Genetic Algorithm with Multiple Crossovers on the Traveling Salesman Problem

Paulo R. Lafeta Ferreira
DigiPen Institute of Technology
Redmond, WA
paulo.ferreira@digipen.edu

Pushpak Karnick
DigiPen Institute of Technology
Redmond, WA
pushpak.karnick@digipen.edu

ABSTRACT

The "'Travelling Salesman Problem"' (TSP) is one of the most notorious NP-Complete problems. Due to its complexity, Genetic Algorithms can be used to get approximate solutions for it. However, good crossovers and strategies must be used to get solutions closer to the optimal within the time constraints. This article describes seven different crossovers and their performances on the TSP. Besides that, we also apply a technique that runs Multiple crossovers during a single execution and analyze the results of it.

Categories and Subject Descriptors

D.2.8 [**Artificial Intelligence**]: Genetic Algorithms—*tsp solution, crossover methods*

Keywords

genetic algorithms, traveling salesman problem, tsp, crossovers

1. INTRODUCTION

1.1 Traveling Salesman Problem (TSP)

The "'Travelling Salesman Problem"' (TSP) is about a traveling salesman that must visit every city in his assigned territory exactly once and then return back to the starting point. The goal is to find the best tour with the lowest possible cost. A possible search space for the TSP problem is a permutation of the n cities. Any single permutation of n cities yields a solution, which is interpreted as a complete tour, with a return home implied as soon as the last city in the sequece is visited.

1.2 Genetic Algorithms

Genetic Algorithms (GA) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such, they represent an intelligent exploitation of a random search within a defined search space to solve a problem. [1] The algorithm can be summarized as:

1. Randomly generate an initial population.
2. Calculate the fitness for each individual.
3. Define selection probabilities for each individual.
4. Select probabilistically individuals from population.
5. Produce an offspring via crossover methods.
6. Repeat step 2 until solution is obtained or maximum number of iterations is reached.

It's also applicable a Mutation step on GA, where you can slightly mutate individuals. However, we are not using that technique in this article.

1.3 Using Genetic Algorithms to solve TSP

Genetic Algorithms can be used to get approximate solutions for TSP. Each individual of the population represents a tour and the fitness of each individual is defined as the total cost (distance traveled) of the tour. As this function is to be minimized, a configuration with a better fitness value necessarily has a lower function value. Each individual can be represented as a string of values that are defined over an alphabet with n elements. The values encoded in this string can be of the following three types: Adjacency, Ordinal and Path. We will discuss these in a later section.

With Genetic Algorithm, a reasonable approach can be done on TSP with high number of cities, generating solutions that are not necessarily optimal, but as close as possible within the time constraints.

2. EXAMPLES OF BIG INPUTS AND SOLUTIONS

Before explaining the strategies used and the algorithm tests, this article will show some big inputs and solutions of TSP by Genetic Algorithm, to exemplify its use and practice. These examples are from the website [3], which has several studies about the subject and results from big Traveling Salesman Problem. Some of them are TSP challenge problems consisting of cities in Argentina through Zimbabwe.

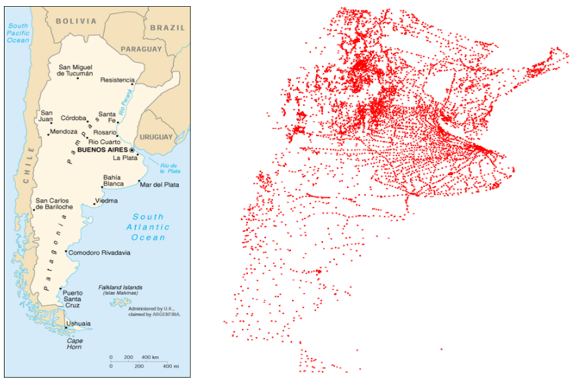

Figure 1: Argentina TSP input: 9152 cities.

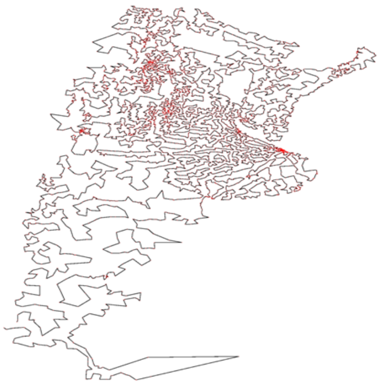

Figure 2: Argentina TSP output. Path size found: 44,232 mi, at maximum, 0.012% greater than optimum solution.

- **Argentina**

  Argentina with 9152 cities still haven't gotten a optimal solution with GA yet. Its input can be seen in Figure 1 and its output on Figure 2.

- **China**

  China, more than 7 times bigger than Argentina, also didn't get an optimal solution yet. Its input can be seen in Figure 3 and its output on Figure 4.

- **World Tour**

  That's a huge input for TSP and required a lot of processing power to find an approximate solution. Its input can be seen in Figure 5 and its output on Figure 6.

  The biggest national TSP solved by this research group with optimal solution until now is **Sweden**. With 24,978 cities, the path size found is 45,049 mi. Figure 7 shows the path solution.

- **Sweden**

  It used 96 clusters dual Intel Xeon 2.8 Ghz and took approximately 3 years. In an usual computer it would take around 84 years.

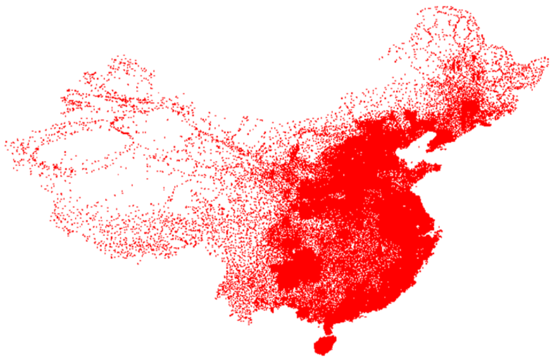

Figure 3: China TSP input: 71,009 cities.

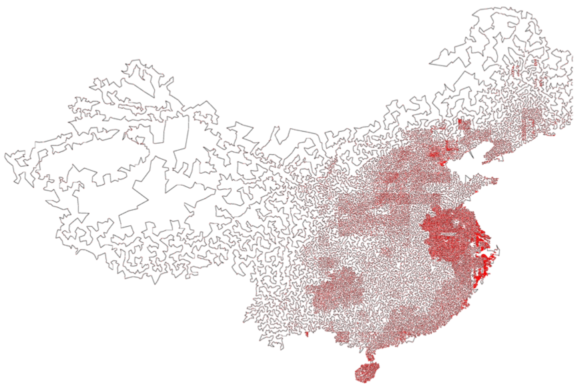

Figure 4: China TSP output. Path size found: 241,188 mi mi, 0.024% near the optimum solution.

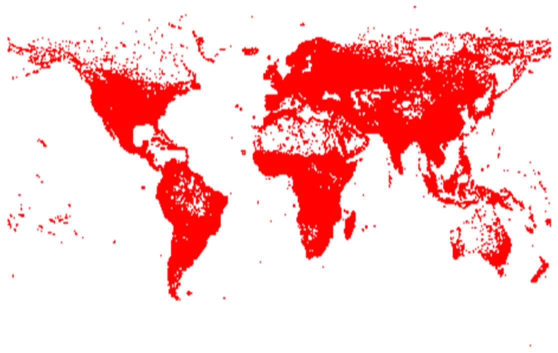

Figure 5: World Tour TSP input: 1,904,711 cities.

Figure 6: World Tour TSP output. Best path size found: 396,967,677 mi, at maximum, 0.04996% greater than optimum solution.

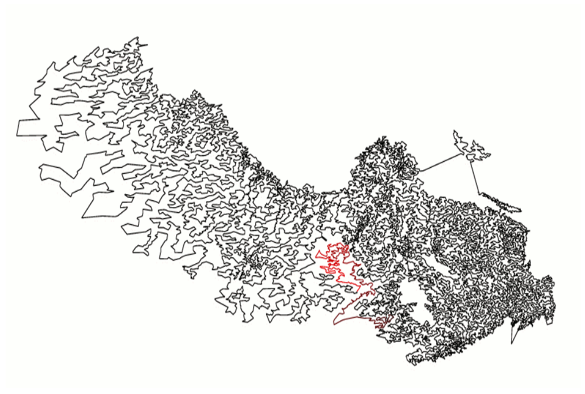

Figure 7: Sweden TSP optimal solution.

After these examples of big inputs, we will verify how important it is to have a good crossover method. With an efficient GA, you save a lot of processing time and get closer to the optimal solution, if not the optimal solution itself. Crossover method is one of the main aspects to the efficiency of a GA, along with a Selection method. In this article, we'll discuss 7 different crossover methods and a technique that use multiple crossovers in one single execution. And also, explain a few strategies to improve the GA.

3. ENCODING TYPES AND CROSSOVERS OPERATORS

In this section we are going to present the different types of encoding to represent the individuals/tours and their crossovers. The crossovers operators gets as input two parents and generates two children to the new population.

3.1 Path

Path encoding assigns at every location the index of the visited node. Every node is assigned a number in the range [0, n-1], and the solutions represent a sequence ordered by the visited cities. For e.g. a tour 5-1-7-8-9-4-6-2-3 is represented as (5 1 7 8 9 4 6 2 3).

3.1.1 Partially Mapped (PMX)

Partially Mapped (PMX) crossover builds an offspring by choosing a sequence of the tour from one parent and preserving the order and position of as many cities as possible from the other parent. A subsequence of the tour is selected by choosing two random cut-points, which serve as boundaries for the swapping operations. For e.g. in the two parents shown below (with cut-points marked by '|') :

p1 = [1 2 3 | 4 5 6 7 | 8 9] and p2 = [4 5 2 | 1 8 7 6 | 9 3]

would produce offspring as follows. First the segments between the cut-points are swapped (we use "'x"' to denote currently unknownĄh)

o1 = [x x x | 1 8 7 6 | x x] and o2 = [x x x | 4 5 6 7 | x x]

This swap also defines a series of mappings, $1 \rightleftharpoons 4$, $8 \rightleftharpoons 5$, $7 \rightleftharpoons 6$, and $6 \rightleftharpoons 7$.

Then, we can fill in additional cities from the original parents for which there is no conflict:

o1 = [x 2 3 | 1 8 7 6 | x 9] and o2 = [x x 2 | 4 5 6 7 | 9 3]

Finally, the first "'x"' in offspring o1 (which should be 1, but there was a conflict) is replaced by the value 4 because of the mapping $1 \rightleftharpoons 4$. We replace the other positions in a similar fashion. The offspring thus generated are:

o1 = [4 2 3 | 1 8 7 6 | 5 9] and o2 = [1 8 2 | 4 5 6 7 | 9 3]

3.1.2 Order (OX)

Order (OX) crossover builds offspring by choosing a subsequence from one parent, and filling in the cities from the other parent such that the relative ordering of visitation is preserved. Referring to our example above, if the parents are p1 = [1 2 3 | 4 5 6 7 | 8 9] and p2 = [4 5 2 | 1 8 7 6 | 9 3]

we begin by first copying the segments inside the cut-points into the corresponding offspring:

o1 = [x x x | 4 5 6 7 | x x] and o2 = [x x x | 1 8 7 6 | x x]

Next, from the second cut-point of one parent, the cities are copied in order from the second parent such that there are no conflicts. For p2 the tour sequence starting from the second cut-point is (9 - 3 - 4 - 5 - 2 - 1 - 8 - 7 - 6). After removing cities 4, 5, 6 and 7 (which have been copied over from the first parent p1, we now have a tour (9 - 3 - 2 - 1 - 8). This sequence is placed in the first offspring starting from the second cut-point:

o1 = [2 1 8 | 4 5 6 7 | 9 3] and similarly, we have o2 = [3 4 5 | 1 8 7 6 | 9 2] The OX crossover exploits the fact that relative order of cities matters to the overall quality of the result.

3.1.3 Cycle (CX)

Cycle (CX) crossover builds offspring in such a way that each city and its position comes from one of the parents. Consider two parents:

p1 = [1 2 3 4 5 6 7 8 9], and p2 = [4 1 2 8 7 6 9 3 5].

produce the first offspring, by taking the first city from the first parent:

o1 = [1 x x x x x x x x]

The next city to choose is 4 because it is Š"just below"' 1 (remember, each city is chosen from both the parents alternately). We position this city according to the position of p1, hence we get:

o1 = [1 x x 4 x x x x x]

The next city to choose is the city from p2 that is Šjust belowŠ 4, i.e. 8. Placing 8 and continuing ahead we get:

o1 = [1 2 3 4 x x x 8 x]

The last city placed was 2. The next city that is below 2 in p2 is 1, which has already been placed. At this point, we have introduced a cycle in the tour, hence we discard the search mechanism and fill in the values from the second parent in the available slots. Thus we now have:

o1 = [1 2 3 4 7 6 9 8 5], and similarly o2 = [4 1 2 8 5 6 7 3 9]

CX preserves the absolute position of the elements in the parent sequence.

3.2 Adjacency

Adjacency encoding uses each position in the string to represent an edge of the tour. So, if the jth position in the sequence contains the value k, then the edge (j, k) exists in the tour. For e.g. the tour 1-2-4-3-8-5-9-6-7 is represented as (2 4 8 3 9 7 1 5 6).

3.2.1 Alternating Edges

Alternating Edges crossover builds the offspring from two parents by randomly choosing an edge from the first parent, then selecting appropriate edge from the second parent, etc. The operator extends the tour by choosing edges from alternating parents. In event of a conflict due to a sub-tour formation (cycles), the operator instead selects a random edge from the remaining edges that does not introduce cycles. For e.g. if the parents are:

p1 = [2 3 8 7 9 1 4 5 6], and p2 = [7 5 1 6 9 2 8 4 3],

the offspring using this operator might be

o1 = [2 5 8 7 9 1 6 4 3],

where the city 6 is introduced at random from the remaining cities, to avoid a cycle because the city 8 was already chosen earlier.

3.2.2 Subtour-chunks

Subtour-chunks crossover constructs an offspring by choosing a random-length subtour from one of the parents, than choosing a random length subtour from the second parent, etc. - the operator extends the tour by choosing random length chunks from alternating parents. As before, if the addition of a chunk introduces a cycle in the partially created tour, the offending edge is replaced by another edge (chosen at random from those available at that point) that avoids the cycle.

3.2.3 Heuristic

Heuristic crossover This operator builds an offspring by choosing a random city as the starting point for the offspringŠs tour. It then compares the two edges that emanate from this city from both the parents and chooses the one which has lower cost. The city on the other end of this edge no serves as the starting point in selecting the shorter edge that leaves this city, and so on. If at some stage, we introduce a cycle in the tour, a random edge is selected from the list of available cities and assigned to the current edge position.

3.3 Ordinal

With a predefined fixed ordering of the cities that is used as a reference point (C), say:

C = [1 2 3 4 5 6 7 8 9]

A tour [1 2 4 3 8 5 9 6 7] is represented as a list l of references, l = [1 1 2 1 4 1 3 1 1]

Each number of l refers to a number in C. For example, take 1 from l, which refers to the first number of C (1). Remove 1 from C and get the next number from l, which is 1 again. That means we have to take the first number of C again, which now is 2. So, so far we 1 - 2 as path. Remove 2 from C and get next number from l, which is 2, so we have to get the current second number from C, which is 4. So the tour now is 1 - 2 - 4. And so on. Continuing this, we get the tour: 1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7.

The advantage of this representation is that the **Cut-and-splice** crossover method actually works in this case as we are working with relative indices in truncated versions of the starting configuration, and this does not lead to conflicts. The disadvantage of this method, other than the computational expense, is the fact that the relatively good ŞlocalŤ subtours in the parent are not preserved.

3.3.1 Cut-and-splice

Cut-and-splice crossover, which involves taking a part from one parent, and splicing it with the remainder from the second parent works without any modification or special processing. However, the potential pitfall that can arise from such a method is that the original cohesiveness of the solution (in terms of city proximity) for part of the solution is not in general preserved after splicing. A nearly optimal subtour may be replaced by a totally random permutation that actually increases the cost. In other words, this crossover is notorious for the presence of local optima that affect the overall search quality.

4. STRATEGIES

4.1 Elitism

Some little improvements were done in GA to increase its performance. One of them is to always maintain the best individual found so far in population, but only if the population size is equal or greater than 4. This is somewhat a form of Elitism, which is a process of selecting individual with a bias towards the better ones. This increased the performance of GA greatly, but it has its risks, as it can make your GA converge quicker, which means that all individuals will be very similar after a certain iteration. Anyway, this strategy showed to be crucial, as it increased greatly the quality of the solutions found.

4.2 Getting out of Local Optima

A local optima of a combinatorial optimization problem is a solution that is optimal (either maximal or minimal) within a neighboring set of solutions. This is in contrast to a global optimum, which is the optimal solution among all possible solutions. [2] `http://en.wikipedia.org/wiki/Local_optimum` . Genetic Algorithms has, as one of its disadvantages, to present a high frequency of being stuck in local optimas. Fortunately, there are ways to outline this.

In our experiments, to avoid being stuck in local optima, we randomize one or two children when both are equal. That means that, if after a reproduction both of the children generated are equal, then we randomize one or both of them to the next population. This allowed our GA to flow better. Before that, it was frequently getting stuck in sub-optima solutions. Another option used is to randomize the entire population, if best solution does not change after n iterations.

5. MULTIPLE CROSSOVERS

After a while, single crossovers methods start lacking variety in their reproductions, as we'll see in the Tests section. Some solutions are possible for this issue, as for example, randomize part or the whole population. But this solution most of the times is not efficient, as you depend more heavily on lucky and you can lose some good sub-tours. Other possible solution proposed and experimented by this article is doing **Multiple Crossovers** in one single execution. That means switching the crossover method being used in real time.

On the experiments made in this work, we switched the crossover method whenever the current crossover showed to have converged and there is no more ”‘creativity”’ to create a better path solution. So, after n iterations without changing the best path solution, we then switch the crossover method to the next one. If the next crossover has a different encoding type of the current one, then a conversion of all individuals of the population is necessary. Conversions can happen between Adjacency - Path, Path - Ordinal and Ordinal - Adjacency.

The crossovers used for this technique in most of the tests are:

- Heuristic (Adjacency)
- Partially Mapped (Path)
- Order (Path)
- Subtour-chunks (Adjacency)

These ones were selected to be used in this technique due to their higher performance compared to the other single crossover methods in the tests. The ones not used are: Cycle (Path), Alternating Edges (Adjacency) and Cut-and-splice (Ordinal). A few tests were done with all of the 7 crossovers, and these confirmed that the multiple crossovers technique works better with only the most efficient methods.

6. TESTS

Some tests were made over inputs given by [3]. The results are an average of two executions. Tables with some results are shown in Appendix A.

6.1 Comparing the Crossover methods

6.1.1 Djibouti

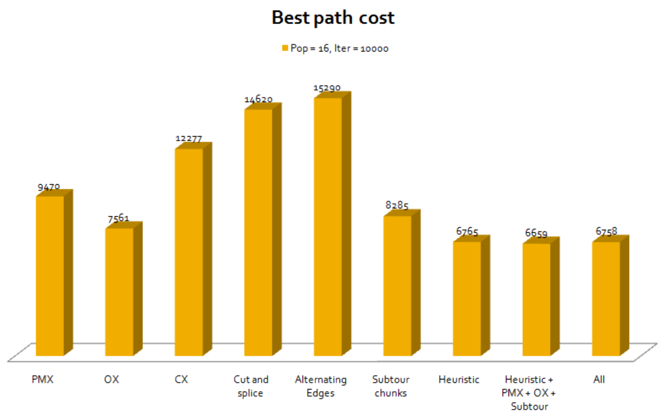

Figure 8: Djibouti results for Population of 16 and 10000 iterations.

The chart in Figure 8 shows the great performance of the Heuristic, Multiple (Heuristic+PMX+OX+Subtour) and All (all 7 crossovers). Order also did good. PMX and Subtour-Chunks didn't have a very well performance and CX, Cut-and-splice and Alternating Edges were the worst. These are the first results shown here, but they reflect the same results from most of the other tests.

In Figure 9, the drawings in black background show the best output of our implementation and the ones in white background is the optimal solution found by [3].

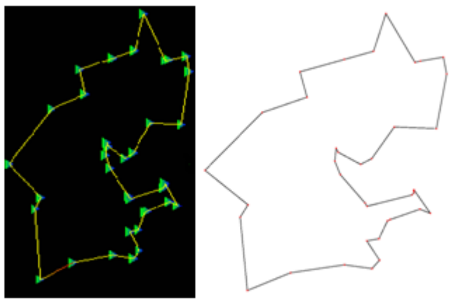

Figure 9: Djibouti: best solution found.

6.1.2 W. Sahara

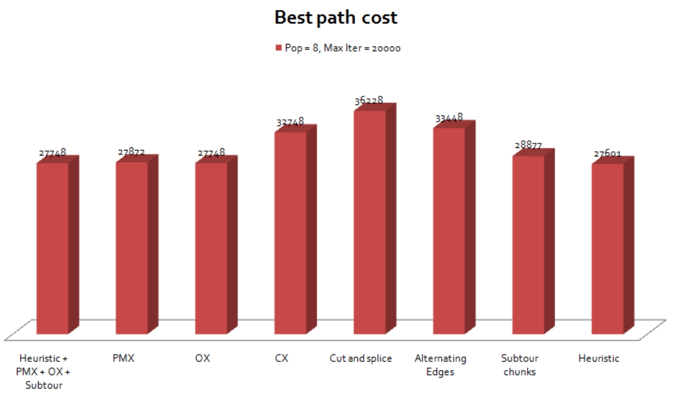

Figure 10: Sahara results for Population of 8 and 20000 iterations. Optimal solution: 27601.

This chart on Figure 10 shows good performance of the Multiple crossovers technique, Heuristic, PMX, OX and Subtour-chunks, similar to the Djibouti input.

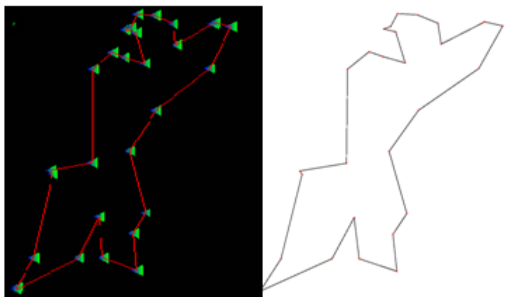

Figure 11: W. Sahara best solution found.

6.1.3 Qatar

In Figure 12, the chart shows results related to a varying number of iterations.

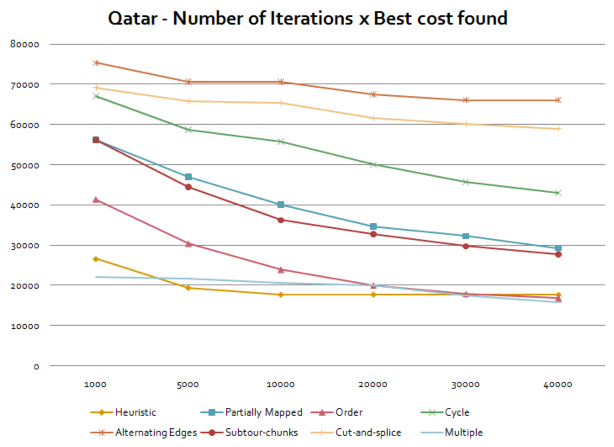

Figure 12: Qatar: Number of Iterations x Best cost found.

To reinforce the earlier results, we have the Multiple, Heuristic and Order as the best crossover results. Followed by Subtour-Chunks, Partially Mapped, Cycle, Cut-and-Splice and Alternating Edges.

In Figure 13 we have tests with different population sizes.

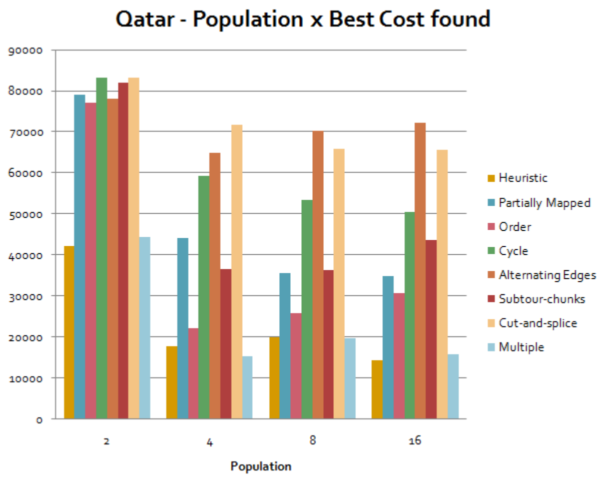

Figure 13: Qatar: Population x Best cost found. Number of iterations: 10000

A little bit of surprise to the population size 4, as some crossovers showed better results with this size, instead of 2, 8, 16. Showed better results with population size 4: Multiple, Order, Alternating Edges and Subtour-Chunks. Others showed better performance on population size 16.

A bigger test, with 200,000 iterations, is shown in Figure 14:

There, both shown approximate solutions to the optimal. Also, **Heuristic** presented a little better solution than Mul-

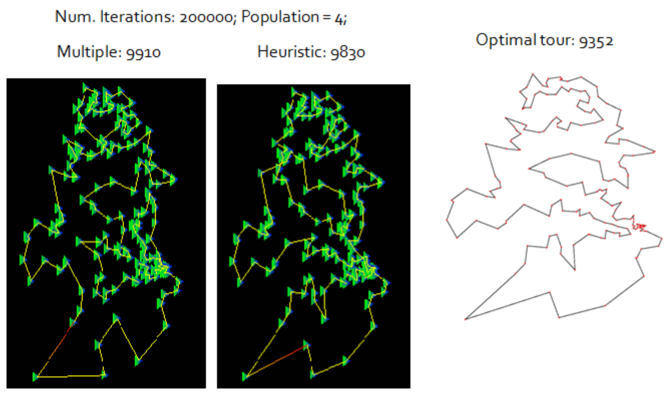

Figure 14: Qatar: approximate solutions obtained shown in the images with black background.

tiple, but we will see in a next, bigger test, that it's not always like that.

6.1.4 Uruguay

The biggest test made was Uruguay, with 734 cities and 400,000 iterations, is shown in Figure 15.

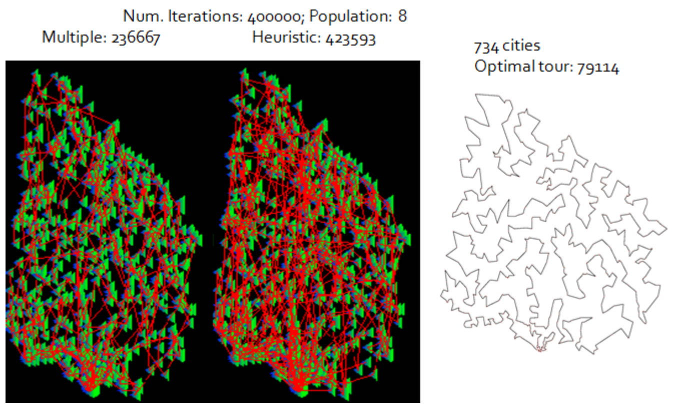

Figure 15: Uruguay: as before, approximate solutions obtained shown in the images with black background.

In this relatively big test, we noticed that the Multiple technique was always converging to a solution. As far as it gets, slower it gets at the same time. But it almost doesn't stop converging, probably due to it's variety of crossover methods acting over it. As for the Heuristic, it seemed to act in an opposite way. It converged quickly to a solution and then it didn't get out of there. Get stucky, depending more on the luck of getting another good random population. It was done around 6 tests with heuristics, all of them got stuck in a solution between 350,000 to 450,000. As for the Multiple, it was almost always converging to a better solution, even if slowly.

6.2 Comparison between Heuristic and Multiple Crossovers technique

Next we have a test with an input of 131 cities. In Figure 16, optimal solution can be seen in the white background image and the approximate solution found by our tests is in the black background image.

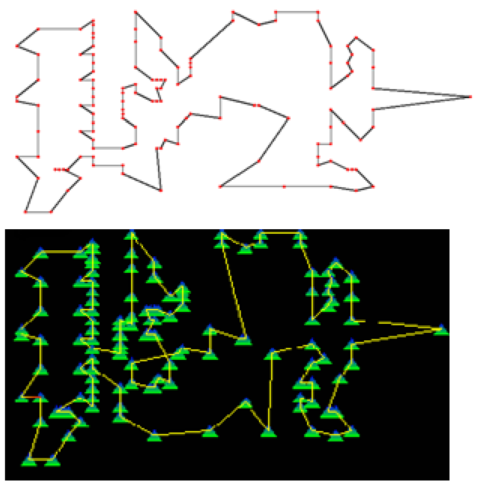

Figure 16: XQF131 output. Optimal and approximate solutions.

In Figure 17, we can see the results for Heuristic crossover only and the Multiple (Heuristic + OX + PMX + Sub-Chunks). The optimal tour has size 564.

Similar to what happened in "'Uruguay"' test, Heuristic converget quickly to a good solution and there seemed to get unchanged. As for the Multiple, it's always converging to a better solution, though in a slower pace.

7. CONCLUSIONS

In this article, we implemented 7 different crossover methods in a GA using some simple strategies such as Elitism and randomizing the offspring. As a different approach, we also did a Multiple crossovers technique that switchs the crossovers in real time during execution, trying to reach the best solution possible.

From all these 7 implemented, Heuristic showed the best performance individually. It converges quickly to a relative good solution. However, after an amount of iterations it gets "'stuck"' in local optima, and randomizing part or the whole population doesn't fix this problem, unless the algorithm gets a lucky configuration.

Heuristic is followed by the crossover **Order**, as it also presented good results. Based on this work, the order of best individual crossovers in the inputs used are:

1. Heuristic (Adjacency)
2. Order (Path)
3. Subtour-chunks (Adjacency)
4. Partially Mapped (Path)
5. Cycle (Path)

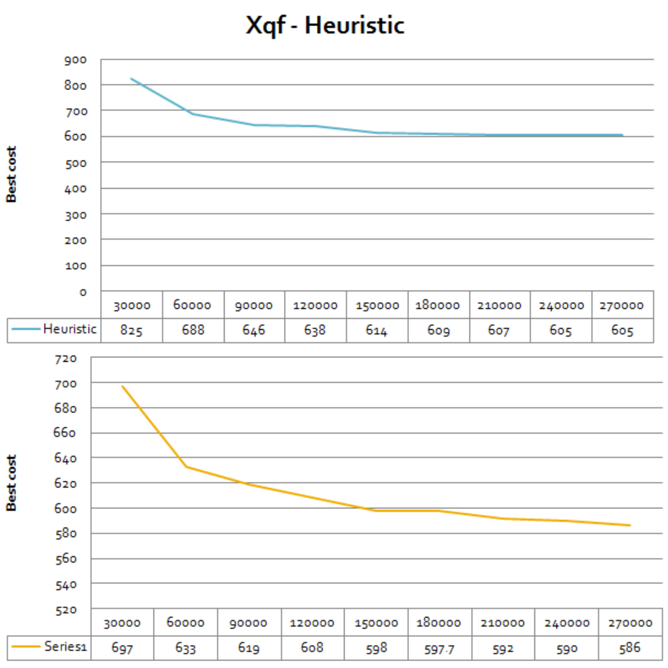

Figure 17: Comparison between Heuristic and Multiple on XQF131 input, varying number of iterations.

6. Cut-and-splice (Ordinal)
7. Alternating Edges (Adjacency)

Ordinal with its **Cut-and-splice** is easy to implement and works well in some GA problems, but seems to not be efficient in TSP specifically.

The Multiple crossovers technique showed the best solutions along the Heuristic for relatively small inputs and few iterations. As for bigger ones and longer executions, the Multiple crossovers highlighted its advantages. Heuristic converges too fast and get ”‘stuck”' easier in local optima. The Multiple crossovers converges a little less quick, but continuously keep finding better solutions, though in a lower pace as time passes by. That said, Multiple technique showed to be a safer bet in long executions and bigger inputs, due to its continuous discovery of new solutions.

Improvements can be made in the Multiple crossovers technique in several ways. One of them is to store information about the crossovers individual performance during execution and then select the current best techniques more frequently, in a similar way of the Selection process from GA, giving more probability for the best ones. Other idea that might improve it is to vary the number of iterations necessary to switch crossover, based on their performance. On this work, the switch occurs after a constant number of times without change of the best solution. Another approach that can also be taken, is to divide the population into groups and apply to each of them an individual crossover. Then switch the crossover of each group independently according to their performance and also some individuals between them.

8. REFERENCES

[1] Genetic algorithms. Available in: `http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html`.

[2] Local optimum. Available in: `http://en.wikipedia.org/wiki/Local_optimum`.

[3] The traveling salesman problem. Available in: `http://www.tsp.gatech.edu/`.

APPENDIX

A. TABLES

Table 1: Djibouti - Population: 8, Iterations: 1000. Optimal solution: 6659

PMX	OX	CX	Cut and Splice	Alternating Edges	Subtour Chunks	Heuristic	Multiple (4)	All
12169	9713	16520	16662	15832	12517	7077	6811	7141

Table 2: Djibouti - Population: 16, Iterations: 10000. Optimal solution: 6659

PMX	OX	CX	Cut and Splice	Alternating Edges	Subtour Chunks	Heuristic	Multiple (4)	All
9470	7561	12277	14620	15290	8285	6765	6659	6758

Table 3: W. Sahara - Population: 8, Iterations: 20000. Optimal solution: 27601

PMX	OX	CX	Cut and Splice	Alternating Edges	Subtour Chunks	Heuristic	Multiple (4)
27872	27748	32748	36228	33448	28877	27601	27748

Table 4: W. Sahara - Population: 16, Iterations: 10000. Optimal solution: 27601

PMX	OX	CX	Cut and Splice	Alternating Edges	Subtour Chunks	Heuristic	Multiple (4)
30662	28554	37684	39612	46438	29866	28103	27601

Table 5: Qatar - Population: 8, Iterations: 40000. Optimal solution: 9352

PMX	OX	CX	Cut and Splice	Alternating Edges	Subtour Chunks	Heuristic	Multiple (4)
29266	16799	43006	66098	27744	58970	17638	15797