Ethan Bills and Elijah Peterson

The TSP algorithm that we created was a genetic algorithm. First we will discuss every function at a high level and then explore each one further after.

To begin we have a method called start which is necessary at the beginning of the TSP algorithm because it randomizes the cities for the upcoming iteration. For the start method we initialize a map of cities by creating a list of cities and randomly distribute them.

We then have a method called totalOrder which basically tests how good a specific route is. After we have a method called select which is pretty self explanatory it selects the best value in the array.

Then we have a method named slice which essentially combines different routes to make a new route. We cross the genes of the two “parents” by checking to see if one is in the other. So we create a while loop and iterate through each parent to see if it’s in the other and add it if it's not.

```
Num = random num
Child = first[0: num]
Count = 0
Stop = length of the parent
        While count < stop
                If parent[count] is not in child
                        child.append(parent[count])
                Count++
Return child
```

The switch method is an attempt to add a mutation by swapping two random cities. Our switch method is a basic swapping algorithm.

For the distance method we find the euclidean distance between two cities using the Pythagorean theorem. We save these distances in an array.

In totalOrder we look up the distances from the distance array and add them to the total distance. The final answer is the reciprocal of the sum. This will be outputted at the end.

The select method is just a simple for loop through the actualSize. For the selection, we iterate through each of our parents and choose the best child. We default our best value to 0 and randomize, continually updating our best value.

```
Best value = 0
// defaults to 0
Count = 0
While count < parent:
        Pick random int
        If parent[int] > best value
                Best = int
                Best value = parent[best]
```

Count ++

Our last method is the TSP method itself where all of our helper methods are called and the actual best path gets decided. The TSP method essentially uses all of the helper methods in completing its formula. It first initializes the cities with the start method. It then selects between distances and then adds them to the path with select and slice. Then it makes some switches based on the mating pool and mutation in an attempt to mutate and randomize in order to find the best solution. It then adds those mutations to the path if they make for a better path.Finally at the end is the code which runs the methods. It gives every variable needed to complete the problem. The following code shows the instantiation of the parents that will be used in selection.

```
While (time < timecap):
    While len(city) < poolsize:
        create  parent 1
        create parent 2
```

After much research and testing, we landed on a 10% mutation rate with a size of 100 and a pool of .5. These numbers were picked to be sure we did not search too little or too much of the problem space. As a too low mutation rate fails to find the optimum and a too high mutation rate leads to overcoverage and a lower change on converging. To actually write to the output file, we opened the file as specified in the system arguments and called the actual TSP function. The function continually outputs the most fit node until there are none left.