Biologically Inspired Computing Assignment 1

UiO : University of Oslo

Hunter Wilhelm Phillips (*hunterp*)

September 21, 2018

GitHub Repository at

https://github.com/robolux/Biologically_Inspired_Computing

Abstract

This project explores algorithms that are used to solve the traveling salesman problem. The three algorithms tested in this assignment were exhaustive, hill climbing, and genetic. The use of metrics and benchmarking allows for analysis pertaining to varying approaches to solving the problem. A boundary analysis was also performed to determine the costs of increased computational time with respect to a decrease in precision. To verify the correlated computation time, a timing sequence was executed to compare with the recorded analysis. The results show that each algorithm has situations where it performs best and should be chosen based on the situation at hand. The hill climbing algorithm works optimally for fast computation time without regard to close accuracy and exhaustive is only useful for problems with smaller city sizes. For this particular case, the genetic algorithm works best when given the ability to run for a long time with large population sizes and city counts.

1. Execution of Code

User Input for all parameters to solve their choice of conditions pertaining to this situation

The onscreen user input instructions should guide you through the necessary steps to execute.

```
python main.py
```

To run selected test cases (as requested by prompt)

Please note that this can take a while due to the use of large genetic algorithm population sizes. These datasets are available in the GitHub repository for convenience.

```
python testing.py
```

All data will be available in …/results

To plot the fitness of the genetic algorithm as Generation vs Distance

```
python plot.py
```

All plots will be in …/results/plots

2. Exhaustive

The first algorithmic solution devised for the traveling salesman problem was an exhaustive search. This method essentially involves brute forcing every combination possible to find the shortest distance between the cities. To characterize this approach, the algorithm was run from n=6 to n=10 cities and benchmarked to compare results. To bound the spectrum, results as seen below in *Table 1* were selected.

Table 1: Exhaustive Search Comparison

	6 cities	10 cities
Best Trip	5018.81	7486.31
Elapsed Time (s)	0.007022	50.84134

The information found in *Table 1* provides the conclusion that the algorithm computational time was quickly growing with the corresponding number of cities. To characterize this, it can be noted that the exhaustive search takes n! number of runs to complete. Estimating the time for completion can be seen below in (1), (2), (3), and (4).

$$n!\, with\; n = 10 \rightarrow 3.6288 \times 10^{6}\; runs \tag{1}$$

$$assuming\; linear\; ratio\; given\; 10\; cities\; takes\; 50.84134\; s\; to\; run \tag{2}$$

$$\rightarrow time\; ratio = \frac{3.63\; x\; 10^{6}}{50.84} = 71374.99$$

$$n!\, with\; n = 24 \rightarrow 6.2045 \times 10^{23}\; runs \tag{3}$$

$$assuming\; the\; same\; time\; ratio\; holds \tag{4}$$

$$\rightarrow\; time = \frac{6.2045 \times 10^{23}}{71374.99} = 8.6927984 \times 10^{18}\; seconds$$

The result shows that it would be not possible in this lifetime to compute a 24-city exhaustive search, with the result being an estimated 275.64 billion years.

3. Hill Climbing

The second algorithmic solution devised for the traveling salesman problem was the use of a hill climbing method. The algorithm was run 20 times with a random starting tour to compensate for the stochastic tendencies of the hill climbing algorithm. The results obtained are showcased in *Table 2* seen below.

Table 2: Hill Climbing Results

	10 Cities	24 Cities
Best Trip (all runs)	7503.1	23624.9
Worst Trip (all runs)	11372.03	30332.14
Average Runs of Best Trip	9679.24	26567.88
Standard Deviation	1010.07	1597.56
Elapsed Time (s)	0.01317	0.01353

It can be seen that the hill climbing algorithm when compared to the exhaustive results found in *Table 1* produces a close result in a much lower computational time. The best trip for each run across the gamut set of 20 can be seen below in *Table 3*.

Table 3: Best Trip for Each Run

Run (#)	10 Cities	24 Cities		Run (#)	10 Cities	24 Cities
1	9280.85	25891.12		11	8182.92	26538.93
2	11372.03	26088.62		12	9890.03	26015.4
3	10818.13	26761.04		13	8748.02	24384.36
4	8506.22	27442.08		14	9013.27	30332.14
5	10875.04	27492.97		15	10292.46	27312.41
6	10749.65	26734.7		16	9815.08	27426.75
7	10561.26	24207.11		17	8693.25	27413.45
8	10502.25	24374.39		18	10487.21	25397.94
9	7503.1	23624.9		19	9717.4	27496.63
10	9388.58	28820.43		20	9188.04	27602.22

Table 2 also shows that the algorithm takes approximately the same time with increasing city size which makes it incredibly useful for large city counts. For example, in an on-line control problem, the speed of the hill climbing algorithm would be much more desired over the accuracy of the exhaustive search, especially with large data set bounding conditions. These results also show that for traveling salesman problems over 10 cities that theoretically the hill climber would be exponentially more efficient.

4. Genetic Algorithm

The second algorithmic solution devised for the traveling salesman problem is a genetic algorithm (GA). This approach allows for lots of customization with respect to how the algorithm is programmed and the specified input parameters. To minimize the number of lost contact between cities that have produced a good solution a partially mapped crossover was used. To mutate the children an inversion was used to suppress too much randomness in the code across generations. The parameters chosen were 100 generations, 0.2 probability, 20 runs, and a tournament size of 10. The population sizes of 10, 40, and 160 were decided upon to provide a wide metric for analysis. The results for city sizes of 10 and 24 can be seen below respectively in *Table 4 & 5*.

Table 4: GA Results for 10 Cities

	population of 10	population of 40	population of 160
Best Trip (all runs)	7486.31	7486.31	7486.31
Worst Trip (all runs)	12035.43	9488.59	8243.97
Average Runs of Best Trip	9760.89	8344.83	7649.60
Standard Deviation	1053.20	568.70	238.40
Elapsed Time (s)	1.1387	4.0568	16.7968

Table 5: GA Results for 24 Cities

	population of 10	population of 40	population of 160
Best Trip (all runs)	24000.24	18356.91	15704.48
Worst Trip (all runs)	29679.49	23593.61	20970.18
Average Runs of Best Trip	25986.70	21153.53	18213.44
Standard Deviation	1613.51	1207.84	1411.05
Elapsed Time (s)	1.9428	7.0168	29.2804

It can be derived from the information in *Table 4 & 5* that an increase in population size correlates with a more precise oscillating convergence on the solution. This comes with the massive cost of computation time though and it can be seen that a population size of 10 handles the task with the same solution in a lower amount of time. This correlates with the large array of input parameter possibilities for this problem. This can be one drawback of a genetic algorithm with the user decisions providing the groundwork for a large amount of uncertainty with the result. There is a myriad of better and worse solutions that exist with different algorithmic setups that provide the ability for reduced computational time. To display the average fitness of each generations' best fit individual, *Figure 1 & 2* were generated for 10 and 24 cities respectively.

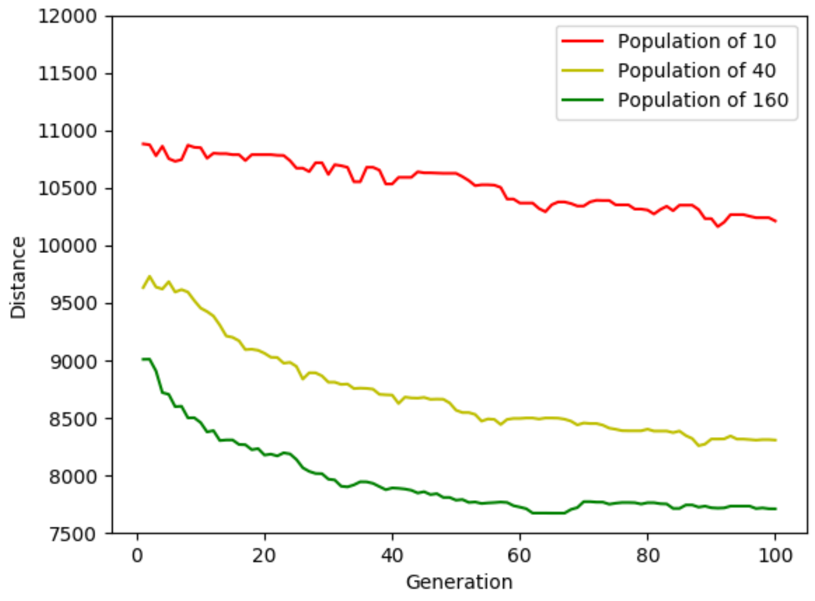

Figure 1: Fitness of Genetic Algorithm with 10 Cities

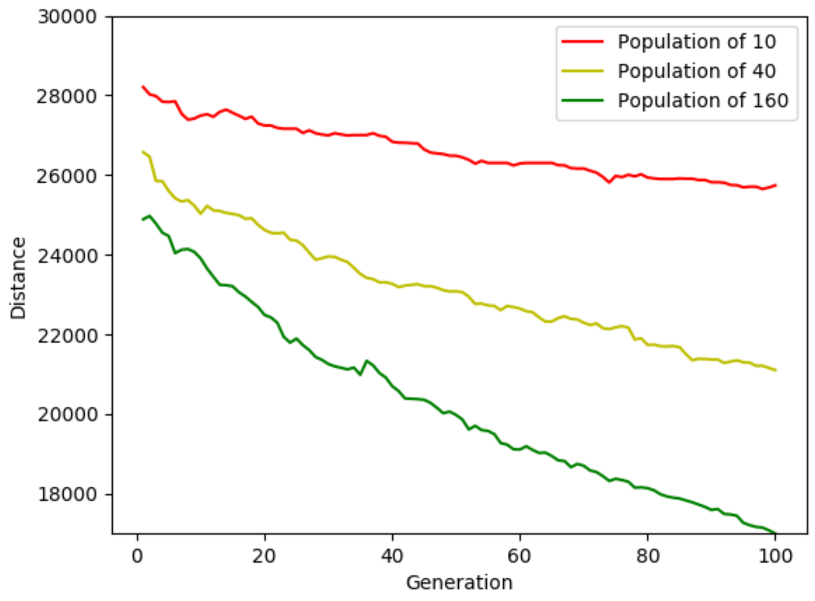

Figure 2: Fitness of Genetic Algorithm with 24 Cities

By analyzing the visual data, it can be seen that the greater the population size, the faster the curve was driven towards a better solution. Although with a city size of 10 it can be determined that populations of 10, 40, and 60 all find the best solution as found by the exhaustive search. When going to larger city sizes, such as 24, it is still converging with growing populations. There is no way to verify this solution convergence since the exhaustive search could not practically run a 24-city case. The running time of the GA enabled searches greater than 10 to be performed which showcases the efficiency compared to the exhaustive search.

To determine the amount of tours completed by the GA the following formula was derived:

$$\begin{aligned} GA\ tours &= (population)(generation\ count)(runs) \\ &= (160)(100)(20) \approx \boldsymbol{320000\ tours} \\ Exhaustive\ tours &= n! \\ &= 10! \approx \boldsymbol{3628800\ tours} \end{aligned}$$

This does not completely account for all possible mutations and extraneous children being produced but provides a close enough approximation. It can be seen that the GA was able to produce a similar result to an exhaustive search with a lot less runs, resulting in a shorter computational time.