Genetic Algorithm: The Traveling Salesman Problem

Emily Strong

Final Draft: December 10, 2017

**Problem**

I created a genetic algorithm to find a solution to the traveling salesman problem. Colloquially, this problem is to find the shortest path for a route that visits each city in a set of cities. Translating this to a graph model, it can be restated as given a complete weighted graph, find the Hamiltonian cycle with the smallest weight.

**Implementation Design**

*Genetic code:* Each allele is an integer corresponding to a vertex. Each gene is an array of integers representing the order in which the vertexes are visited. While this may not be the most elegant way to represent the data, it simplifies mutation and crossing over and guarantees that every gene created will be a valid Hamiltonian cycle.

*Gene expression:* Each adjacent pair of alleles corresponds to an edge, as well as the last and first alleles. The edge weights are stored in a Bag data structure.

*Fitness function:* Since I am evaluating for the shortest path, the fitness function is the inverse of the net weight of the cycle. That is, f(g) = 1/w(g) where w(g) = Σ(edge weights). A cycle with a smaller weight will have a larger fitness value and vice versa.

*Mutation:* I am doing swap mutations of randomly selected alleles. For example:

Parent: 1 | 7 | 2 | 4 | 5 | 3 | 6

swap indexes 1 and 2

Child: 7 | 1 | 2 | 4 | 5 | 3 | 6

*Crossing Over:* Since a pure crossing over would result in an invalid solution, I must effectively repair the gene afterward. To accomplish this, I randomly select pairs of genes, randomly select an index, and perform a swap mutation based on the value the other parent has at that index. For example:

Parent 1: 1 | 3 | 2 | 4 | 5 | 7 | 6

Parent 2: 2 | 7 | 1 | 3 | 6 | 5 | 4

swap index 3 ↑

In Parent 1 we find the index of the value that has index 3 in Parent 2 and swap that with the value at index 3. We repeat this process for Parent 2.

Child 1: 1 | 4 | 2 | 3 | 5 | 7 | 6

Child 2: 2 | 7 | 1 | 4 | 6 | 5 | 3

*Evolution:* The population is stored as a linked list. For each generation, after mutations and crossing over are performed I calculate the fitness function for each individual

**Results**

**Conclusion**