**Analysis of Algorithms**

**Assignment 4**

**Ryan Farber, Zach Singleton, Huarui Wu**

**Pseudo-code**

To solve the euclidean traveling salesman problem we decided to use the 2 opt local search algorithm that was first proposed by Croes in 1958. The idea behind the 2 opt optimization algorithm is to take a route that crosses over itself and reorder it such that you identify the shortest distance between two points. The way the algorithm works is to select two edges and then swap any two points from two edges in order to find that shortest distance. The problem with this approach is if you are swapping a given two points, you end up having two cycles after connecting every single point, as opposed to one --- This makes it difficult to confirm the validity of the output of the algorithm.

**2-Opt Optimization Steps:** 1) Find two edges and their endpoints. 2) Swap endpoints.

**2-OPT heuristic:**

1. T = some starting tour  
2. noChange = true  
3. **repeat**  
4. **for all** possible edge-pairs in T  
5. T' = tour by swapping end points in edge-pair  
6. **if** T' < T  
7. T = T'  
8. noChange = false  
9. **break** // Quit loop as soon as an improvement is found  
10. **endif**  
11. **endfor**  
12. **until** noChange  
13. **return** T

For each pair of edges, there are also two ways to swap endpoints.

For example, given edges E1 = (A1,B1) and E2 = (A2, B2)

1) swap A1, and A2

2) swap A1, and B1

The two swaps will create different resulting edges, which will affect the length of the tour. So for each possible edge-pair, two swaps are made. Each are compared against the current minimum tour length, to determine whether it is an optimization.

A swap could also create a nonvalid tour. This happens when a swap of endpoints creates two disconnected cycles in the graph. If this happens, it violates the TSP. Therefore, after finding a swap that shortens the tour, we also check to see if the new tour created by the swap is valid. This is done with a simple algorithm looping through the tour.

**TourValidator:**

current\_city = first\_city

count = 0

while (current\_city is not visited):

current\_city set visited

current\_city = next\_city

count++

if (count < tour length)

return false

else

return true

**Analysis**

For smaller problems our current algorithm setup consistently yields solutions with a very high level of efficiency. However, for larger problems our algorithm has ineffective in generating viable paths in a reasonable amount of time. This is due to the fact that the algorithm is O(n^(2+)). Over the last week, we have been able to make improvements in our java code that improve our efficiency over larger problem sets. However, this is not a good algorithm for very large input sets.

That said, given the pattern of increased data set size followed by efficiency improvements, we have reached a conclusion about this algorithm. Based on our observations, we can say with confidence that no guarantees can be made on the efficiency of a 2-Opt solution, only that the longer time spent the better the solution will likely be. Additionally, we have observed that our tour generates crossovers where the path crosses itself.

**Conclusion**

Based on our observations, and reading that we’ve done about other approaches, the traveling salesman algorithm works best on small to mid-sized data sets (100 or less data points). The 2-opt optimization does not produce the optimal results but is much faster than any algorithm that does.

**Compile and Execute Procedures**

1) compile TSP.java normally:

javac TSP.java

2) execute TSP with java passing in input file name as the first argument

java TSP example\_input.txt