Mark Harmer
CS 776 - Assignment 3

Symmetric Traveling Salesman Problem

Encoding:
I used a simple permutation encoding where a gene represented a single city and the ordering translates to the route the salesman takes; no city is represented twice and all cities are represented. I looked into other methods such as using genes to represent links between two cities, however I wanted to see how well the GA would perform with simple permutation encoding.

Selection:
I used elitism with an elite size of 15% of the population. I used a population size of 50. I tried several population sizes ranging from 30 to 500 and 50 was the lowest population size in which the GA still did well.

Crossover:
I used a crossover method that would simply do a regular crossover then "fixup" individuals with copies of cities by removing them and randomly replacing them with cities it didn't contain, I also used one-point crossover. This method didn't work too well, the GA would still get stuck with bad "solutions". I then tried two-point crossover that I observed marginally created better solutions than before.

At this point I created a graphical utility to observe some of the solutions the GA was creating. It was apparent that the path it generated was still tangled, and some sections should really be reversed. At this point I came up with the idea of keeping two-point crossover but allowing a 50% chance for the child to reverse each section of the crossover section (3 in total for 2-point). This greatly increased the rate and quality of solutions the GA found. This however still requires a "fixup" of the individual to be done which is quite expensive.

After some initial querying search engines, as I assumed, this reverse method has been done before. Although it doesn't appear to be attributed to a single person, perhaps because of it's simplicity.

I found that the crossover was a large contributor to how well the GA found good solutions.

Mutation Operator:
I used a swap city mutation operator that would take one city in the individual and swap it with another. Initially I swapped cities that were side-by-side but then moved to just two random cities. I observed that any mutation rate higher than 0.01 hindered the GA more than helping it. I used a mutation rate of 0.005 per gene.

Generations:
After a few test runs I found the GA generally found a good solution within generation 2500, however for my final data collection experiments I allowed it to run for 3000.

Changing Fitness Function Over Time:
I had an idea to utilize the fitness function in a piecewise manner, where the GA would change it's fitness function to incorporate more gene evaluations as it reached certain milestone generations. My thinking was that it would attempt a local search and find a good solution to the first 8 genes, then expand to finding a solution to 12 genes, and so on. This would result in a sharp drop in the fitness whenever the GA hit these milestones, however it would apply to all individuals. After implementing this and experimenting with it it became apparent that it didn't work well. I believe it was operating more as a greedy search and wasn't looking at the entire problem.

Numerical Results:
I ran the experiments on the Eil51 and Eil76 datasets. For the Eil51 dataset the GA actually does pretty well with many of the runs within 90% quality.

For the Eil76 dataset the percentage distance in quality are smaller than the previous dataset because of a harder problem.
The same number of generations was used to compare the two problems. If I had more time I would rerun the datasets for at least 5000
generations.

Overview of experiment setup:
Population Size: 50
Elitism: 0.15 (= 8 individuals saved per generation)
Mutation Rate: 0.005
Reverse Crossover Section Probability: 0.5 (per section, 3 sections total in two-point crossover)
Runs: 30
Generations: 3,000

Eil51 - Optimum length: 426.
Quality = 95%, the route mark is 448.
Reliability = 30% (9 runs out of 30)
Speed = 134,690

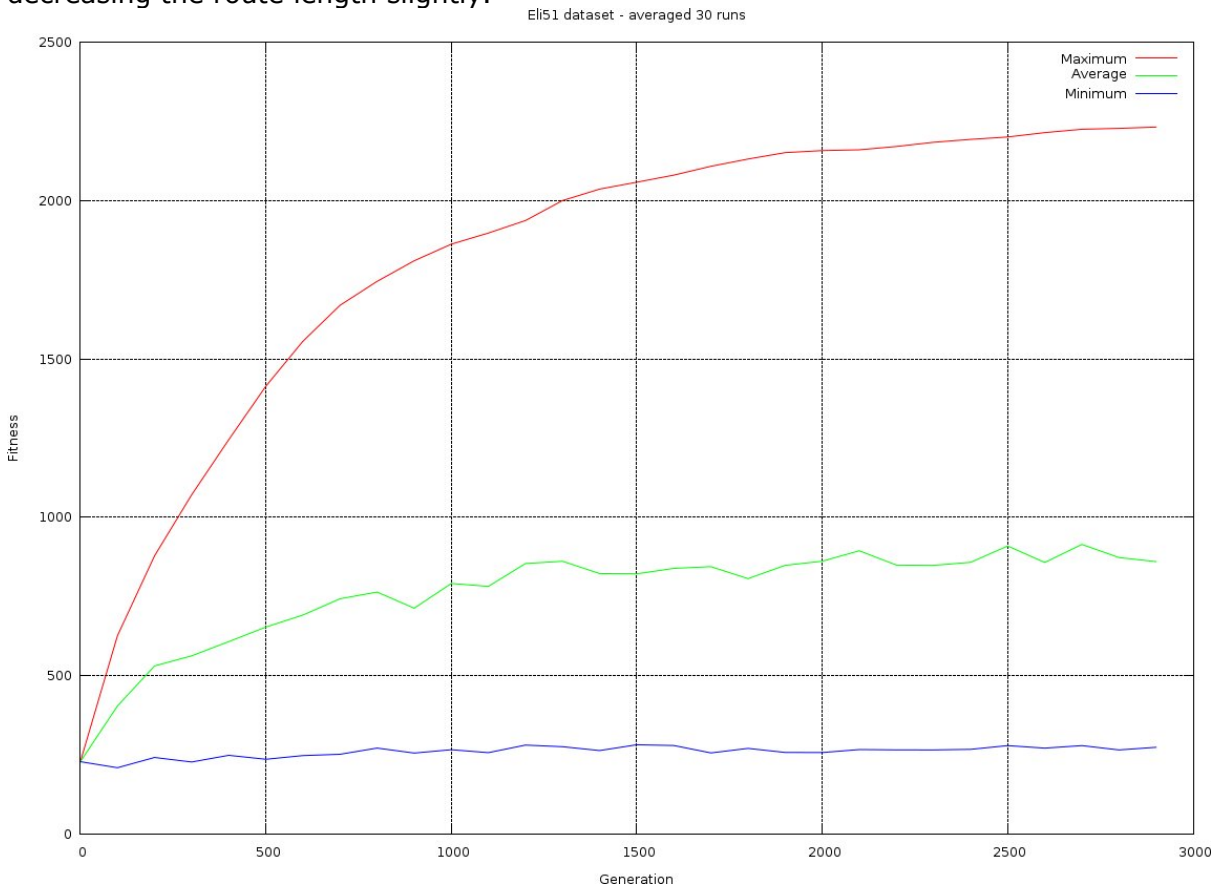
Quality = 90%, the route mark is 473.
Reliability = 90% (27 runs out of 30)
Speed =83,305

Eil76 - Optimum length: 538.
Quality = 85%, the route mark is 633.
Reliability = 30% (9 runs out of 30)
Speed = 143,921

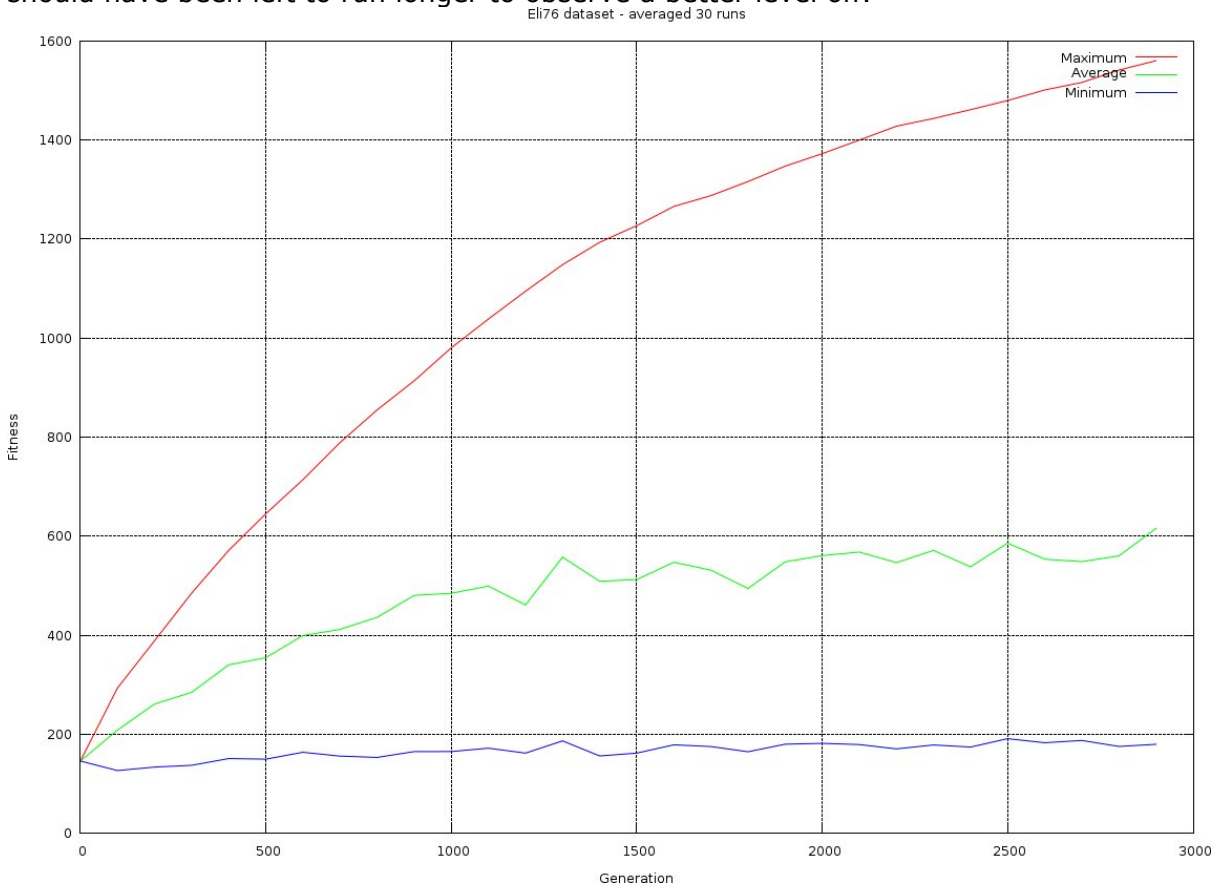
Quality = 80%, the route mark is 673.
Reliability = 93% (28 runs out of 30)
Speed = 109,711

Graphical Results:

Eil51 dataset: After 2,000 generations the GA finds a good solution and slowly revises it decreasing the route length slightly.

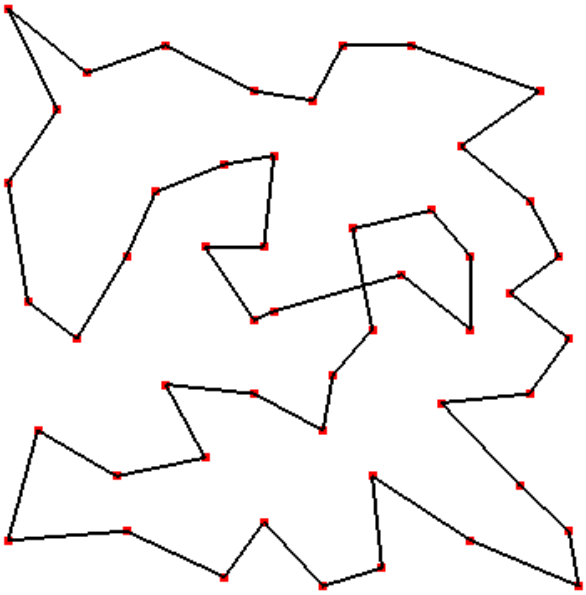


Eil76 dataset: It takes longer for the GA to work up a good solution as observed from the averaged maximum in comparison with the Eil51 dataset, in hindsight this dataset should have been left to run longer to observe a better level off.

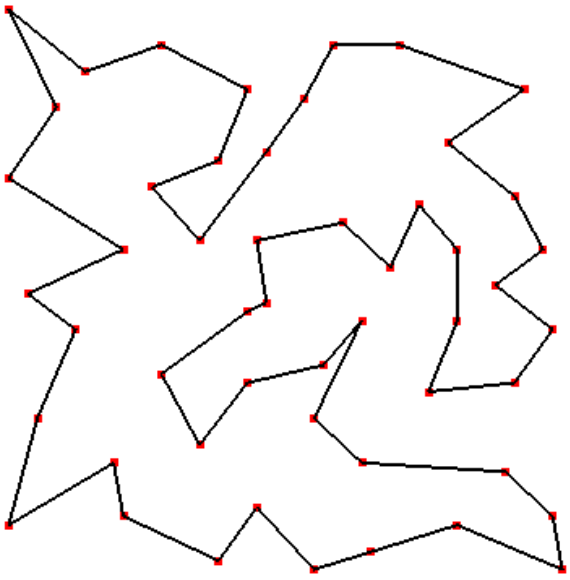


Visualization Tool:
The visualization tool I used to debug/experiment with the GA TSP.

This one is the Eil51 dataset near the end of the 3000 generations. As you can see the GA does well but the center area could still be optimized if a random reverse crossover were generated.

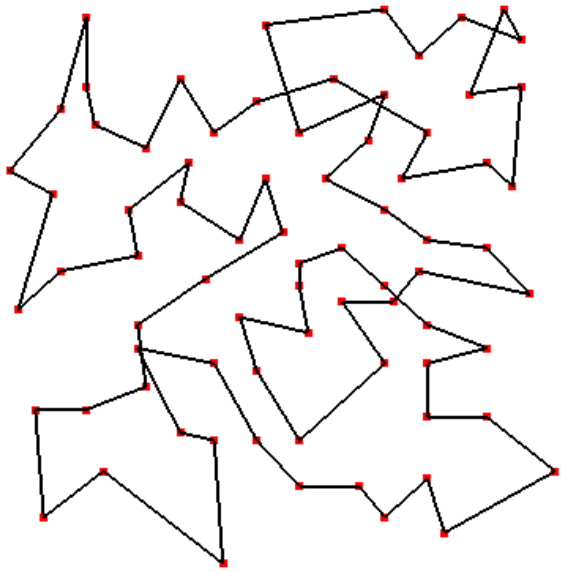


Below is the Eil51 dataset with the GA run at the end of 6,000 generations. You can see it found a good solution, the route length for this was 452 which is a quality of 94%.



Below is a Eil76 dataset image near the end of 3000 generations. This GA could find a

better solution given more time to work. In single run experiments it finds a good solution in ~6000 generations.



The image below is from the Eil76 dataset run over 6,000 generations with a route length of 592. This is a quality of 90.8%. Given enough time the GA should be able to unwrap the upper left quadrant that is still tangled.