The Vehicle Routing Problem with Time Windows - Part II: Genetic Search

Jean-Yves Potvin[ab]
Samy Bengio[b]

[a] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), Canada H3C 3J7
[b] Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), Canada H3C 3J7

potvin@iro.umontreal.ca
bengio@iro.umontreal.ca

Abstract. This paper is the second part of a work on the application of new search techniques for the vehicle routing problem with time windows. It describes GENEROUS, the GENEtic ROUting System, which is based on the natural evolution paradigm. Under this paradigm, a population of solutions evolves from one generation to the next by "mating" parent solutions to form new offspring solutions that exhibit characteristics inherited from their parents. For this vehicle routing application, a specialized methodology is devised for merging two vehicle routing solutions into a single solution that is likely to be feasible with respect to the time window constraints. Computational results on a standard set of test problems are reported, and comparisons are provided with other heuristics.

KeyWords. Genetic algorithm, Vehicle routing, Time windows.

Section 0. Introduction

The vehicle routing problem with time windows (VRPTW) has already been defined in the first part of this work.[7] However, for the sake of completeness, we will restate the problem here: given a central depot, a homogeneous fleet of vehicles and a set of customers with known demands (e.g., some quantity of goods to be delivered), find a set of closed routes, originating and ending at the depot, that service all customers at minimum cost.[4,10]

In addition, capacity and time window constraints must be satisfied by each route. In the first case, the total quantity of goods to be delivered cannot exceed the capacity of the vehicle servicing the route. In the second case, a time window or time interval is specified at each customer location in order to constrain the service time. In the hard time window case, a vehicle cannot arrive at a customer location after its time window's upper bound. However, the vehicle can arrive at a customer location before the lower bound. In this case, a waiting time is added to the route. Hence, the routing costs to be minimized include the distance (or travel time), and the waiting time. In this paper, the first objective is to service all customers with the minimum number of routes and, for the same number of routes, to minimize the total route time.

Many references to exact and heuristic algorithms for solving this problem may be found in [7]. In this paper, we will rather focus on two recent approaches based on genetic algorithms. In [11], a genetic algorithm is designed to find good clusters of customers, within a "cluster-first route-second" problem-solving strategy. Once the clusters are identified by the genetic search, classical insertion and post-optimization procedures are applied to produce the final routes.

The genetic algorithm proposed in [2] is representative of a well-know approach to combinatorial optimization problems with side constraints. It is based on the hybridization of a genetic algorithm with a greedy heuristic. The greedy heuristic inserts the customers one by one into the routes, using a fixed a priori ordering of the customers. Under this scheme, the genetic algorithm searches for a good ordering of customers, while the construction of the feasible solution (based on this ordering) is handled by the greedy heuristic. Specialized genetic operators are defined by the authors to create new orderings from old ones. These operators use a global precedence relationship among the customers. For example, it is generally desirable to insert customer i before customer j during the greedy insertion phase, if the time window at customer i occurs before the time window at customer j. Accordingly, this relationship is used by the genetic operators to push customers with early time windows at the front of the orderings.

Although the two above algorithms are based on the evolution paradigm, they are very different from our problem-solving approach. In the following, Section 1 first introduces the evolution paradigm, and describes a very simple genetic algorithm. Then, Section 2 introduces GENEROUS, the GENEtic ROUting System, and explains how it solves the VRPTW. Finally, computational results on a standard set of test problems are reported in Section 3.

Section 1. A Simple Genetic Algorithm

A genetic algorithm is a randomized search technique operating on a population of individuals (solutions). The search is guided by the fitness value of each individual. A simple genetic algorithm can be summarized as follows.

(1) *Representation*. Encode the characteristics of each individual in the initial population as a chromosome (typically, a chromosome is a bit string). Set the current population to this initial population.

(2) *Reproduction*. Select two parent chromosomes from the current population. The selection process is stochastic, and a chromosome with high fitness is more likely to be selected.

(3) *Recombination*. Generate two offspring from the two parents by exchanging pieces of genetic material (crossover).

(4) *Mutation*. Apply a random mutation to each offspring with a small probability.

(5) Repeat steps (2), (3) and (4), until the number of chromosomes in the new population is the same as in the old population.

(6) Set the current population to the new population of chromosomes.

This procedure is repeated for a fixed number of generations, or until convergence to a population of similar individuals is obtained. Then, the best chromosome generated during the search is decoded into the corresponding individual.

In the above procedure, the reproduction phase is aimed at propagating good solution features from one generation to the next, via a bias in the selection process towards the best chromosomes. Then, the recombination phase combines the characteristics of the two selected chromosomes on a single offspring, in the hope that this offspring will be of higher fitness than both parents. An example of a one-point crossover is shown in Figure 1, with a cross point randomly chosen between the second and third bit.

1	0	$\vert$	1	0	0	(parent 1)
$\underline{0}$	$\underline{0}$	$\vert$	$\underline{1}$	$\underline{1}$	$\underline{1}$	(parent 2)
1	0		$\underline{1}$	$\underline{1}$	$\underline{1}$	(offspring 1)
$\underline{0}$	$\underline{0}$		1	0	0	(offspring 2)

Figure 1. One-point crossover on two bit strings

The mutation operator applies to a single chromosome, and is aimed at maintaining a minimum level of diversity in the population. If a chromosome is a bit string, the mutation operator flips a bit value from 0 to 1, or from 1 to 0 within the string, with a very small probability at each position. A good level of diversity in the population is critical for a thorough exploration of the search space. If a population is composed of similar chromosomes, the genetic search cannot make any progress because the offspring look like their parents.

Section 2. A Genetic Search for the VRPTW

Classical genetic algorithms work on a population of chromosomes that encode the characteristics of the corresponding individuals. Their robustness comes from their ability to evolve good individuals, even if the reproduction, crossover and mutation operators are applied at the level of the encoding (and do not exploit any useful information about the individuals, apart from their fitness value). In other words, classical genetic algorithms do not decode the chromosomes as individuals in order to guide the search process.

In the work to be described here, the "representation" issue, or the encoding of a solution within a chromosome, is not addressed. This issue is avoided because it is very difficult to encode multiple routes on a chromosome, and to design crossover operators that would generate feasible offspring at the encoding level. Accordingly, GENEROUS directly applies the genetic operators to the individuals (solutions), and heuristic information about the problem is used to guide the search. Hence, the system departs from the theoretically-founded practice, and simply exploits the general problem-solving methodology of genetic algorithms, namely, the evolution of a population of solutions, and the generation of better solutions through a recombination of good characteristics of two parent solutions on a single offspring solution.

In the following sections, the reproduction, recombination and mutation phases of GENEROUS are described.

2.1 Reproduction phase.

During the reproduction phase, parent solutions are selected from the current population. The selection process is stochastic and biased toward the best solutions. Here, the concept of fitness relates to solution quality, and solution A is better than solution B if:

(a) Solution A has fewer routes than solution B

or

(b) Solutions A and B have the same number of routes, but the total route time of solution A is smaller.

In order to bias the selection process towards the best solutions, a linear ranking scheme is used.[13] First, all solutions in the current population are ranked according to their quality, that is, the best solution has rank 1, and the worst solution has rank P, where P is the size of the population. A fitness value is associated to the solution of rank i as follows:

$$\text{fitness}_i = \text{MAX} - [(\text{MAX} - \text{MIN}) \times (i-1)/(P-1)]$$

$$\text{with } \text{MAX} = 1.6, \text{MIN} = 0.4.$$

For example, when P=5, $\text{fitness}_1=1.6$, $\text{fitness}_2=1.3$, $\text{fitness}_3=1.0$, $\text{fitness}_4=0.7$ and $\text{fitness}_5=0.4$. Then, a fitness-proportional selection scheme is applied to these values. Namely, the selection probability p_i for a solution of rank i is:

$$p_i = \frac{\text{fitness}_i}{\Sigma_{j=1,\dots,P}\ \text{fitness}_j} = \frac{\text{fitness}_i}{P}.$$

It is worth noting that the summation over all fitness values in the population is equal to P, because MIN + MAX = 2, and the average fitness is equal to 1. Since P different selections (with replacement) must be performed on the current population in order to select P parents and generate P new offspring, the expected number of selections E_i for a solution of rank i is:

$$E_i = P \times p_i = \text{fitness}_i \ .$$

Hence, fitness_i is also equal to the expected number of selections for the solution of rank i. For example, the best solution with fitness MAX=1.6, is expected to be selected 1.6 times on average over P different trials.

In order to reduce the variance associated with pure proportional selection (i.e., each solution can be selected between 0 and P times over P different trials), stochastic universal selection or SUS was applied to the fitness values. This approach guarantees that the number of selections for a given solution is at least the floor, and at most the ceiling, of its expected number of selections.[1]

2.2 Recombination Phase

During the recombination phase, two parent solutions are merged into a single one, so as to guarantee the feasibility of the new solution. Two types of crossover operators are used, namely, a "sequence-based" crossover and a "route-based" crossover. They are described in the following sections.

2.2.1 Sequence-Based Crossover (SBX)

This crossover operator is illustrated in Figure 2. First, a link is randomly selected and removed from each parent solution. Then, the customers that are serviced before the break point on the route of parent-solution_1 are linked to the customers that are serviced after the break point on the route of parent-solution_2 (c.f. the black nodes). Finally, the new route replaces the old one in parent-solution_1. A second offspring can be created by inverting the role of the parents.

In a feasible solution, customers with early time windows are typically scheduled at the beginning of a route. Conversely, customers with late time windows are typically scheduled at the end of the route. Hence, by linking the first customers on a route of parent-solution_1 to the last customers on a route of parent-solution_2, the time window constraints are likely to be satisfied.

Unfortunately, the new solution is rarely valid, because some customers are duplicated or unrouted in the process. For example, in

Figure 2, two customers are now located on two different routes, and two other customers are unrouted. Accordingly, a repair operator is applied to the offspring to generate a new feasible solution. This operator deals with the infeasible offspring in the following way:

(a) If a customer appears twice in the new route, one of the two copies is removed from the route. If a customer appears once in the new route, and once in an old route, the customer is removed from the old route. Obviously, these situations are easy to solve, because a feasible route remains feasible when customers are removed (note that a vehicle can arrive at a customer before the time window's lower bound).

(b) If a customer is unrouted, then this customer is inserted at the feasible insertion place that minimizes the additional detour.

Obviously, unrouted customers are more difficult to handle, because there is no guarantee that there is a feasible insertion place for each one of them. If this situation occurs, the offspring is discarded, and a new selection round is initiated to select two new parents. The failure to insert the unrouted customers is the main source of infeasible offspring. On Solomon's test problems,[9] about 50% of the offspring are infeasible. Consequently, the crossover operator must be applied about 2P times in order to generate a new population of feasible offspring of size P.

2.2.2 Route-Based Crossover (RBX)

This operator creates an offspring by replacing a route of parent-solution$_2$ by a route of parent-solution$_1$. As for the sequence-based crossover, RBX is likely to generate offspring with unrouted customers and customers that are duplicated. In Figure 3, the route with black customers in parent-solution$_1$, replaces the corresponding route in parent-solution$_2$. As we can see, a customer is now located on two different routes, while another customer is unrouted. Hence, the repair operator introduced in Section 2.2.1 is applied to the offspring in order to generate a feasible solution.

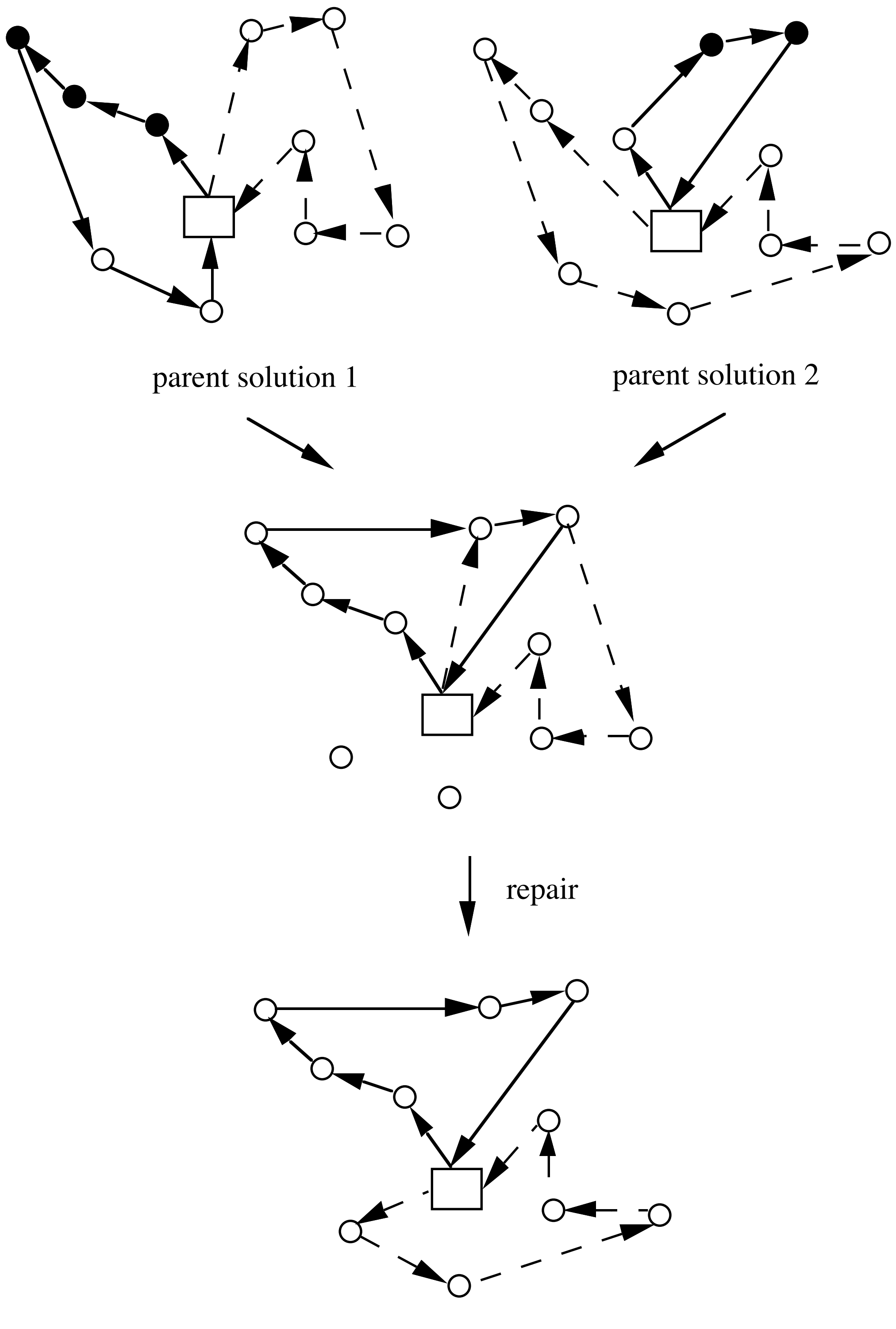

Figure 2. The SBX crossover operator

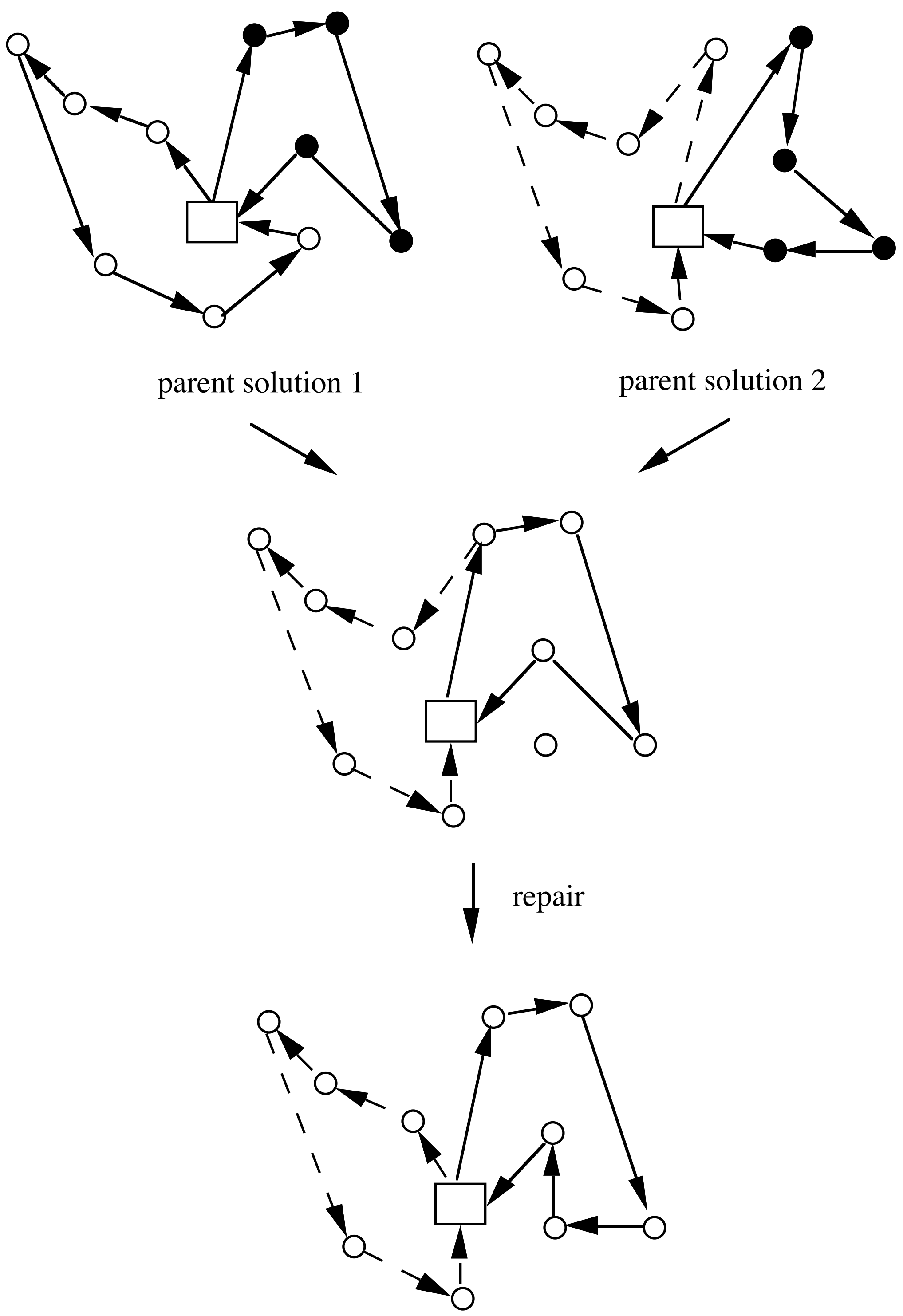

Figure 3. The RBX crossover operator

2.3 Mutation Phase

The crossover operators of Section 2.2 can improve the total route time, but they can hardly reduce the number of routes. Hence, mutation operators are useful to empty routes with only a few customers. In the following sections, we describe three different types of mutation operators available in GENEROUS.

2.3.1 One-level exchange (1M)

This mutation operator first selects a route, and then, tries to move the customers out of this route. This operator can be summarized as follows.

```
Step 1.  Select $route_1$.
Step 2.  For each $customer_1$ in $route_1$, do:
             find the feasible insertion place for $customer_1$ that
             minimizes the detour (over all routes other than $route_1$)
             if there is such a feasible insertion place, then insert
             $customer_1$ at this place.
```

As we can see, the procedure is applied to each customer in the selected route. Obviously, a route can be saved if a feasible insertion place is found for each customer in the selected route. Figure 4a illustrates a one-level exchange for a single customer (c.f. the black node).

It is worth noting that the route selection procedure is stochastic, and is biased towards smaller routes. Namely, the selection probability of a route with n customers is twice as much as the selection probability of a route with $2n$ customers. In this way, the procedure focuses on small routes which are easier to empty. Note also that the total route time can increase in the process.

2.3.2 Two-level exchange (2M)

In a one-level exchange, a customer is moved from one route to another. However, it can be difficult to add a new customer in the second route, due to the capacity and time window constraints. Accordingly, a two-level exchange tries to make room for the new customer by moving another customer out of the second route. The two-level exchange can be summarized as follows.

```
Step 1.  Select $route_1$.
Step 2.  For each $customer_1$ in $route_1$, do:
           for each $customer_2$ in the other routes, do:
             if $customer_1$ can be feasibly inserted at the
             location of $customer_2$, then:
               2a. find the feasible insertion place for $customer_2$
                   that minimizes the detour (over all routes other
                   than $route_1$).
               2b. if there is such a feasible insertion place for
                   $customer_2$, then perform the two-level exchange,
                   and go to Step 2.
```

Figure 4b illustrates a two-level exchange involving two customers (c.f. the black nodes), and three different routes. As for the one-level exchange, the selection of $route_1$ is biased towards smaller routes, and the total route time can increase after an exchange.

2.3.3 Local search (LSM)

In order to provide a means to locally optimize a solution, a mutation operator based on Or-opt exchanges is also available.[6] This operator reduces the total route time via Or-opt exchanges until a local optimum is found.

This local search mutation can be summarized as follows:

Step 1. Start with an initial feasible solution

Step 2. Try to improve the current solution by considering a sequence of consecutive customers, and by moving this sequence at another feasible location within the same route or within another route.

Step 3. Repeat Step 2 until no additional improvement is found.

In Step 2, all sequences of three consecutive customers, two consecutive customers and a single customer are considered (in this order), and a move is performed as soon as an improvement is found. Then, the procedure is repeated with the new solution, until a local optimum is found.

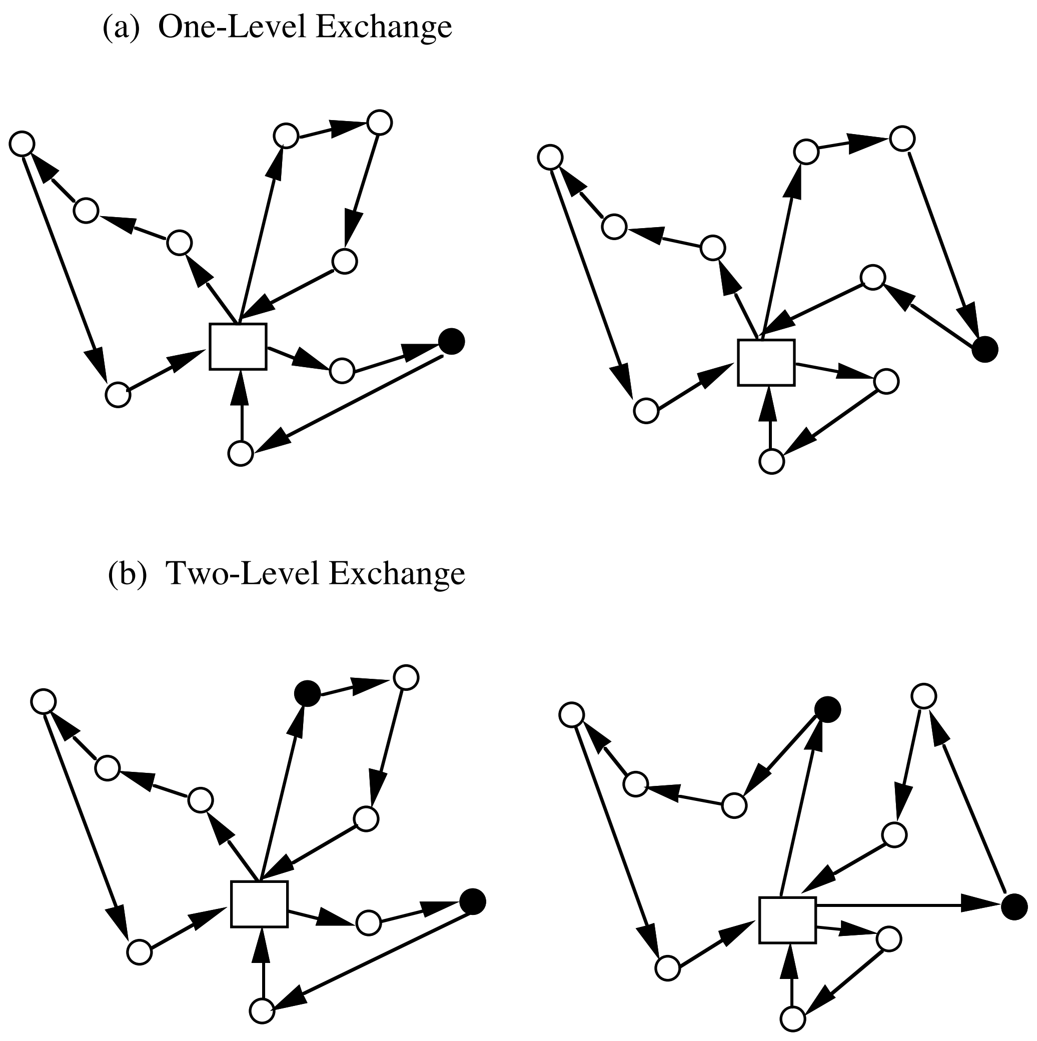

Figure 4. The mutation operators 1M and 2M

Section 3. Computational Results

The algorithm of Section 2 was applied on a standard set of 100-customer Euclidean VRPTWs.[9] In these problems, the travel times are equal to the corresponding Euclidean distances. The geographical data were either randomly generated using a uniform distribution (problem sets R1 and R2), clustered (problem sets C1 and C2) or mixed with both randomly distributed and clustered customers (problem sets RC1 and RC2). Problem sets R1, C1 and RC1 have a narrow scheduling horizon. Hence, only a few customers can be serviced by the same vehicle. Conversely, problem sets R2, C2 and RC2 have a large scheduling horizon, and more customers can be serviced by the same vehicle. More details about these problems may be found in [9].

3.1 Experiments on Solomon's problems

The results on Solomon's problems are summarized in Tables 1a, 1b, and 1c. The average number of routes, distance, waiting time, route time and computation time (in minutes and seconds on a SPARC10 workstation) are shown for each method and each set of problems. Route time or schedule time refers to the sum of travel time (or distance), waiting time and unload time. Note that a total unload time value of 9,000 must be added to the sum of travel time and waiting time for the problems in sets C1 and C2, and a value of 1,000 in the other sets.

In the Tables, I1 refers to the solutions reported in Solomon's paper for the best insertion heuristic I1,[9] while GENEROUS refers to the solutions produced with our algorithm, using the following settings:

(a) The initial population is created by running Solomon's I1 heuristic with randomly generated parameter settings.

(b) The mutation and crossover rates are both equal to 0.6. Hence, 60% of the parents are modified by one of the two crossover operators, which are equally likely to be applied. The remaining parents are copied without any modification. Finally, 60% of the offspring are modified by one of the three mutation operators, which are equally likely to be applied.

(c) The population size is set to 150.

(d) The number of generations is set to 50.

(e) Generation replacement with elitism is applied. Hence, the best solution at a given generation is preserved in the next generation.

The Tables show the evolution of the solutions at generation 0, 20 and 50 (c.f. GENEROUS-00, GENEROUS-20 and GENEROUS-50). The Tables show that GENEROUS significantly improves Solomon's solutions with respect to the number of routes and route time. The improvement with respect to the number of routes is particularly meaningful, due to the acquisition and maintenance costs associated with each vehicle. The genetic algorithm is computationally expensive, but the additional computational requirements are easily justified in terms of fleet reduction.

Note also that the number of generations was set to 50 in order to save the maximum number of routes over the 56 test problems. However, it is clear that this number can be reduced for many problem sets. In particular, the results at generation 20 and at generation 50 are almost the same for sets C1, C2, R2 and RC2. The following generations slightly reduce the total route time, and only one additional route is saved in problem set R2 (at generation 49). This observation is interesting, because the problems of type 2, namely C2, R2 and RC2, involve large routes with many customers per route, and are more computationally expensive than the problems of type 1. By setting the maximum number of generations to 20 for these problems, it is possible to obtain solutions that compare to the solutions obtained at generation 50, at only 40% of the computation cost.

Finally, we observed that GENEROUS found the same solution in set C1 for seven problems out of nine. The total route time of this solution is 9828.9. The final sequence of customers was different on the other two problems, and the total route time was slightly higher at 9874.6 and 9865.0, respectively. However, the solution at 9828.9 is also feasible for these two problems. Hence, it seems that the same sequence of customers is optimal for all problems in set C1, and that the optimal route time is 9828.9. If this hypothesis is right, GENEROUS found the optimum on seven problems in set C1, and failed to find the optimum on the other two problems.

R1 12 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	13.6	1436.7	258.8	2695.5	---
GENEROUS-00	13.5	1424.4	238.7	2663.1	---
GENEROUS-20	12.8	1327.2	131.2	2458.4	4:24
GENEROUS-50	12.6	1296.8	110.6	2407.4	11:19

R2 11 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	3.3	1402.4	175.6	2578.1	---
GENEROUS-00	3.2	1321.6	159.3	2480.9	---
GENEROUS-20	3.1	1167.1	49.7	2216.8	14:55
GENEROUS-50	3.0	1117.7	65.3	2183.0	39:44

Table 1a. Results on problems of type R

C1 9 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	10.0	951.9	152.3	10104.2	---
GENEROUS-00	10.0	947.1	99.9	10046.9	---
GENEROUS-20	10.0	846.4	0.0	9846.4	4:08
GENEROUS-50	10.0	838.0	0.0	9838.0	10:01

C2 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	3.1	692.7	228.6	9921.4	---
GENEROUS-00	3.3	784.2	212.0	9996.2	---
GENEROUS-20	3.0	596.9	0.0	9596.9	15:52
GENEROUS-50	3.0	589.9	0.0	9589.9	41:22

Table 1b. Results on problems of type C

RC1 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	13.5	1596.5	178.5	2775.0	---
GENEROUS-00	13.1	1589.5	129.4	2718.9	---
GENEROUS-20	12.8	1463.6	83.2	2546.8	4:30
GENEROUS-50	12.1	1446.2	63.7	2509.9	11:13

RC2 8 problems	Number of Routes	Distance	Waiting Time	Route Time	Computation Time (min:sec)
I 1	3.9	1682.1	273.2	2955.4	---
GENEROUS-00	3.9	1634.9	228.3	2863.2	---
GENEROUS-20	3.4	1402.4	97.3	2499.7	13:12
GENEROUS-50	3.4	1360.6	101.0	2461.6	35:34

Table 1c. Results on problems of type RC

3.2 Alternative implementations

In the original implementation, a random choice is made among the available crossover and mutation operators. In order to assess the effectiveness of these operators, six different implementations were tested by fixing the choice of the crossover and mutation operators. Namely, six different genetic algorithms were obtained through the cross-product {SBX, RBX} x {1M, 2M, LSM}. These algorithms were applied to problem set RC1, and the results are shown in Table 2. In this Table, the algorithms are referred to as GEN-<crossover>-<mutation>, where <crossover> and <mutation> refer to a particular genetic operator.

This Table shows that the local search mutation LSM provides better solutions than 1M and 2M. In particular, LSM is quite effective for minimizing the total route time. On the other hand, LSM alone cannot save as many routes as the original implementation. In fact, this operator can quickly reduce the diversity found in the initial population if is applied too frequently, because it drives the solutions towards the same local minima. This phenomenon is referred to as "premature convergence". Accordingly, the diversity is better preserved by intermixing LSM with other simple mutation operators like 1M and 2M.

RC1 8 problems	Number of Routes	Route Time	Computation Time (min:sec)
I 1	13.5	2775.0	---
GENEROUS	12.1	2509.9	10:58
GEN-SBX-1M	12.9	2731.5	1:59
GEN-SBX-2M	12.9	2729.1	2:45
GEN-SBX-LSM	12.6	2521.7	22:32
GEN-RBX-1M	12.9	2722.2	3:38
GEN-RBX-2M	12.8	2732.2	5:11
GEN-RBX-LSM	12.5	2515.2	28:35

Table 2. Comparison of different genetic implementations on set RC1

3.3 Comparison with other heuristics

Table 3 shows the average number of routes produced by GENEROUS, and the best results obtained by other researchers on the 56 test problems of Solomon. The number of routes is the first objective to be minimized in all cases. Consequently, this objective provides a common basis to compare the methods. However, it is not possible to compare the results on the basis of the secondary objective, because it is not the same in all cases. Some authors use the route time, while others use the distance, or even a weighted sum of distance and route time.

The other reported heuristics include Solomon's I1 heuristic,[9] the parallel insertion heuristic PARIS,[8] the greedy randomized adaptive search procedure GRASP,[5] the cyclic transfer algorithm CTA,[12] the genetic sectoring algorithm GIDEON,[11] and the tabu search heuristic TABU.[7] In the case of CTA and TABU, the solutions reported are the best solutions obtained with different parameter settings.

The results in Table 3 show that GENEROUS outperforms nearly all other heuristics on the 56 test problems, by minimizing the total number of routes. However, many heuristics do as well as GENEROUS on sets C1, C2, and RC2, while TABU does better on set R1. Clearly, GIDEON and TABU are the most competitive approaches. A Wilcoxon signed-rank test showed a significative difference between GENEROUS and GIDEON, as well as GENEROUS and TABU, on problem set RC1 only (i.e., the hypothesis that the difference is not significative can be rejected at a level of confidence of 90% in both cases).

Set	I1	PARIS	GRASP	CTA	GIDEON	TABU	GENEROUS
R1	13.6	13.3	13.1	13.0	12.8	12.5	12.6
R2	3.3	3.1	3.1	3.1	3.2	3.1	3.0
C1	10.0	10.7	10.6	10.0	10.0	10.0	10.0
C2	3.1	3.4	3.4	3.0	3.0	3.0	3.0
RC1	13.5	13.4	12.8	13.0	12.5	12.6	12.1
RC2	3.9	3.6	3.6	3.7	3.4	3.4	3.4

Table 3. Average number of routes for different heuristics

With respect to computation time, it is very difficult to compare the various approaches, due to different languages and hardware. However, GRASP looks as the most efficient approach. The authors report computation times under one second on problem set R1, using a IBM RISC 6000. For the same problem set, GENEROUS is the more computationally expensive, with 11 minutes of computation time on a SPARC10 workstation. With respect to its closest competitors, GIDEON takes an average of 99.9 seconds on a SOLBOURNE 5/802, while each run of TABU requires between two and five minutes of computation time on a SPARC10 workstation (depending on the parameter settings).

3.4 Comparison with optimal solutions

The results of GENEROUS were also compared to the optimal solutions reported in [3]. The authors solved a few problems in Solomon's test set, namely, problems R101 and R102 in set R1, and problems C101, C102, C106, C107, C108 in set C1. The computational results are shown in Table 4. These results were obtained by truncating the distances at the first decimal place, as in [3]. The two numbers in each cell are the number of routes and total distance, respectively. The solutions produced by Solomon's I1 heuristic,[9] TABU[7] and GENEROUS are shown in the first three columns. The optimal solutions are reported in the last column.

Table 4 shows that the optimum was found by GENEROUS on problems C101, C102, C106, C107 and C108. This observation is in line with the results reported in the previous section for set C1 with non truncated distances. As opposed to the tabu search heuristic, GENEROUS found the minimum number of routes on problem R102 (but not the optimal distance). On problem R101, GENEROUS found one more route than the optimum, like TABU. Hence, there is still room for improvement on the two problems of type R, in particular R101.

Problem	I1	TABU	GENEROUS	OPTIMUM
R101	21 1867.1	19 1650.7	19 1669.4	18 1607.7
R102	19 1699.5	18 1471.8	17 1532.1	17 1434.0
C101	10 851.4	10 827.3	10 827.3	10 827.3
C102	10 966.7	10 827.3	10 827.3	10 827.3
C106	10 916.0	10 827.3	10 827.3	10 827.3
C107	10 902.4	10 827.3	10 827.3	10 827.3
C108	10 853.1	10 827.3	10 827.3	10 827.3

Table 4. Comparison between I1, TABU, GENEROUS and optimal solutions

Section 4. Conclusion

In this paper, a genetic approach to the VRPTW was described. This approach has produced solutions that are competitive with the best solutions reported thus far by other researchers. As for the tabu search heuristic,[7] it is possible that better solutions will emerge by allowing infeasible solutions during the genetic search. This line of research will be explored in the near future.

Acknowledgments. We would like to thank François Guertin for his help during the computational experiments. Also, this research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

References

1. J.E. Baker, 1987. Reducing Bias and Inefficiency in the Selection Algorithm, in Proceedings of the Second Int. Conf. on Genetic Algorithms, Cambridge, MA, pp. 14-21.

2. J.L. Blanton and R.L. Wainwright, 1993. Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms, in Proceedings of the Fifth International Conference on Genetic Algorithms, Champaign, IL, pp. 452-459.

3. M. Desrochers, J. Desrosiers and M.M. Solomon, 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows, *Operations Research 40*, 342-354.

4. J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, 1992. Time Constrained Routing and Scheduling, Technical Report G-92-42, Groupe d'études et de recherche en analyse des décisions, Université de Montréal, Montréal, Canada.

5. G. Kontoravdis and J. Bard, 1992. Improved Heuristics for the Vehicle Routing Problem with Time Windows, Working Paper, Operations Research Group, The University of Texas at Austin, Austin, TX.

6. I. Or, 1976. Traveling Salesman-type Combinatorial Problems and their relation to the Logistics of Blood Banking, Ph.D. thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL.

7. J.Y. Potvin, T. Kervahut, B.L. Garcia and J.M. Rousseau, 1995. The Vehicle Routing Problem with Time Windows - Part I: Tabu Search, *ORSA Journal on Computing* (in this issue).

8. J.Y. Potvin and J.M. Rousseau, 1993. A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows, *European Journal of Operational Research 66*, 331-340.

9. M.M. Solomon, 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints, *Operations Research 35*, 254-265.

10. M.M. Solomon and J. Desrosiers, 1988. Time Window Constrained Routing and Scheduling Problems, *Transportation Science 22*, 1-13.

11. S.R. Thangiah, 1993. Vehicle Routing with Time Windows using Genetic Algorithms, Technical Report SRU-CpSc-TR-93-23, Computer Science Department, Slippery Rock University, Slippery Rock, PA.

12. P. Thompson and H. Psaraftis, 1993. Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems, *Operations Research 41*, 935-946.

13. D. Whitley, 1989. The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best, in Proceedings of the Third Int. Conf. on Genetic Algorithms, Fairfax, VA, pp. 116-121.