MATHEMATICAL COMMUNICATIONS
Math. Commun. **18**(2013), 359–375


# Comparison of eight evolutionary crossover operators for the vehicle routing problem

Krunoslav Puljić[1,*] and Robert Manger[1]

[1] *Department of Mathematics, University of Zagreb, Bijenička cesta 30, HR-10 000 Zagreb, Croatia*

Received December 6, 2011; accepted May 6, 2013

**Abstract.** This paper deals with evolutionary algorithms for solving the vehicle routing problem. More precisely, the paper is concerned with eight evolutionary crossover operators, which have originally been designed for the traveling salesman problem but can also be used for vehicle routing. The considered crossovers are tested on a set of well known benchmark problem instances. The obtained experimental results clearly show that the behavior and relative ranking of the operators within the vehicle routing environment is different than within the traveling salesman environment.

**AMS subject classifications**: 90C27, 90C35, 90C59, 68W40

**Key words**: vehicle routing problem, evolutionary algorithms, crossover operators, traveling salesman problem, experiments

## 1. Introduction

The *vehicle routing problem* (VRP) [19] is an interesting combinatorial optimization task, which deals with scheduling a fleet of vehicles to distribute goods between a depot and customers. A set of routes for vehicles should be determined, which are in some sense optimal, e.g. the shortest or the cheapest possible. Certain constraints should be taken into account, such as customer demands or vehicle capacities.

*Evolutionary algorithms* (EAs) [12] are a popular metaheuristic, which tries to solve optimization problems by imitating processes observed in nature. An EA maintains a population of feasible solutions to a particular problem instance. Evolution of those solutions takes place through application of evolutionary operators such as selection, crossover, mutation, etc.

An important property of the VRP is that it is computationally very hard. Indeed, its instances with more than 30 or 40 customers cannot be solved to optimality, but only approximately by metaheuristics. Thus it makes sense to consider applications of EAs to the VRP. Another important property of the VRP is that it resembles very much the well known *traveling salesman problem* (TSP) [9]. Therefore it seems plausible to assemble an EA for the VRP by reusing components that have originally been designed for the TSP.

In recent years there have been many attempts to solve the VRP by EAs, for instance [1, 2, 3, 4, 10, 13, 15]. Most of those attempts rely on evolutionary operators

*Corresponding author. *Email addresses:* nuno@math.hr (K. Puljić), manger@math.hr (R. Manger)

 

borrowed from the TSP. The authors usually choose operators that are considered best for the TSP. However, it is not clear whether the same components can function equally well within the VRP setting.

The aim of this paper is to test suitability of eight crossover operators for the VRP. All of them have previously been used for the TSP, but not with equal success. The paper will try to establish a relative ranking of the operators within the VRP environment and see if that ranking is the same as for the TSP.

The paper is organized as follows. After this introduction, some necessary preliminaries about the VRP, TSP and EAs are listed in Section 2. Section 3 gives more details about the chosen crossover operators. Section 4 describes the overall design of our EAs that have been used for testing the crossovers. Section 5 presents the obtained experimental results. The final Section 6 gives a conclusion.

## 2. VRP, TSP and EAs

In this paper we consider the standard *capacitated* VRP or CVRP [19], which is described as follows. Let $G = (V, A)$ be a complete directed graph, where $V = \{0, 1, 2, \ldots, n\}$ is a vertex set and $A$ is an arc set. Vertices $i = 1, 2, \ldots, n$ correspond to the customers, and vertex 0 corresponds to the depot. A nonnegative cost $c_{ij}$ is assigned to each arc $i \to j$, and it represents the travel cost spent to go from vertex $i$ to vertex $j$. Each customer vertex $i$ is associated with a nonnegative demand $d_i$ to be delivered, and the depot 0 has a fictitious demand $d_0 = 0$. A set of $K$ identical vehicles, each with the capacity $C$, is available at the depot. The CVRP consists of finding a collection of $\leq K$ elementary cycles in $G$ with minimum total cost such that:

- each cycle visits the depot vertex 0,
- each customer vertex $i = 1, 2, \ldots, n$ is visited by exactly one cycle,
- the sum of the demands $d_i$ of the vertices visited by a cycle does not extend the vehicle capacity $C$.

The cycles constituting the solution to a VRP instance specify optimal routes for the vehicles delivering goods from the depot to the customers. Thereby the demand of each customer is satisfied and no vehicle is overloaded.

The chosen variant of the VRP can obviously be considered as a generalization of the TSP. Indeed, the TSP can be defined as a special case of our VRP where the number of vehicles $K$ is equal to 1, and the capacity $C$ of the vehicle is infinite.

An EA is a randomized algorithm which maintains a population of *chromosomes*. Each chromosome represents a feasible solution to a given instance of an optimization problem. The population is iteratively changed, thus giving a series of population versions usually called generations. It is expected that the best chromosome in the last generation represents a near-optimal solution to the considered problem instance.

An EA consists of many building blocks, which can be chosen and combined in various ways. Consequently, there is a wide variety of possible EAs for the same

optimization problem. Here follows a list of most important building blocks and design options.

- The *data structure* used to represent a chromosome.
- The *initialization procedure* that produces the initial population of chromosomes.
- The *evaluation procedure* used to evaluate a chromosome to give some measure of its "goodness" or "fitness". The goodness measure is related to the objective function of the original optimization problem.
- The evolutionary operators that produce new chromosomes from old ones. There exist unary evolutionary operators, called *mutations*, which create new chromosomes (mutants) by a small random change in a single chromosome. There also exist higher order operators called *crossovers*, which create new chromosomes (children) by combining parts from several (usually two) chromosomes (parents).
- The *selection procedure*, used to find "good" chromosomes for crossover, or "bad" chromosomes that will be discarded from the population.
- The *termination condition*, which determines when the whole evolutionary process should be stopped.

As already mentioned before, EAs for solving the VRP are often assembled from components that have originally been designed for the TSP. Indeed, it has been quite common to represent the chromosome for the VRP as a permutation of customer vertices $1, 2, \ldots, n$ [11]. With such representation, it is also possible to reuse a variety of evolutionary operators that transform permutations into permutations. For instance we can use:

- *order crossover* (OX) [7, 10, 11, 15, 17],
- *partially mapped crossover* (PMX) [11, 18],
- *edge recombination crossover* (ERX) [11, 17],
- *cycle crossover* (CX) [10, 11],
- *alternating edges crossover* (AEX) [8, 11, 14],
- *heuristic greedy or random or probabilistic crossover* (HGreX or HRndX or HProX) [8, 11, 18],
- *mutation by inversion* (IM) [10],
- *mutation by reinsertion* (RM) [11],
- *swap mutation* (SM) [20].

It is important to note that, contrary to the TSP, a chromosome in form of a permutation does not uniquely determine a solution to the VRP. Namely, such chromosome must be interpreted as a concatenation of vehicle routes, and many different combinations of routes can produce the same concatenation. Consequently, using permutations as chromosomes makes the evaluation procedure much more complicated, since prior to computing the objective function the permutation must be splitted into feasible individual routes. If we insist on optimal evaluation, i.e. on such splitting that minimizes the objective function, then the split procedure from [15] must be used. It is a relatively complicated procedure based on solving auxiliary shortest-path problems, whose complexity per evaluation is as high as $\mathcal{O}(n^2)$. Another possibility is to use a simpler split procedure based on the greedy approach. Then it is assumed that the first vehicle serves as many customers from the initial part of the chromosome as possible regarding the capacity $C$, the second vehicle serves as many customers as possible from the subsequent part of the chromosome, etc. With its linear complexity, the greedy procedure is relatively fast although not necessarily optimal.

Apart from the components that originate from the TSP, EAs for the VRP may also contain more general components applicable to any kind of problem. For instance, selection is usually accomplished by *tournament selection* [12], where a predefined number of chromosomes is picked up randomly, and then the best or the worst of them is selected. The initial population is often formed by some constructive heuristics or randomly. The termination condition is usually based on the elapsed time or the number of evaluations. Most algorithms support *elitism* [5], i.e. the best chromosome is never discarded.

## 3. Crossover operators

As stated before, this paper is concerned with crossover operators for the VRP that have been borrowed from the TSP. In the previous section we have already listed eight such crossovers: OX, PMX, ERX, CX, AEX, HGreX, HRndX and HProX. Appropriate references have also been given. Still, since the mentioned operators are in focus of our interest, we now describe each of them in detail and give some illustrative examples. Note that all considered crossovers work with chromosomes that are represented as permutations of customer vertices. Such chromosome is sometimes interpreted as a big circle (either undirected or directed) spanning the whole graph. Costs of the involved arcs can also be relevant.

### 3.1. Order crossover (OX)

The OX operator acts as follows: it copies a part of the child chromosome from the first parent and constructs the remaining part by following the vertex ordering in the second parent. More precisely, two cut points are randomly selected, and the part of the first parent located between those cut points is copied to the child. The remaining positions in the child are then filled one at a time, starting after the second cut point, by considering the customer vertices in order found in the second parent (wrapping around when the end of the list is reached).

For instance, let the two parents and the two cut points "|" be as follows:

$$p_1 = (1\ 2\ 3 \mid 5\ 4\ 6\ 7 \mid 8\ 9),$$
$$p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3).$$

Then the first child $c_1$ is:

$$c_1 = (2\ 1\ 8 \mid 5\ 4\ 6\ 7 \mid 9\ 3).$$

If we exchange the roles of the two parents $p_1$ and $p_2$, we can obtain the second child:

$$c_2 = (3\ 5\ 4 \mid 1\ 8\ 7\ 6 \mid 9\ 2).$$

## 3.2. Partially mapped crossover (PMX)

The PMX operator is similar to OX. Again we have two cut points given. The first parent's part between the two cut points is first copied to the child. Then the remaining parts of the child are filled with the remaining vertices so that their absolute positions are inherited as much as possible from the second parent.

For instance, let the parents $p_1$ and $p_2$ and the cut points "|" be the same as in the previous OX example. By switching the roles of $p_1$ and $p_2$ we immediately construct two children $c_1$ and $c_2$ initialized as

$$c_1 = (*\ *\ * \mid 5\ 4\ 6\ 7 \mid\ *\ *),$$
$$c_2 = (*\ *\ * \mid 1\ 8\ 7\ 6 \mid\ *\ *).$$

Here $*$-s denote positions that are not settled yet. Note that the two initialized sections of $c_1$ and $c_2$ define a mapping:

$$5 \to 1,\ 4 \to 8,\ 6 \to 7,\ 7 \to 6.$$

Next, the vacant parts of $c_1$ are filled (if possible) with vertices from $p_2$ that happen to be on the same positions. We obtain

$$c_1 = (*\ *\ 2 \mid 5\ 4\ 6\ 7 \mid 9\ 3).$$

The two remaining positions should be filled with 4 and 5. But since 4 and 5 are already present in $c_1$, we replace them according to the above mapping with 8 and 1, respectively. Thus the completed first child is

$$c_1 = (8\ 1\ 2 \mid 5\ 4\ 6\ 7 \mid 9\ 3).$$

The second child $c_2$ is completed by an analogous procedure, and it looks as follows:

$$c_2 = (5\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid 4\ 9).$$

### 3.3. Edge recombination crossover (ERX)

The ERX operator ignores arc directions and interprets a chromosome as an undirected cycle of *edges*. The idea is that the child should inherit as many edges from the parents as possible, and that edges that are common to both parents should have priority. The procedure of constructing the child is based on the concept of neighborhood. A neighbor of a vertex is another vertex that is adjacent to the original one either within the first or within the second parent cycle. The procedure starts from an arbitrary vertex and proceeds in steps. In each step, the next vertex is chosen among the neighbors of the previous one. If there are more feasible choices, then the neighbor is chosen whose own list of neighbors is the shortest.

For instance, let the two parents be

$$p_1 = (1\,2\,3\,4\,5\,6\,7\,8\,9),$$
$$p_2 = (4\,1\,2\,8\,7\,6\,9\,3\,5).$$

We can start constructing the child $c$ from the vertex 1. Then the second vertex in $c$ should be chosen among the neighbors of 1, and these are 2, 4 and 9. Since 2 itself has three neighbors, 4 three neighbors, and 9 four neighbors, the choice should be restricted to 2 or 4. If we randomly select 4, then the child is initialized as

$$c = (1\,4\,*\,*\,*\,*\,*\,*\,*).$$

In the next step, we choose the third vertex in $c$ among the neighbors of 4, and these are 1, 3 and 5. But 1 is already present in $c$, so the choice restricts to 3 or 5. Since 3 has four neighbors of its own, and 5 has three neighbors, we must choose 5, and the child becomes

$$c = (1\,4\,5\,*\,*\,*\,*\,*\,*).$$

By proceeding in the same fashion, we finally obtain

$$c = (1\,4\,5\,6\,7\,8\,2\,3\,9).$$

### 3.4. Cycle crossover (CX)

The CX operator tries to realize the following idea: a vertex should be copied into the child from one parent, but its position should be inherited from the other parent.

For instance, let the two parents $p_1$ and $p_2$ be the same as in the previous ERX example. We start by picking the first vertex from $p_1$ and by copying it into the child $c_1$. Thus $c_1$ is initialized as

$$c_1 = (1\,*\,*\,*\,*\,*\,*\,*\,*).$$

Next we select the vertex from $p_2$ at the same position 1 - it is 4. We copy 4 into $c_1$ but to the position where it resides in $p_1$, i.e. to position 4. Thus we have:

$$c_1 = (1\,*\,*\,4\,*\,*\,*\,*\,*).$$

Next we select the vertex from $p_2$ at the same position 4 - it is 8. We copy 8 into $c_1$ to the position where it can be found in $p_1$, i.e. to position 8. Indeed:

$$c_1 = (1 * * 4 * * * 8 *).$$

By proceeding with two more steps in the same fashion, we obtain

$$c_1 = (1\ 2\ 3\ 4 * * * 8 *).$$

Now the last inserted vertex is 2 at position 2. It means that the next choice should be the vertex from $p_2$ at the same position - it is vertex 1. But 1 is not feasible since it is already present within $c_1$. In such situation, we use only $p_2$ to complete $c_1$, i.e. all remaining positions in $c_1$ are filled with the vertices from $p_2$ at the same positions. The final $c_1$ looks as follows:

$$c_1 = (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5).$$

By reversing the roles of the two parents, we can also obtain the other child:

$$c_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9).$$

## 3.5. Alternating edges crossover (AEX)

The AEX operator interprets a chromosome as a directed cycle of arcs. The child cycle is formed by choosing in alternation arcs from the first and from the second parent, with some additional random choices in case of infeasibility.

For instance, let the two parents be:

$$p_1 = (5\ 1\ 7\ 8\ 4\ 9\ 6\ 2\ 3),$$
$$p_2 = (3\ 6\ 2\ 5\ 1\ 9\ 8\ 4\ 7).$$

The procedure starts by choosing the arc $5 \to 1$ from $p_1$ as the first arc. So the child is initialized as

$$c = (5\ 1 * * * * * * *),$$

Next, the arc from $p_2$ is added that goes out from 1, i.e. $1 \to 9$. Thus the child becomes

$$c = (5\ 1\ 9 * * * * * *).$$

Next, the arc from $p_1$ going out from 9 is added, etc. After few steps the following partially formed child is obtained:

$$c = (5\ 1\ 9\ 6\ 2\ 3 * * *).$$

The arc going out from 3 should be chosen from $p_2$, but such choice is infeasible since it would close the circle too early. To avoid this situation, one of the remaining unvisited vertices is picked up randomly, for instance 7. Thus the child becomes

$$c = (5\ 1\ 9\ 6\ 2\ 3\ 7 * *).$$

From this point the ordinary procedure can be resumed again by choosing the arc $7 \to 8$ from $p_1$, and then $8 \to 4$ from $p_2$. The completed child looks as follows:

$$c = (5\ 1\ 9\ 6\ 2\ 3\ 7\ 8\ 4).$$

### 3.6. Heuristic crossovers (HGreX, HRndX, HProX)

The HGreX operator bears some resemblance to AEX. The child cycle is formed by choosing from each vertex the cheaper of the two respective parent arcs. In case of infeasibility, some additional choices are made.

Let the two parents $p_1$ and $p_2$ be the same as in the previous AEX example. Suppose that the involved arcs have the following costs:

$$p_1 \ldots c_{51} = 2, c_{17} = 2, c_{78} = 6, c_{84} = 8, c_{49} = 3, c_{96} = 6, c_{62} = 4, c_{23} = 4, c_{35} = 3,$$
$$p_2 \ldots c_{36} = 6, c_{62} = 4, c_{25} = 2, c_{51} = 2, c_{19} = 6, c_{98} = 8, c_{84} = 8, c_{47} = 3, c_{73} = 5.$$

The procedure starts from a randomly chosen vertex, for instance 5. The arcs in both parents going out from the vertex 5 are considered. In our particular case, both parents use the same arc $5 \to 1$. So the child is initiated as

$$c = (5\ 1\ *\ *\ *\ *\ *\ *\ *).$$

Next, the parent arcs leaving the vertex 1 are considered, i.e. $1 \to 7$ with the cost $c_{17} = 2$ and $1 \to 9$ with the cost $c_{19} = 6$, respectively. Among them $1 \to 7$ is cheaper, thus it is copied into the child:

$$c = (5\ 1\ 7\ *\ *\ *\ *\ *\ *).$$

Next, the cheaper among the parent arcs $7 \to 8$ and $7 \to 3$ is selected, so that the child becomes

$$c = (5\ 1\ 7\ 3\ *\ *\ *\ *\ *).$$

Next, the arcs $3 \to 5$ and $3 \to 6$ are considered. The first of them is cheaper but infeasible since it would close the cycle too early. So exceptionally, the more expensive arc $3 \to 6$ is chosen, and the child becomes

$$c = (5\ 1\ 7\ 3\ 6\ *\ *\ *\ *).$$

The procedure proceeds in a similar manner, and after the next step we have

$$c = (5\ 1\ 7\ 3\ 6\ 2\ *\ *\ *).$$

Then the arcs $2 \to 3$ and $2 \to 5$ have to be considered, but both of them are infeasible since they lead to already visited vertices. In such situation the operator randomly generates a prescribed number of feasible arcs, e.g. $2 \to 4$, $2 \to 8$ and $2 \to 9$, and picks up the cheapest among them. Supposing that $2 \to 8$ is the cheapest, the child would finally be completed in the following way:

$$c = (5\ 1\ 7\ 3\ 6\ 2\ 8\ 4\ 9).$$

The remaining two operators, HRndX and HProX, can be considered as simple variants of HGreX. They consist of the same steps but use different criteria for choosing an arc in a particular step. While HGreX applies the greedy approach and always selects the cheapest among the considered arcs, HRndX and HProX make random choices. Thereby HRndX picks up any arc with equal probability, while HProX gives more chance to a cheaper arc. For the same parents $p_1$ and $p_2$ as shown before, HRndX or HProX could produce a different child than HGreX, for instance it could be

$$c = (5\ 1\ 7\ 3\ 6\ 2\ 9\ 8\ 4).$$

## 4. Design of experiments

Our experiments are based on running and testing several variants of a simple EA. All variants conform to the same generic structure shown in Figure 1, but they differ in two aspects:

- choice of a crossover operator,
- presence or absence of mutation.

There are nine possible choices for crossover combined with two possibilities regarding mutation, thus giving altogether eighteen variants.

```
EvolutionVRP( ) {
    input the VRP instance and the EvaluationLimit;
    initialize the population P with 30 chromosomes;
    evaluate the whole population P;
    EvaluationCount = 0;
    while (EvaluationCount < EvaluationLimit) {
        // crossover
        Parent1 = a "good" chromosome from P selected by tournament;
        Parent2 = a "good" chromosome from P selected by tournament;
        Child = crossover of Parent1 and Parent2;
        leave Parent1 and Parent2 in P as they are;
        evaluate Child; EvaluationCount += 1;
        insert Child into P by taking into account similarity;
        // mutation (optional)
        Rnd = a random integer between 1 and 100;
        if (Rnd == 1) { // probability 1%
            Parent = a randomly chosen chromosome
            from P which is not the best one in P;
            Mutant = mutation of Parent;
            evaluate Mutant; EvaluationCount += 1;
            replace Parent in P by Mutant;
        }
    }
}
```

Figure 1: *Generic pseudocode of our EA for solving the VRP*

The first eight choices of crossovers feature OX, PMX, ERX, CX, AEX, HGreX, HRndX and HProX, respectively. Within one choice, a single operator is run in isolation as the only available form of crossover. The purpose of such separate testing is to measure performance of particular operators and to obtain their relative

ranking. As the ninth choice, a mix of eight crossovers is considered, where any of them is activated randomly with equal probability. The goal of such additional testing is to investigate possible synergies among the operators. Mutation, when present, is realized as a mix of three standard operators: IM, SM and RM, so that any of them is selected randomly with equal probability. By activating or deactivating mutation operators, we can explore their influence on crossovers. Namely, it is expected that mutations could help crossovers to escape from local optima and achieve better performance.

In our EA, the chromosome is represented as a permutation of customer vertices, so that the above listed crossover and mutation operators can be applied. The initialization procedure simply produces 30 random permutations. According to our preliminary tests, the population size 30 seems to be just adequate. Tournament selection of a "good" chromosome is performed with three participants within a tournament, while a "bad" chromosome is chosen by using two participants. Evaluation is always accomplished according to the optimal split procedure. The whole process is stopped after a given number of evaluations.

In the course of the algorithm, the population $P$ changes due to insertions of new chromosomes or replacements of existing chromosomes. All insertions rely on the concept of similarity. We say that two chromosomes are *similar* if their goodness values differ in less than 1% of the best value within the population. Insertion "by taking into account similarity" means the following.

- If there exists another chromosome in $P$ that is similar to the new one, then the better of those two "twins" is retained in $P$ and the other one is discarded.
- If there is no similar chromosome, then the new one is retained in $P$, and some other "bad" chromosome from $P$ is selected by tournament and discarded.

It is easy to check that according to our rules the size of $P$ always remains the same, i.e. 30. Indeed, whenever a chromosome is inserted, another one is discarded. Note that our algorithm supports a kind of elitism, namely the best chromosome is discarded only if it is replaced by an even better chromosome. Note also that the algorithm is purely evolutionary or non-hybrid. Indeed, it is built of evolutionary procedures and operators, and it does not incorporate elements of any other heuristic or metaheuristic.

## 5. Results of experiments

In order to perform experiments, we have implemented our EA for the VRP as a C++ program. In our experiments, we have tested the implemented algorithm on seven benchmark VRP instances from the well known Christofides-Mingozzi-Toth collection [6]. Table 1 gives some basic parameters of those instances, including the costs of their best known solutions as recorded by [6]. Each of the previously described eighteen variants of the EA has been evaluated on each of the seven VRP instances, thus giving altogether 126 experiments. In all program runs, the number of evaluations has been set to 4000000.

Since our algorithm is randomized, its repeated execution on the same input data with the same number of processes usually produces slightly different results. To

| VRP instance | Number of customers ($n$) | Number of vehicles ($K$) | Vehicle capacity ($C$) | Cost of the best known solution |
|---|---|---|---|---|
| CMT01 | 50 | 5 | 160 | 524.61 |
| CMT02 | 75 | 10 | 140 | 835.26 |
| CMT03 | 100 | 8 | 200 | 826.14 |
| CMT04 | 150 | 12 | 200 | 1028.42 |
| CMT05 | 199 | 17 | 200 | 1291.45 |
| CMT11 | 120 | 9 | 200 | 1042.11 |
| CMT12 | 100 | 10 | 200 | 819.56 |

Table 1: *Benchmark VRP instances from the CMT collection*

amortize this effect of randomization, each experiment from our agenda has been repeated 30 times. Consequently, all measured values reported in the forthcoming tables and figures are in fact averages obtained over 30 repetitions. The corresponding coefficients of variation are all below 0.05, thus assuring that the reported averages are accurate and stable.

The results of experiments are summarized in Tables 2 and 3. Thereby Table 2 corresponds to the nine variants of the EA where no mutation is applied, while Table 3 comprises the variants with mutations included. Both tables are organized in the same way: a row corresponds to a particular problem instance and a column to a particular EA variant characterized by a certain choice of crossover operators. Thus a table entry presents the solution of the corresponding instance by the corresponding variant. The solution is described by its cost and as the relative error with respect to the best known solution. The average error for a chosen variant over all instances is also given. The best solution for a chosen instance over all variants is marked by bold face.

Figures 2 and 3 give a more detailed presentation of results for a single chosen problem instance. Thereby Figure 2 refers to the EA variants without mutation, and Figure 3 to the variants with mutation, respectively. In both figures, each graph corresponds to a particular crossover operator, and it shows how the current solution improves depending on the number of evaluations. Only the four best performing operators are presented. The labels on the left figure margin denote the absolute solution cost, while the labels on the right margin refer to the relative cost compared to the best known solution.

From the data presented in Tables 2-3 and Figures 2-3, the following facts can be observed.

- Mutation always improves performance, apparently by helping crossovers to escape from local minima.
- Mixing different crossover operators produces better results than using any operator alone. Thus a synergy among crossovers seems to exist. Improvement is more visible when there is no mutation.

| Problem | Crossover operator | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Instance | OX | PMX | ERX | CX | AEX | HGreX | HRndX | HProX | mix |
| CMT01 | 558.71 | 1097.21 | 571.25 | 1106.12 | 540.97 | 543.40 | 549.62 | 546.66 | **537.95** |
| | 6.5% | 109.1% | 8.9% | 110.8% | 3.1% | 3.6% | 4.8% | 4.2% | 2.5% |
| CMT02 | 893.49 | 1732.40 | 916.81 | 1723.13 | 869.14 | 866.68 | 891.43 | 878.86 | **861.33** |
| | 7.0% | 107.4% | 9.8% | 106.3% | 4.1% | 3.8% | 6.7% | 5.2% | 3.1% |
| CMT03 | 928.37 | 2296.97 | 913.93 | 2330.28 | 868.12 | 863.54 | 937.53 | 902.19 | **845.11** |
| | 12.4% | 178.0% | 10.6% | 182.1% | 5.1% | 4.5% | 13.5% | 9.2% | 2.3% |
| CMT04 | 1268.33 | 3465.57 | 1264.64 | 3437.10 | 1132.83 | 1104.28 | 1301.62 | 1192.74 | **1078.08** |
| | 23.3% | 237.0% | 23.0% | 234.2% | 10.2% | 7.4% | 26.6% | 16.0% | 4.8% |
| CMT05 | 1649.36 | 4630.01 | 1695.02 | 4526.89 | 1446.95 | 1398.50 | 1720.65 | 1531.35 | **1379.61** |
| | 27.7% | 258.5% | 31.2% | 250.5% | 12.0% | 8.3% | 33.2% | 18.6% | 6.8% |
| CMT11 | 1321.74 | 4198.80 | 1303.84 | 4150.51 | 1079.38 | 1088.12 | 1278.92 | 1102.38 | **1053.33** |
| | 26.8% | 302.9% | 25.1% | 298.3% | 3.6% | 4.4% | 22.7% | 5.8% | 1.1% |
| CMT12 | 867.75 | 2632.77 | 883.59 | 2666.87 | 830.24 | 829.39 | 852.33 | 829.18 | **821.94** |
| | 5.9% | 221.2% | 7.8% | 225.4% | 1.3% | 1.2% | 4.0% | 1.2% | 0.3% |
| average | 15.7% | 202.0% | 16.6% | 201.1% | 5.6% | 4.7% | 15.9% | 8.6% | 3.0% |

Table 2: *Solutions obtained without mutation*

| Problem | Crossover operator | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Instance | OX | PMX | ERX | CX | AEX | HGreX | HRndX | HProX | mix |
| CMT01 | 550.10 | 555.51 | 553,84 | 629.02 | 535.67 | 537.62 | 546.74 | 541.32 | **534.57** |
| | 4.9% | 5.9% | 5.6% | 19.9% | 2.1% | 2.5% | 4.2% | 3.2% | 1.9% |
| CMT02 | 884.54 | 889.95 | 886.18 | 1016.13 | **860.27** | 866.47 | 884.79 | 870.89 | 860.36 |
| | 5.9% | 6.5% | 6.1% | 21.7% | 3.0% | 3.7% | 5.9% | 4.3% | 3.0% |
| CMT03 | 882.85 | 894.23 | 907.12 | 1136.65 | 858.43 | **845.73** | 916.18 | 878.27 | 846.54 |
| | 6.9% | 8.2% | 9.8% | 37.6% | 3.9% | 2.4% | 10.9% | 6.3% | 2.5% |
| CMT04 | 1162.69 | 1200.99 | 1251.28 | 1717.23 | 1113.62 | 1085.53 | 1273.10 | 1154.64 | **1077.89** |
| | 13.1% | 16.8% | 21.7% | 67.0% | 8.3% | 5.6% | 23.8% | 12.3% | 4.8% |
| CMT05 | 1541.30 | 1564.66 | 1705.16 | 2388.64 | 1431.84 | 1381.22 | 1699.21 | 1498.21 | **1380.08** |
| | 19.3% | 21.2% | 32.0% | 85.0% | 10.9% | 7.0% | 31.6% | 16.0% | 6.9% |
| CMT11 | 1258.44 | 1261.56 | 1276.88 | 1659.36 | 1076.62 | 1068.69 | 1227.27 | 1080.36 | **1052.21** |
| | 20.8% | 21.1% | 22.5% | 59.2% | 3.3% | 2.6% | 17.8% | 3.7% | 1.0% |
| CMT12 | 842.40 | 897.14 | 851.94 | 1231.51 | 823.78 | 825.18 | 849.82 | 826.24 | **822.53** |
| | 2.8% | 9.5% | 4.0% | 50.3% | 0.5% | 0.7% | 3.7% | 0.8% | 0.4% |
| average | 10.5% | 12.7% | 14.5% | 48.7% | 4.6% | 3.5% | 14.0% | 6.6% | 2.9% |

Table 3: *Solutions obtained with mutation*

- If only a single crossover operator is used, then, depending on the problem instance and whether mutation is used or not, the best performance is accomplished either by AEX or HGreX.

- Among single crossovers HProX also produces very good results, although it never happens to be the best.

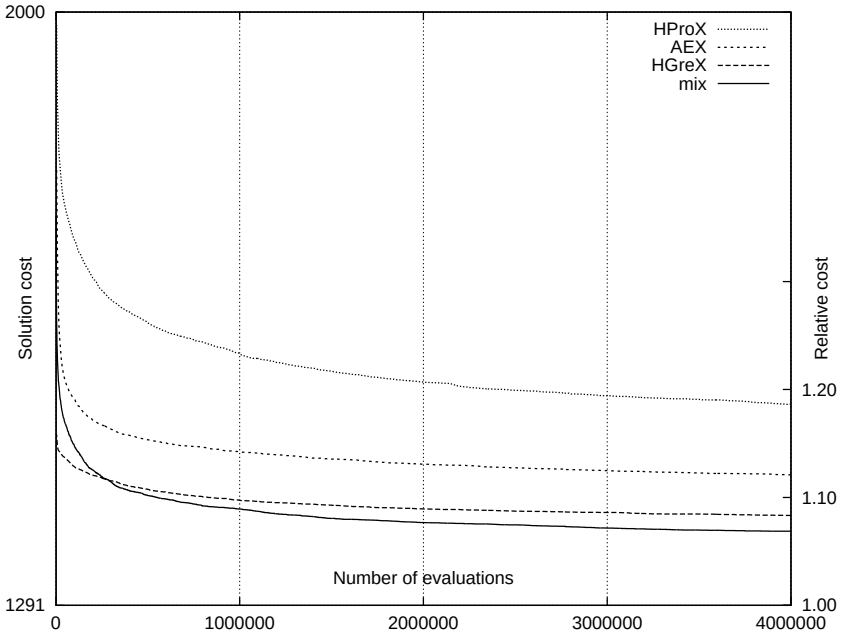


Figure 2: *Details for CMT05 - solution cost vs. number of evaluations - no mutation*

- Operators OX, ERX and HRndX are usable, but their performance is inferior to AEX, HGreX or HProX.
- Exceptionally bad performance is provided by PMX and CX (if there is no mutation) and again by CX (if mutation is present).

In order to verify the above observations, we have also made an adequate statistical analysis. By assuming that columns of Tables 2 and 3 are random and independent samples, we conducted a series of statistical hypotheses tests. More precisely, for each pair of crossover operators we tested the hypothesis that one of the operators performs better than the other. Thereby the performance of an operator has been characterized by its mean relative error computed over the whole set of problem instances with known solutions. Consequently, each test was in fact a standard test for the difference of means based on small samples and the Student's t-distribution [16]. In our analysis the mix of eight crossovers was treated as an additional crossover. The experiments with or without mutation have been analyzed separately.

The outcomes of our hypotheses testing have been summarized in Tables 4 and 5. Table 4 corresponds to the experiments without mutation, while Table 5 corresponds to the experiments with mutation. A single table row contains on its right-hand side the list of crossovers that are inferior to the crossover on its left-hand side. All reported inferiorities are statistically significant at the significance level 0.05 [16].

In Table 4 or 5, each crossover is ranked according to its number of inferior

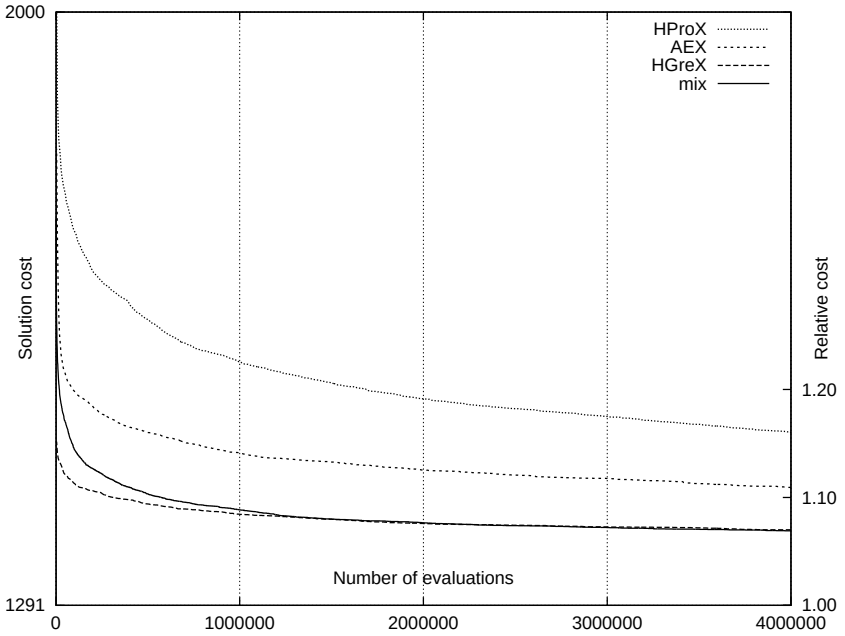


Figure 3: *Details for CMT05 - solution cost vs. number of evaluations - mutation included*

crossovers. The rankings in the two tables are somewhat different at the bottom, but they are almost the same at the top. As expected, the best rank in both tables belongs to the mix of operators. Also the best or the second-best are AEX and HGreX followed by HProX. Again according to the expectations, PMX and CX share the worst rank in Table 4, while CX alone is the worst in Table 5.

As we see, the presented statistical analysis supports most of our previous observations about crossovers. Indeed, the difference between any of the three highest-ranked operators and any of the five lowest-ranked operators is statistically significant at the significance level 0.05. Still, the differences among some pairs of the higher-ranked operators are not statistically significant. This is for instance true for HGreX vs. HProX, or AEX vs. HProX or the mix vs. HGreX or the mix vs. AEX. Thus we cannot rule out the possibility that HProX can provide equally good performance as HGreX or AEX. Also, it is quite possible that HGreX or AEX alone can produce the same quality of results as the mix of all eight operators.

Note that the above rankings can be collated with the reports from [8, 11, 17]. Those reports describe the behavior of the same crossovers within their original TSP environment. Indeed, according to [11, 17], ERX should produce the best results, while OX should be the second best. Moreover, in [11] it is claimed that PMX and CX are inferior to OX. Similarly, [8, 11] state that both HGreX and AEX give discouraging results, although HGreX seems to be better than AEX. Obviously, our observations differ in several aspects from the cited reports, thus showing that the operator rankings valid for the TSP cannot be applied directly to the VRP.

| Crossover operator | The list of inferior crossovers |
|---|---|
| mix | HProX, OX, HRndX, ERX, CX, PMX |
| HGreX | OX, HRndX, ERX, CX, PMX |
| AEX | OX, HRndX, ERX, CX, PMX |
| HProX | ERX, CX, PMX |
| OX | CX, PMX |
| HRndX | CX, PMX |
| ERX | CX, PMX |
| CX | - |
| PMX | - |

Table 4: *Outcomes of statistical hypotheses testing - no mutation.*

| Crossover operator | The list of inferior crossovers |
|---|---|
| mix | OX, PMX, HRndX, ERX, CX |
| HGreX | OX, PMX, HRndX, ERX, CX |
| AEX | OX, PMX, HRndX, ERX, CX |
| HProX | PMX, ERX, CX |
| OX | CX |
| PMX | CX |
| HRndX | CX |
| ERX | CX |
| CX | - |

Table 5: *Outcomes of statistical hypotheses testing - mutation included.*

Note also that the study presented in this paper bears some resemblance to the study in [4]. Indeed, both papers evaluate VRP crossovers on roughly the same set of benchmark problem instances. Still, there are some differences that make direct comparison of results hard and dubious. The main difference is that our paper considers only crossovers borrowed from the TSP, while [4] introduces a specially tailored operator (turning out to be their best). In addition, [4] uses a slightly different chromosome representation, and a hybrid algorithm that combines evolution with local search.

## 6. Conclusion

Since the VRP has very much in common with the TSP, it is expected that both problems can be solved by similar EAs where the same components are reused. In this paper we have investigated how certain crossover operators originally designed for the TSP behave when they are applied to the VRP. More precisely, we have con-

sidered the following eight crossovers: OX, PMX, ERX, CX, AEX, HGreX, HRndX and HProX.

According to our experiments, all considered operators are suitable for the VRP, and they produce the best results when they are combined together. However, if only one operator is used in isolation, then the best performance is provided by HGreX or AEX. Very good results also are obtained by HProX. On the other hand, PMX and CX produce exceptionally poor solutions.

The obtained relative ranking of operators is quite different than within the TSP. Indeed, the most popular crossovers and apparently the best for the TSP are OX and ERX. Yet within the VRP environment both OX and ERX are clearly outperformed by AEX, HGreX or HProX.

In this paper we were only interested in exploring relative strengths and weaknesses of various crossover operators. We did not aim to design the most competitive algorithm for the VRP so far. Consequently, we have restricted ourselves to simple and purely evolutionary processes with no additional hybridizations or solution improvements such as local search. Still, we believe that the obtained results could serve as guidelines for construction of more sophisticated algorithms.

## References

[1] E. Alba, B. Dorronsoro, *Solving the vehicle routing problem by using cellular genetic algorithms*, in: *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization - EvoCOP 2004*, (J. Gottlieb and G. R. Raidl, Eds.), LNCS, Vol. 3004, Springer Verlag, Berlin, 11–20, 2004.

[2] E. Alba, B. Dorronsoro, *Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm*, Inform. Process. Lett. **98**(2006), 225–230.

[3] J. Berger, M. Barkaoui, *A parallel hybrid genetic algorithm for the vehicle routing problem with time windows*, Comput. Oper. Res. **31**(2004), 2037–2053.

[4] C. Bermudez, P. Graglia, N. Stark, C. Salto, H. Alfonso, *Comparison of recombination operators in panmictic and cellular GAs to solve a vehicle routing problem*, Inteligencia Artificial **46**(2010), 33–44.

[5] I. Choi, S. Kim, H. Kim, *A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem*, Comput. Oper. Res. **30**(2003), 773–786.

[6] B. Dorronsoro, *The VRP Web*, Languages and Computation Sciences Department, University of Malaga, `http://neo.lcc.uma.es/radi-aeb/WebVRP`

[7] A. Gog, C. Chira, *Comparative analysis of recombination operators in genetic algorithms for the traveling salesman problem*, in: *Hybrid Artificial Intelligent Systems - 6th International Conference Proceedings, Part II*, (E. Corchado, M. Kurzynski, M. Wozniak, Eds.), LNCS, Vol. 6679, Springer Verlag, Berlin, 2011, 10–17.

[8] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, D. Van Gucht, *Genetic algorithms for the traveling salesman problem*, in: *Proceedings of the 1st International Conference on Genetic Algorithms*, (J. J. Grefenstette, Ed.), Lawrence Erlbaum Associates, Mahwah NJ, 1985, 160–168.

[9] G. Gutin, A. P. Punnen (Eds.), *The Traveling Salesman Problem and Its Variations*, Kluwer Academic Publishers, Dordrecht NL, 2002.

[10] H-S. Hwang, *An improved model for vehicle routing problem with time constraint based on genetic algorithm*, Computers and Industrial Engineering **42**(2002), 361–369.

[11] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, S. Dizdarevic, *Genetic algorithms for the travelling salesman problem: a review of representations and operators*, Artificial Intelligence Review **13**(1999), 129–170.

[12] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Edition, Springer, New York, 1995.

[13] L. S. Ochi, D. S. Vianna, L. M. A. Drummond, A. O. Victor, *A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet*, in: *Parallel and Distributed Processing*, (J. D. P. Rolim, Ed.), LNCS, Vol. 1388, Springer Verlag, Berlin, 1998, 216–224.

[14] P. Pongcharoen, D. J. Stewardson, C. Hicks, P. M. Braiden, *Applying designed experiments to optimize the performance of genetic algorithms used for scheduling complex products in the capital goods industry*, J. Appl. Stat. **28**(2001), 441–455.

[15] C. Prins, *A simple and effective evolutionary algorithm for the vehicle routing problem*, Comput. Oper. Res. **31**(2004), 1985–2002.

[16] M. Spiegel, L. Stephens, *Schaum's Outline of Statistics*, Fourth Edition, McGraw-Hill, New York, 2011.

[17] T. Starkweather, S. McDaniel, K. E. Mathias, L. D. Whitley, C. Whitley, *A comparison of genetic sequencing operators*, in: *Proceedings of the 4th International Conference on Genetic Algorithms*, (R. K. Belew, L. B. Booker, Eds.), Morgan Kaufmann, San Francisco CA, 1991, 69–76.

[18] K. C. Tan, L. H. Lee, Q. L. Zhu, K. Ou, *Heuristic methods for vehicle routing problem with time windows*, Artificial Intelligence in Engineering **15**(2001), 281–295.

[19] P. Toth, D. Vigo (Eds.), *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2002.

[20] Y. Zhong, M. H. Cole, *A vehicle routing problem with backhauls and time windows: a guided local search solution*, Transportation Research Part E: Logistic and Transportation Review **2**(2005), 131–144.