Travelling Salesperson - Genetic Algorithm

Sudarshan Rao Mundkur

Computer Science Department, Purdue University

Fort Wayne, IN, 46805

[munds01@pfw.edu](mailto:munds01@pfw.edu)

**Problem Statement**: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?

Given this, 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.

**Approach:**

Gene: a City (represented as distance matrix with index as city number)

Individual/ Chromosome: a single route.

Population: a collection of possible routes

Parents: two routes that are used to combine and create new route

Mating pool: a collection of parents that are used to create our next population

Fitness function: Tells how good each route is or how short the distance is.

To minimize the route distance, we consider an inverse of the distance, hence a larger fitness function value is desirable.

Mutation: To introduce variation in the population by randomly swapping two cities in a route

Elitism: To select the best individuals for the next generation

**Working of the program:**

|  |  |
| --- | --- |
| Function Name | Functionality |
| calcDistance | Calculates the distance for each possible route in the population and assigns a fitness values. |
| rankRoutes | Ranks the routes based on the fitness function calculated in calcDistance(). The fitness function is calculated by evaluating 1 / Distance. |
| selectParents  matingPool | Select the elite parents and add it into the population array for next generation. The approach used here to select parents for cross over is **fitness proportionate selection**. Mating pool functions is used to convert the route index into route values. |
| generateOffsprings  crossOver | The selected parents are used to create offsprings by using the ordered cross over. 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. |
| mutate  mutatePopulation | The output list from the cross over method is then mutated based on the mutation rate, where the two cities are randomly selected in a route and swapped. |
| nextGeneration | Call the function sequentially rankRoutes, selectParents, matingPool, generateOffsprings, mutate population. |
| geneticAlgorithm() | Main function which calls all other functions in order. Creates an initial symmetric distance matrix. Creates the first population of possible routes. Calls iteratively nextGeneration function and plots the graph Distance vs Generation. |

**Observation / Test-Case:**

* Test set - 1

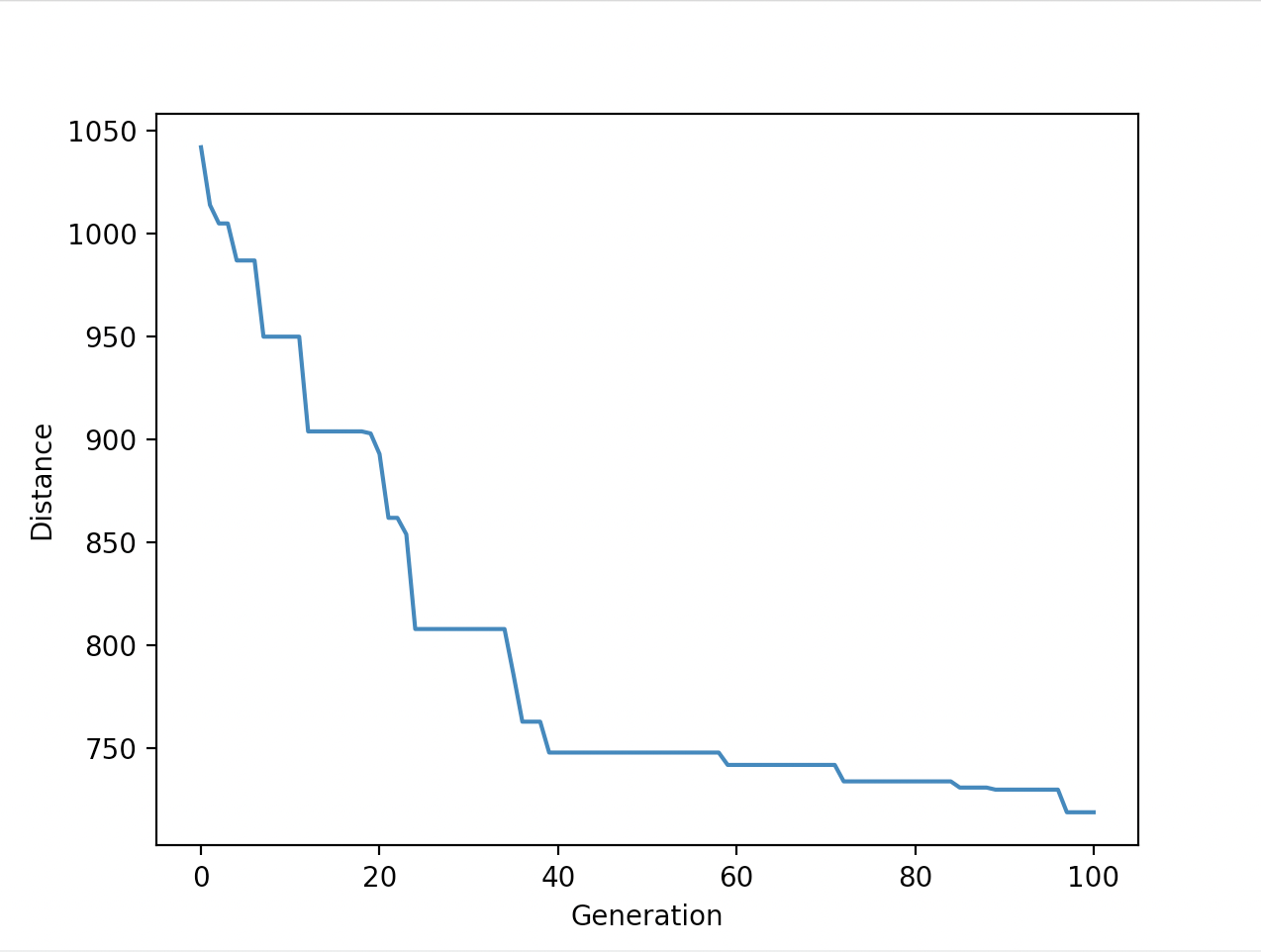
Please enter number of cities: 100

Please enter number of populations: 100

Please enter number of generations: 100

Please enter elite number of parents: 5

Please Enter mutation rate: 0.02



* Test Set - 2

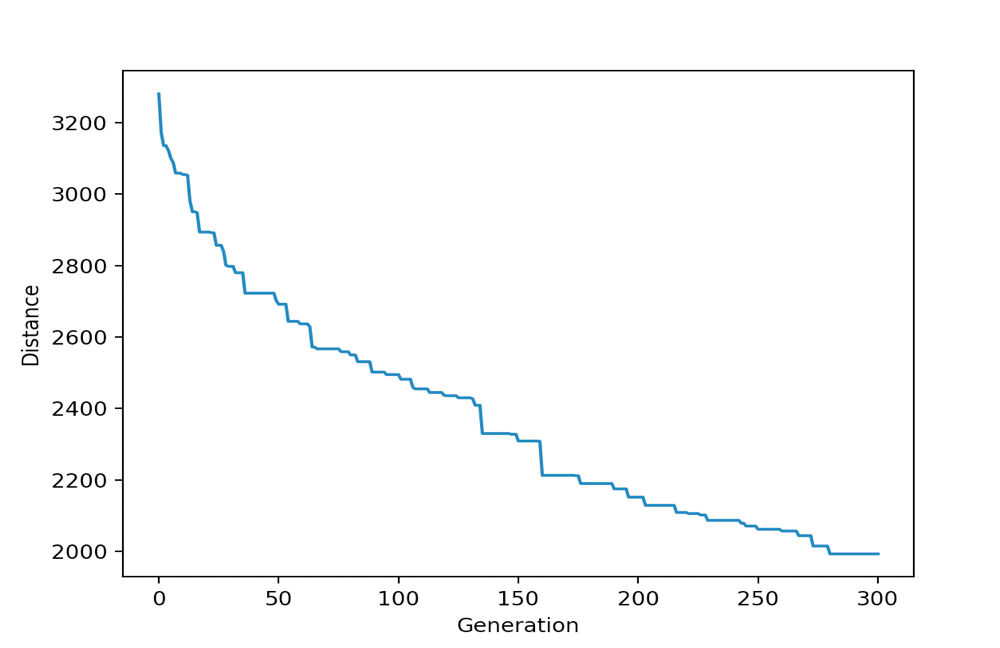
Please enter number of cities: 300

Please enter number of populations: 200

Please enter number of generations: 300

Please enter elite number of parents: 20

Please Enter mutation rate: 0.01



* Test Set - 3

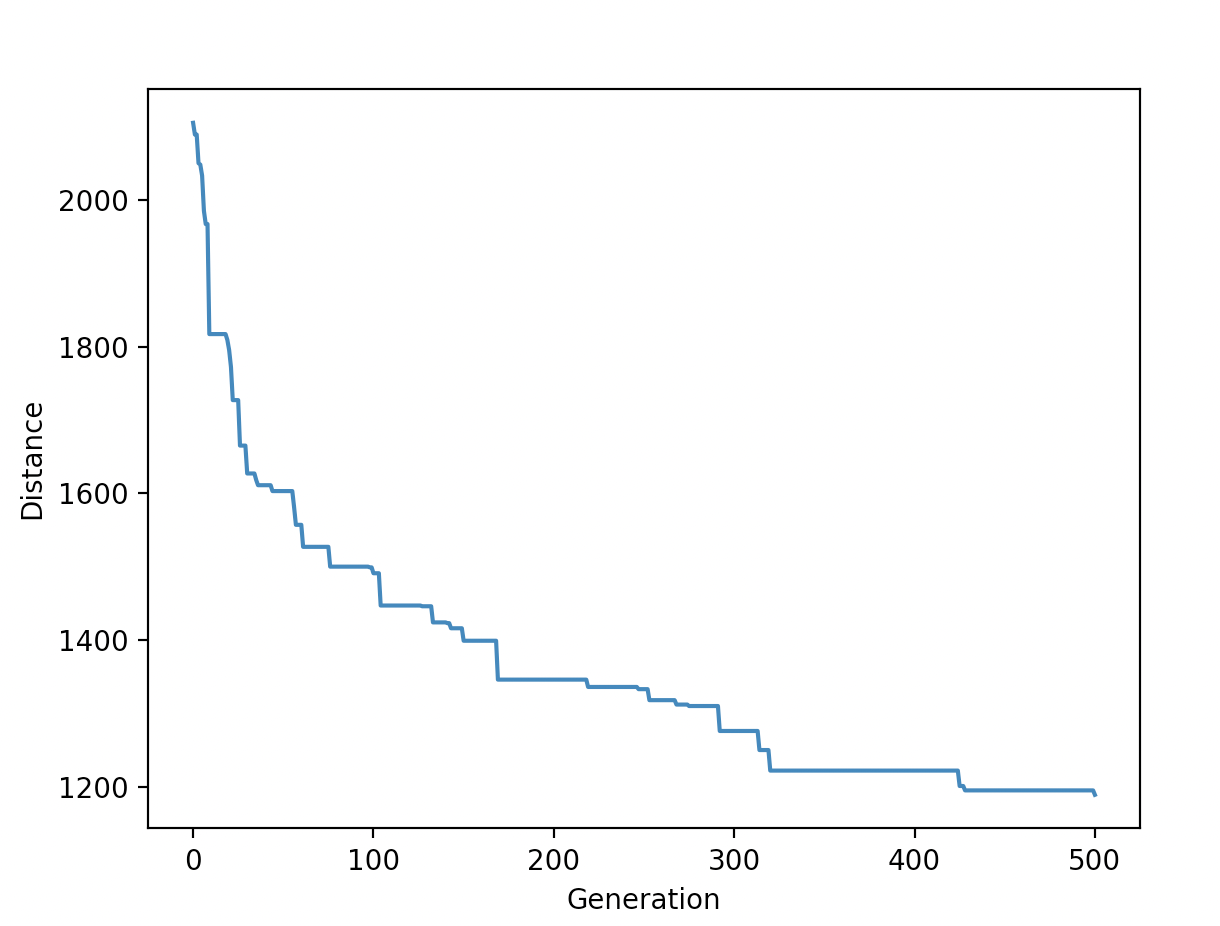
Please enter number of cities: 200

Please enter number of populations: 250

Please enter number of generations: 500

Please enter elite number of parents: 25

Please Enter mutation rate: 0.02



* Test Set - 4

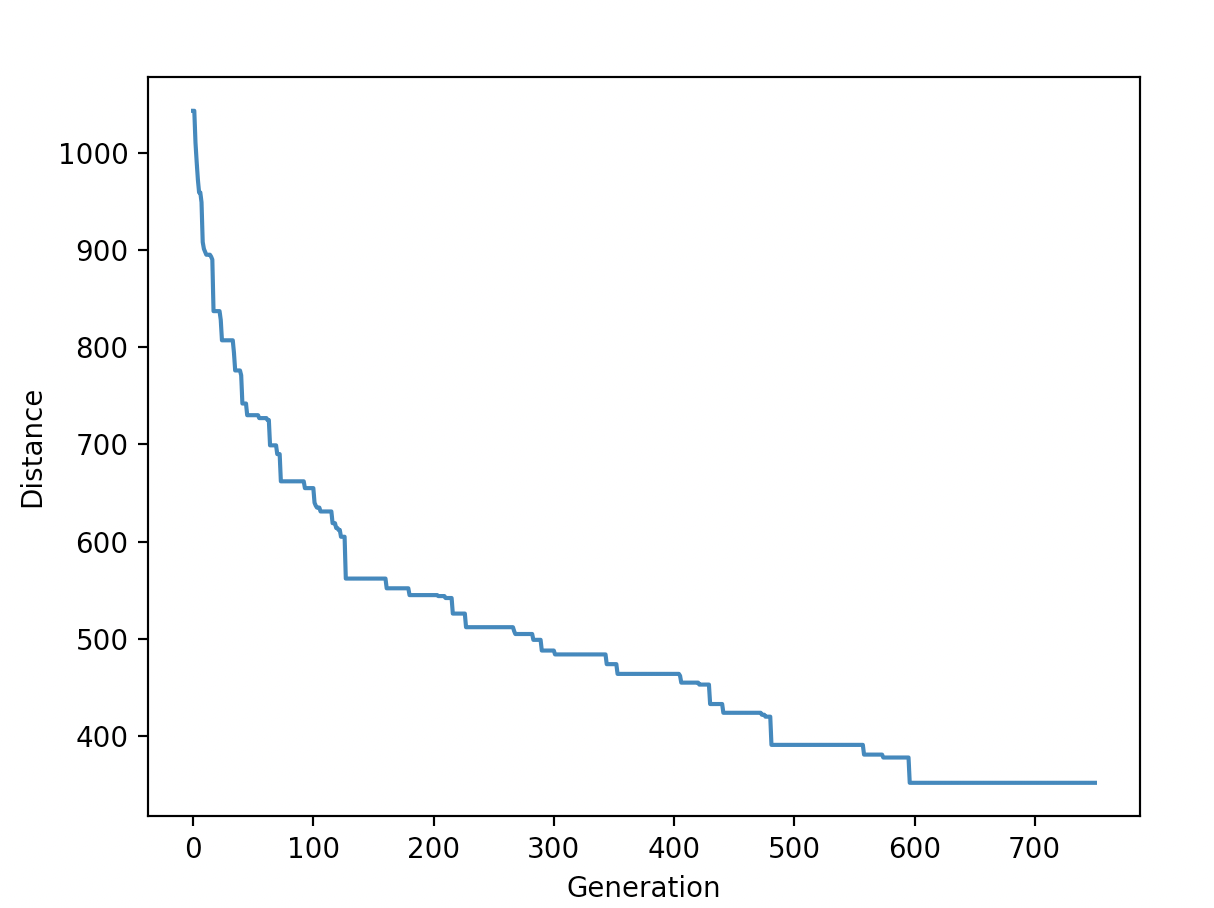
Please enter number of cities: 100

Please enter number of populations: 150

Please enter number of generations: 750

Please enter elite number of parents: 15

Please Enter mutation rate: 0.02



* Test Set - 5

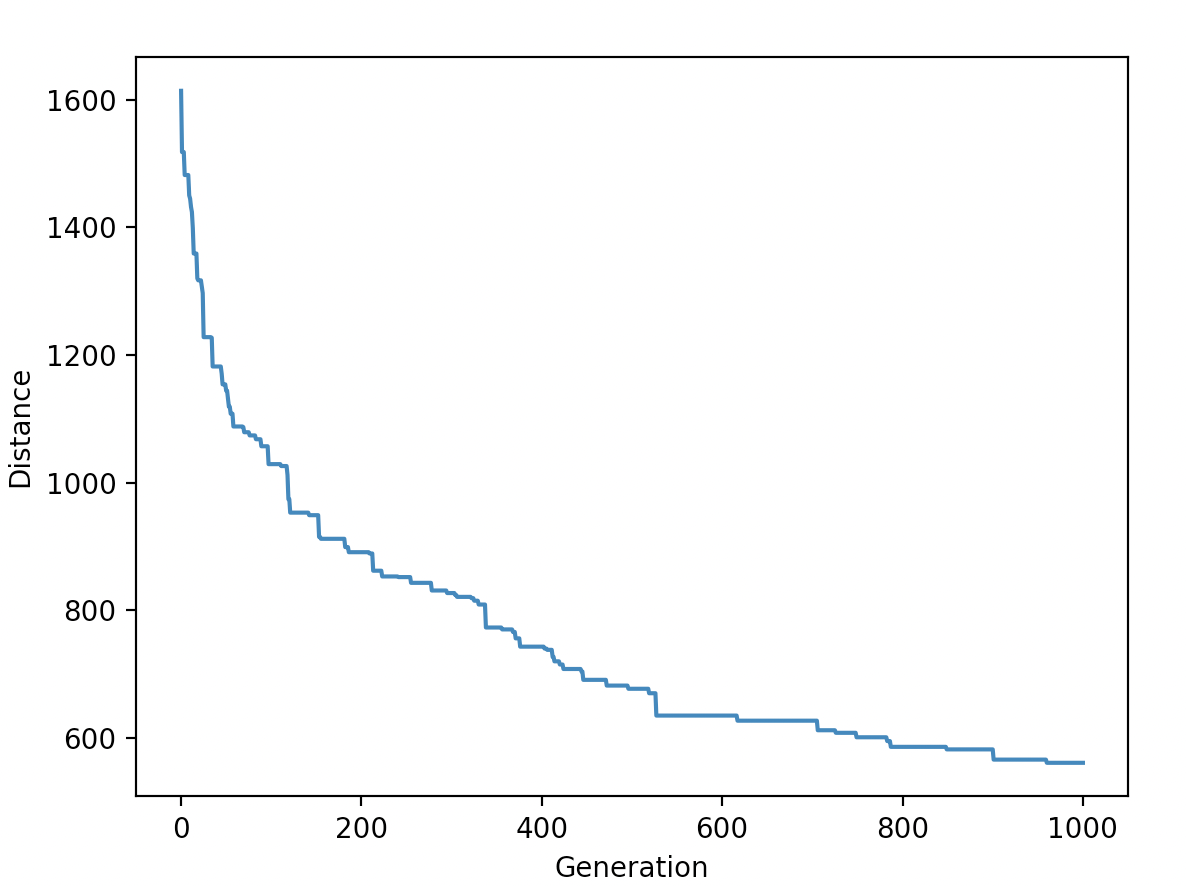
Please enter number of cities: 150

Please enter number of populations: 150

Please enter number of generations: 1000

Please enter elite number of parents: 20

Please Enter mutation rate: 0.01



* Test Set - 6

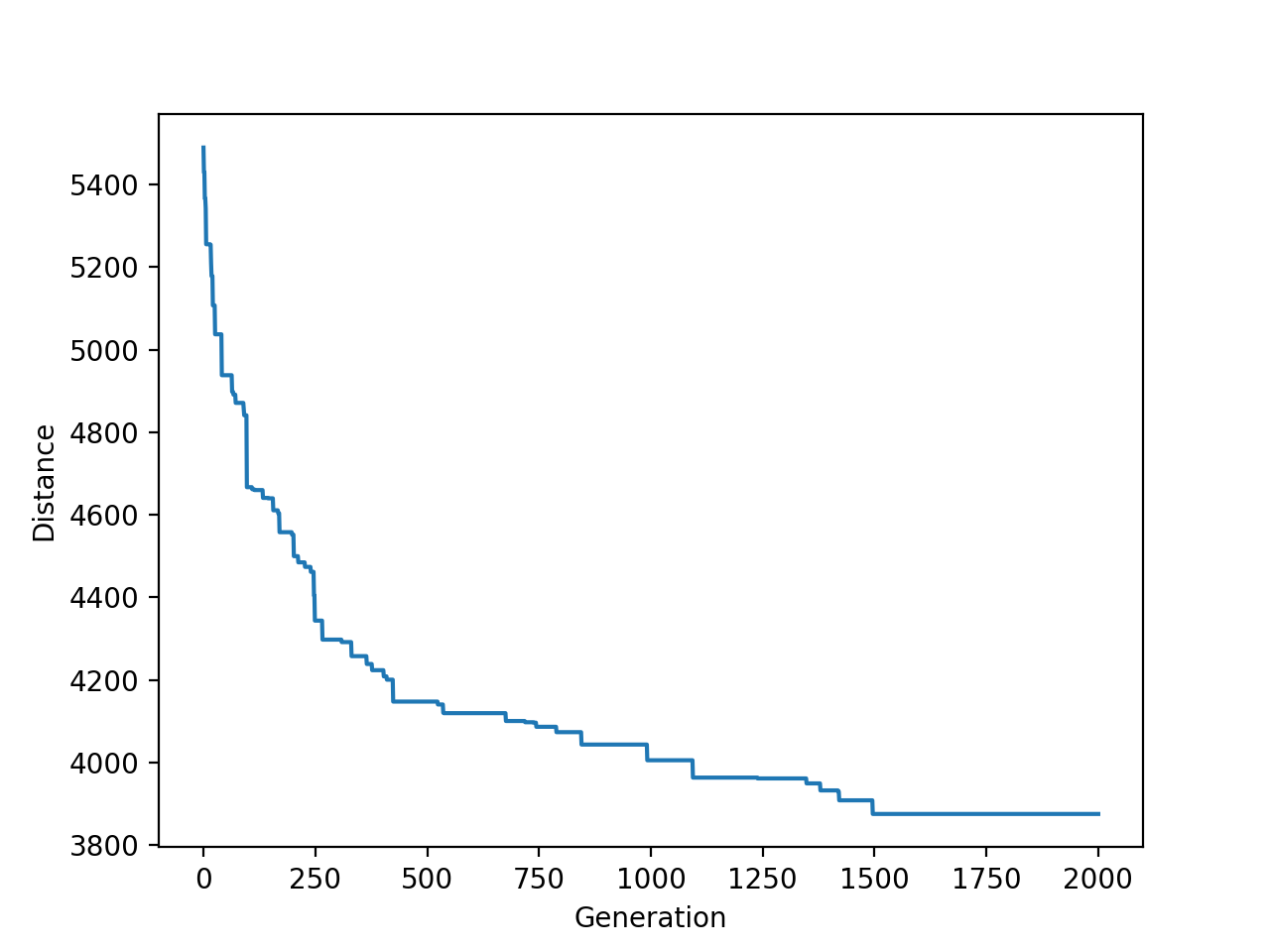
Please enter number of cities: 500

Please enter number of populations: 500

Please enter number of generations: 2000

Please enter elite number of parents: 25

Please Enter mutation rate: 0.02



The last test case tested extreme boundary conditions with 500 cities, 500 different routes and iterated for 2000 generations. The program took 1 hour to execute.

**Conclusion:**

After observing the above output results,

* The time taken to compute the best route increased exponentially when the number of generations were increased.
* The graph shows local minimas fort he best routes. The number of local minima found decreased after adding elitism.
* As the number of generation is increased, an optimal solution is found for the problem.