41	B42	B51	B52
Customers		8	8	12	12	21	21	29	29	50	50
% Windows		90	75	90	75	90	75	90	75	90	75
Optimal	TT	6564	6384	162	162	354	354	4755	4755	5631	5632
Solution	TD	6564	6384	158	158	350	341	4753	4753	5631	5557
	CPU*	.4	.4	.7	.7	11.5	13.1	9.1	303.0	349.2	1148.3
Shorter	TT	6564	6384	162	162	354	354	4755	4755	5631	5637
Edge	TD	6564	6384	158	158	350	341	4753	4753	5624	5562
	CPU	.08	.07	.10	.13	.17	2.48	.22	108.30	.42	835.87
Longer	TT	6564	6384	162	162	354	354	4755	4755	5631	5637
Edge	TD	6564	6384	158	158	350	341	4753	4753	5624	5562
	CPU	.07	.07	.12	.13	.15	2.65	.25	51.98	.38	853.22
Most	TT	6564	6384	162	162	354	354	4755	4755	5631	5637
Nodes	TD	6564	6384	158	158	350	341	4753	4753	5624	5562
	CPU	.08	.12	.10	.12	.15	2.88	.22	43.60	.40	836.87
Randomly	TT	6564	6384	162	162	354	354	4755	4755	5631	5632
Combined	TD	6564	6384	158	158	350	341	4753	4753	5624	5557
	CPU	.10	.07	.08	.10	.18	1.87	.18	6.73	.43	953.53
Nearest	TT	6564	6384	162	162	354	354	4755	4755	5631	5637
Nodes	TD	6564	6384	158	158	350	341	4753	4753	5624	5562
	CPU	.12	.13	.15	.17	.23	3.77	.33	148.33	.48	1063.67

Legend:

TT: Total Route Time

TD: Total Route Distnace

CPU*: The CPU time used by the earliest closing time crossover operator to reach optimal solution

CPU: CPU seconds, Solbourne 5/802

References

[1] Baker, E. (1983). An Exact Algorithm for the Time Constrained Traveling Salesman Problem, Operations Research 31, 938-945.

[2] Baker, E. and J.R. Schaffer (1986). Solution Improvement Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints, American Journal of Mathematical and Management Sciences, Special Issue 6 (3-4), 261-300.

[3] Christofides, N., A. Mingozzi and P. Toth (1981). State-Space Relaxation Procedures for the Computation of Bounds Problems, Networks 11, 145-164.

[4] Davis, L. and M. Steenstrup (1987). Genetic Algorithms and Simulated Annealing: An Overview, in *Genetic Algorithms and Simulated Annealing*, in Davis (ed.), Morgan Kaufmann, Los Altos, California , 1-11.

[5] Eilon, S., C. Watson-Gandy and N. Christofides (1971). *Distribution Management*, Griffin Press, London, England.

[6] Goldberg, D.E. and R. Lingle, Jr (1985). Alleles, Loci, and the Traveling Salesman Problem, in Grefenstette (ed.), Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, 154-159.

[7] Goldberg, D.E. (1989). *Genetic Algorithms in Search Optimization, and Machine Learning*, Addison -Wesley, New York, New York.

[8] Grefenstette, J.J. (1984). *A User's Guide to GENESIS*, Technical Report CS-84-11, Computer Science Department, Vanderbilt University, Nashville, Tennessee.

[9] Grefenstette, J.J., R. Gopal, B.J. Rosmaita and D. Van Gucht (1985). Genetic Algorithms for the Traveling Salesman Problem, in Grefenstette (ed.), Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 160-168.

[10] Holland, J.H. (1975). *daptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan.

[11] Lenstra, J. and A. Rinnooy Kan (1981). Complexity of Vehicle Routing and Scheduling Problems, Networks 11, 221-227.

[12] Nygard, Kendall E. and Cheng-Hong Yang (1992). Genetic Algorithm for the Traveling Salesman Problem with Time Windows, *Computer Science and Operations Research: New Development in Their Interfaces*, Pergamon Press, Oxford, England, 411-423.