**TSP – Search with Genetic Algorithms**

Blake Emmons

[CECS]

Speed School of Engineering

University of Louisville, USA

[blake.emmons@louisville.edu](mailto:blake.emmons@louisville.edu)

1. **Introduction**

In this project we were tasked with solving the traveling salesman problem using a genetic algorithm with a two variable parameters with at least two settings. We did this to develop understanding of some more advanced pathing algorithms that have the ability to adapt to the current circumstances and reach an answer through iteration upon a solution rather than the usual search methods. Performing this project helps to enhance current knowledge of algorithms, tree pathing, and search design.

1. **Approach**

The first thing that needed to be done in order to develop my genetic algorithm was to generate my starting population. In order to do this the nodes provided are looped through and then a random number is selected in the range of 0 to the length of the node list and a node is selected to be appended to the current path. This path is then returned and placed into a list of paths. From here the lengths of each path is calculated and stored for later use. This process continues until the desired population size is reached, in this case 100. The user is then asked for input on which mutation rate to use and which of the implemented crossover methods to use.

Next, the generation cycle is entered. A loop is performed until the number of generations defined is reached and then the final steps of the program are performed to begin displaying the relevant data. In this loop the generation is calculated. This function starts by sorting a copy of the list of lengths to pass into other functions. A loop is entered to loop through each of the paths in the current population. This list chooses 2 parents through a tournament selection, where the smallest distance path in the tournament will be made a parent, with tourney sizes of 2. Tourney sizes of two were chosen to preserve genetic diversity in the populations. Parent 2 is chosen with more care so that the same parent is not chosen twice.

After parents are chosen they are input into the crossover function. The crossover function that was used in this assignment was found in Genetic Algorithm Solution of the TSP

Avoiding Special Crossover and Mutation by Gokturk Ucoluk from the Middle East Technical University in Ankara, Turkey. The idea printed in this paper was to convert the given paths into a chromosomal form using mathematical inversions (the sortedness of the current list). The numbers are selected in this chromosome based upon how many numbers to the left of the number selected, indicated by the index in the chromosome array. For example, if the first box of the chromosomal array was selected and there are two numbers bigger than 1 to the left of it in the path then the number placed into this box is 2. The beauty of this conversion is that a crossover can be performed without having to worry about duplicate numbers or similar issues as long as the chromosome boxes are not swapped indexes. In my case I used two different methods of splitting the chromosome. In the first method the chromosome is split at a random location, in a method similar to the algorithm in the paper discussed above. One side, in my case the right, is then swapped between the chromosomes. In the other method that was used two random indexes are chosen and then the subarrays of paths in the 2 lists between the two indexes are swapped. After this crossover is performed the chromosome is reverted back to normal path form. The crossover is repeated a number of times equal to the population amount as to provide an ample number of children to choose from. A final crossover between the best path and a random other path is performed to preserve the good genes of the best path.

Upon completion of the crossover function the mutation function is entered. The mutation method used is a simple insertion mutation found in a paper linked in the references section called Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, where one node in the path is inserted into a random new location in the path. To implement this the population is looped through, to provide mutation opportunity to all nodes, and a random number between 0 and the value determined by the user in the beginning, either 1000 or 100 depending on the percentage, is generated. The loop checks if this number is <=0 and this gives the user a 1% mutation rate if the range chosen from is 1000 or 0.1% mutation rate if the range chosen from is 100. A copy of the Population length list is then created and then the paths with the worst values are removed until the population is back down to 50.

Finally, in order to dynamically display the best path routes drawing of the path is updated every time a new one is made.

1. **Results**

The algorithm performs moderately well when performed for a large number of generations, but it still does not arrive at an exact solution and it takes considerable time. The fewer generations that are calculated the farther from the goal path. Although fewer generations does perform more quickly. When compared to the other approaches that we have used to solve the TSP problem the created genetic algorithm performs much more slowly and not as effectively. The minimum finish time for a population of 100 for 500 generations and for any of my variations of the algorithm was 66.18104 seconds, while the average was around 70 seconds. It should be noted that these time values were attained with the dynamic graphing of the path turned off as to save time. This value is significantly slower than previous TSP Algorithms. Especially when compared the the greedy heuristic algorithm which when retested finished 100 nodes in 0.72866 seconds. Further, the genetic algorithm created is not very accurate, with an average path length found of around 3190 when the best path was well below 1000 with the TSP Greedy Heursitic algorithm finding a minimum length of 951.83. Given, the genetic algorithm results would improve with larger numbers of generations often getting around 2000 for around 2000 generations.

Based on the results of the testing of the variables in my solution the clear best variable combination is the 1% mutation rate and the crossing with the middle range. This was decided based on the table in Figures 1-3 below in 3.2 Results. This particular variable combination outperformed the rest in all categories except for time. It had the lowest average best path, beating out the 1% right hand switch by a small margin. The other statistics show the 1% mutation rate and crossing with the middle range clearly in the lead. With a minimum distance 100 less than any of the others, the smallest max value, and an incredibly low standard deviation in comparison to the others. The time was also only slower by a small margin.

* 1. **Data**

The data the algorithm uses to create the optimized paths comes from a series of .tsp files that hold a series of coordinates labeled by node. In order to determine how my algorithm performed I used a series of different metrics. First, I used a timer to time how long my program took to execute from when the algorithm type was run to when the optimized path was rendered. Then to get a good idea of the average time, I ran my program 5 times for each given input file. The program was also run 5 times to find minimum and maximum values, as well as the standard deviation for time, number of mutations, and path distance.

* 1. **Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best Distance | | | | |
| n | MR (1%)/RC | MR (1%)//MC | MR (0.1%)//RC | MR (0.11%)//MC |
| 1 | 2840.173 | 2860.698 | 3426.509 | 3234.939 |
| 2 | 2855.62 | 2889.602 | 3792.898 | 3597.695 |
| 3 | 3301.569 | 2851.425 | 3389.249 | 3573.203 |
| 4 | 2998.863 | 2738.802 | 3492.071 | 3276.71 |
| 5 | 2925.23 | 3004.119 | 3273.594 | 3479.489 |
| avg | 2984.291 | 2868.9292 | 3474.8642 | 3432.4072 |
| min | 2840.173 | 2738.802 | 3273.594 | 3234.939 |
| max | 3301.569 | 3004.119 | 3792.898 | 3597.695 |
| standard deviation | 188.2100544 | 94.88267036 | 194.6917135 | 167.7762782 |

Figure 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (in seconds) | | | | | |
|  | n | MRA/1CI | MRA/2CI | MRB/1CI | MRB/2CI |
|  | 1 | 66.18104 | 73.8187 | 68.32704 | 74.18488 |
|  | 2 | 72.3424 | 71.41519 | 66.50785 | 71.22217 |
|  | 3 | 71.92933 | 72.86723 | 66.95432 | 71.08492 |
|  | 4 | 71.996 | 70.80138 | 69.77467 | 71.09494 |
|  | 5 | 71.13227 | 72.44375 | 67.73398 | 71.585 |
| avg |  | 70.716208 | 72.26925 | 67.859572 | 71.834382 |
| min |  | 66.18104 | 70.80138 | 66.50785 | 71.08492 |
| max |  | 72.3424 | 73.8187 | 69.77467 | 74.18488 |
| standard deviation |  | 2.573659328 | 1.190891101 | 1.279505137 | 1.329502118 |

Figure 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | n | mutations for rate a | mutation for rate b |
|  | 1 | 376 | 26 |
|  | 2 | 391 | 33 |
|  | 3 | 376 | 26 |
|  | 4 | 380 | 42 |
|  | 5 | 379 | 33 |
| avg |  | 380.4 | 32 |
| min |  | 376 | 26 |
| max |  | 391 | 42 |
| standard deviation |  | 6.188699379 | 6.595452979 |

Figure 3

My program outputs the path distance of the best found path, the best found path, the number of mutations occurred, and the time to run.

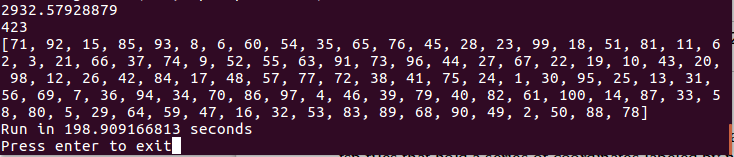


Figure 4 – Terminal Output

My program also displays a graphical representation of the path using the networkx and pyplot libraries as shown in Fig 5, which showcases the produced least distance path. These libraries are also used to display the Best Paths over generations with cost as the y-axis and generations as the x-axis as shown in Fig 6.

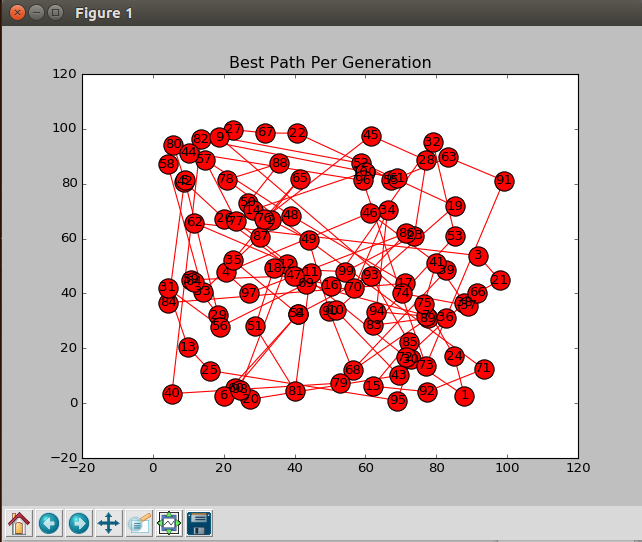


Figure 5 – Graph of best found path for generations 500 1% rate of mutation and Middle crossover

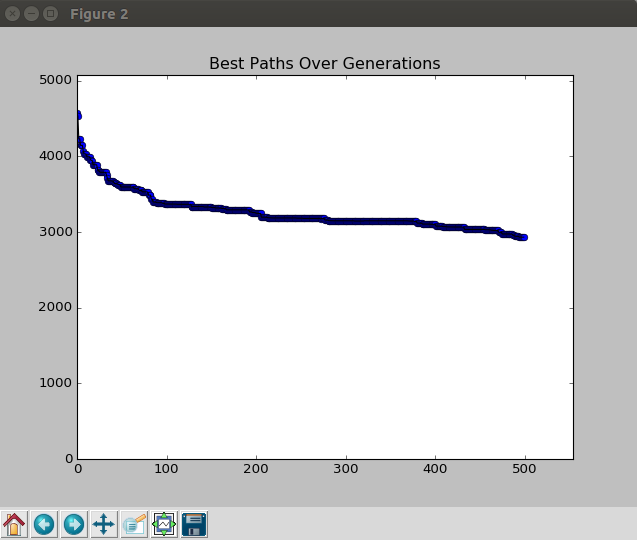


Figure 6 – 1% Middle Crossover

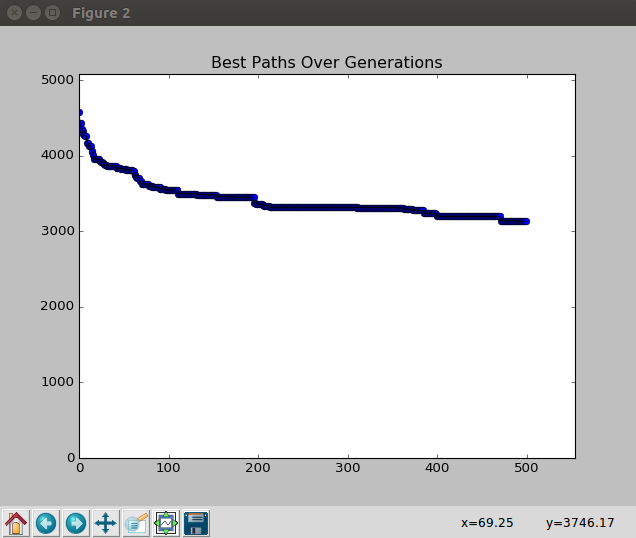
F

Figure 7 – 1% Right Crossover

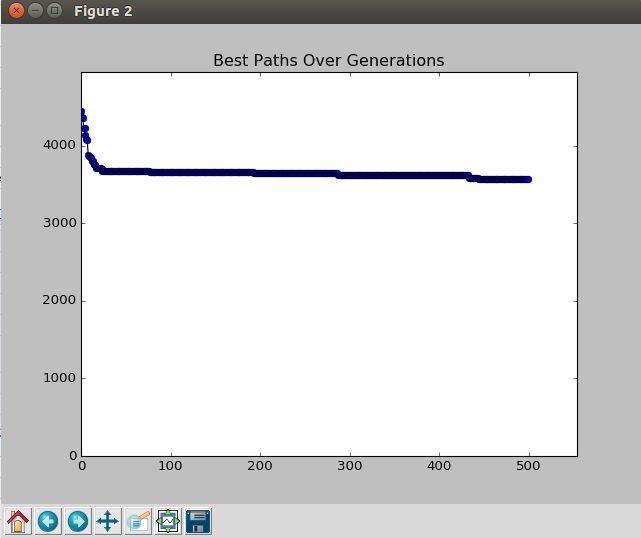


Figure 8 – 0.1% right crossover

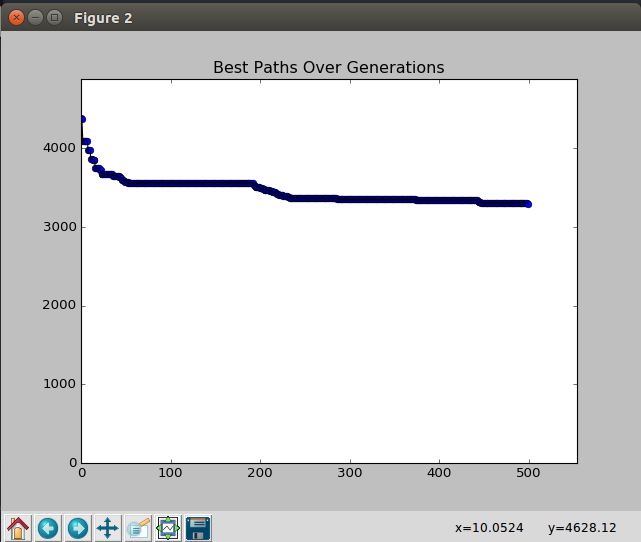


Figure 9 – 0.1% middle crossover

1. **Discussion**

The main thing that can be inferred from my results is that an genetic search algorithm is different than what we have used before. It has a far more complex approach to solving problems that is not necessarily straight forward. I can see how algorithms of this nature would be useful for situations where traditional search algorithms will not function effectively. This type of algorithm has the heavy drawback that it is not guaranteed to find the optimal solution and is slower, but it has possible use cases in a larger variety of fields.

The biggest problem that I encountered while performing this project was with repeating parents and children. I kept finding myself with situations where my population would converge very early on down to 1 or 2 different specimens causing infinite loops or errors. This held my progress up significantly. It also caused me to switch form roulette style selection to a tournament style and led me to a few other mistakes in my code. This gave me a greater understanding of what factors effected the GA .

If I were to perform this project again. I would start immediately with tournament style selection as it would hopefully prevent me from experiencing the amount of issues I was experiencing. Further I would also find a bit more diverse choice of variables. The ones I chose didn’t end up causing huge differences in final behavior and it would have been more interesting to see a more drastic change.

Using GA’s taught me a great deal of respect for the thought that goes into the more complex problems of computer science. I did not realize initially how complex it would be initially to come up with effective crossovers or mutations rather than just random ones. It also showed me that there are more indirect ways to solve problems and that they have their place. I could easily see other areas where genetic algorithms would be more easily used to learn about complex systems where as other simple searches don’t really teach the user much.

All in all I think that GA’s are an incredibly interesting problem solving technique. I believe that they have an incredible applicability to the world around us. While they may be slow I would say that their problem solving power can be easily underestimated.

1. **References**

<https://networkx.github.io/>

<http://matplotlib.org/users/pyplot_tutorial.html>

<http://www.sc.ehu.es/ccwbayes/docencia/kzmm/files/AG-TSP-operadoreak.pdf>

<http://user.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf>

<http://stackoverflow.com/questions/31815454/animate-graph-diffusion-with-networkx>

<http://stackoverflow.com/questions/28269157/plotting-in-a-non-blocking-way-with-matplotlib>