CS551 – Introduction to Artificial Intelligence Assignment-2

Solving Travelling Salesman Problem with Genetic Algorithm

Adem ALDEMIR
MS. in Computer Science at Ozyegin University
adem.aldemir@ozu.edu.tr

Abstract. Travelling Salesman Problem (TSP) is an optimization problem that aims navigating given a list of city in the shortest possible route and visits each city exactly once. When number of cities increases, solution of TSP with mathematical methods becomes almost impossible. Therefore it is better to use heuristic methods to solve the problem.

In this study, for the solution of TSP, Genetic Algorithms which are based on heuristic techniques were used and changed mutation rate, number of generations parameters. After that performances were compared. Genetic Algorithm which is an evolutionary algorithm is inspired by biological changes and it uses operators such as natural selection, reproduction, tournament, crossover and mutation. Population evolution is occurred by applying these operators iteratively and as the other heuristic techniques, it gives exact or approximate solutions.

As a result, the low value of the mutation rate significantly reduces the distance value in the TSP problem, while the population size and generation size also the tournament will not be greatly effective here.Multiple values are taken into consideration and the results are given in the section Experiments and Results.

Keywords. *Travelling Salesman Problem, Genetic Algorithm, Mutation, Crossover, Tournament, Python*

1. Introduction

Traveling Salesman Problem (TSP) is an important NP-hard problem in combinatorial optimization. According to TSP, there is a salesman and he wants to sell his goods in different cities. Therefore, the salesman leaves a city, visits each of the cities exactly once and returns back to the starting point. The aim of the problem is to provide the possible shortest route to the salesman, among n cities whose distances between each city are known. Determining the route, initially there are different n cities which the salesman may choose. In the first city, the salesman may choose one of the different n-1 cities. In the second city, the salesman may choose one of the different n-2 cities. As a result, the salesman have right to choose though n! different tour. For this reason, number of cities increases, solution of TSP with mathematical methods becomes almost impossible. So, it is better to use heuristic methods to solve such problems.

Genetic algorithm (GA) is an artificial intelligence search method that uses the process of evolution and natural selection theory and is under the umbrella of evolutionary computing algorithm and is significant optimization technique, offer heuristic methods for NP-hard problems such as TSP. It is an efficient tool for solving optimization problems. It uses random search techniques. It works using population of individuals. Thus, it starts searching the set of the point, not one point and reaches the global optimum. Integration among (GA) parameters is vital for successful (GA) search. Such parameters include mutation rate, population size, tournament size and number of generation that are important issues in (GA). However, each operator of GA has a special and different influence. The impact of these factors is influenced by their probabilities. This paper reviews various methods for choosing mutation size, number of generations and population ratios in GAs.

2. Travel Salesman Problem (TSP)

We can explain the Travel Salesman Problem as “Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.” Before you set off on your journey you'll probably want to plan a route so you can minimize your travel time. Let's say we've found a route that we believe is optimal. However, how can we test if it's really the optimal route?

Practically, we can’t. To understand why it's so difficult to prove the optimal route let's consider 3 locations. To find a single route, we first have to choose a starting location from the three possible locations on the map. Next, we'd have a choice of 2 cities for the second location, then finally there is just 1 city left to pick to complete our route. This would mean there are 3 x 2 x 1 different routes to pick in total. For this example, there are only 6 different routes to pick from. A factor of the important reasons that make this problem difficult is factorial because factorials are that they grow in size quickly. If we want to find the shortest route for our map of 20 locations we would have to evaluate 2432902008176640000 different routes. Even with modern computing power, this is terribly impractical, and for even bigger problems, it's close to impossible. The genetic algorithm is included in the game here. Genetic algorithm (GA) is one of the best heuristic algorithms that have been used widely to solve the TSP instances.

There are *two important rules* to keep in mind:

- Each city needs to be visited exactly one time
- We must return to the starting city, so our total distance needs to be calculated accordingly.

3. Solution of TSP with Genetic Algorithm (GA)

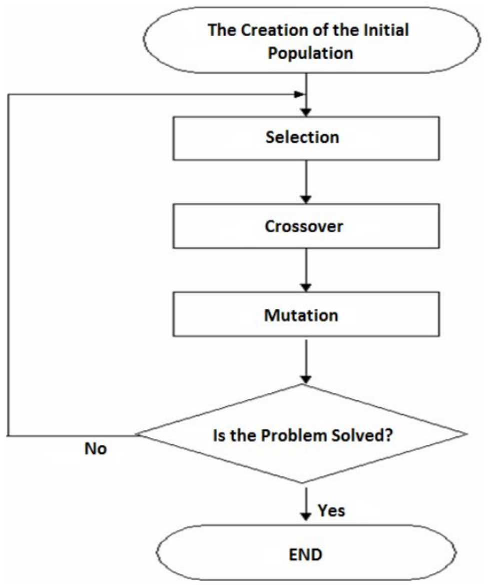
A genetic algorithm is a search heuristic that is inspired by Charles Darwin’s theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. Randomness plays a substantial role in the structure of genetic algorithms, and it is the main reason genetic algorithms keep searching the search space. Genetic algorithms create an initial population of randomly generated candidate solutions, these candidate solutions are evaluated, and their fitness value is calculated. The fitness value of a solution is the numeric value that determines how good a solution is, higher the fitness value better the solution. In the Genetic Algorithm, a population of potential solutions termed as chromosomes and individuals is evolved over successive generations using a set of genetic operators called tournament, crossover, and mutation. First of all, based on some criteria, every chromosome is assigned a fitness value and then the tournament operator is applied to choose relatively fit chromosomes to be part of the reproduction process. In the reproduction process, new individuals are created through the application of Crossover and Mutate operators.

A few definitions, rephrased in the context of the TSP:

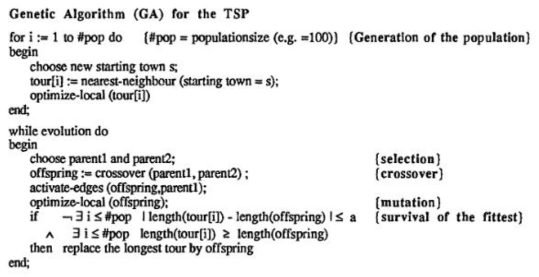
Gene: a city (represented as (x, y) coordinates)
Individual (aka “chromosome”): a single route satisfying the conditions above
Population: a collection of possible routes (i.e., collection of individuals)
Parents: two routes that are combined to create a new route
Mating pool: a collection of parents that are used to create our next population (thus creating the next generation of routes)
Fitness: a function that tells us how good each route is (in our case, how short the distance is)
Mutation: a way to introduce variation in our population by randomly swapping two cities in a route
Elitism: a way to carry the best individuals into the next generation

3.1 The Basic Steps of Genetic Algorithm

The algorithm starts generating a population randomly and continues using operators such as natural selection, crossover and mutation iteratively. The population in the each iteration is called as a generation. In each generation, the value of the fitness (usually objective function) of every individual in the population is evaluated. The more appropriate individuals are stochastically selected from the current population and modified with operators of crossover and mutation. Thus, new chromosomes are produced from the older generation. Then the new generation is used in the next iteration of the algorithm. The algorithm terminates when either a certain number of iterations or a satisfactory fitness level has been reach.



The Parallel Problem Solving from Nature book has been the biggest source of help in improving the Genetic Algorithm. It is inspired by the following algorithm. You can examine the algorithm.



```
Genetic Algorithm (GA) for the TSP

for i := 1 to #pop do     {#pop = populationsize (e.g. =100)}  {Generation of the population}
begin
    choose new starting town s;
    tour[i] := nearest-neighbour (starting town = s);
    optimize-local (tour[i])
end;

while evolution do
begin
    choose parent1 and parent2;                                    {selection}
    offspring := crossover (parent1, parent2) ;                    {crossover}
    activate-edges (offspring,parent1);
    optimize-local (offspring);                                    {mutation}
    if    ¬∃ i ≤ #pop  | length(tour[i]) - length(offspring) | ≤ a  {survival of the fittest}
       ∧   ∃ i ≤ #pop  length(tour[i]) ≥ length(offspring)
    then  replace the longest tour by offspring
end;
```

3.2 Basic Concepts for Genetic Algorithm

Initialization / Population - Create an initial population. This population is usually randomly generated and can be any desired size, from only a few individuals to thousands.

Fitness Calculation - Each member of the population is then evaluated and we calculate a 'fitness' for that individual. The fitness value is calculated by how well it fits with our desired requirements. These requirements could be simple, 'faster algorithms are better', or more complex, 'stronger materials are better but they shouldn't be too heavy'. The GAs are used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as F(x) = 1/f(x) , where f(x) is the objective function. Since TSP is a minimization problem; we consider this fitness function, where f(x) calculates the cost (or value) of the tour represented by a chromosome.

Selection - We want to be constantly improving our populations' overall fitness. Selection helps us to do this by discarding the bad designs and only keeping the best individuals in the population. There are a few different selection methods and in this paper, we will use the *tournament selection* method. In more detail like this, chromosomes are copied into the next-generation mating pool with a probability associated with their fitness value. By assigning to the next generation a higher portion of the highly fit chromosomes, reproduction mimics the Darwinian survival-of-the-fittest in the natural world.
In a natural population, fitness is determined by a creature's ability to survive predators, pestilence, and other obstacles to adulthood and subsequent reproduction. In this phase, no new chromosome is produced. In tournament selection, a group of i individuals are randomly chosen from the population. This group takes part in a tournament and an individual with highest fitness value wins. In many cases i is chosen to be two, and this method is called binary tournament selection. To further enhance this selection, i is selected to be five. And the best of these five is selected. In order to speed up convergence and avoid creating large number of generation, **elitism** technique can be applied as a selection technique (Asllani and Lari, 2007). In this technique best l members from the current generation are selected to form the mating pole for next generation. It is applied to ensure gradual improvement of the solution.

In **Parallel Problem Solving from Nature**[**], he describes it under the title of *Selection and Survival of the Fittest* as follows.

As in natural surroundings it holds on average: "the better the parents the better the offspring" and "the offspring is similar to the parent". Therefore it is on the one hand desirable to choose the fittest individuals more often, but on the other hand not to often, beause otherwise the diversity of the population decreases. In our implementation we select the best individuals 4 times more often than the worst Furthermore we only accept an offspring as a new member of the population, if it differ enough from the other individuals, that means here its fitness differ from all the other individuals at least about the amount a. After accepting a new individual we remove one of the worst in order to hold the populationsize constant. In our implementation we remove the worst, because the algorithm is not sensible against this selection.

Crossover - During crossover, we create new individuals by combining aspects of our selected individuals. The hope is that by combining certain traits from two or more individuals we will create an even 'fitter' offspring which will inherit the best traits from each of its parents. If our individuals were strings of 0s and 1s and our two crucial rules didn't apply, we could simply pick a crossover point and splice the two strings together to produce an offspring. However, the TSP is unique in that we need to include all locations exactly one time. To abide by this rule, we can use a special breeding function called **ordered crossover**. In the ordered crossover, we randomly select a subset of the first parent string and then fill the remainder of the route with the genes from the second parent in the order in which they appear, without duplicating any genes in the selected subset from the first parent.

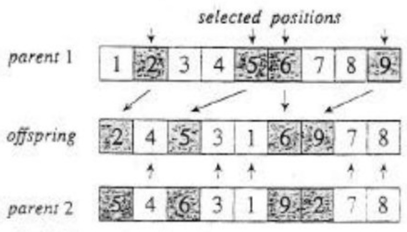
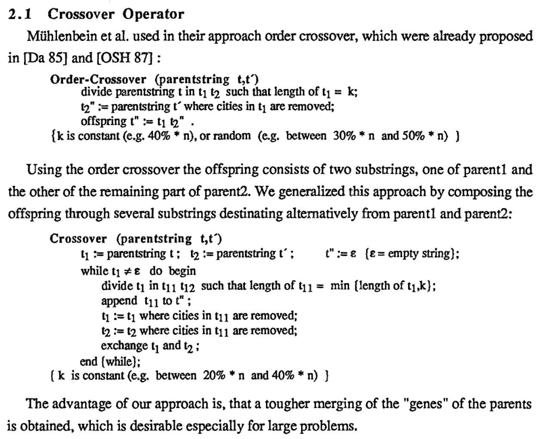


Figure 3.16. Illustration of the order-based crossover operator.

Where did I get inspired when writing the crossover function?

The picture below belongs to the **Parallel Problem Solving from Nature** book. I developed my crossover script by following the rules and warnings recommended in that book.Likewise, I wrote mutate function using that book.



2.1 Crossover Operator

Mühlenbein et al. used in their approach order crossover, which were already proposed in [Da 85] and [OSH 87] :

```
Order-Crossover (parentstring t,t´)
        divide parentstring t in $t_1$ $t_2$ such that length of $t_1 = k$;
        $t_2$" := parentstring t´ where cities in $t_1$ are removed;
        offspring t" := $t_1$ $t_2$" .
{k is constant (e.g. 40% * n), or random (e.g. between 30% * n and 50% * n) }
```

Using the order crossover the offspring consists of two substrings, one of parent1 and the other of the remaining part of parent2. We generalized this approach by composing the offspring through several substrings destinating alternatively from parent1 and parent2:

```
Crossover (parentstring t,t´)
        $t_1$ := parentstring t ;  $t_2$ := parentstring t´ ;      t" := $\varepsilon$ {$\varepsilon$ = empty string};
        while $t_1 \neq \varepsilon$ do begin
            divide $t_1$ in $t_{11}$ $t_{12}$ such that length of $t_{11}$ = min {length of $t_1$,k};
            append $t_{11}$ to t" ;
            $t_1$ := $t_1$ where cities in $t_{11}$ are removed;
            $t_2$ := $t_2$ where cities in $t_{11}$ are removed;
            exchange $t_1$ and $t_2$ ;
        end {while};
{ k is constant (e.g. between 20% * n and 40% * n) }
```

The advantage of our approach is, that a tougher merging of the "genes" of the parents is obtained, which is desirable especially for large problems.

Mutation - We need to add a little bit randomness into our populations' genetics otherwise every combination of solutions we can create would be in our initial population. Mutation typically works by making very small changes at random to an individual's genome. Especially, TSP has a special consideration when it comes to mutation. Again, if we had a chromosome of 0s and 1s, mutation method would simply mean assigning a low probability of a gene changing from 0 to 1, or vice versa. However, since we need to abide by our rules, we can't drop cities. Instead, we'll use exchange mutation. This means that, with specified low probability, two cities will swap places in our route. We'll do this for one individual in our *mutate* function.

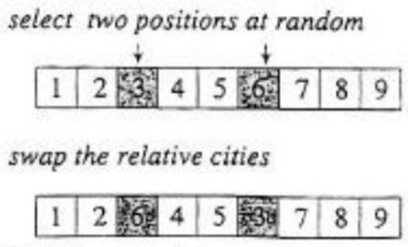


Figure 3.23. Illustration of reciprocal exchange mutation.

a. Tournament

To select a number of individuals from the population in the mating pool. Based on the previously calculated fitness value, the best individuals based on a threshold are selected. In a K-way tournament selection, we select k-individuals/cities and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation. In this way many such tournaments take place and we have our final selection of candidates who move on to the next generation.

Algorithm for Tournament :
1.Select k individuals from the population and perform a tournament amongst them
2.Select the best individual from the k individuals
3. Repeat process 1 and 2 until you have the desired amount of population

b. Evolve

Evolve function Evolves a population over one generation. Within the Evolve function, first, check the elitism parameter. If Elitism is enabled, it keeps our best individuals. Next, select the fittest individuals with *Tournament* function and then create offsprings with the *Crossover* function. Finally, mutate the parents using the *Mutate* function. That is, we wrote the evolve function to use all of our genetic algorithm functions by following the algorithm below.

Our basic genetic algorithm in evolve function works roughly as follows:

```
Initialize population of optimum's searchers
Begin
   {tournament / selection}
   choose parentl and parent2;
   {crossover}
   offsprings combination (parentl,parent2);
   {mutation}
   optimize-local (offspring);
   {survival of the fittest}
   ifsuited(offspring)
   then replace optimum's searcher with worst
fitness by offspring
Repeat
End
```

4. Experiments and Results

This section analyses the obtained results.I visualized the initial and final results with graphics using the python matplotlib library. The most fun part of this research is to see the results and to track which parameter affects the results more. All figures written in the table can be found in the FIGURES section below.

Table 1

Figures	Initial Results	Final Results
Figure 1	1849.26	997.60
Figure 2	1926.73	871.11
Figure 3	1725.65	1003.55
Figure 4	1781.73	978.76
Figure 5	1851.50	1044.58
Figure 6	1836.16	923.99
Figure 7	1850.83	1219.58
Figure 8	1822.68	1151.48
Figure 9	1673.24	1112.31
Figure 10	1904.96	990.74
Figure 11	1866.47	871.11
Figure 12	1876.35	897.53
Figure 13	1759.53	944.78
Figure 14	1685.86	938.76
Figure 15	1873.11	957.4
Figure 16	1734.58	1367.60
Figure 17	1596.27	1203.00
Figure 18	1798.77	1342.08
Figure 19	1775.63	934.09
Figure 20	1727.11	871.11
Figure 21	1684.17	947.94
Figure 22	1826.64	897.53
Figure 23	1715.80	902.9

Figure 24	1694.98	1300.43
Figure 25	1732.58	1308.60
Figure 26	1777.45	1235.88

In this experiment, I reached the initial and final outputs (table 1) using the below variables and values.

```
popSize = [50, 200, 400]
mutation_rate = [0.030, 0.1, 0.2]
num_gen = [100, 200, 300]
elitisim = [True]
```

What we want in this experiment is to bring the final output to the minimum value. This means that we have found the best optimal result. The best final result in the table is 871.11. In other words, it means the shortest route that will provide the lowest cost among 20 cities.

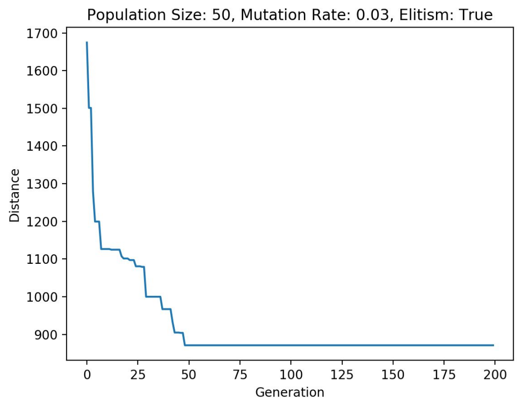


Figure 2

When we look at the chart values in Figure 2, the Population Size is 50 and the Mutation Rate is 0.03. Let's look at Figure 11, Figure 20, which are other figures that give better results.

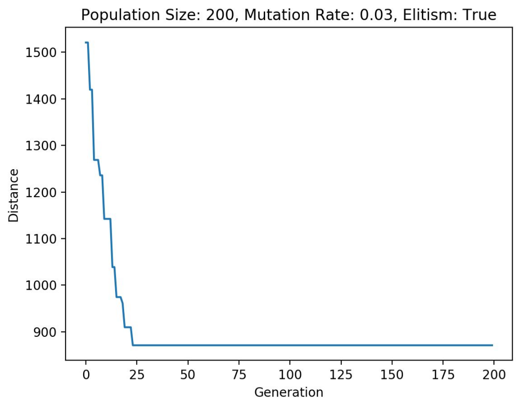


Figure 11

The parameters in Figure 11 are as follows Population Size 200, Mutation Rate 0.03. And the other best result.

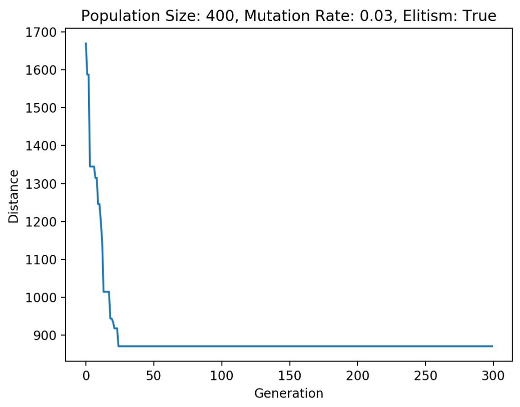


Figure 20

The parameters in Figure 20 are as follows Population Size 400, Mutation Rate 0.03. In Figure 2 it is not effective after Generation 50 and Distance is at the saturation level after this value. Well, let's look at Population Size. Population Size is 50 in Figure 2, 200 and 400 in Figure 11 and 20 respectively. These graphs continue to give us the best result. When we look at Mutation Rate, it is 0.03 in 3 charts that give the best results. This value is the smallest number we give in the mutation_rate array. At this point, we reached an uncertain result in the experiment. Mutation Rate is the biggest factor affecting distance. But to conclude this, let's look at the results in the table, where we get the highest distance values highlighted with red.

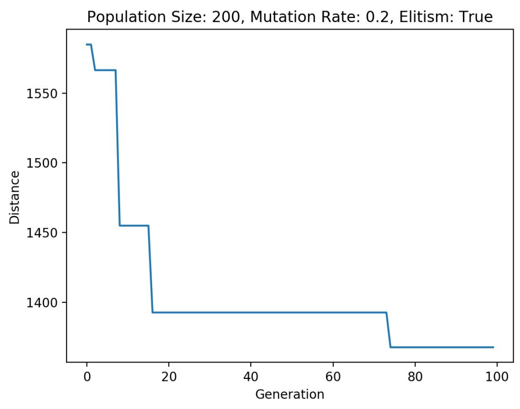


Figure 16

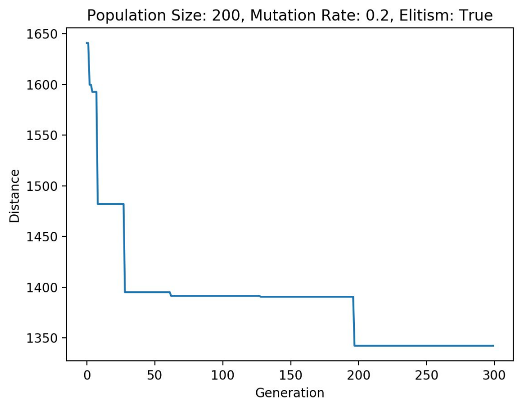


Figure 18

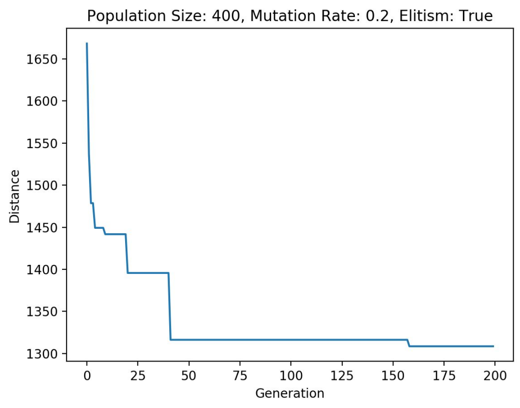


Figure 25

In Figure 16, Population Size is 200, Mutation Rate is 0.2. 100 generations were created. This chart shows the distance as 1367 after the 73rd generation. In Figure 18, Population Rate is 200, Mutation Rate 0.2, 300 generation was created. The distance after the 200th generation always showed 1348. In Figure 25, Population Size is 400, Mutation Rate is 0.2 and 200 generations are created. The distance between the 40th Generation and the 160th generation is approximately 1320. After the 160th generation, the distance value was constant at 1308.

5. Conclusion

Based on our conclusion from these experiments, algorithms prove that Mutation Rate is important for the Travel Salesman Problem (TSP). The following researchers explained how GA parameters matters and how to should use them in the Genetic Algorithm.

The crossover and mutation rates was proofed as key elements to success in the search in the GAs [*]. DeJong [2] suggested optimal range values for population size to be in the range of [50–100], mutation parameter rate to be (0.001), and high mutation rates leads the search to be random.

Grefenstette [3] designed a Meta-GA provided optimal parameters value for the simple GAs, and showed that small population size ranging from (20) to (40) is associated with high crossover rates went in harmony with low mutation rates. In general, his findings revealed that mutation rates above 0.05 were not useful for the optimal performance of (GAs). Accordingly, the researcher suggested another set of parameters with population size value of 30 individuals, a mutation rate of 0.01 and two-point crossover with 0.95 rates [3].

Schlierkamp-Voosen [4] investigated the dynamic behavior of mutation and crossover rates on GA regarding the binary functions. The GA parameter mutation and crossover rates were compared to their optimal solution performance. The mutation rates were most effective in small size populations. While the Crossover rates were depending on the size of the population, and mutation rates were the most robust search in small population size.The analysis of the interaction between mutation and crossover rates showed that the mutation rates normally increased with the size of population. Furthermore, it showed that the mutation rate = 1/n (where n was the size of problem) works efficiently. Using crossover and mutation together, achieved better results than using one of them alone. The crossover appeared to be more efficient than the mutation in larger population size, on the other hand, the mutation was more efficient in small populations [4].

References

[*] Beasley, D.; Martin, R.R.; Bull, D.R. An overview of genetic algorithms: Part 1 Fundamentals. Univ. Comput. 1993, 15, 56–69.

[1] DeJong, K. Analysis of the Behavior of a Class of Genetic Adaptive. Ph.D. Thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[2] Beasley, D.; Martin, R.R.; Bull, D.R. An overview of genetic algorithms: Part 1 Fundamentals. Univ. Comput. 1993, 15, 56–69.

[3] Grefenstette, J. Optimization of control parameters for genetic algorithms. IEEE Trans. Syst. Man Cybern. 1986, 16, 122–128. [CrossRef]

[4] Schlierkamp-Voosen, D. Optimal interaction of mutation and crossover in the breeder genetic algorithm. In International Conference on Genetic Algorithms; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993; 648p.

Parallel Problem Solving from Nature, 1st Workshop, PPSN I Dortmund, FRG, October 1–3, 1990 Proceedings, Hans-Paul SchwefelReinhard Männer

S.M. Soak, and B. H. Ahn. "New Genetic Crossover Operator for the TSP". ICAISC, 2004

"Genetic algorithms in search, optimization, and machine learning" Book by David E. Goldberg

https://hackernoon.com/genetic-algorithms-explained-a-python-implementation-sd4w374i

Using Genetic Algorithm with Combinational Crossover to Solve Travelling Salesman Problem. https://www.researchgate.net/publication/301453105_Using_Genetic_Algorithm_with_Combinational_Crossover_to_Solve_Travelling_Salesman_Problem [accessed Apr 11 2020]

FIGURES

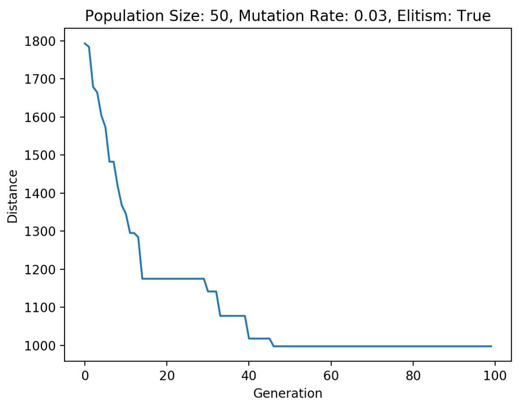


Figure 1

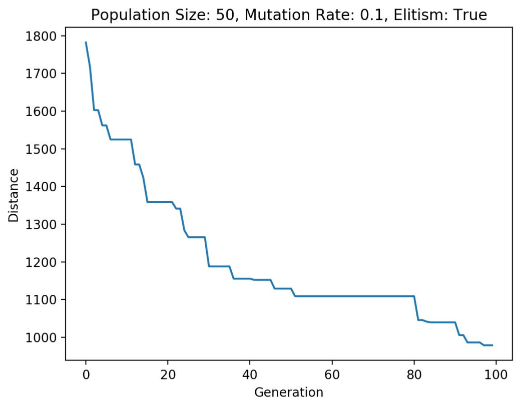


Figure 4

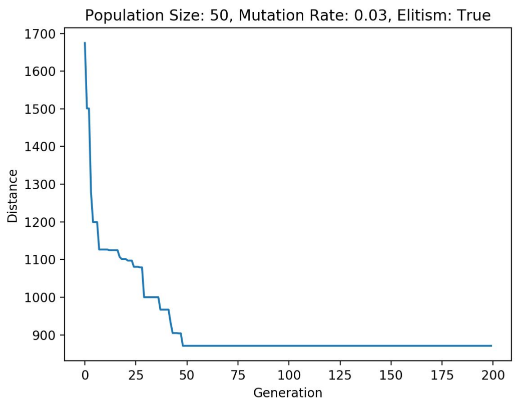


Figure 2

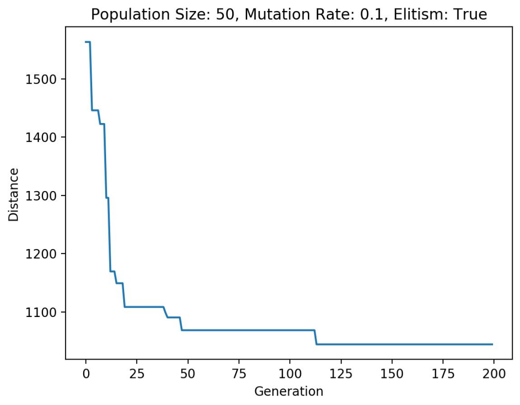


Figure 5

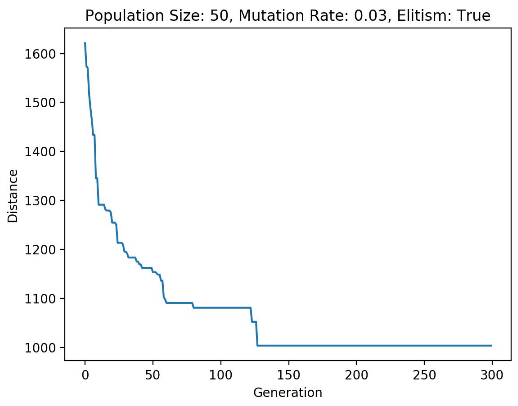


Figure 3

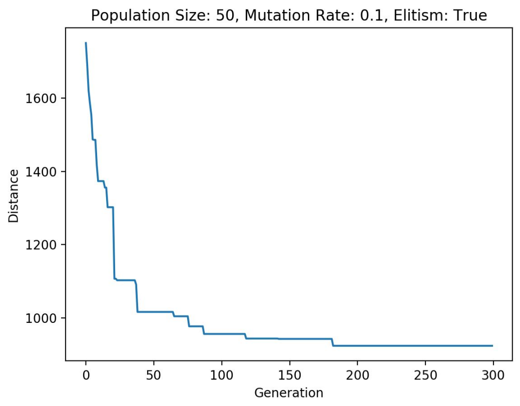


Figure 6

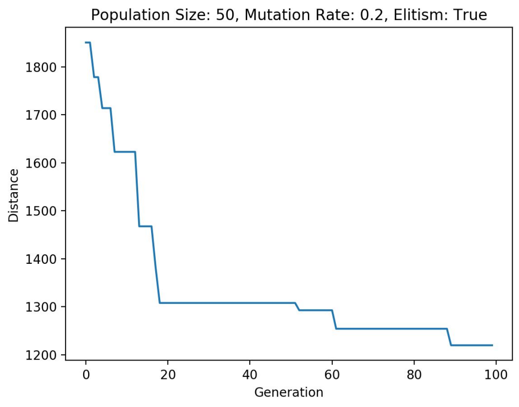


Figure 7

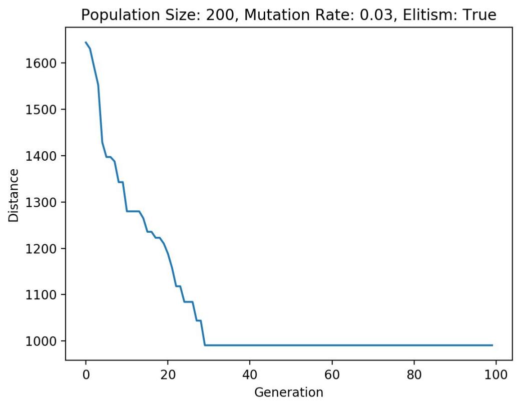


Figure 10

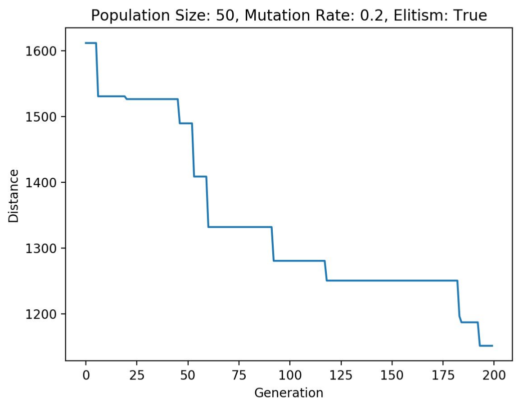


Figure 8

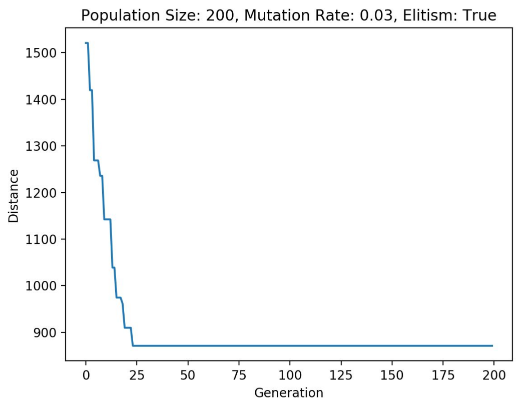


Figure 11

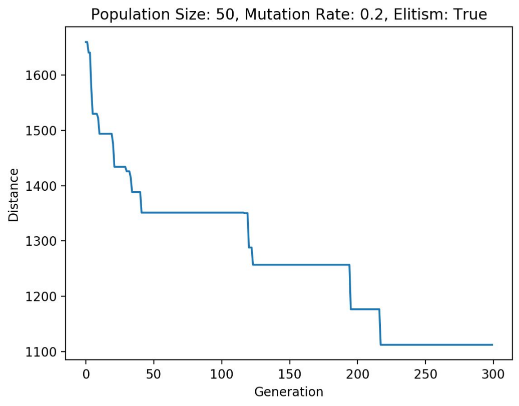


Figure 9

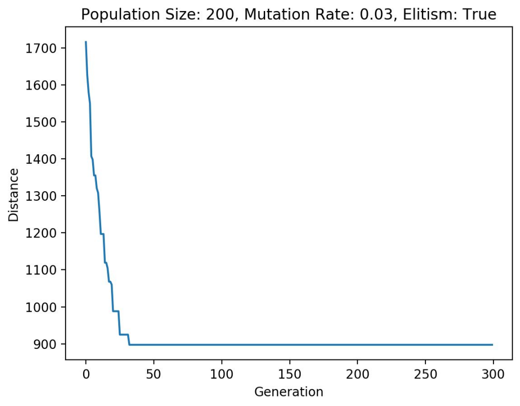


Figure 12

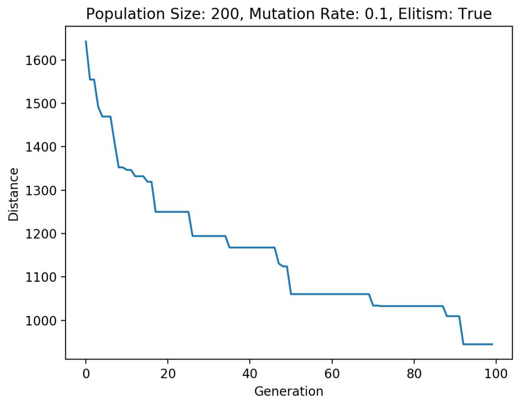


Figure13

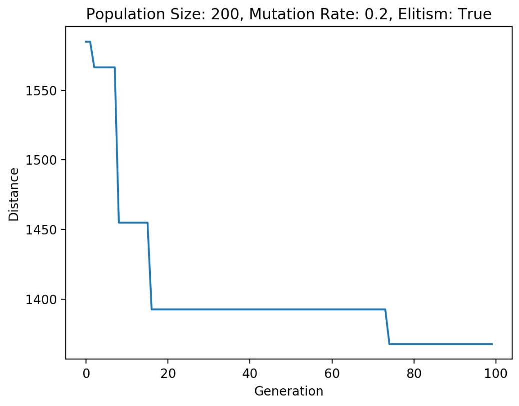


Figure 16

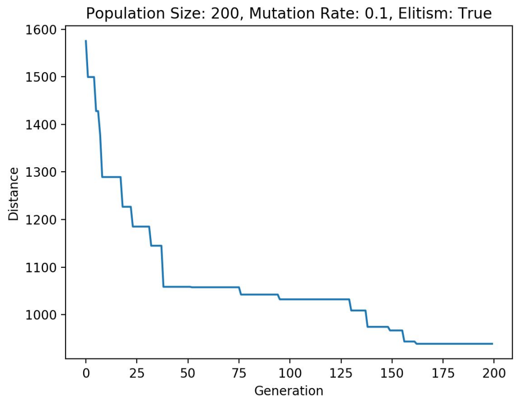


Figure 14

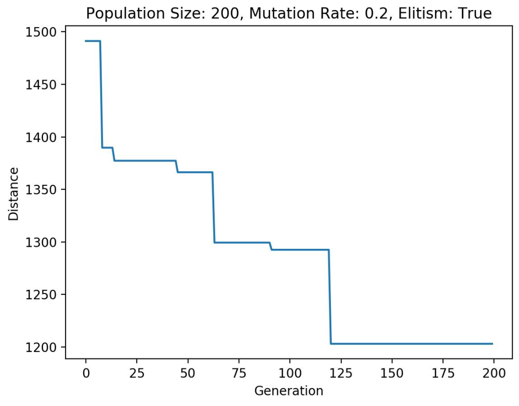


Figure 17

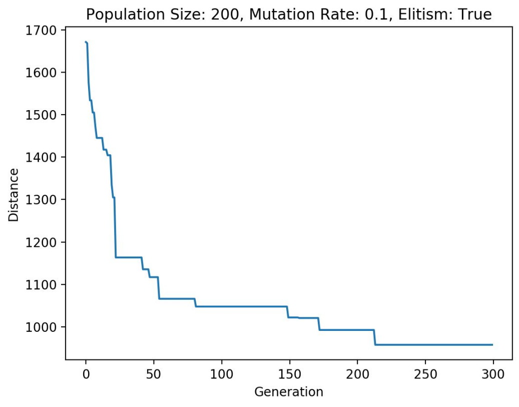


Figure 15

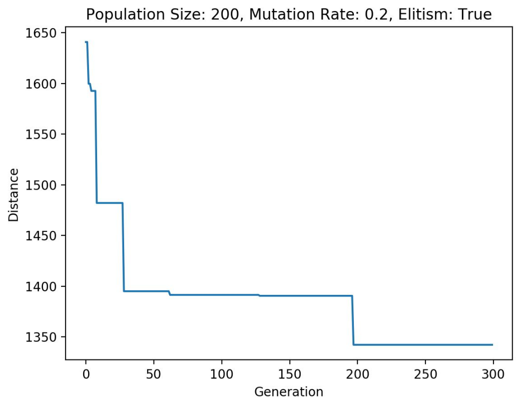


Figure 18

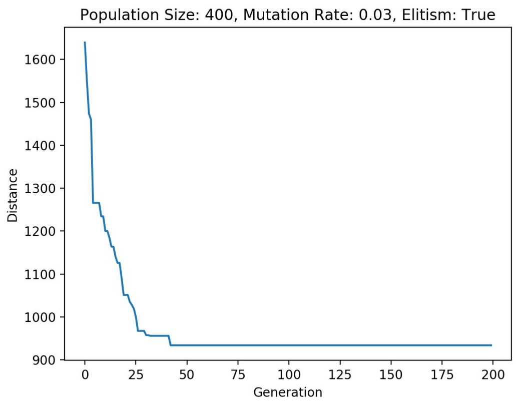


Figure 19

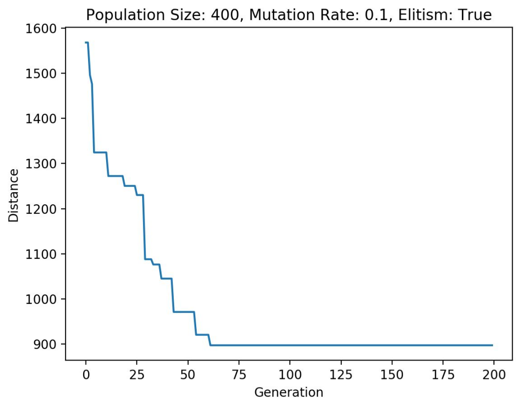


Figure 22

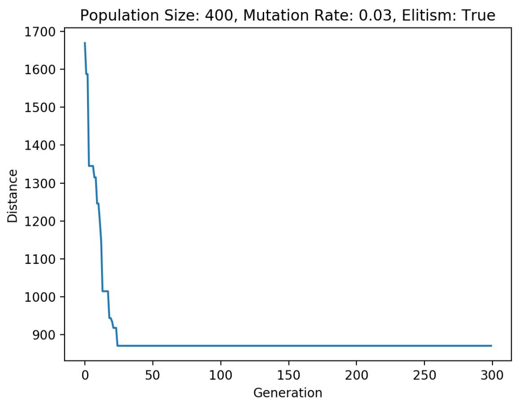


Figure 20

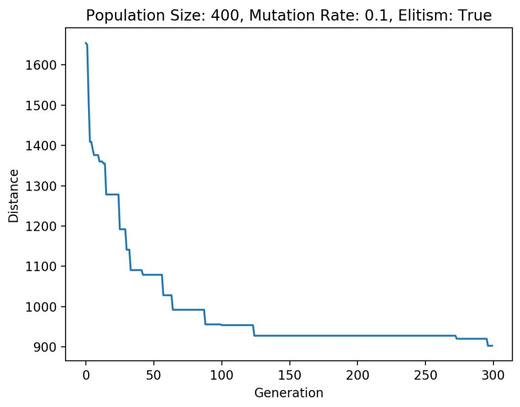


Figure 23

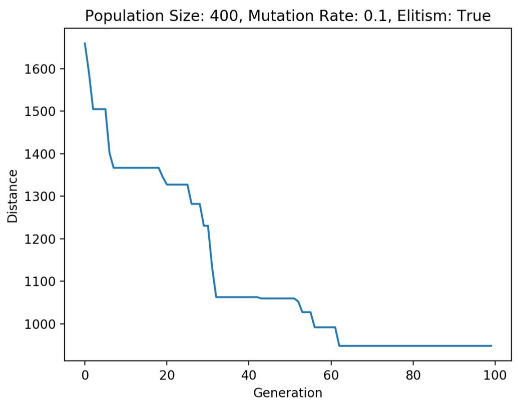


Figure 21

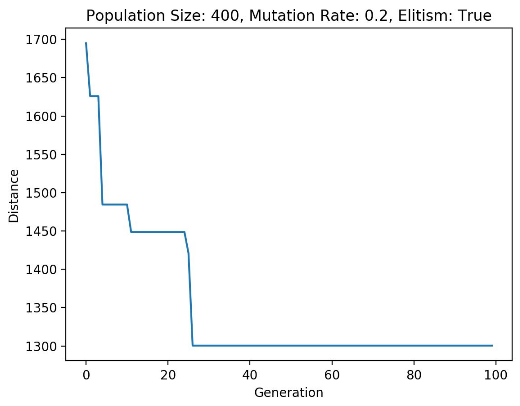


Figure 24

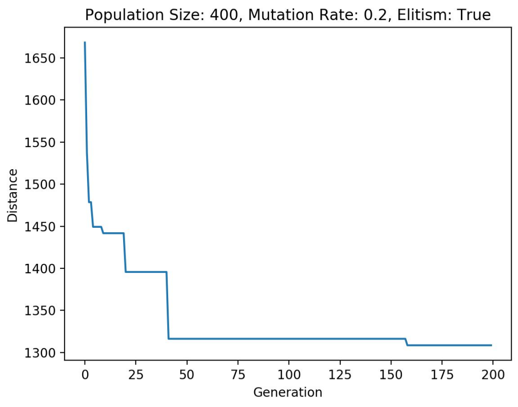


Figure 25

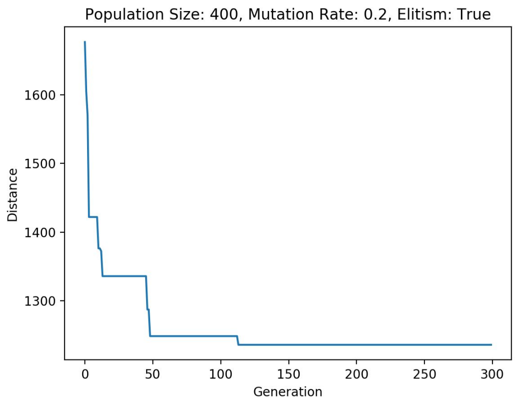


Figure 26