Abstract

Traveling Salesman Problem consists in finding the shortest path from one city to the remaining and only passing trough each of them one time. The solution present in this document is based on a Genetic Algorithm. This is one of my projects while working in this area. If you want to have the code (Matlab), please send me an email.

Mail	florez.fernandez.david@gmail.com

Contents

1 Introduction

The Traveling Salesman Problem (TSP) [1] is a very well known problem in computational optimization. The problem consists in a number of cities and the objective is to find the path of minimum distance that joins every city but only visiting one city one time. Figure 1 shows an easy example of how graphically can be represented cities, distances and paths. Each node in the node represents a city and each arc between them represents the path from one to the other. It a very often used problem in computation task for comparing the performance of new optimization methods. The history of the problem is not clear at all. Some experts dates the birth on traveling handbook for sales man in 1832. But in that text any mathematical representation was missed. Later on the same century the mathematician Hamilton and Kirkman were the first authors of a formal representation of the problem. Hamilton defines what is known as Hamiltonian Cycle which definition is a path in an undirected or directed graph that visits each vertex exactly once. Lately in the 20th century the problem starts to be very relevant in the recently created field of computation. One the most important applications of the TSP is its use to build layouts of actual printed circuits.

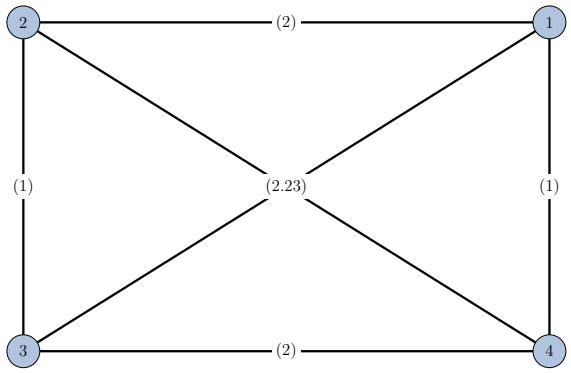

Figure 1: All possible paths with distance in parentheses from city to city for 4 cities problem.

The problem has a very vast methods to be resolved. Briefly they can be classified in:

- **Exact Algorithms**: All of those algorithms for finding exact solutions (they will work only for small problem sizes, for bigger they are very slow). Some examples are: brute force search, Held-Karp algorithm (we will revisited this one in other section of the document), branch-and-bound algorithms, cutting-plane method...
- **Heuristics Algorithms**: All of those algorithms that deliver probably good solutions, but which could not be proved to be optimal. Some examples are: nearest neighbor (NN) algorithm, Christofides algorithm, Match Twice and Stitch (MTS), Ant colony optimization, Pairwise exchange, V-opt heuristic...
- **Genetic Algorithms**: As general purpose algorithms for optimization this kind of methods work fast and they reach reasonably good solutions. This document is focused in this kind of resolution.

After this brief introduction in history and methods applied to resolution, the rest of the document will cover the mathematical formulation of the problem (Section 2), the computational complexity of the problem (Section 3), the implemented genetic solution program description (Section 4), the results obtained from a range of configurations and comparatives versus other methods and bounds (Section 5), and finally the conclusions of this study (Section 6).

[1] Some material was read from Reference [1]

2 Mathematical Formulation

Here we will follow Reference [2]. Suppose a Graph G with m nodes and n arcs and a cost $c_{i,j}$ associated with each arc (i, j). Then the TSP can be formulated as: Find the shortest path (less expensive) from node 1 to node m in graph G. The shortest is measured like the sum of the cost (ie distance) of each arc in the path.

The TSP can be explained in the context of flow problems. It is only necessary design a graph to send one unit of flow from node 1 to node m with minimum cost. Then the mathematical formulation is as shown in Equation (1) where the sums and restrictions are taken on arcs of G. $I_{i,j}$ indicates if the arc is in the path or not.

$$
\begin{aligned}
&Minimize \sum_{i=1}^{m}\sum_{j=1}^{m} c_{i,j} I_{i,j} \\
&subject\ to \sum_{j=1}^{m} I_{i,j} - \sum_{k=1}^{k} I_{k,i} = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i \in \{1, m\} \\ -1 & \text{if } i = m \end{cases} \\
&\text{With } I_{i,j} \in \{0, 1\} \quad i, j = 1, 2, 3, \ldots, m
\end{aligned} \tag{1}
$$

The costs $c_{i,j}$ are defined as the euclidean distance from node i to node j in two dimensions x and y with coordinates (x_i, y_i) and (x_j, y_j), i.e., $c_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

3 Computational Complexity

TSP problem can not be resolved with polynomial-time algorithms: on city inputs of size m, their worst-case running time is not bounded by $O(m^k)$ for any constant k. TSP cannot be solved by any computer, no matter how much time is provided. This characteristic is not unique for the TSP, there are a whole group of problems with this kind of property. They are known as NP-hard problems.

The most notable characteristic of NP-hard problems is that no fast solution to them is known. That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows. As a consequence, determining whether or not it is possible to solve these problems quickly, is one of the principal unsolved problems in computer science today. To see the demonstration of NP-hardness of TSP see Reference [3].

As long as TSP is a NP-hard problem, when the number of cities m of the problem increases, the optimal solution is more and more difficult to find it. A genetic algorithm can converges to a solution but it is quite difficult to ensure whether or not it is the optimal solution or just a local optimum (Premature convergence).

4 Genetic Algorithm

Until this point the complexity of the TSP was clearly exposed and how difficult is its resolution by exact methods. Now a genetic algorithm will be described and implemented to demonstrate its big usefulness. The author has followed the general scheme of the algorithm as seen in Reference [4]. The algorithm scheme implemented as shown in Figure 2.

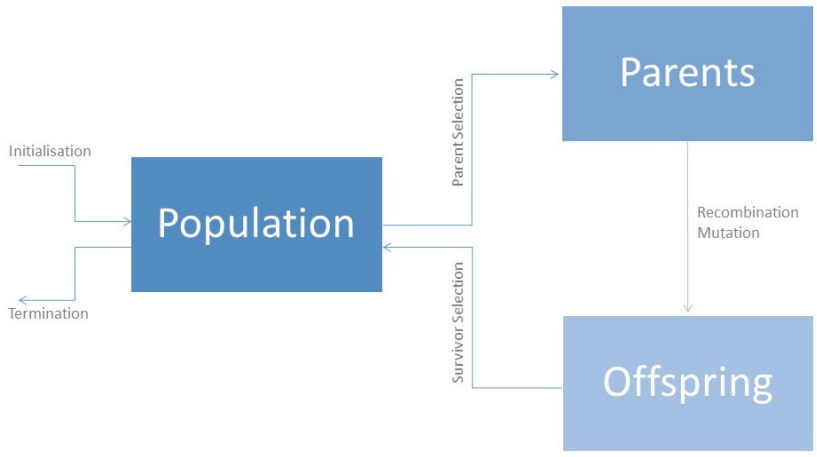

Figure 2: Scheme of the algorithm implemented.

The software chosen for implementing the algorithm is Matlab (Version 7.10 R2010a). It was programmed with the object oriented programming (oop) tool in Matlab. The author has decided the implementation with objects instead of functional programming to start the building of a library along the course and to be able to have 'recyclable' methods and operators. Following the details of the implementation will be detailed. Table 1 shows a summary of the methods implemented.

4.1 Input data

Some functions were elaborated in Matlab to create and read .txt with the coordinates of the cities. Those functions are: 'Ciudades.m' and 'readCSV.m'. The first one create random coordinates for m cities. The problem was reduce, without loss of generality, to coordinates inside the $[0,1] \times [0,1]$. On the other side the 'readCSV.m' file extracts the information from any csv file with two columns and so much rows as cities. It is also possible to use other input data, only changing the input of the coordinates data to the new one. The 'readCSV.m' extracts so many cities from the file as desired.

4.2 Chromosome representation

The first step in a genetic algorithm is the selection of a proper representation. This representation is made in comparation with nature and because of this it is usual named chromosome representation. Folling the nature, a chromosome encode the information neccesary to understand the phenotype. In our TSP problem as we see in Section 2 there are many possible paths passing trough every city. So the question is How can it be coded in a simply way? The answer here presented is permutations. Given the problem with m cities then every permutation of the integers from 1 to m is a possible path. The set of all permutation from 1 to m is denoted by $Perm(1, m)$. To clarify how the chromosome coded with permutation some examples are shown in Figure 3 for a TSP with 4 cities.

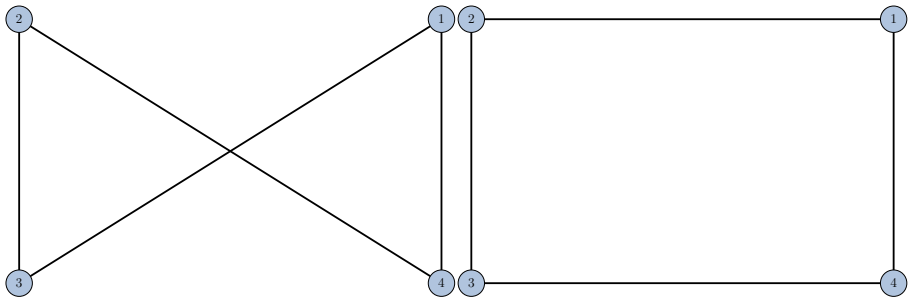

Figure 3: Phenotypes of genotype (1,3,2,4) in the left and genotype (1,2,3,4) in the right.

Note that using this representation is trivial that the number of possible solutions of the TSP is $(m-1)!$. So the optimum path must be found in $(m-1)!$ paths. Figure 4 shows how sharply the number of possibilities grows.

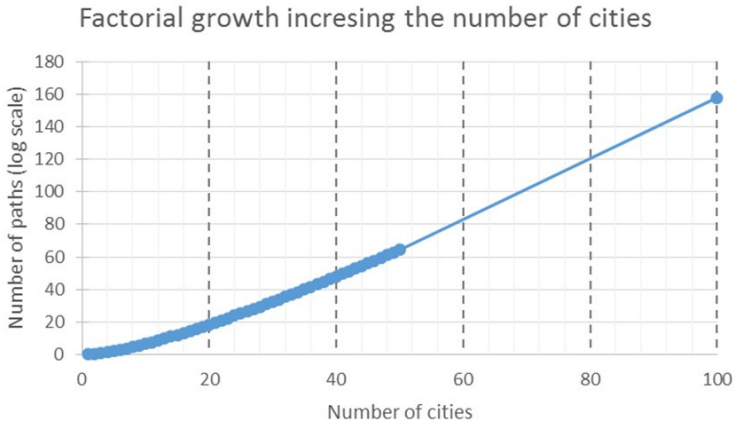

Figure 4: Explosive growth of feasible paths when the number of cities grows.

Summarizing:

Given m cities denoted by 1, ... , m then all possible paths belongs to $Perm(1, m)$.

4.3 Fitness function

The role of fitness function is representing the requirements the population should adapt to. As long as the library used in the developing of this exercise was though as modular and scalable, only is prepared to threat maximization of the objective function. Because of this the fitness function is implemented as a solely .m file in Matlab and calculated formally as the equivalent maximum (see Equation (2)).

$$\text{Natural TSP objective } := Min(c_{i,j}) \Longleftrightarrow \text{Maximization objective } := Max(\frac{1}{c_{i,j}}) \tag{2}$$

Then the fitness equation implemented is $Max(\frac{1}{c_{i,j}})$. Some considerations have to be considered with this function. If the distance between two cities is zero then the function will be infinity. This kind of problem is not possible under the code because the cities are obteined with pseudorandom sequence of numbers, and the probability of getting two same random numbers is zero.

Summarizing:

Fitness Function $:= Max(\frac{1}{c_{i,j}})$.

4.4 Initialization

The initialization or first generation of the problem is done randomly. In general, it could be said that the typical progress curve of an evolutionary process makes heuristic initialization methods unnecessary (see chapter 2 of Reference [4]). It is assumed a constant population size in each generation. And also it is important to input a population size even.

Summarizing:

Initialization is random.

4.5 Parent Selection

The role of parent selection is to distinguish among individuals based on their quality, and, in particular, to allow the better individuals to become parents in the next generation. The author has implemented a very easy method random that is not good in order to get good solutions and another one much better, the fitness proportional.

Fitness Proportional

In the Fitness Proportional operator, a Roulette wheel method was implemented. The description of the method is the following:

- The first step is to calculate the cumulative fitness of the whole population through the sum of the fitness of all individuals, $\sum_{i=1}^{m} f_i$.
- After that, the probability of selection P_{Sel} is calculated for each individual as being $P_{Sel}(i) = \frac{f_i}{\sum_{i=1}^{m} f_i}$.
- Then, an array is built containing cumulative probabilities of the individuals. So, m random numbers are generated in the range 0 to 1 and for each random number an array element which can have higher value is searched for.
- Therefore, individuals are selected according to their probabilities of selection.

Summarizing:

Parent Selection could be random or fitness proportional.

4.6 Parent Recombination

Recombination or Crossover operator mergues information from two parent (at least) into one or two offspring genotypes. It is a stochastic operator: the choices of what parts of each parent are combined, and how this is done, depend on random drawings. The idea behind the recombination is to create an offspring that combines the information of both parents. It was implemented: Edge and One Point Crossover (1PX).

Edge

This crossover method strives to introduce the fewest paths possible. The idea here is to use as many existing edges, or node-connections, as possible to generate children. Edge Recombination typically out performs Partially Mapped Crossover (PMX) and Ordered crossover, but it takes longer to compute. Only one offspring per recombination is created. It is calculated as following:

- Construct a adjacency matrix for each parent and merge them.
- Pick an initial element at random and put it in the child.
- Set the variable current_element = entry.
- Set all references in adjacency matrix to current_element to zero.
- Examine the adjacent elements for current_element:
  - If there is a common edge, pick that to be the next element.
  - Otherwise pick the entry in the list of adjacent elements with the shortest list of adjacents.
  - Ties are split at random.
- In the case of reaching an empty list, a new element is chosen at random.

One Point Crossover

This is another crossover operator very easy but much more fast than Edge. This method do:

- The first step is choose one crossover point at random, and copy the segment until that point from each parent to the child.
- Starting from the beginning look for elements in the second part of the parent that have not been copied in the previous step.
- Select those elements and copy into the child.

Summarizing:

Parent Recombination could be Edge or One Point Crossover.

4.7 Mutation

Mutation is a variation operator that uses only one parent and create one child by applying some kind of randomized change to the representation. Three methods have been implemented:

Swap

This operator chooses randomly two positions in the string and swapping their values. One example can be the following:

$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \longrightarrow 1\ 5\ 3\ 4\ 2\ 6\ 7\ 8\ 9$

Inversion

Inverse mutation works by randomly selecting two positions in the strings and reversing the order in which the values appear between those positions. It is the basic move behind the k-opt search for TSP. Following an example of this mutation:

$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \longrightarrow 1\ 5\ 4\ 3\ 2\ 6\ 7\ 8\ 9$

Slide

Slide mutation selects two random positions and then slide the first value of the selected segment in front of the rest of the values. An example is shown below:

$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \longrightarrow 1\ 3\ 4\ 5\ 2\ 6\ 7\ 8\ 9$

Summarizing:

Mutation could be Swap, Inversion or Slide.

4.8 Variation operator probabilities

It was implemented two probabilities affecting both variation operators, Recombination and Mutation. If this probabilities are denoted just for notation as P then the distributions of probability of variation operators are as shown in Equation (3) with k a binary variable of success or fail, i.e., mutate or not mutate or recombinate or not recombinate.

$$f(k, P) = \begin{cases} P & \text{if } k = 1 \\ 1 - P & \text{if } k = 0 \end{cases} \tag{3}$$

Note that this kind of implementation does not maintain the ratio of mutation or offspring equal in every trial, because it is stochastic. This is what happens in nature, there are some odds of have a child after the mating and there is a probability of suffering a mutation.

4.9 Survivor Selection

The role of survivor selection is to distinguish among individuals based on their quality. It is an idea similar to parent selection but it is used in a different stage of the genetic algorithm. The survivor selection takes place after the creation of the offspring from the selected parents. As said before the population size must be constant which means that any selection (based in quality or age based) must to be done. In contrast to parent selection, which is stochastic, survivor selection is often deterministic. Two methods have been implemented:

Best Based

This operator just takes the best in the new population (parents + offspring) in terms of fitness function. It takes as much as the population size. This can lead to very rapid convergence as the population tends to rapidly focus on the fittest member. It is commonly used in conjunction with large populations.

Elitism

This method takes by pairs all the chromosomes in the population and compare their fitness pair by pair. Then the worst of the pair is deprecated. This operator is good for use with binary recombination methods, for example the One Point Crossover. Note that the recombination method applied must give two offspring per two parents and with probability one to maintain the population size.

Summarizing:

Survivor Selection could be Best Based or Elitism.

4.10 Termination Condition

The maximum number of generations can be defined. Anyway a termination condition was introduced to not allow the algorithm continues in a stationary state. That condition is: the last 25

generations does not provide different solutions.

Initialization	Parent Selection	Recombination	Mutation	Survivor Selection
Random	Random	Edge	Swap	BestBased
	Fitness Proportional	1PX	Inversion	Elitism
			Slide	

Table 1: Summary of implemented methods in the TSP exercise.

4.11 Parameter configuration

With all the details of the algorithm described, now it is necessary to define different configurations to test the program. Two configurations will be used, Configuration A (ConfigA) and Configuration B (ConfigB). The main difference between them is the different recombination and survivor method. Table 2 presents the definition.

	ConfigA	ConfigB
Parent Selection	Fitness proportional	Fitness proportional
Recombination	1PX	Edge
Mutation	To be contrasted	To be contrasted
Survivor	Elitism	Best Based

Table 2: Description of the configuration tested in the TSP exercise.

5 Results

In this section all tests carried out with the algorithm will be exposed. Tests will be done in three blocks depending on the size of the problem:

- Small problem: Only 10 cities will be used.
- Medium problem: 50 cities will be considered.
- Big problem: 100 cities will be analyzed.

The first thing to do is to choose the best size population, then the best mutation operator and after that some analysis about the probability of mutation and recombination. It is important to note that ConfigA with Elitism survivor method, does not allow the use of recombination that does not create an offspring of the same size than the population. This is relevant because it is not possible making anything different than probability of recombination not equal to one.

5.1 Small problem

Ten cities are extracted from the file 'cities.csv'. Figure 5 shows the coordinates of the first ten cities. First of all we are going to analyze the size of the population. Using the ConfigA without mutation and with recombination always, we resolved the problem with different number of size population. Table 3 shows the distance for the path found when the algorithm termination is

reached and the iteration or generation number. As we will demonstrate after the optimal distance for this problem is approximately 2.66. The optimal path is only get when the population has a size bigger enough. Because of this we have decided to take the population size 500. It is necessary enough variety in the first generation to initialize the algorithm.

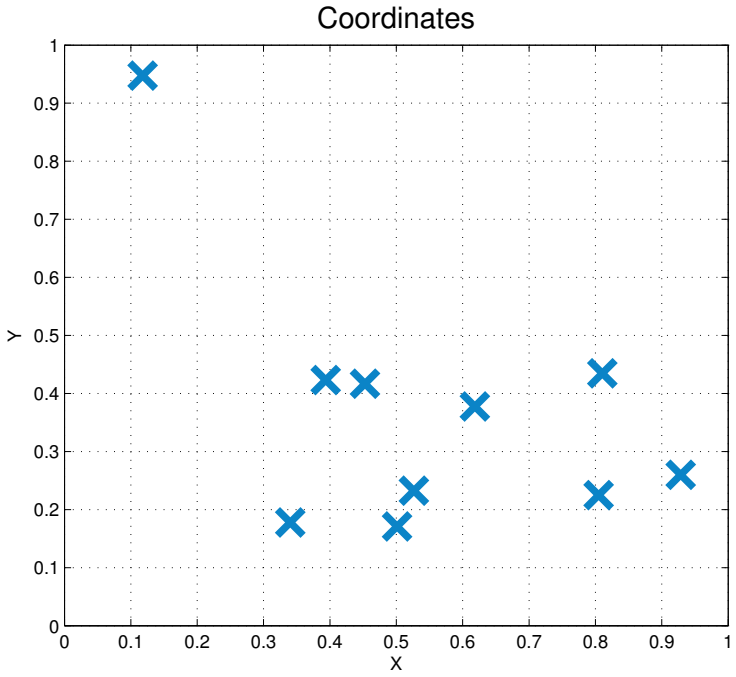

Figure 5: Coordinates of the 10 cities for the small problem.

PopulationSize	Stop Iteration	Optimum path distance
50	35	2.75
100	36	2.87
300	34	2.66
500	42	2.66

Table 3: Comparison of iteration and distance for the small problem changing the size of the population for ConfigA.

We have compared our genetic algorithm versus an exact version. We use a program based on dynamic programing[2] that for small problems return the optimal path. The function is based on the paper by Held and Karp from 1962 (See Reference [5]). The Dynamic Programming is guaranteed to provide the accurate (optimal) result to the TSP, but the time complexity of this algorithm is $O(2^n \cdot n^2)$, which limits the use of this algorithm to 15 cities or less. Using this function with our cities the optimal path distance is 2.66 approximately.

Now we are going to study the different mutation methods. It is computationally reasonable make some trials with different mutation methods. The test consists in analyzing the variation power induced just by mutation operator. Using ConfigB and a mutation probability high enough to introduce enough variety ($P_{mut} = 15\%$) thirty trials were run. Table 4 summarizes the distance and number of iterations obtained when the algorithm terminate. The best results in terms of times of getting the right path and the less volatility is for the Inversion method. We are going to use Inversion in the rest of the tests.

[2]Thanks to Elad Kivelevitch, the programmer of this code

Swap		Slide		Inversion		
Distance	**Generations**	**Distance**	**Generations**	**Distance**	**Generations**	
2.7501	60	**2.6688**	**196**	**2.6688**	**244**	
2.6688	**78**	2.7789	227	**2.6688**	**253**	
2.6688	**182**	**2.6688**	**68**	2.6869	278	
2.7794	304	2.732	214	**2.6688**	**347**	
2.8705	52	2.6869	259	2.7571	144	
2.759	287	2.6869	207	**2.6688**	**277**	
2.7927	163	2.7794	85	2.6869	302	
2.8585	108	2.6869	160	**2.6688**	**500**	
2.7789	256	**2.6688**	**57**	**2.6688**	**167**	
2.8467	51	**2.6688**	**96**	2.7201	221	
2.6869	136	2.7201	145	2.7201	291	
2.6688	**136**	2.7833	292	2.8103	133	
2.8517	66	**2.6688**	**454**	2.7975	500	
2.7201	299	2.826	500	2.7833	26	
2.7571	51	2.8123	124	**2.6688**	**143**	
2.7201	103	2.732	344	2.7571	362	
2.732	227	**2.6688**	**200**	2.6869	284	
2.6869	168	**2.6688**	**81**	2.6869	482	
2.7794	171	**2.6688**	**444**	2.732	67	
2.8123	132	**2.6688**	**265**	2.6869	500	
2.7501	254	**2.6688**	**354**	2.6869	62	
2.8716	192	2.8108	268	2.7571	500	
2.7904	240	2.6869	119	2.7571	281	
2.7201	374	2.8123	500	**2.6688**	**236**	
2.6869	203	2.6869	70	**2.6688**	**283**	
2.7571	148	2.732	51	**2.6688**	**213**	
2.849	500	2.8435	147	**2.6688**	**355**	
2.7571	267	**2.6688**	**244**	2.732	112	
2.8589	218	2.826	219	**2.6688**	**226**	
2.7975	142	2.7833	218	2.7927	500	
2.767	187.103	2.723	220.345	**2.706**	268.586	**Average**
0.065	105.662	0.062	133.032	**0.045**	136.307	**Std Deviation**
3		11		**12**		**Times Optimum**

Table 4: Distance for the last optimum and number of generations used to reach that optimum. In the Bottom, Average, Standard deviation and number of times the optimal path was obtained in the 30 trials.

After many test we can conclude about the probability of mutation and recombination that both are useful tools to calibrate the solution finding. This is because in some problems it is necesary more explotation, i.e., no so much diferent chromosomes but concentrate in some solution space. Instead other problems need more exploration, i.e., more different chromosomes dispersed in the total space of possible solutions to find the optimum. For 10 cities problems we have seen that for ConfigB a $P_{mut} = 10\%$ and $P_{Recomb} = 60\%$ and for ConfigA (remember that this configuration does not allow $P_{Recomb} \neq 100\%$) a $P_{mut} = 10\%$ or more if exploration is needed.

With all these parameters and selections chosen, we proceed to compare ConfigA and ConfigB. With such simply problem both configurations will going to be good in terms of finding the shortest path. Like an example Figure 6 plots the progress for both methods looking for the minimum. In this trial ConfigB outperforms ConfigA because it finds the minimum path earlier.

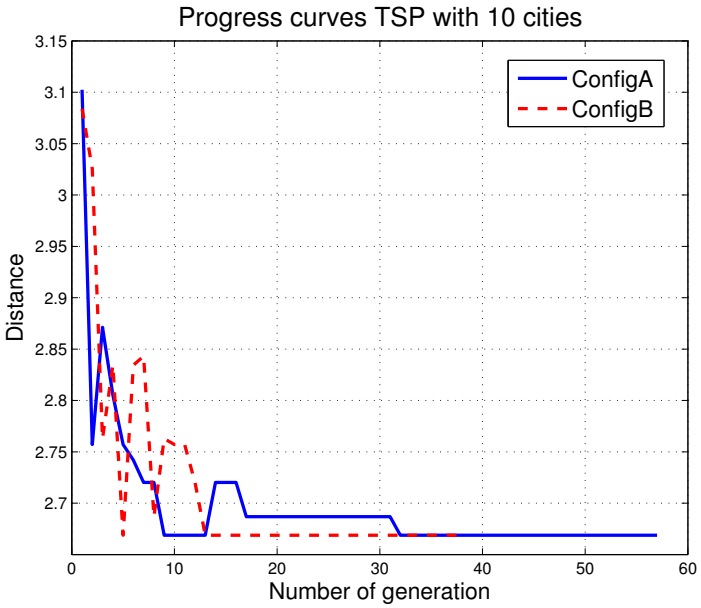

Figure 6: Progress curves for ConfigA and ConfigB.

5.2 Medium problem

Fifty cities are extracted from the file 'cities.csv'. Figure 7 shows the coordinates of the first fifty cities. We are going to make some analysis for medium problems, first we are going to compare ConfigA and ConfigB, and second we are going to established some bounds for the 50 cities TSP. We decided to run 50 times each configuration and compare the results. Figure 8 presents the otimum distances and generations needed to obtain that optimal path for each configuration. It is quite clear that ConfigB is better. One explanation on the bad performance of ConfigA is that it has to much exploration with $P_{Recomb} = 100\%$ a $P_{mut} = 10\%$ and easily it finds local minimums in line with the lower number of generations compared versus ConfigB.

There is a very useful theoretical result in Reference [6] that allow us to calculate some bounds for our problem when cities with coordinates in the $[0, 1] \times [0, 1]$ are under consideration. In that paper it is established β for the theorem:

Theorem 5.1 *(Beardwood, Halton & Hammersley,1959) Let $X_1, X_2, \ldots, X_n, \ldots$ be i.i.d. uniformly distributed random variables in $[0, 1] \times [0, 1]$. There exists a universal constant β such that*

$$\lim_{n\to\infty} L(X_1, \ldots, X_n) = \beta \cdot \sqrt{n}$$

with probability 1.

The authors of the paper concluded with this:

Theorem 5.2 *Steinerberger We have*

$$\frac{5}{8} + \frac{19}{5184} \leq \beta \leq \beta_{BHH} - \epsilon_0$$

for some explicit

$$\epsilon_0 > \frac{9}{16} 10^{-6}$$

and $\beta_{BHH} = 0.92037\ldots$

We have contrasted ConfigB from some number of cities with those bounds (we have suppose $\beta \leq 0.9$) and results are shown in Figure 9. We noted that in general the genetic algorithm are inside those bounds, indicating the good implementation and quality of the genetic programing with this kind of problems.

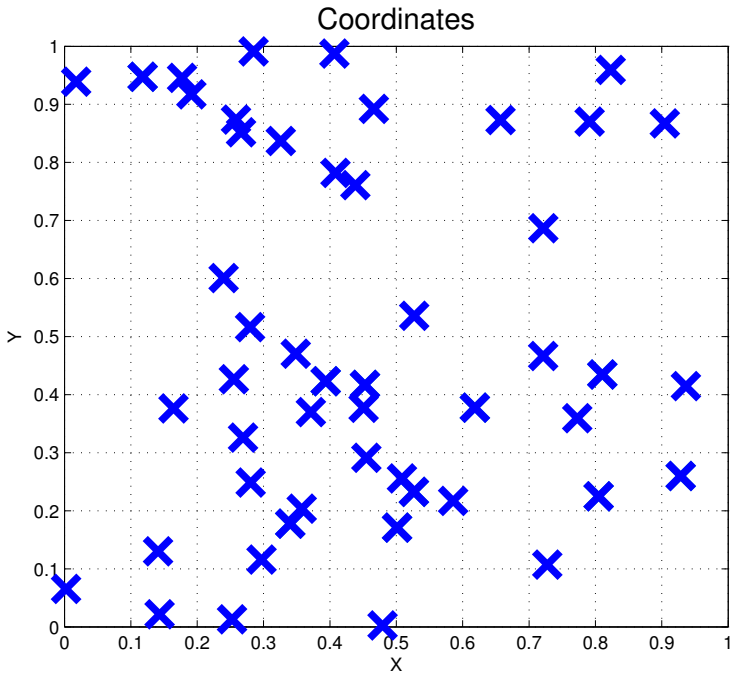

Figure 7: Coordinates of the 50 cities for the medium problem.

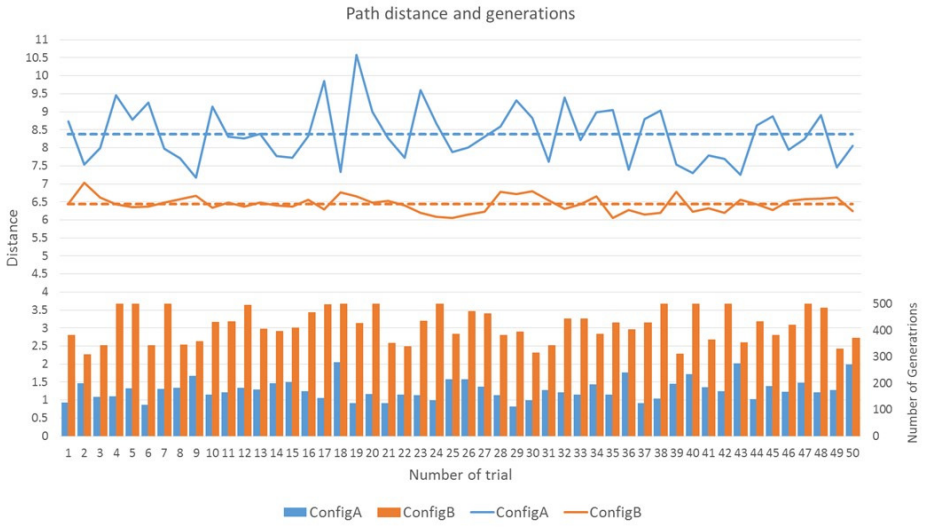

Figure 8: Distance for the optimum path found by ConfigA (Blue) and ConfigB (Red) and the number of generations used to reach the optimum. Dotted lines are the average for each configuration.

5.3 Big problem

one hundred cities are extracted from the file 'cities.csv'. Figure 10 shows the coordinates of the first one hundred cities. For 100 cities problem with a common computer it will take a long time make more than some trials on the algorithm so we are going to extract some heuristics conclusions. But we can again calculate some bounds to ensure the result is inside that boundaries. For this big problem we prefer the use of ConfigB because it is quicker than ConfigA and we will try to exploit the best. We start with $P_{Mut} = 10\%$ *and we obtained distance of 12.6174. It looks like a problem of not enough exploration. Then we repeat with* $P_{Mut} = 50\%$ *and the distance was 12.7611, not much different from the one with less probability of mutation. And both are far from the 9 derived from the upper bound. Now we repeat but this time with a maximum number of generations of 1000 and* $P_{Mut} = 40\%$*. With this new parameters the distance obtained was 11.3982. If now the size of the population is input as 1000, the distance is 9.2984.*

For more than 50 cities it was run the program with ConfigB with the last parameter described and the results are recollected in Figure 9. The results for problems with more than 50 cities are very difficult to find a path inside the bounds. It would require other operators maybe a better survivor selection.

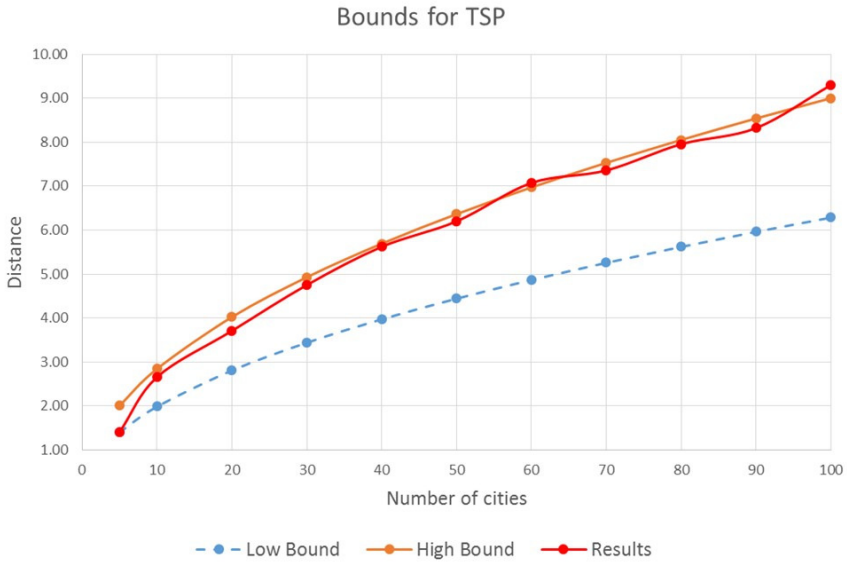

Figure 9: Upper bound and lower bound for TSP with different number of cities and our results in red.

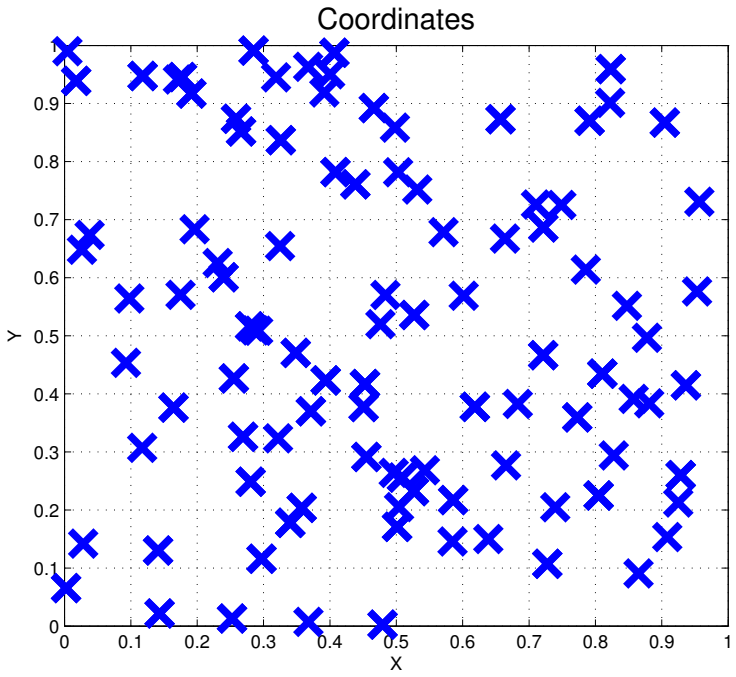

Figure 10: Coordinates of the 100 cities for the small problem.

6 Conclusions

After this study there are some conclusion to remark. About the Problem in general terms:

- Problems NP-hard are a quite difficult to resolve. The TSP has demonstrate how difficult is to achieve an optimal solution when the number of cities are just one hundrend.
- Genetic algorithm are a very good tool to resolve this kind of problem. They are quicker that exact version resolution methods and they are very flexible in order to get the optimum.
- It is very helpful have some contrast value from theory when it is possible.

About the program coded:

- Population size of 500 is good when we have around 50 cities or less. But more are required when the problem size grows.
- Inversion method is the best mutation operator of three implemented.
- ConfiB is much slower than ConfigA but it is also better to find the solution.
- For problems with many cities some new fatures must be introduced in the program to improve the performance.
- P_{Mut} and P_{Recomb} are very useful to control how much variety is needed in a problem (exploration vs exploitation).

References

[1] Wikipedia. *Travelling salesman problem (http://en.wikipedia.org/wiki/Travelling_salesman_problem).*

[2] John J. Jarvis Mokhtar S. Bazaraa. *Programacion Lineal y flujo en redes.* Noriega Limusa, 1991.

[3] H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein. *Introduction to Algorithms.* The Mit Press, 1990.

[4] J.E. Smith A.E. Eiben. *Introduction to Evolutionary Computing.* Springer, 2007.

[5] Richard M. Karp Michael Held. A dinamic programming approach to sequencing problems. 1962.

[6] Stefan Steinerberger. New bounds for the travelling salesman constant. 2010.