# Genetic Algorithms
Homework 3

Ciobotaru Mihai, 2B3 Lavinia Cibotariu, 2E2

January 14, 2022

**Abstract**

The purpose of this study is to prove the efficiency of Genetic Algorithms and Simulated Annealing algorithm in finding good enough solutions for the Symmetric Travelling Salesman Problem. Implementation and parameters of these algorithms influence their results very much and, based on the results I have obtained, I have found genetic algorithms give better results than simulated annealing but the time it takes to obtain them is exponentially worse. In order to conduct this study, I have tested the algorithms for 8 instances with a sample size of 50 for genetic algorithms and 100 for simulated annealing.

# 1 Introduction

The Travelling Salesman Problem is a classical combinatorial optimisation problem with instances where the points are distributed in a non-uniform manner. It requires searching a set of given nodes, which can represent cities or places, to find a route that passes through each node exactly once and has minimal route/tour length. It also belongs to the class of combinatorial optimization problems known as NP-complete because it has no "quick" solution and the complexity of calculating the best route will increase when you add more destinations to the problem.

This paper's purpose is to present the behaviour and efficiency of a basic genetic algorithm(GA) and of heuristic search algorithm, Simulated Annealing(SA), in finding good enough solutions for the Symmetric Travelling Salesman Problem.

The study will be archived by testing the algorithms on some well-known instances of the combinatorial problem. After we obtain the results we will compare between them to analyse the behaviour of the algorithms implemented and draw conclusions.

# 2 Genetic Algorithm

## 2.1 Implementation generalities

**Genetic algorithms** are a family of algorithms inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures in order to reach the best possible solution it can find.

An implementation of a genetic algorithm begins with a population of chromosomes and after that, for multiple iterations called generations, there are used reproductive methods (genetic operators) in such a way that the chromosomes which represent a better solution for the target problem are given more chances to 'reproduce' than those chromosomes which represent poorer solutions. The 'goodness' of a solution is typically defined with respect to the current population.

It is a good idea to use a genetic algorithm when the search space is so large that using an exhaustive search would be impossible. More than that, genetic algorithms do not provide an exact global maxima/minima but they offer a good enough solution in a good amount of time. Other heuristic search algorithms such as Hill Climbing may be better to use if the function to be tested is uni-modal.

In nature, the individual that has better survival traits will survive for a longer.This in turn provides it a better chance to produce offspring with its genetic material. In time, the population will consist with lot of genes from the superior individuals and less from the inferior individuals. The existence of competition among individuals of a population determine them to get fitter.

The concept described above is also present in genetic algorithms.As a consequence, mostly only the fittest solutions get passesd to the next generation. By random chance, a small number of unfit solutions are also picked in order to prevent getting stuck in a local accumulation point.A very small number of unfit solutions are closer to the global maxima/minima that a fit solution. There are many selection functions to choose from, but the fitness function(used to evaluate a solution) is dependent on the problem.

### 2.2 Genetic operators

**Crossover** is a genetic operator in which the parents exchange information(genetic material) at a random position in the chromosome. In some genetic algorithms the offspring are added to the new population and in other algorithms they replace their parents.

**Mutation** is an asexual recombination operator(genetic operator) in which the information of a chromosome is slightly modified in order to maintain the diversity of the population. The mutation operator ensures that the algorithms doesn't converge in a local maxima/minima and gives the random chance to find a better result.

## 3 Simulated Annealing

### 3.1 Implementation generalities

In simulated annealing, the equivalent of temperature is a measure of the randomness by which changes are made to the path, seeking to minimise it. When the temperature is high, larger random changes are made, avoiding the risk of becoming trapped in a local minimum (of which there are usually many in a typical travelling salesman problem), then homing in on a near-optimal minimum as the temperature falls.

The fundamental idea is to accept moves resulting in solutions of worse quality than the current solution in order to escape from local minima. The probability of accepting such a move is decreased during the search through parameter temperature. SA algorithm starts with an initial solution and then a candidate solution is then generated from its neighbourhood.

## 4 Methods used

### 4.1 Algorithm

**Algorithm 1** Genetic Algorithm

**Require:** $generationsNum > 0 \ and \ i = 0$

```
1: population ← randomPopulation()
2: while i < generationsNum do
3:     evaluatePopulation(population)
4:     selectPopulation(population)
5:     mutatePopulation(population)
6:     crossoverPopulation(population)
7: end while
8: return population.bestSol()
```

## 4.2 Mutation

The mutation function picks at random with an equal probability for each between the inversion operator and the swap operator.

### 4.2.1 Inversion operator

The inversion operator chooses two random indexes of the chromosome and invert the order of the gens between those two indexes. An example of the the inversion operator for indexes 2 and 7 is: initial chromosome $c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}$ and after inversion the chromosome mutates to: $c_1c_7c_6c_5c_4c_3c_2c_8c_9c_{10}$

### 4.2.2 Swap operator

The swap operator chooses two random indexes of the chromosome and swaps the gens marked by those two indexes. An example of the the swap operator for the same indexes as above is: initial chromosome $c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}$ and after inversion the chromosome mutates to: $c_1c_7c_3c_4c_5c_6c_2c_8c_9c_{10}$

## 4.3 Crossover

The crossover operator picks randomly two indexes. After that, for each information between the indexes, it swaps gens. In order for a gene from a chromosome to be swapped with a gene from another chromosome, we need to find the index of the gene from first chromosome in the second chromosome. Once found, we swap the index we want to swap with that index found. We do this for every index we want to swap. In order to accomplish this we need to make copies of the chromosomes and cross them over once at a time. An example of the crossover operator for the chromosomes $c_1c_2c_3c_4c_5c_6c_7$ and $c_7c_6c_5c_4c_3c_2c_1$ and indexes 2 5 is :The first chromosome becomes $c_1c_6c_5c_4c_3c_2c_7$ and the second one - $c_7c_2c_3c_4c_5c_6c_1$

## 4.4 Fitness function

The fitness function sorts the population from the best to the worst and then for the solution at index i we give it the fitness (i+1) divided by the size of the population + 1. After that we translate each fitness in the [-1,1] interval and use the function shown bellow to get values between [0.5,1] for fitness.

$$\frac{1}{\left(1+e^{-2\cdot\left(x^3+1\right)}\right)}$$

## 4.5 Selection

The selection function used for the genetic algorithm is a 2-way tournament selection with a tour size of 200. Tournament selection is a method of selecting solutions in which there are picked n random solutions (n being the tournament size) and there are played k-way tournaments ( 2 in our case ) until a single solution wins and this is being repeated for how many solutions we need. In the first solution is better than the second then the first wins. However if the first solution is worse there is a 5% probability that is still wins.

## 4.6 Simulated annealing

The simulated annealing algorithm starts with a temperature of 2500 and for each iteration it generates solutions until one is accepted or until it generated 200 solutions. A solution is accepted when is better than the current one or if it worse it has a probability of $e^{temperature-diff}$ where diff is the absolute difference between the evaluation of the candidate and the evaluation of the current solution. The temperature decreases by 2 for each iteration passed until it hits a minimum of 75.

| | Genetic Algorithm | | | | Simulated Annealing | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Worst** | **Average** | **Best** | **Avg Time (s)** | **Worst** | **Average** | **Best** | **Avg Time (s)** | Optimal solution |
| **berlin52** | 8105.9055 | 7718,8328 | 7544.3705 | 42.8924 | 8975.5631 | 8126.5515 | 7544.9809 | 0.0506 | 7542 |
| **rat99** | 1400.7575 | 1325.6648 | 1252.5691 | 78.1130 | 1488.0764 | 1409.2399 | 1319.3125 | 0.0777 | 1211 |
| **eil101** | 703.6401 | 683,5397 | 661.3014 | 49.4192 | 750.2805 | 724.1895 | 687.3264 | 0.0654 | 629 |
| **ch150** | 7667.3540 | 7162.8522 | 6717.6324 | 51.3923 | 8424.3340 | 7743.9133 | 7222.4940 | 0.1876 | 6528 |
| **pr226x** | 93720.614 | 88342,998 | 79861.449 | 49.2351 | 111882.791 | 100992.249 | 89412.690 | 0.3849 | 80369 |
| **a280** | 3372.6252 | 3266.9390 | 3093.41816 | 53.3145 | 5860.8684 | 5281.5979 | 4580.5624 | 0.1289 | 2579 |
| **tsp225** | 4593.4086 | 4477,0934 | 4281.8765 | 51.6102 | 6056.2969 | 5604.9882 | 5075.4112 | 0.1884 | 3916 |
| **gil262** | 2900.4316 | 2819,3013 | 2731.6788 | 58.7486 | 5068.0980 | 4513.3788 | 4131.5316 | 0.3480 | 2378 |

# 5 Experiment

For this experiment we will be comparing the time and value result‘s of a Genetic Algorithm and Simulated Annealing in relation with the Symmetric Travelling Salesman Problem.Each algorithm will target to find the optimal tour of the instances. We are interested in finding the best,worst and average time and values and inserting them in tables to have a good visual representation of the results. For the genetic algorithm we will use a sample with size 50 and for the simulated annealing - 100. Genetic Algorithm uses a population of 200 solutions that goes through 2000 generations and as for Simulated Annealing, the algorithm goes through 2000 iterations.

# 6 Comparison

The results shown above depict the efficiency of each algorithm in finding the global minima of the instances provided. Based on the results we can conclude how efficient genetic algorithms in finding an optimal solution but, as drawback, in order to obtain a result the algorithm takes too much time. We can observe that Simulated Annealing gives slightly worse results but it can obtain them even 100 times faster that genetic algorithm. As shown above the results of the genetic algorithms are more concentrated, meaning that the interval in which the results are is smaller compared to the ones from the other heuristic search algorithm. We can observe both algorithms tend to get worse results then the numbers of cities get larger but simulated annealing gets exponentially worse results.

# 7 Conclusion

This study proved the efficiency of genetic algorithms and simulated annealing in relation with finding the best tour for TSP. As a final remark, with the help of this research, we can conclude that, if we want to get results fast, we should use Simulated Annealing, otherwise we should use Genetic Algorithm in order to get better results.

# References


[1] Mathew, Tom V. "Genetic algorithm." Report submitted at IIT Bombay (2012). | *http://datajobstest.com/data-science-repo/Genetic-Algorithm-Guide-[Tom-Mathew].pdf*

[2] Wang SC. (2003) Genetic Algorithm. In: Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science, vol 743. Springer, Boston, MA | *https://link.springer.com/chapter/10.1007/978-1-4615-0377-4_6f*

[3] Noraini Mohd Razali, John Geraghty: Genetic Algorithm Performance with Different Selection Strategies in Solving TSP | *http://www.iaeng.org/publication/WCE2011/WCE2011*$_p$*p1134 – 1139.pdf*

[4] Wikipedia - Genetic algorithm |
*https://en.wikipedia.org/wiki/Genetic_algorithm*

[5] Wikipedia - Simulated annealing |
*https://en.wikipedia.org/wiki/Simulated_annealing*

[6] Geeks for geeks: Simulated Annealing |
*https://www.geeksforgeeks.org/simulated-annealing/*

[7] Eugen Croitoru: Teaching: Genetic Algorithms |
*https://profs.info.uaic.ro/∼eugennc/teaching/ga/*