**Project 4 – Genetic Algorithm**

Zachary Shumate

[CECS]

Speed School of Engineering

University of Louisville, USA

znshum01@louisville.edu

1. **Introduction** (What did you do in this project and why?)

This project had me once again solve the Traveling Salesman Problem except this time, I used a genetic algorithm to find the solution. This genetic algorithm took the complete list of cities and generated many routes between them and then proceeded to use crossover methods and mutations to create altered ‘children’ which became better and better as generations passed. I implemented four combinations of two crossover functions and two mutation rates. This approach did not result in the shortest possible cycle through these approaches, but was able to produce good solutions for a large number of cities that other, simpler algorithms would tame immensely longer to complete.

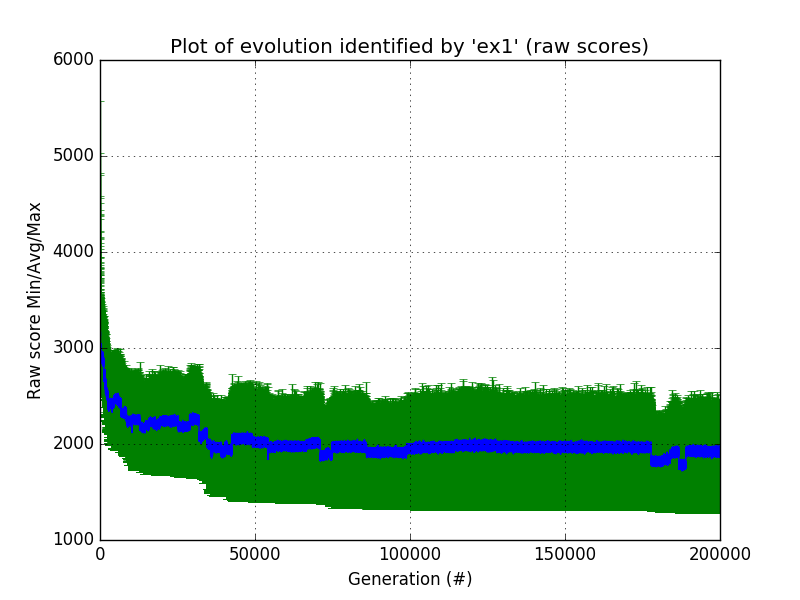
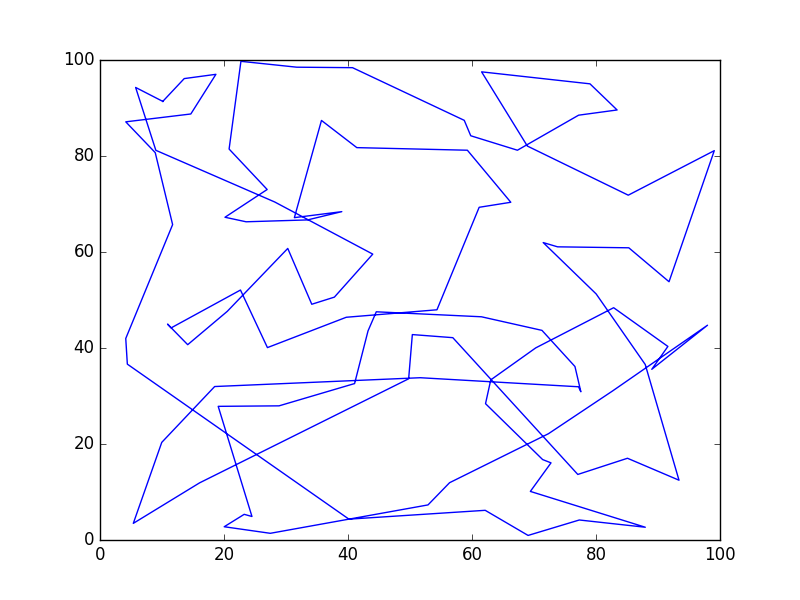
1. **Approach** (Describe algorithm you are using for this project)

This Python script accepts the files of the city coordinates which are passed into a list. These coordinates are then passed into code from Pyevolve, an imported library, designed to run genetic algorithms that proceeds to populate generations with routes that are improved from their ‘parent’ routes using a combination of an *edge recombination* or a *cut and crossfill* crossover function, provided by the library, and a mutation rate of 0.02 or 0.002. This program ran each combination for 200,000 generations. I ran each scenario multiple times and stopped upon realizing that for each combination the improvement curve flattened out, showing diminishing returns, and the final route cost was approximately, if not exactly, the same.

1. **Results** (How well did the algorithm perform?)
   1. **Data** (Describe the data you used.)

The data for this project was a provided file with 100 cities. Each city had unique coordinates.

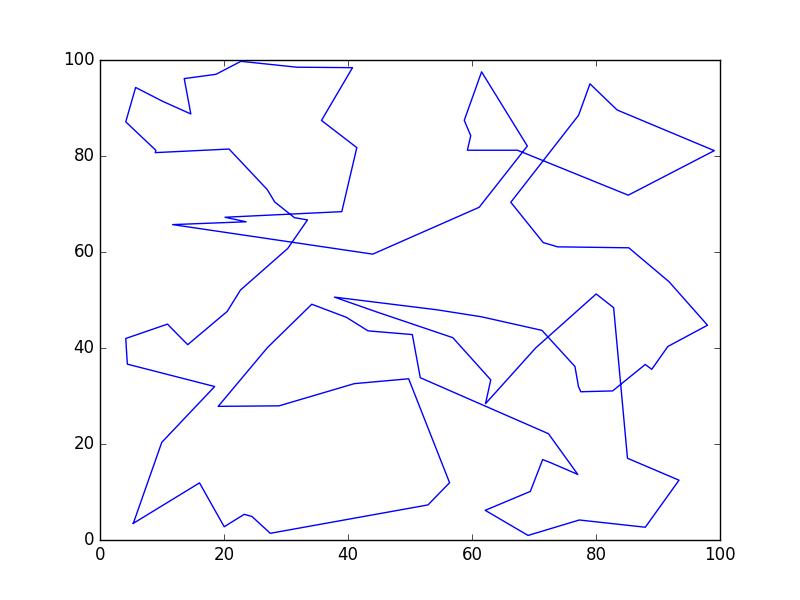
* 1. **Results** (Numerical results and any figures or tables.)

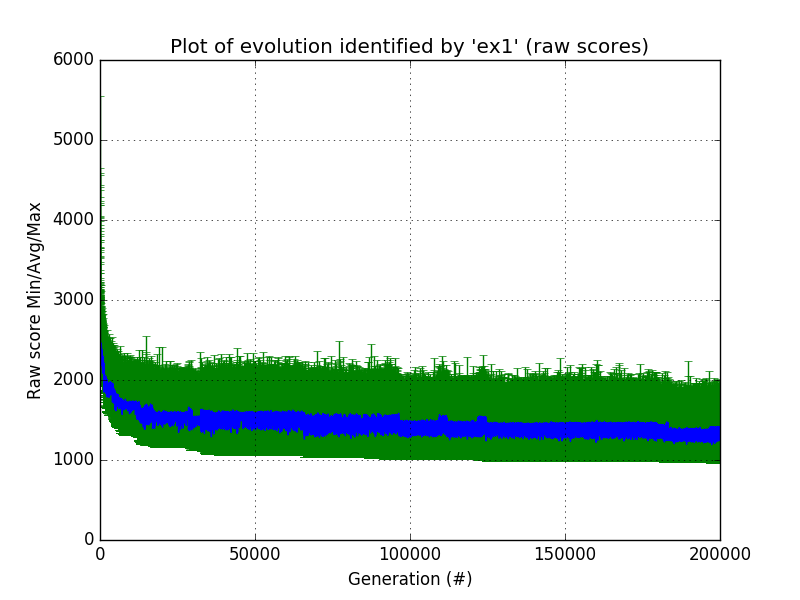
Dataset 1A (Edge Recombination, 0.02):

Gen. 200000 (100.00%): Max/**Min**/Avg Fitness(**Raw**) [2269.99(2241.05)/1231.95(**1282.41**)/1891.66(1891.66)]

Total time elapsed: **3781.328 seconds**.

**List**: [43, 79, 4, 13, 48, 11, 17, 86, 3, 32, 58, 63, 34, 96, 46, 98, 45, 33, 95, 64, 87, 75, 47, 1, 76, 25, 49, 77, 26, 66, 21, 51, 99, 54, 27, 62, 31, 44, 60, 18, 90, 2, 52, 22, 85, 40, 37, 70, 23, 72, 69, 15, 89, 24, 39, 12, 28, 9, 6, 88, 74, 16, 92, 10, 68, 7, 53, 50, 55, 97, 59, 5, 19, 78, 67, 84, 35, 20, 36, 65, 38, 73, 93, 82, 71, 29, 42, 0, 91, 94, 14, 80, 83, 30, 61, 41, 57, 56, 8, 81]

Dataset 2A (Edge Recombination, 0.002):

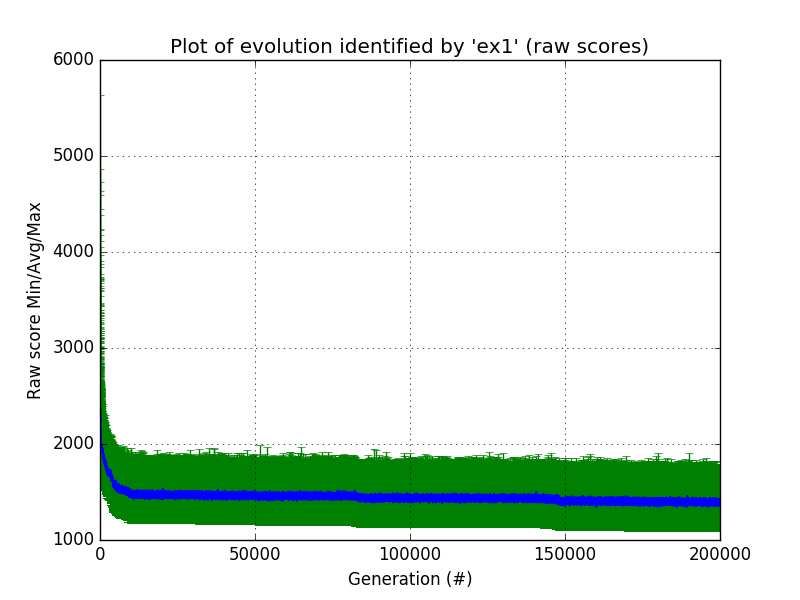
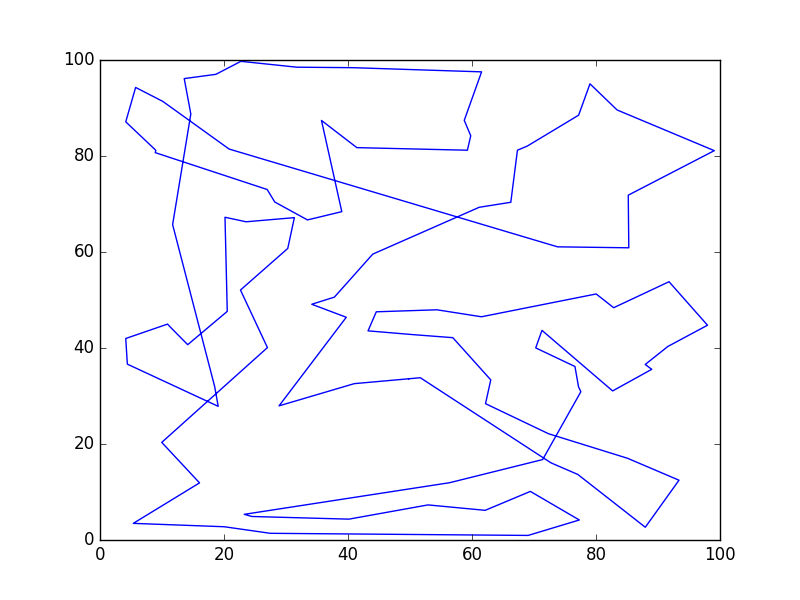


Gen. 200000 (100.00%): Max/**Min**/Avg Fitness(**Raw**) [1641.77(1757.36)/1079.10(**957.00**)/1368.14(1368.14)]

Total time elapsed: **4230.812 seconds**.

**List**: [39, 12, 28, 83, 30, 58, 63, 32, 3, 34, 86, 1, 75, 13, 49, 77, 41, 4, 57, 79, 43, 56, 81, 8, 26, 66, 21, 87, 64, 47, 25, 76, 61, 48, 45, 60, 44, 51, 99, 95, 54, 18, 90, 62, 31, 27, 33, 85, 22, 52, 2, 20, 65, 36, 37, 35, 88, 6, 74, 16, 92, 98, 11, 10, 69, 93, 82, 73, 40, 38, 23, 70, 0, 91, 94, 14, 42, 71, 29, 72, 84, 9, 15, 68, 46, 17, 96, 55, 50, 53, 7, 89, 67, 78, 80, 19, 97, 59, 5, 24]

Dataset 1B (Cut and Crossfill, 0.02):

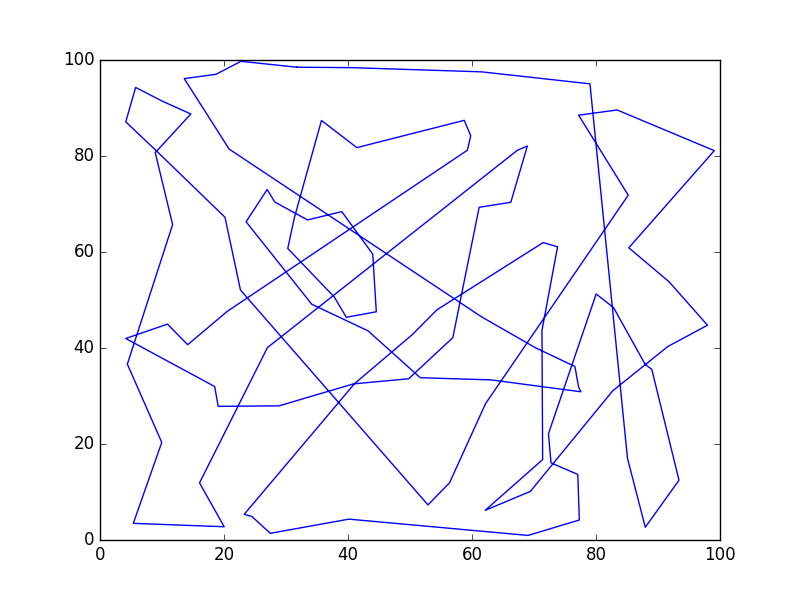


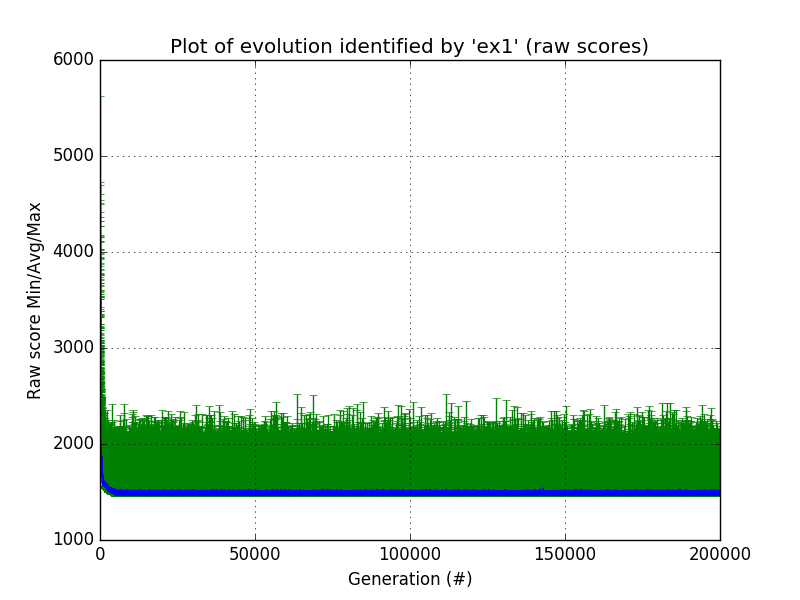
Gen. 200000 (100.00%): Max/**Min**/Avg Fitness(**Raw**) [1664.55(1608.22)/1013.45(**1089.33**)/1387.12(1387.12)]

Total time elapsed: **2503.694 seconds**.

**List**: [89, 7, 53, 50, 46, 17, 11, 48, 45, 33, 54, 60, 27, 31, 62, 90, 18, 52, 22, 85, 77, 43, 79, 57, 4, 41, 49, 13, 1, 47, 87, 64, 95, 99, 51, 44, 21, 66, 26, 8, 81, 56, 61, 28, 55, 83, 30, 58, 63, 32, 3, 25, 76, 75, 86, 34, 96, 12, 24, 39, 5, 19, 94, 91, 42, 14, 78, 80, 97, 59, 67, 71, 88, 6, 74, 73, 16, 35, 36, 37, 65, 20, 2, 38, 40, 92, 98, 10, 68, 15, 69, 93, 82, 84, 23, 70, 0, 72, 29, 9]

Dataset 2B (Cut and Crossfill, 0.002):





Gen. 200000 (100.00%): Max/**Min**/Avg Fitness(**Raw**) [1793.79(1909.93)/1469.69(**1459.92**)/1494.83(1494.83)]

Total time elapsed: **2363.059 seconds**.

**List**: [66, 26, 8, 81, 77, 92, 73, 74, 6, 88, 93, 9, 68, 17, 76, 49, 13, 1, 47, 48, 10, 46, 11, 86, 75, 87, 64, 51, 99, 95, 3, 32, 63, 58, 30, 28, 55, 50, 53, 89, 69, 45, 33, 60, 54, 96, 24, 5, 39, 12, 83, 61, 41, 56, 43, 79, 57, 4, 25, 34, 78, 67, 82, 18, 27, 62, 90, 52, 2, 20, 65, 35, 42, 14, 71, 16, 22, 85, 98, 15, 7, 59, 97, 19, 80, 94, 91, 72, 29, 84, 40, 38, 37, 36, 70, 0, 23, 31, 44, 21]

1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

Each of these took around an hour to run through each of the 200,000 generations and produce a resulting route. I received the best route from using the Edge Recombination crossover function and a mutation rate of 0.002. From my testing, it seemed to me that the Edge Recombination function produced more consistent results over the Cut and Crossfill function. I did not have many issues with implementation and analysis thanks to the Pyevolve library that I discover to aid in the grunt work of the algorithm.

I have determined from this that genetic algorithms are great tool to find solutions to problems with lots of possibilities which would be greatly inefficient with other appropriate algorithms, but genetic algorithms would be a poor solution to simpler problems where there are algorithms that can get the very best route in an efficient amount of time. Genetic algorithms are able to solve problems that other algorithms would be unable to solve as well since generations can be quickly generated and evaluated individually. This can allow for solutions to be generated whenever the algorithm needs to stop, or the algorithm can be left to continue working thus getting ever closed to the optimal solution.

1. **References** (If you used any sources in addition to lectures please include them here.)

I used Pyevolve to help with the genetic algorithm as it has many functions to generate generations and run crossovers as well as mutations. I also used one its example programs from the site as a basis and of my code.

<http://pyevolve.sourceforge.net/0_6rc1/>