Assignment 4:
Optimization with Genetic Algorithms
Neuronal and evolutionary computing
2023 - 2024

Pawel Puzdrowski

2024-01-13

Contents

0.1 Git repository

The repository that holds the code for this assignment can be found on this website: https://github.com/Pawel712/NEC-Assignment4.git

0.2 Description of the chromosome, and adaptations done to the algorithm

0.2.1 Translation of the problem to chromosome

Firstly the dataset is imported with help of tsplib95 library. A simple tsplib95.load() function is used to import the dataset and get proper structure of the problem so it can be handled by the genetic algorithm. The genetic algorithm has several stages and this stages are inspired from geeksforgeeks source [3]

- Initialization
- Fitness score calculation
- Elitism
- Selection
- Crossover
- Mutation
- Population update

At the initialization stage the original population is created with create_initial_population() function. The cities are extracted from the dataset and tours are created randomly.

Fitness score calculation is done with fitness() function where following fitness score calculation is done:

$$\frac{1}{calculate_distance()}$$

Function calculate_distance() is basically calculating the total distance of a tour. Then if the distance is high the fitness score will get low and if the distance is low the fitness score will get high. Every tour is assigned one fitness score.

The part where elitism is done, it's taken from this source [4]. Elitism is done with elitism() function. The best tour from the population is chosen and added to the new population. This way the best tour is always saved in the new population and the evolution is faster.

Selection is choosing the parents for the mating part. Depending on the method the parents are choosen differently. Crossover section is creating the children from the chosen parents and depedning on the method the childrens inheritance of parents genetics is done. Mutation part has the responsibility to perform mutations on the children and as well mutation has different methods to perform mutations so the mutations can vary.

To finish off the algorithm, all the parents creates a new child and at the end a new population has arrised and its selected to do the same mating process again depending on how many generations are specified.

The different selection methods were taken from this source [4] and the different selection methods are listed below

0.2.2 Methods for selection/crossover/mutation

Selection methods: [4]

- Fitness proportionate selection (selectionFPS) - parents are choosen randomly, but the parents with higher fitness score has higher chance of being selected
- Tournament selection (selectionTS) - 4 parents are choosen randomly from the set and the parent with highest fitness score and the second highest score is chosen to mate.

Crossover methods: [5]

- One point crossover - a randomly point is chosen on the tour and the tour is splitted in two parts. Visualization of one point crossover is shown in figure 1
- Uniform crossover - each gene (city) is treated seperately instead of divided the parent in to one or more parts. Basically a flip-coin is done on every city if its included or not. Visualization of uniform crossover is shown in figure 2

Mutation methods: [6]

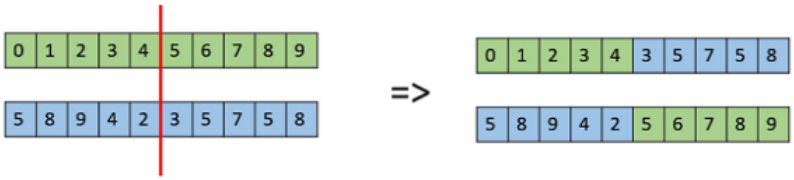


Figure 1: One point cross

Figure 2: Uniform crossover

- One bit flip mutation - only one bit is mutated which in this case means that one city is swapped with another random city from the same tour. Visualization of one bit mutation is shown in figure 3
- Scramble mutation - a random set of cities is chosen and swapped with another random set of cities. Visualization of scramble mutation is shown in figure 4

0.2.3 How the population size is chosen

After doing coople of test with population size, the best and efficient results are obtained when the population size of the size of the dataset or little bit higher. Double, tripple or higher population size doesn't result in better path.

0.2.4 How stationary state is identified

To identify stationary state for the datasets various tests were performed on the parameters: population size, number of generations, and mutation rate, and then

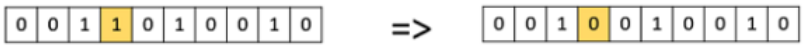


Figure 3: One bit mutation

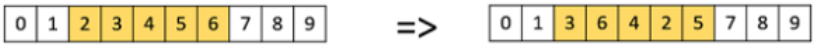


Figure 4: Scramble mutation

checking the plot of evolution of the minimum total traveling distance.

0.3 The results of executing the code for 5 problems of different sizes

0.3.1 Description of the datasets

smalldataset.tsp
Dataset consisting of only seven cities. This dataset is a modified att48.tsp where couple of points from the bigger dataset were taken and then added to the smaller dataset. This was done because there were no dataset to download that had less than 10 cities.

burma14.tsp
This dataset consist of 14 cities and was taken from this source [1].

att48.tsp
This dataset consists of 48 cities and was taken from this source [2].

ch150.tsp
Bigger dataset consisting of 150 cities and was taken from this source [1].

a280.tsp
The biggest dataset consisting of 280 cities and was taken from this source [1]. Executing the program with bigger cities than that results in very long execution time.

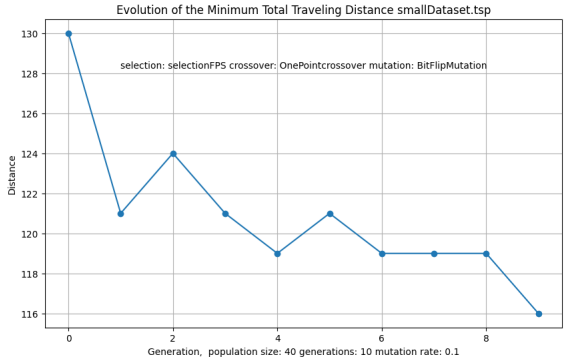
Pop size	Gen	Mutation	Selection	Cross	Mutation	Min Dist
20	100	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	116
40	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	116
40	10	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	116
40	100	0.2	selectionTS	uniformCrossover	scramblemMutation	116

Table 1: Result dataset smallDataset.tsp

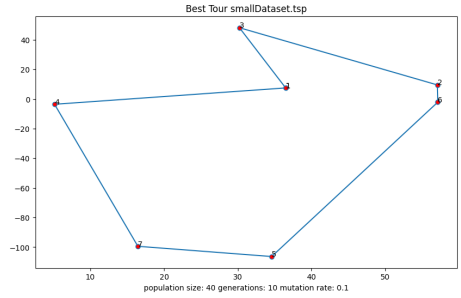
0.3.2 The results obtained with at least 6 different combinations of parameters

smalldataset.tsp

The result from trying out different parameters for smalldataset is shown in table 1. The best results and evolution of best tour are shown in figures 5. Because this is a very small dataset, it doesn't need much adjusting parameters to reach the best tour.



(a) Evolution of best tour



(b) Best tour

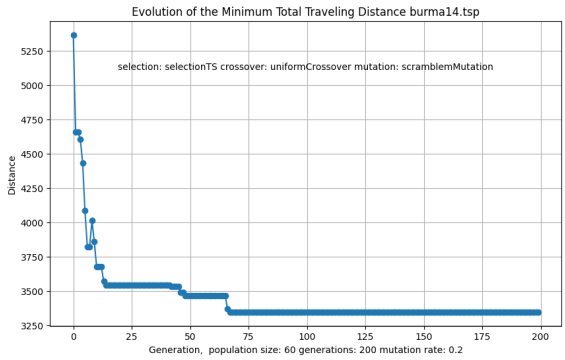
Figure 5: Plots for small dataset

Pop size	Gen	Mutation	Selection	Cross	Mutation	Min Dist
40	10	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	4814
40	100	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	4149
40	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	3642
60	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	3513
60	200	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	3346
60	300	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	3416
60	200	0.2	selectionTS	uniformCrossover	scramblemMutation	3346

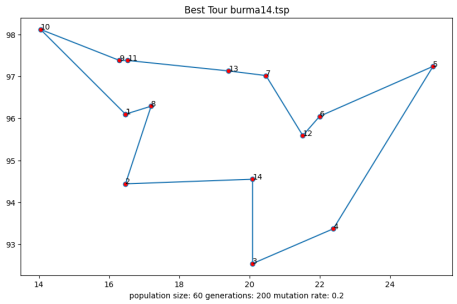
Table 2: Result dataset burma14.tsp

burma14.tsp

The result from trying out different parameters for burma14.tsp is shown in table 2. The best results and evolution of best tour are shown in figures 6.



(a) Evolution of best tour



(b) Best tour

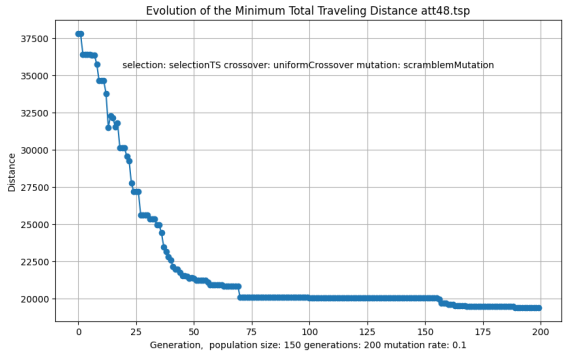
Figure 6: Plots for burma14.tsp

Pop size	Gen	Mutation	Selection	Cross	Mutation	Min Dist
60	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	31623
100	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	32467
100	200	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	29300
100	300	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	25441
100	200	0.1	selectionTS	uniformCrossover	scramblemMutation	21004
150	200	0.1	selectionTS	uniformCrossover	scramblemMutation	19408

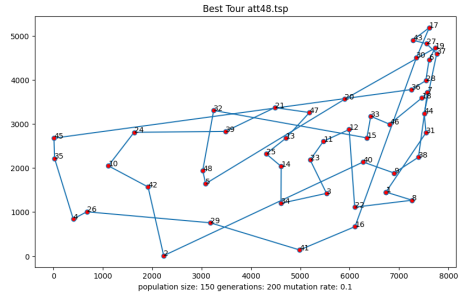
Table 3: Result dataset att48.tsp

att48.tsp

The result from trying out different parameters for att48.tsp is shown in table 3. The best results and evolution of best tour are shown in figures 7.



(a) Evolution of best tour



(b) Best tour

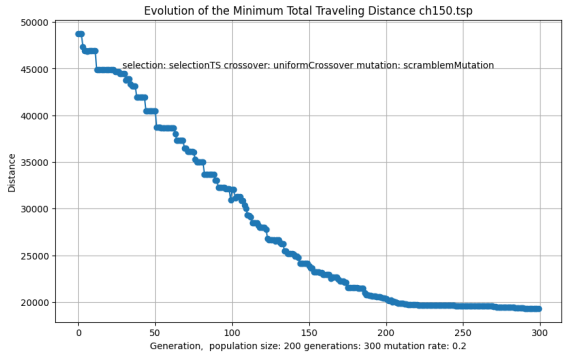
Figure 7: Plots for att48.tsp

Pop size	Gen	Mutation	Selection	Cross	Mutation	Min Dist
200	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	45815
200	100	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	47187
200	150	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	45323
200	150	0.2	selectionTS	uniformCrossover	scramblemMutation	25049
200	300	0.2	selectionTS	uniformCrossover	scramblemMutation	19321

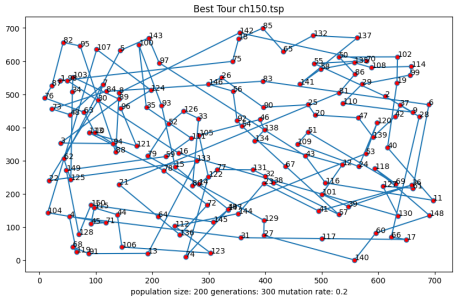
Table 4: Result dataset ch150.tsp

ch150.tsp

The result from trying out different parameters for ch150.tsp is shown in table 4. The best results and evolution of best tour are shown in figures 8.



(a) Evolution of best tour



(b) Best tour

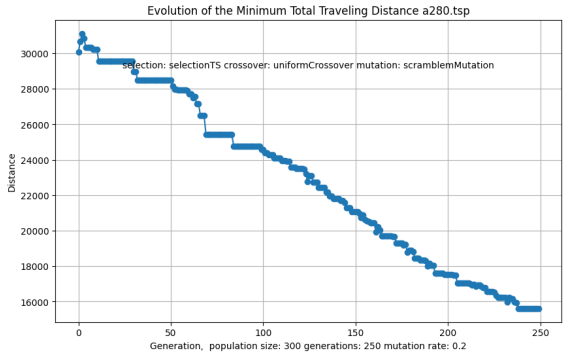
Figure 8: Plots for ch150.tsp

Pop size	Gen	Mutation	Selection	Cross	Mutation	Min Dist
300	100	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	29223
300	100	0.1	selectionFPS	OnePointcrossover	BitFlipMutation	30058
300	150	0.2	selectionFPS	OnePointcrossover	BitFlipMutation	28584
300	150	0.2	selectionTS	uniformCrossover	scramblemMutation	20839
300	250	0.2	selectionTS	uniformCrossover	scramblemMutation	15619

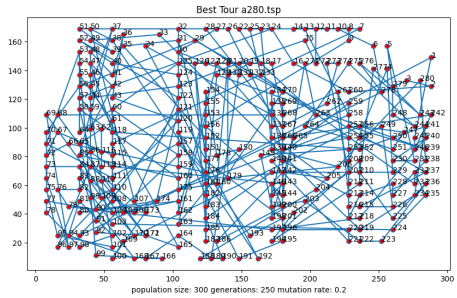
Table 5: Result dataset a280.tsp

a280.tsp

The result from trying out different parameters for a280.tsp is shown in table 5. The best results and evolution of best tour are shown in figures 9.



(a) Evolution of best tour



(b) Best tour

Figure 9: Plots for a280.tsp

Bibliography

[1] Link to source. Accessed 2024-01-05.

[2] Data for the traveling salesperson problem. Link to source. Accessed 2024-01-05.

[3] Geeksforgeeks. Traveling salesman problem using genetic algorithm. Link to source, 2023. Accessed 2024-01-05.

[4] Eric Stoltz. Evolution of a salesman: A complete genetic algorithm tutorial for python. Link to source, 2018. Accessed 2024-01-05.

[5] Tutorialspoint. Genetic algorithms - crossover. Link to source. Accessed 2024-01-08.

[6] Tutorialspoint. Genetic algorithms - mutation. Link to source. Accessed 2024-01-08.