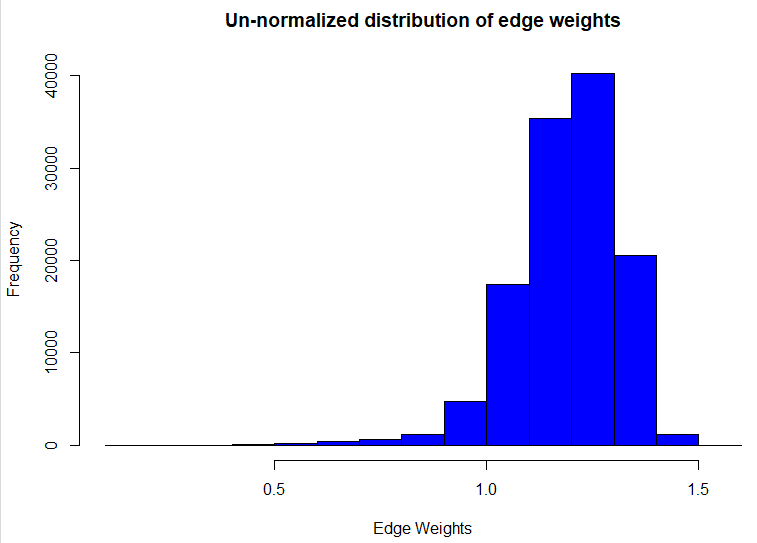
**ECE 232E Project 4**

Group members: Kushagra Rastogi (304640248)

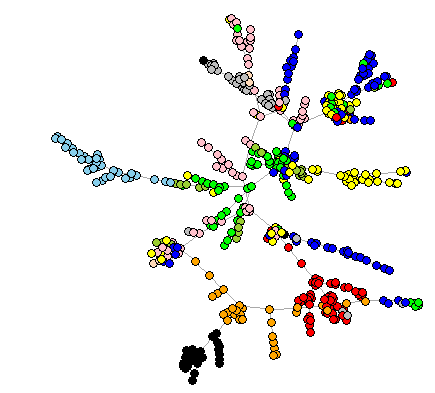
Jonathan Lee (104840173)

**Part 1: Stock Market**

1. The formula represents a Pearson correlation coefficient. By Cauchy-Schwartz inequality, we can show that the upper bound is 1 and the lower bound is -1. We use log-normalized returns it helps in reducing the skewness of the data. taking the log of the returns reduces the variance in the return which gives us a better idea about the relative changes in stock prices and returns.
2. The histogram is shown below.



1. The MST is shown below.



Yes, there is a pattern in the MST. Nodes in the same sector tend to be connected in the MST and nodes in different sectors tend to be disconnected. This can be seen from the plot above since nodes with the same color are clustered together and these clusters are pretty homogenous. This is because nodes in the same sectors are highly correlated with each other while nodes in different sectors are poorly correlated with each other.

1. When , then . When , then . The difference between the two is large. This is because the first method exploits the MST cluster structure and only looks at sectors of neighboring nodes which is advantageous since stocks belonging to the same sector form Vine clusters. On the other hand, the second method calculates a naïve probability estimate by calculating the probability a node belongs to a sector so it considers all the nodes in the graph.
2. The MST for weekly data is shown below.



The nodes belonging to the same sector are clustered less closely together and less well-separated than the MST in question 3. The similarity is that in both MSTs, nodes in the same sector are clustered together. However, the clustering is better with daily data rather than weekly data. Also, the MST in this question seems to have a more circular structure than the MST in question 3. We are unsure whether this is due to the plotting function in R or the difference in data.

**Part 2: Let’s Help Santa!**

1. Nodes = 2649 and edges = 1004955
2. Since the dataset does not contain street address, we present the pairwise distances between nodes as suggested on Piazza.

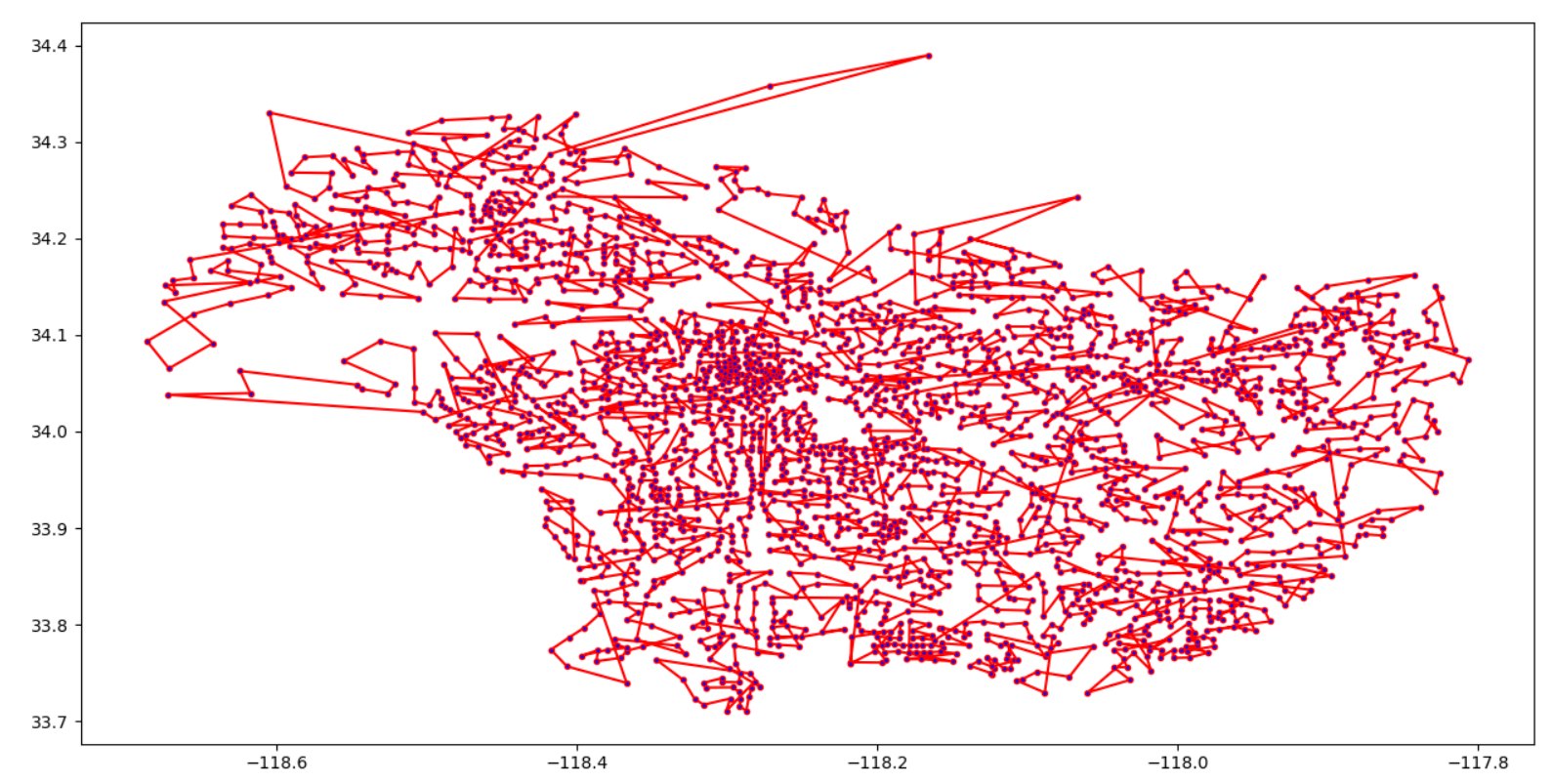
|  |  |  |
| --- | --- | --- |
| Node 1 | Node 2 | Distance between nodes (miles) |
| 3 | 13 | 0.6182092075476802 |
| 6 | 2015 | 0.620388615817233 |
| 12 | 22 | 0.8631734769272965 |
| 17 | 19 | 0.5507395509049474 |
| 21 | 22 | 0.7779597729281168 |

The distances between nodes is around 0.7 mile on average. This makes intuitive sense since the minimum spanning tree consists of edges that combine to give the lowest weight/cost possible. This means edges with the lowest weights will be chosen. Since, the weights represent the mean travel time, a low weight means the commute between two nodes is small and hence the distance between the nodes is small. Thus, the MST consists of nodes that are close to each other in terms of distance.

1. We are assuming that the randomly sampled triangles are not unique. Percentage = 92.7 %
2. We found the 1-approximate tour using the following algorithm:
3. Create a MST
4. Create a multigraph of T
5. Find an Eulerian walk of T and embedded tour
6. For each edge in the embedded tour, search it in the MST. If the edge does not exist, then find the shortest path between the nodes that contain the edge.
7. Compute the cost as the sum of the weights of the edges in the embedded tour. If the shortest path is computed, then the cost of the edge is the cost of the shortest path.

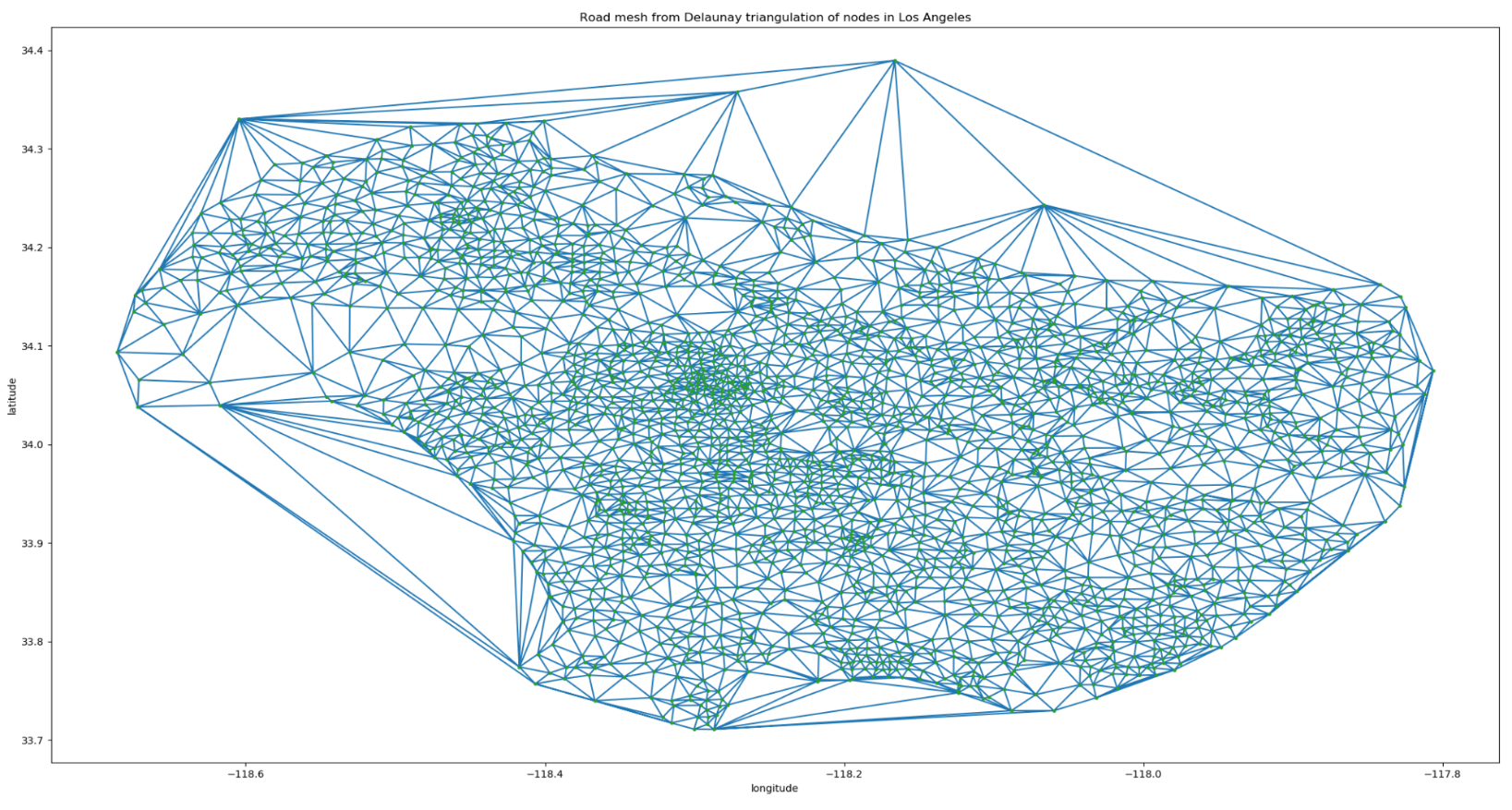
We know that MST cost < Optimal TSP cost < Approximate TSP cost < 2\*MST cost. Since the MST cost is a lower bound on the optimal TSP cost, we use that in the calculation for . From the Eulerian walk we found, we get MST cost = 269084.54500000016 and 1-approximate tour cost = 421489.3149999998. Then we get –

Thus, 1 < Optimal TSP cost < 1.56638 < 2.

1. The trajectory is plotted below.

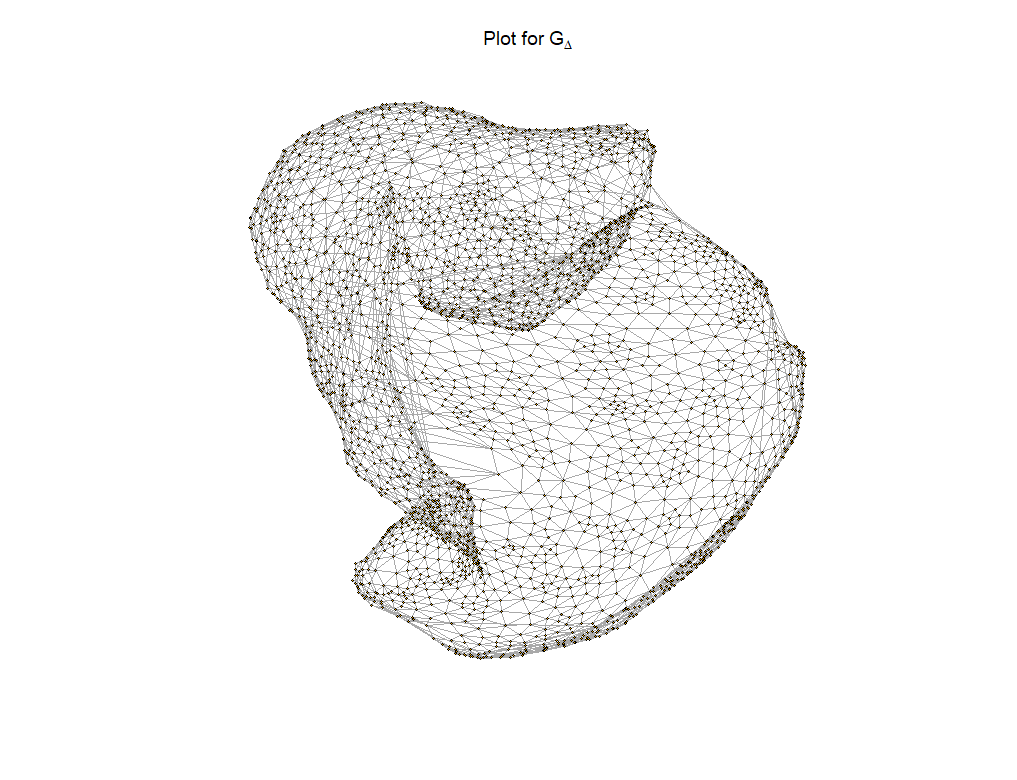
The blue dots represent the cities and the red lines represent the paths. The trajectory makes intuitive sense since most edges are not overlapping. However, there are overlapping edges since the original graph is not fully connected. The original graph is only connected which means that we have to find shortest paths between two nodes if an edge doesn’t exist between them. These shortest paths can cause overlapping edges. That being said, the 1-approximate algorithm only produces an approximate algorithm and hence the trajectory is correct.

**11.** We now take a deeper look into analyzing traffic flow, that is, finding the maximum traffic that can flow between two locations. The metric we will use for traffic flow is cars/hour and is computed in the later sections. In order to analyze traffic flow, the generation of a roadmap of the entire dataset is required. The algorithm we will be using is the Delaunay triangulation method, and it will be applied to all of the nodes in the giant connected component that was processed in the earlier sections. The resulting road mesh from Delaunay triangulation is shown below:

****

The road mesh generated above seems to depict road structures of Los Angeles, although there are a few exceptions. For instance, there are many edges on the road mesh that do not actually exist. This is due to the fact that Delaunay’s triangulation method generates additional edges such that every triangle fit inside a circumcircle.

The graph shown below corresponds to the road mesh of Los Angeles, whose nodes are locations, and edges are the mean travel times from one location to another.



**12.** With the graph of , the next step in the analysis of traffic flow of Los Angeles is to calculate the traffic flow of each road in the road mesh, that is, find the maximum capacity of the number of cars per hour on each road/edge. Assumptions are given below:

1. Each degree of lat/lon is approximately 69 miles

2. The length of each car is about 0.003 miles

3. Each car maintains a safety distance of 2 seconds to the next car

4. Each road has 2 lanes in each direction

5. No traffic jam

The derivation for calculating traffic flow for each road is as follows:

1. Total distance = [ (Vcar \* mean travel time) / 3600 ] miles
2. Car length + gap length = [ (0.003 + (Vcar \* 2)/3600) ] miles
3. Number of cars on road = [ 2 \* ((Vcar \* mean travel time) / 3600) / (0.003 + Vcar / 1800) ] cars

= [ (Vcar \* mean travel time) / (5.4 + Vcar) ] cars

1. Traffic flow = 3600 / mean travel time \* (Vcar \* mean travel time) / (5.4 + Vcar)

= [ (3600 \* Vcar) / (5.4 + Vcar) ] cars/hr

Where Vcar is the average velocity of the car on each road. We use the distance equation d = v \* t to calculate the total distance form one location to another, where the total distance is approximated through the Pythagorean theorem of the latitude/longitude coordinates between the two nodes. The car length and gap length are computed as one unit. Afterwards, we find the number of cars on the road by taking the total distance of the road and dividing it by the car length + gap length. Note that we multiply this quantity by 2 given that each road has 2 lanes in a particular direction. This value is then divided by the mean travel time to get the final traffic flow value. Note that we divide by 3600 for the mean travel time because the units of time in terms of traffic flow is in hours, but the mean travel time data is in seconds.

We then apply this equation to all of the edges and their endpoints to find the traffic flow of all the roads in Los Angeles that were generated by Delaunay’s triangulation algorithm.

**13.** In this section, we explore the concept of maxflow from a source node to a destination node. The source and destination coordinates considered are Malibu: [34.04o, -118.56­­o] and Long Beach: [33.77o, -118.18o], respectively. However, upon viewing the coordinates via Google Maps, we notice that the coordinate of [34.04o, -118.56­­o] was actually located in Palisades, not Malibu. Therefore, we instead use [34.0259, -118.7798], which are the coordinates of Malibu provided by Google. To find the corresponding Census Tract Dataset coordinates of these two locations, we selected the coordinates that were closest to Malibu and Long Beach in Euclidean Distance (L-2 norm):

Source coordinate (Malibu) -> [34.03809095384618, -118.67225393333337]

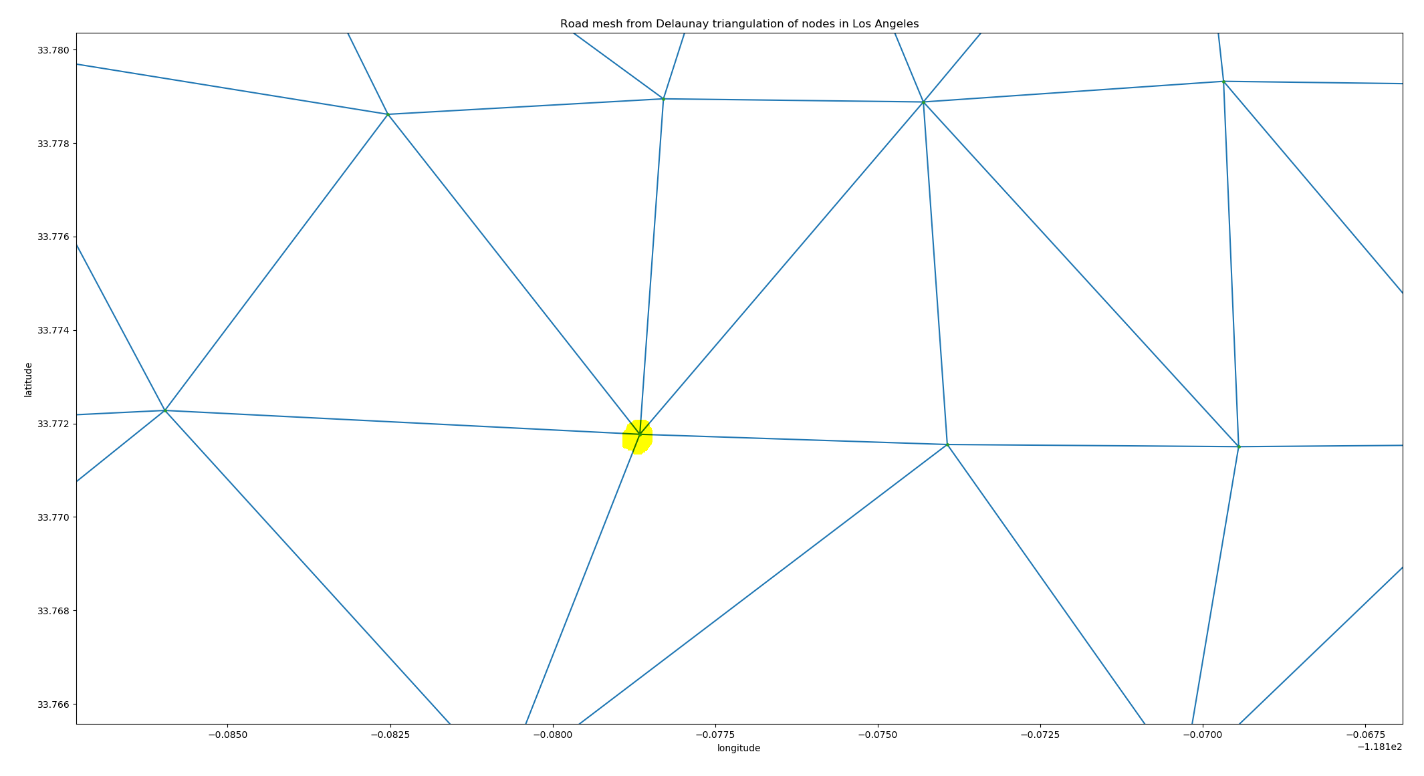
Destination coordinate (Long Beach) -> [33.771767700000005, -118.17865950000001]

Note that we will be using the two coordinates depicted above for sections 13 to 15. Once the coordinates have been finalized, applying the maxflow function from Malibu to Long Beach produced a value of 13095.49 cars/hour, implying that the maximum capacity of traffic flow from Malibu to Long Beach is approximately 13095 cars every hour. The number of edge-disjoint paths between the two coordinates is computed to be 6.

From the road maps of the nodes of Malibu and Long Beach depicted below (highlighted in yellow), respectively, it is easy to see the degrees of each node:

****

*Malibu Roadmap*

****

*Long Beach Roadmap*

Malibu seems to have 6 outgoing edges/roads, whereas Long Beach has 6 incoming edges/roads. Coincidentally, the out-degree of Malibu and in-degree of Long Beach are the same; therefore, the number of edge-disjoint paths is 6. However, if Long Beach, were instead to have 5 incoming edges, the number of edge-disjoint paths would decrease to 5; since the edge-disjoint paths from Malibu to Long Beach must not share any edges, one of the outgoing roads from Malibu will eventually converge with another road from Malibu in order to traverse one of the 5 roads toward Long Beach. In other words, the minimum of the outgoing edge count from the source node and the incoming edge count from the destination node will give the number of edge-disjoint paths.

**14.** As of now, we have been processing the entirety of . However, recall that the road mesh contains unreal roads of Los Angeles due to the consequence of Delaunay’s triangulation method mentioned in the previous section. It can be observed from the road mesh in section 11 that there are some roads that extend beyond the coastline, namely, along the beaches of Los Angeles. For instance, there is an unreal road connecting Malibu to the Agua Amarga Canyon over the ocean. Our approach in removing such instances is by utilizing Google Maps to follow a coastline path from the centroid coordinates starting from Malibu Pier and ending at Cabrillo Beach. This path is sufficient in partitioning the graph from the unreal edges that extend beyond the coastline. In other words, any edge that falls below/south of the generated path will be removed from . The path is shown below.

**Malibu Pier: (34.03809095384618, -118.67225393333337)**

**Big Rock: (34.03975774911659, -118.61724170318024)**

**N Marquette St: (34.043940760683725,-118.54279579914532)**

**Palisades Beach: (34.022316000000004 ,-118.51373178260872)**

**Venice Beach: (33.98657492857143, -118.4737257142857)**

**Toes Beach: (33.955592483870966, -118.45335495967741)**

**Dockweiler Beach: (33.932255009615375, -118.4384201153846)**

**El Porto Beach: (33.90297007407407, -118.42332207407408)**

**Hermosa Beach: (33.870485456647394, -118.40540964161855)**

**Seaside Lagoon: (33.84558560119048, -118.39609398511907)**

**Redondo Beach: (33.83420097142857, -118.39177008571423)**

**Avenue H Beach Lifeguard Tower (33.82029325301204, -118.38821546987946)**

**Torrance Beach: (33.81164102590671, -118.3843589067357)**

Note that the red sub-path path covers the concavity of the coast from Malibu to Torrance. We then proceed with the same path to form a convex sub-path along

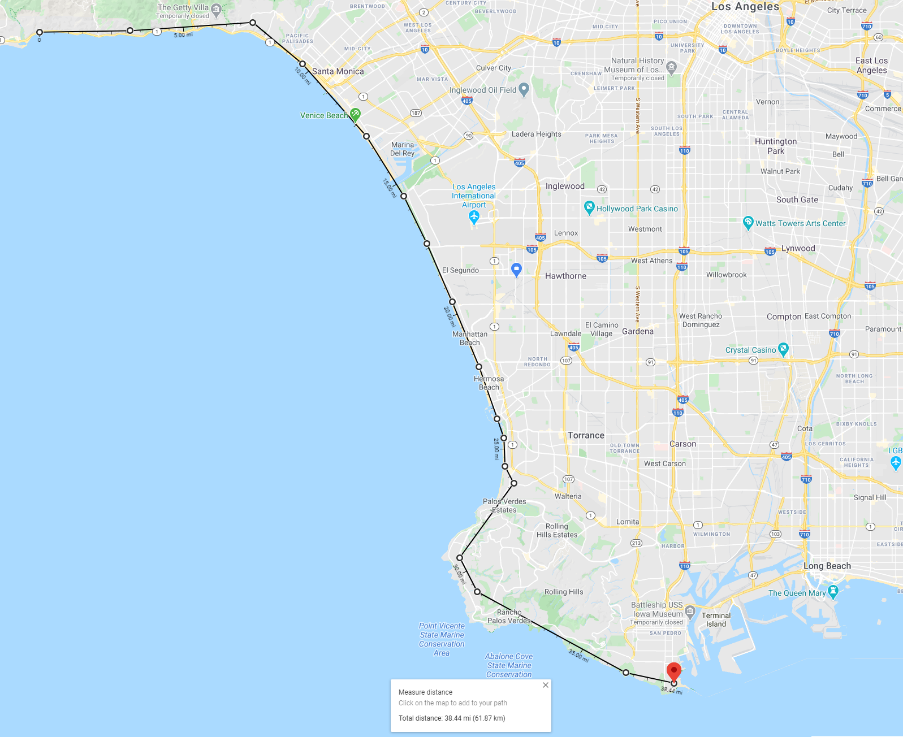
**Agua Amarga Canyon: (33.774103109090916, -118.41714115454548)**

**Los Verdes Golf Course: (33.75689578693178, -118.40648371022725)**

**White Point Beach: (33.7146981025641, -118.31700520512823)**

**Cabrillo Beach: (33.710754580645165, -118.28703954032257)**

Adding the tail of the red sub-path and the head of the green sub-path give us the entire path of real roads from Malibu to Cabrillo Beach. The aforementioned path is shown on the next page via Google Maps:

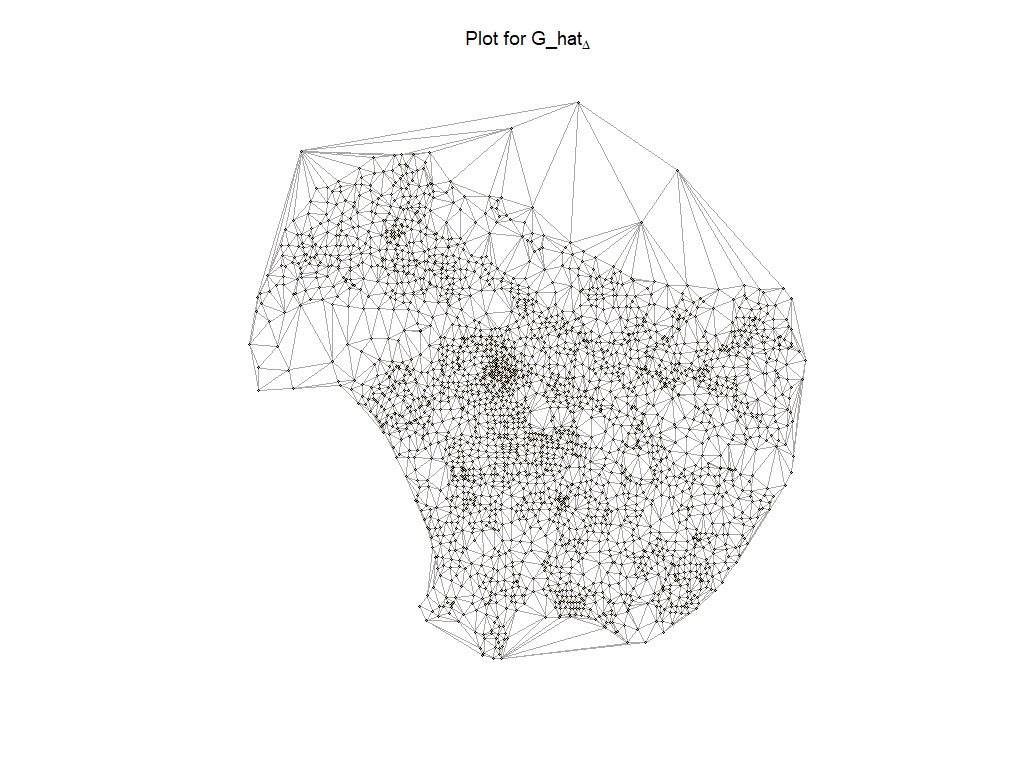
****

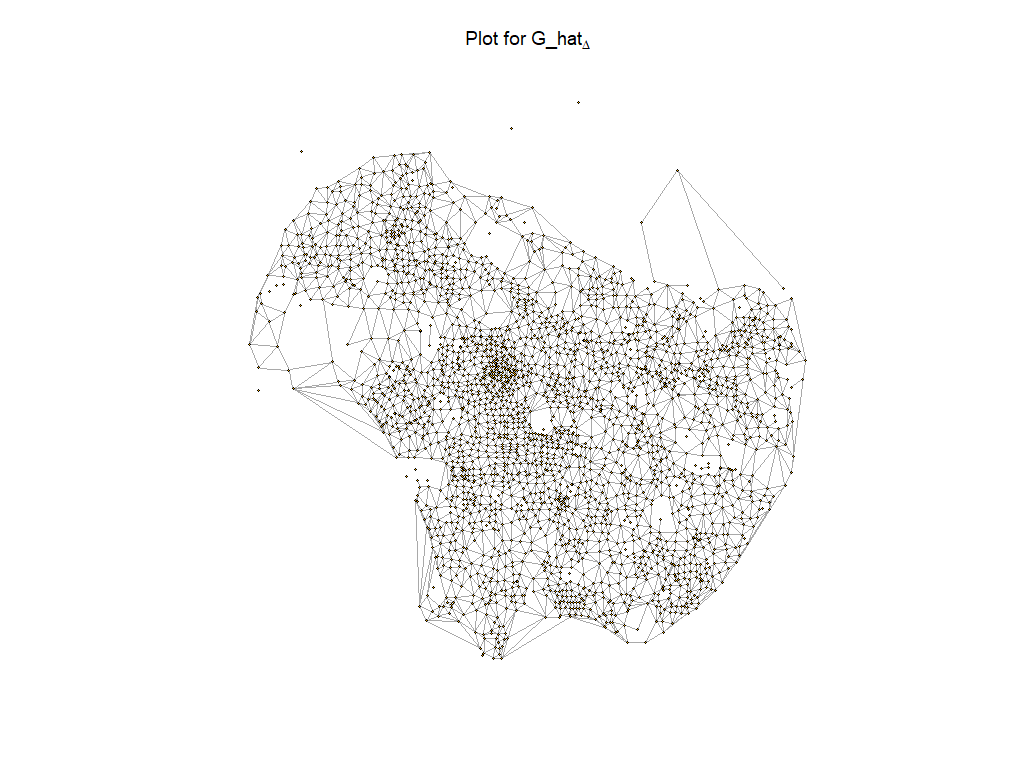
Using the information above, we can determine that the roads generated south of this path are unreal as it reaches beyond the coastline into the ocean. There are two cases we analyzed in remedying this issue:

1. Solely eliminating the edges off the shoreline
2. Eliminating the edges off the shoreline along with other unreal edges through thresholding

It is important to note that real roads/edges are neglected during the removal process. As reference, the original plot from triangulation is shown below, followed by the first case and second case eliminations:







**15.** Looking at the first case, we only remove the unreal edges below the coastline. As shown, we see the source node of Malibu noted by the red circle, and Long Beach somewhere within the green circle. Upon calculating the maxflow, we found the value to remain the same, that is, 13095.49 cars/hour. The reasoning behind the fact that the maxflow stayed the same is because the sum of capacities of the remaining outgoing edges of Malibu is still greater than 13095.49 (the remaining outgoing roads from Malibu can still handle traffic flow greater than 13095.49 cars/hour). Moreover, there still exists many paths that can be taken from Malibu to Long Beach as depicted above, with capacities of each road varying from around 600 to 2500 cars/hour. Hence, eliminating the roads off the shoreline did nothing in decreasing the maxflow, as the amount of flow going out of Malibu can take many other paths to reach Long Beach. This is analogous to filling water from one source to a sink, where the removal of an edge is similar to clogging up a particular pipe/edge: if one pipe is clogged, the waterflow can always traverse through another path of available pipes to reach its destination. However, the calculation of the number of edge-disjoint paths from source to destination seems to have decreased from 6 to 4. The result of this comes from the elimination of 2 unreal edges that were connected to Malibu below the coastline, i.e., Malibu -> Agua Amarga Canyon and Malibu -> Los Verdes Golf Course. Due to this removal, the out-degree of Malibu is now 4, while Long Beach remained the same at 6. As mentioned previously, taking the minimum of the two degrees is equivalent to finding the number of edge-disjoint paths between Malibu and Long Beach. Hence, the number of edge disjoint paths is equal to 4.

Looking into the second case, we now apply a threshold such that it not only removes the roads below the coastline, but also some roads along the hills of Topanga and more. Our approach was to take the minimum mean travel time of the roads below the coastline and set that as the threshold. Any unreal edge that exceeds this threshold will be eliminated from the original graph.

It can easily be seen from the graph generated from case 2 that Malibu, depicted in the red circle, no longer has any outgoing edges or roads, which implies that all of the roads pointing to Malibu that were generated by Delaunay’s triangulation algorithm are all in fact unreal roads. Applying the maxflow of this new graph from Malibu to Long Beach, which again lies within the green circle, is 0 cars/hour. The reasoning behind a 0 traffic flow is because Malibu is now an isolated node after the removal of unreal edges above the set threshold. With no outgoing edges from Malibu to the rest of the road mesh, the total amount of flow it can output to Long Beach is 0.

The number of edge-disjoint paths is also 0, and it can be shown by taking the minimum of the out-degree of Malibu and the in-degree of Long Beach.

**Part 3: Define your own task**

In this part, we explore the feasibility and examine the accuracy of different approximate algorithms to solve the TSP problem on the World TSP dataset. We compare the performance of these algorithm with the performance of the 1-approximate algorithm on the World TSP dataset.

There are several algorithms that have been developed in an effort to closely approximate the optimal solution of TSP problems:

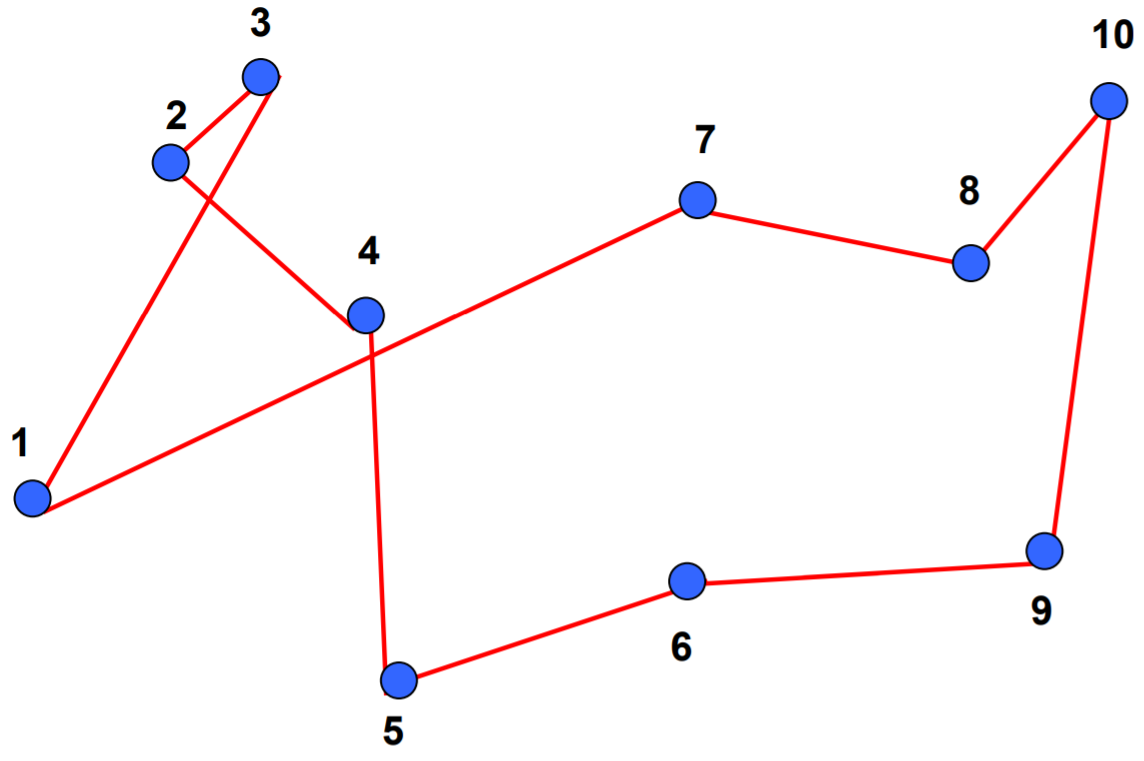
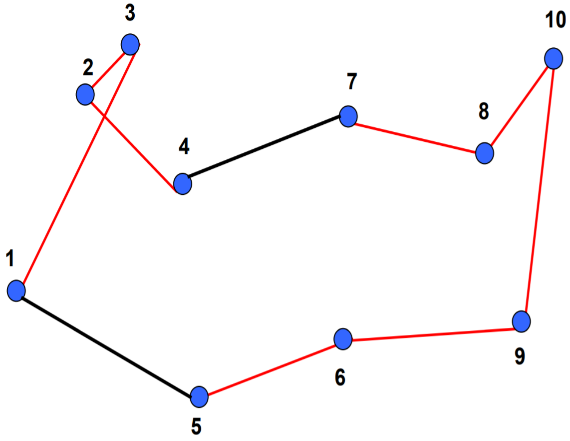
1. Nearest neighbor (greedy algorithm)
2. Lin-Kernighan (LK) heuristics
3. Randomized improvement (tabu search, 2-opt, 3-opt, cross entropy, ant colony, simulated annealing)

Along with approximation algorithms, there do exist exact algorithms that find the optimal solution; however, the computation time limits the size of TSP datasets to a dramatically low number (around 300-500 cities):

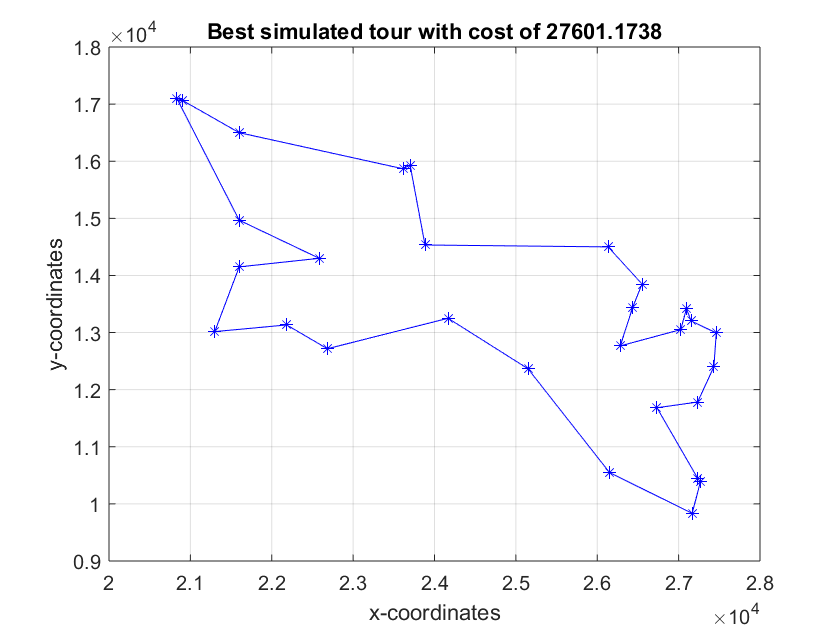
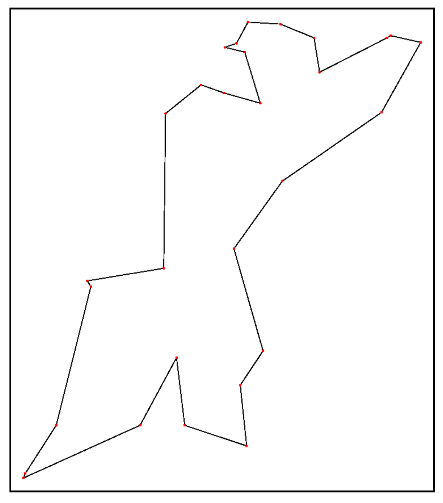
1. Branch-and-bound
2. Progressive improvement techniques (Linear programming)
3. Branch-and-cut

For the purposes of our task, we will be taking a look at approximate algorithms, specifically, simulated annealing and the 2-opt heuristic, in solving TSP datasets provided by the University of Waterloo (<http://www.math.uwaterloo.ca/tsp/world/countries.html>).

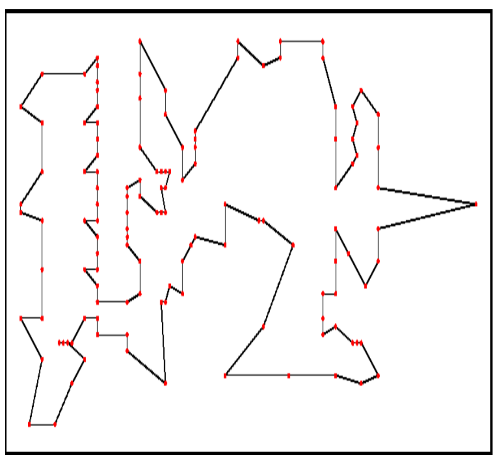
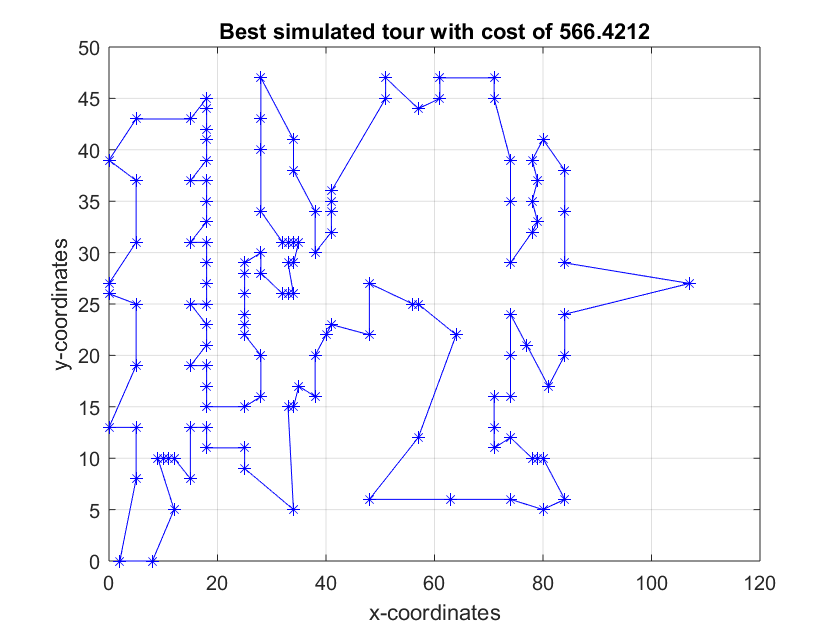
The first algorithm we implemented was 2-opt. The idea behind this local search algorithm is to repeatedly improve on the cost of a Hamiltonian path through removing any crossings of the path. How this is done is by first generating a random tour and then applying a node inversion technique on the scrossing of edges:



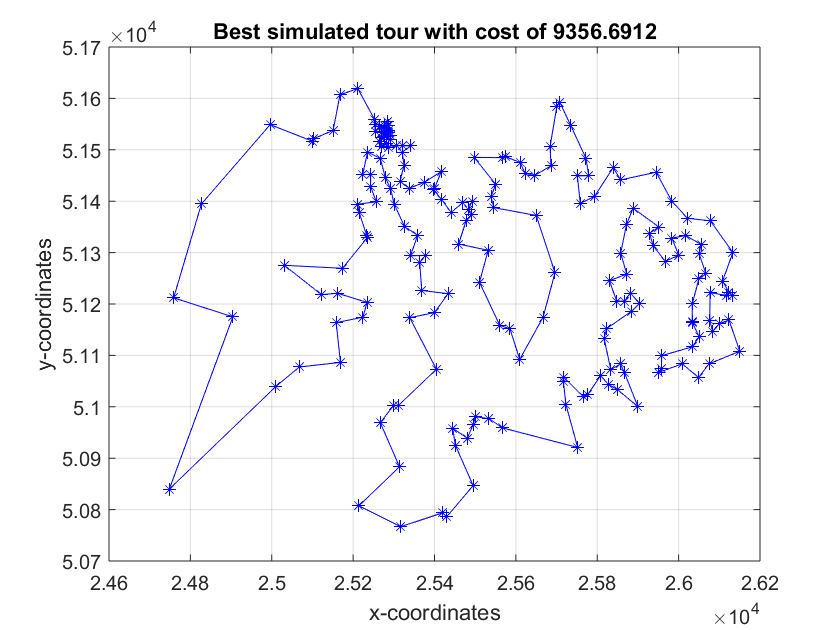
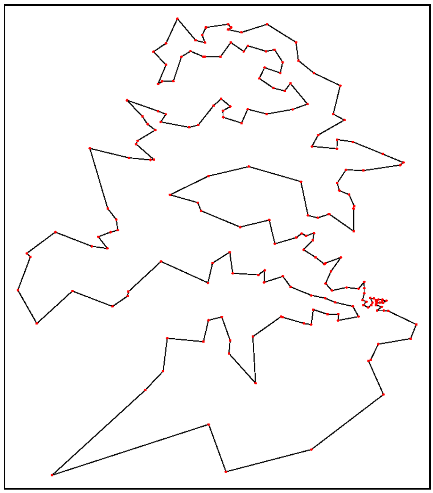
In essence, the inversion process changed the left tour of 1->3->2->4->5->6->9->10->8->7->1 to the right tour of 1->3->2->4->7->8->10->9->6->5->1. This process is done repeatedly until no other crossings/improvements can be made to reduce the cost of the tour. Running this algorithm is fairly quick with a simulation runtime of < 2 sec for TSP problems <200 nodes. Several results are shown below comparing the 2-opt tours (left) to the optimal tours (right).



*Western Sahara: 29 cities, optimal length of 27601*

**

*XQF131 VLSI instance: 131 nodes, optimal length of 564*

**

*Qatar: 194 cities, optimal length of 9352*

Notice that the 2-opt algorithm at lower count cities generates a fairly low error percentage of approximately 0-2% from the optimal solution. The main intentions of producing these results, however, was not actually gauge on the speed in which it finds a solution. For the country of Qatar, 2-opt was able to find a solution in about 1.21 sec. Having knowledge of its speed and accuracy, we will later scale the TSP problem up to 8000 nodes and compare its performance with the 1-approximate algorithm and the method of simulated annealing. Note that we wrote the 2-opt algorithm along with simulated annealing, which we will discuss in the latter half of this section, in Matlab rather than Python due to its speed.

For the next algorithm, we take a look at the simulated annealing algorithm. The motivation behind implementing this method, similar to 2-opt, is its speed and small deviation from the global optimum. However, there are far more precautionary measures that need to be taken in order for simulated annealing to perform at its peak, which equates to better performance than the 2-opt heuristic.

To start, we first had to understand the idea behind simulated annealing, which was first inspired by the concept annealing in metal (metallurgy): the altering of the properties of a material through heating the metal to an extremely high temperature followed by an extremely slow cooling rate to a specific temperature. One purpose of doing this is to enhance the electrical conductivity of a metal.

The same concept applies to simulated annealing in the traveling salesman problem, where its objective is to improve on the length of a Hamiltonian path by a balancing act of exploration and exploitation. We begin with an arbitrary initial tour and a high starting temperature which is used to “simulate” the concept of annealing by attempting to explore for better paths of the tour. The higher the temperature, the higher probability the algorithm attempts to explore, that is, favoring moves that that produce a higher cost than a lower cost. The motivation of doing this is primarily to escape from premature convergence to a local optimum that is worse than the global optimum. The cooling schedule is described as follows:

* + - *Metal heated to*
    - *Metal slowly cooled to*

with an acceptance probability function P that does one of two moves:

* + - *Higher temperature P favors seeking greater cost*
    - *Lower temperature P favors seeking minimal cost*

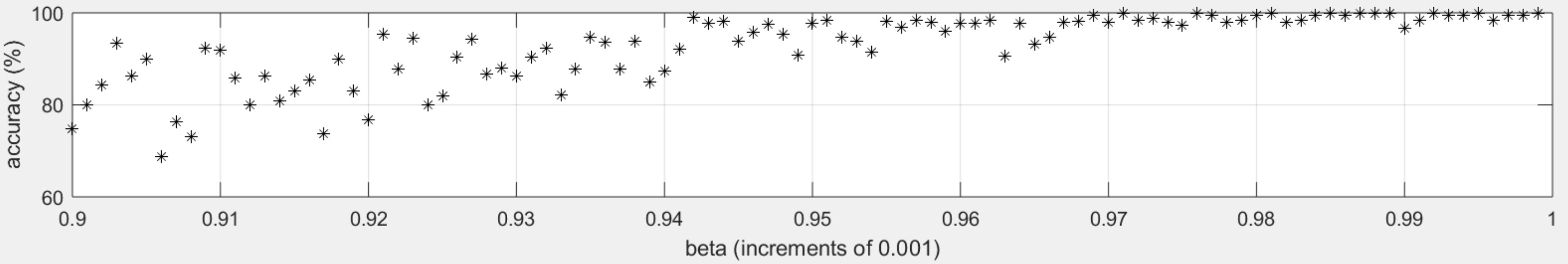
As the temperature cools down, the algorithm will start favoring moves that result in a tour with a lower cost than the original tour. There are three “candidate” moves that we use to perturb the tour to another valid tour (Hamiltonian cycle):

1. Randomly invert a sub-path of the original tour (same move as the 2-opt algorithm).
2. Swapping the positions of two nodes in the path of the original tour.
3. Taking one node and slipping it into the position of another node.

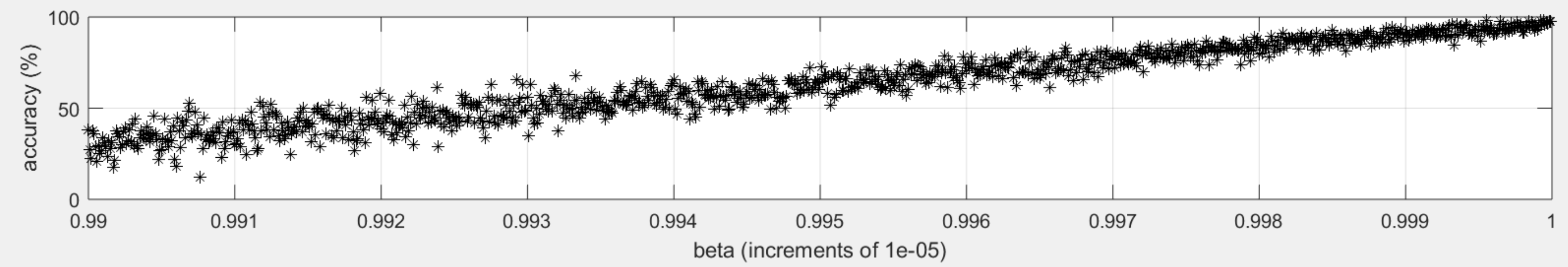
If the random draw against the probability acceptance function P is < P, we accept the move with

the greatest cost of the 3 “candidate” moves if and only if it generates a cost greater than the original tour (exploration). Otherwise, we accept the move with the lowest cost of the 3 “candidate” moves if and only if it generates a cost less than the original tour (exploitation). The only way a tour remains unchanged during an iteration of simulated annealing is when the greatest “candidate” cost during exploration is lower than the cost of the original tour or when the lowest “candidate” during exploitation is higher than the cost of the original tour. The algorithm will stop once the temperature has cooled to its final temperature *TF.*

Shown below are accuracies of the method of simulated annealing when varying the cooling rate for the same three TSP problems used for testing the speed of 2-opt:



*Western Sahara: 29 cities*



*XQF131 VLSI instance: 131 nodes*

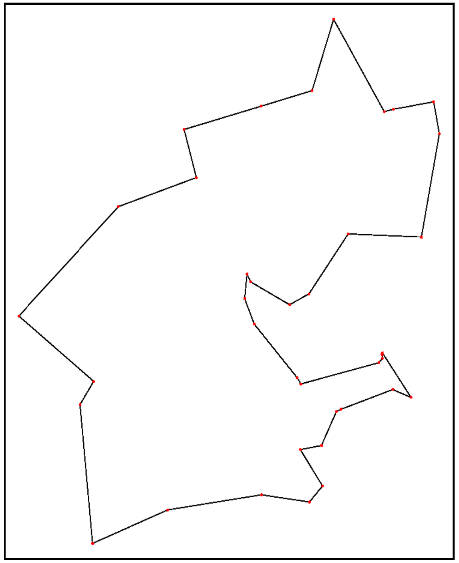


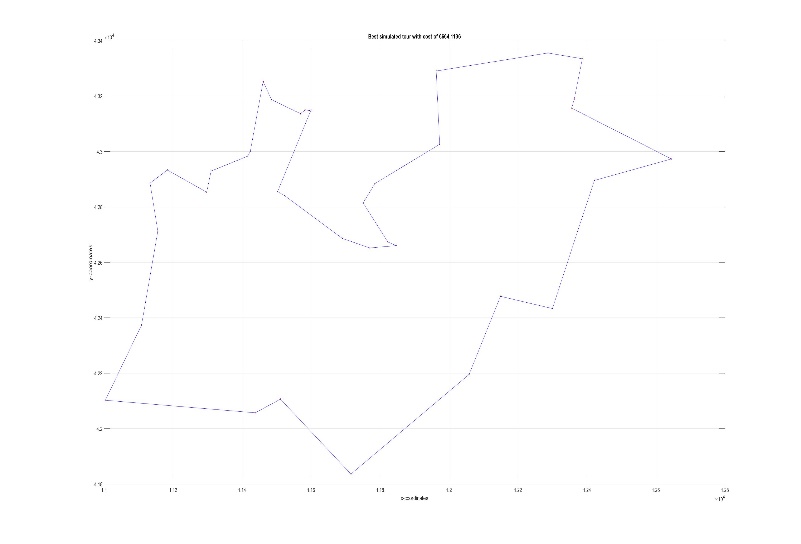
*Qatar: 194 cities*

It is easily seen above that as slowly approaches 1, which implies the cooling process is slower, the accuracy tends to 100%. This, however, is a tradeoff between speed and accuracy. If is close to 1, there are more iterations to go through before cooling to the final temperature (increasing runtime), but more exploration and exploitation allows it to find a better local optimum. The initialization of the starting and final temperatures are also important parameters to consider when fine tuning the performance of the tour. In the next section, we will compare the performance metrics of the 1-approximate algorithm, the 2-opt algorithm, and the method of simulated annealing.

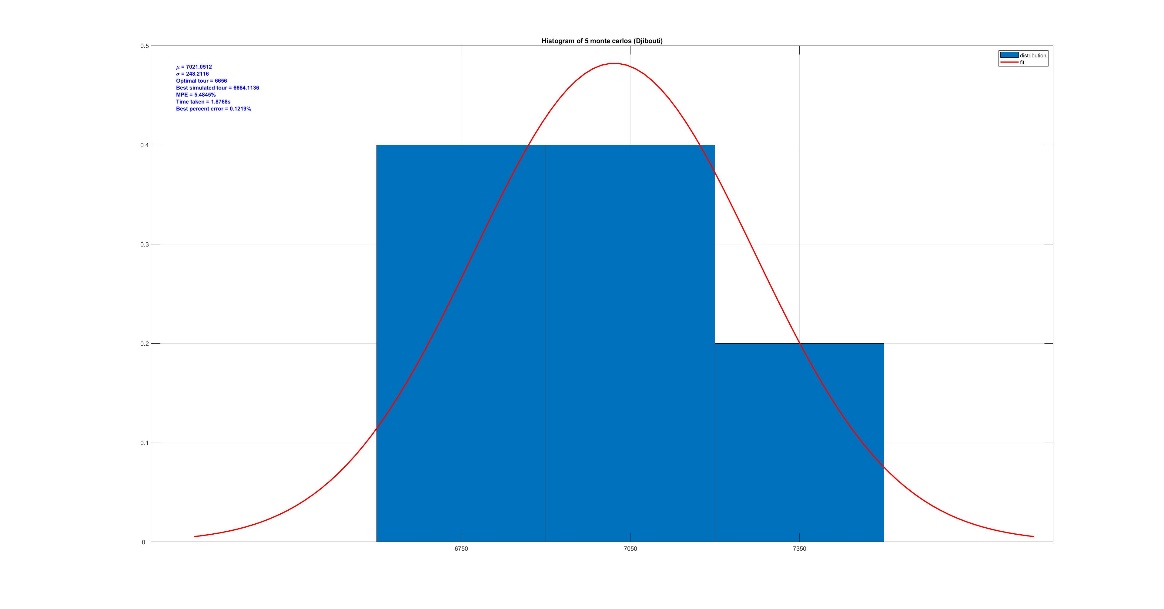
We now test the accuracies of the 1-approximate, 2-opt, and simulated annealing algorithms on a set of 5 countries generated from the World TSP dataset:

1. Djibouti (38 cities)
2. Oman (1979 cities)
3. Canada (4663 cities)
4. Tanzania (6117 cities)
5. Egypt (7146 cities)

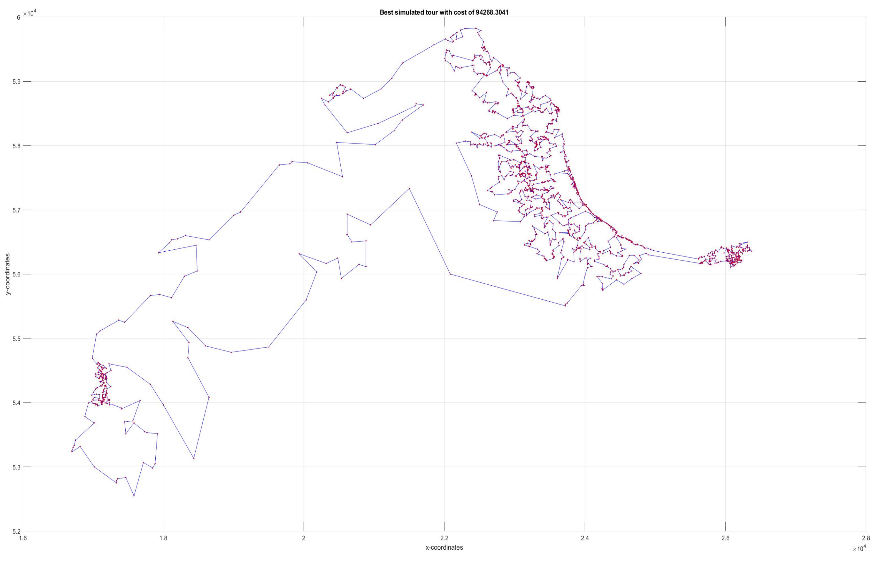
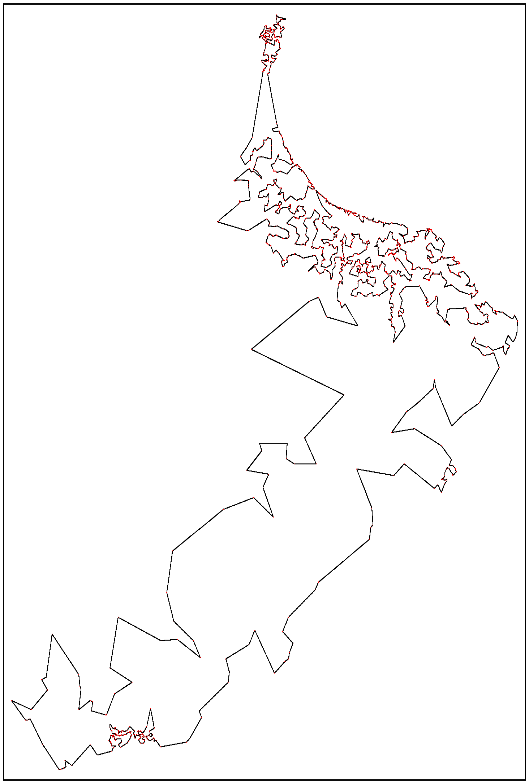
The choice of these countries is to test the accuracy over a wide range of graph sizes. We first utilized simulated annealing with a cooling rate of 0.9, initial temperature of 108, and a stopping temperature of 10-4 over 5 Monte Carlo runs. The simulated tour (left) vs. optimal tour (right) comparisons and statistic performances of each country are illustrated below. Keep in mind that for the histograms, results falling under leftmost bin are closest to the optimal solution (left is better).



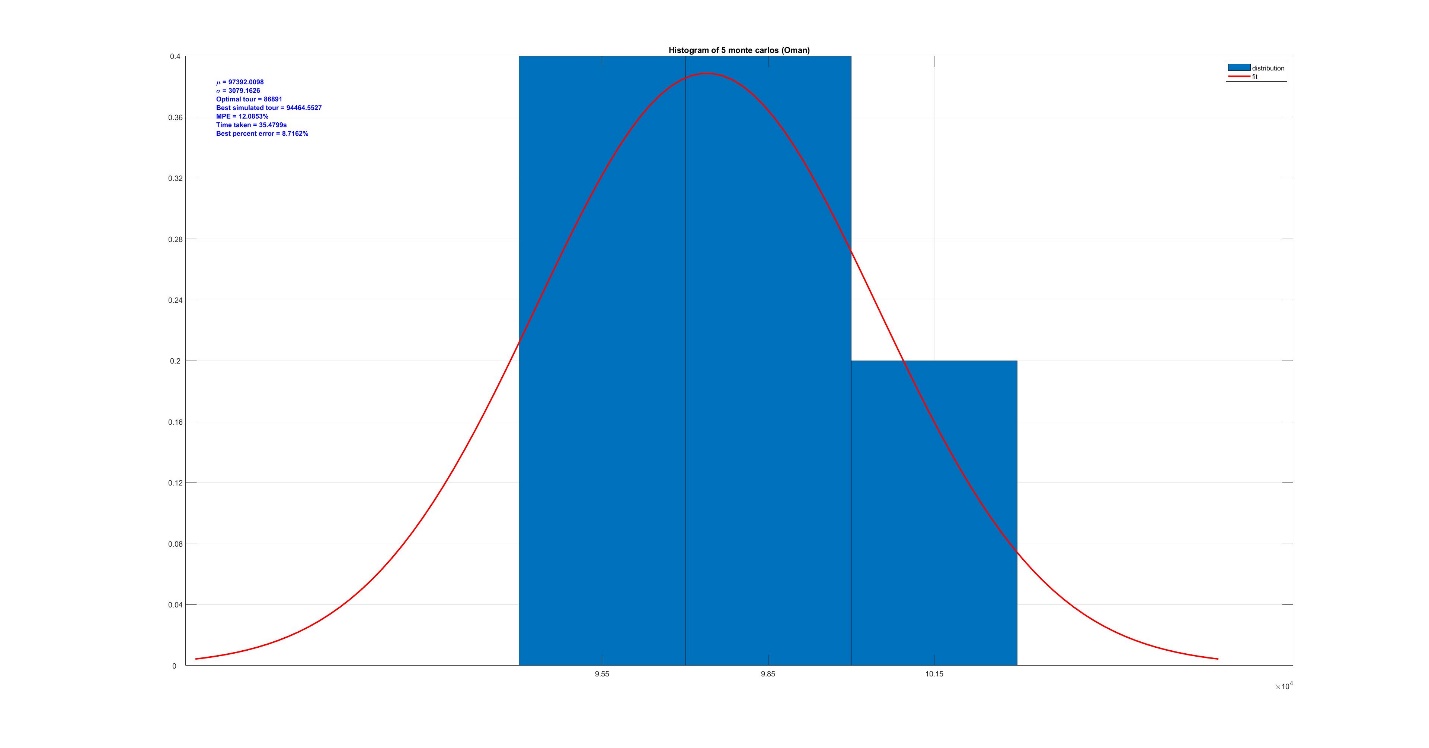
*Djibouti: 38 cities, optimal length of 6656*



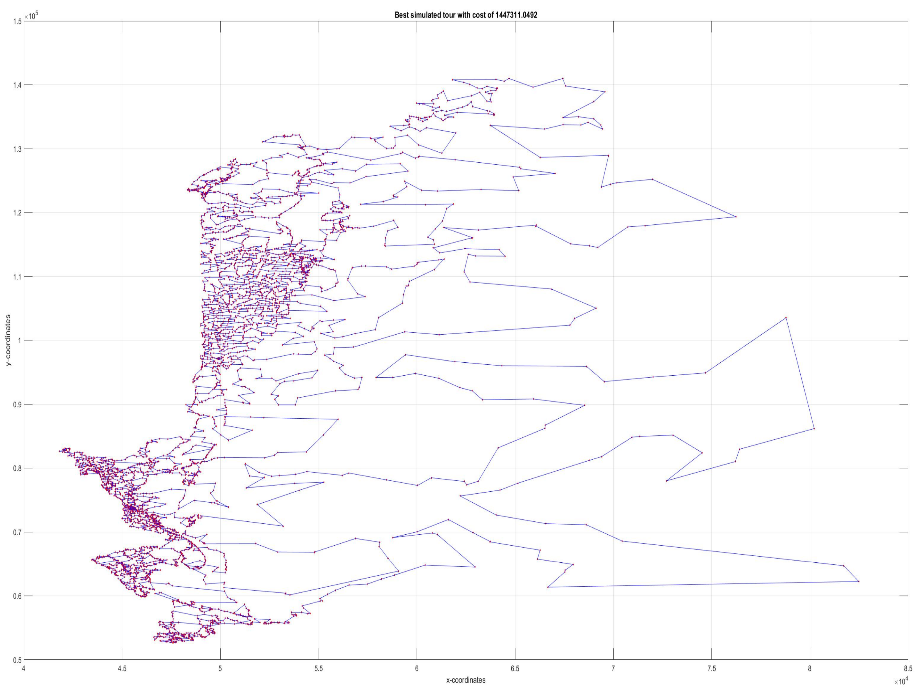
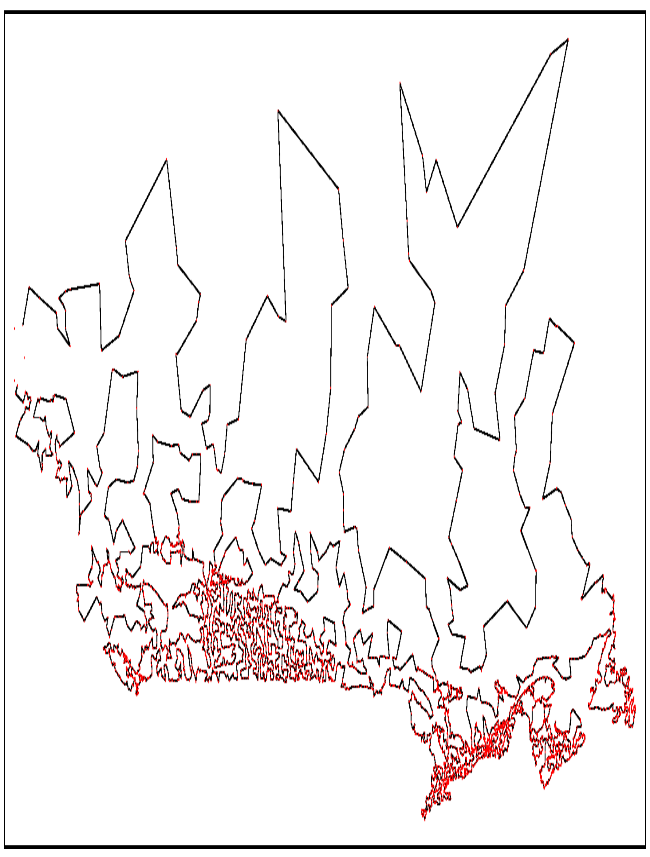
*Djibouti Performance: Mean percent error of 5.4845%, best simulated cost error of 0.1219%*



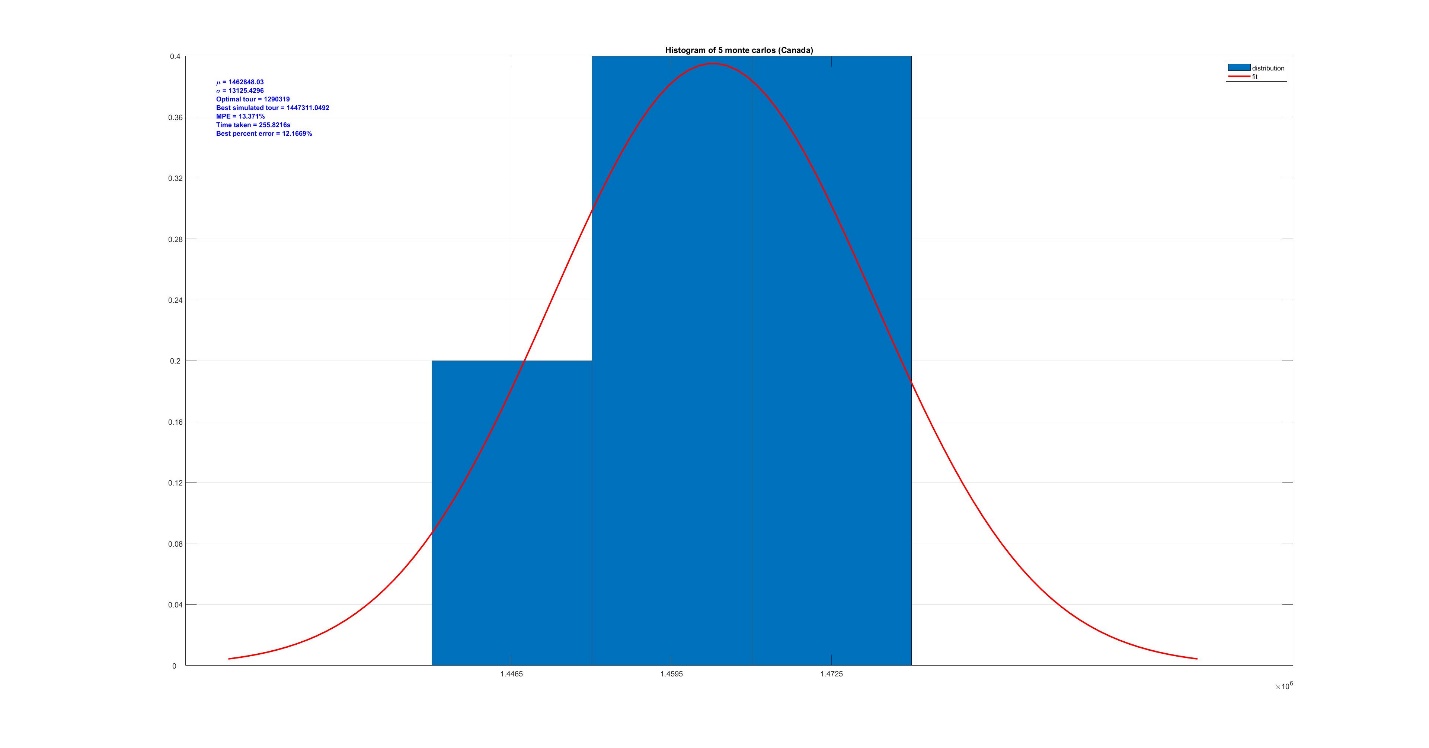
*Oman: 1979 cities, optimal length of 86891*

**

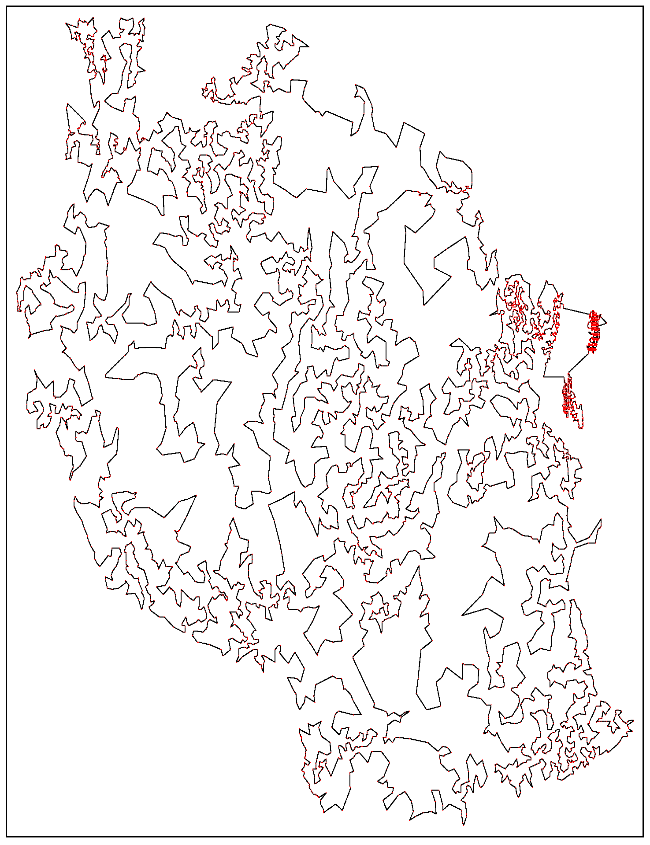
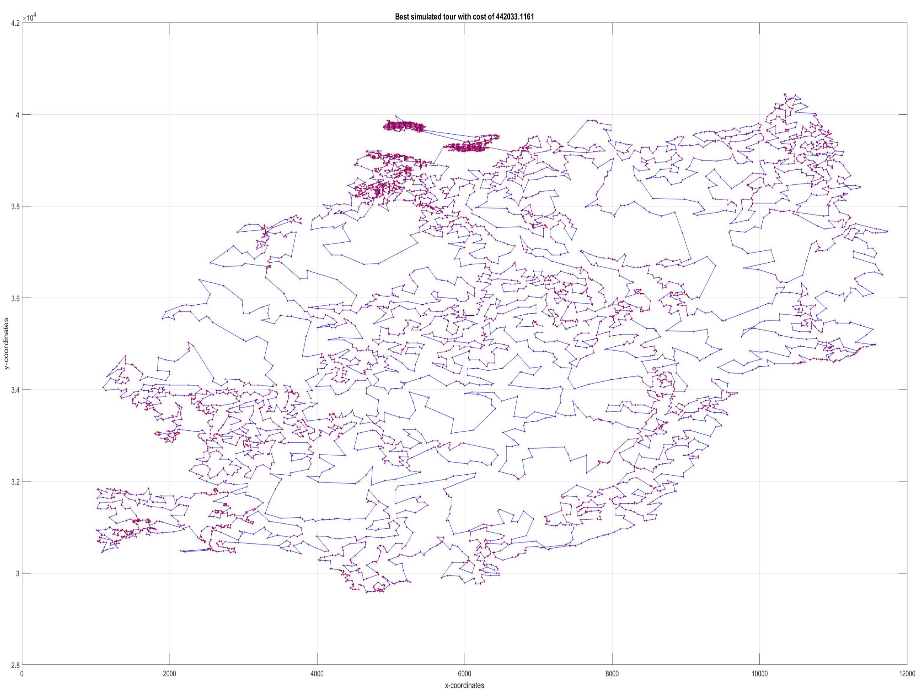
*Oman Performance: Mean percent error of 12.0853%, best simulated cost error of 8.7162%*



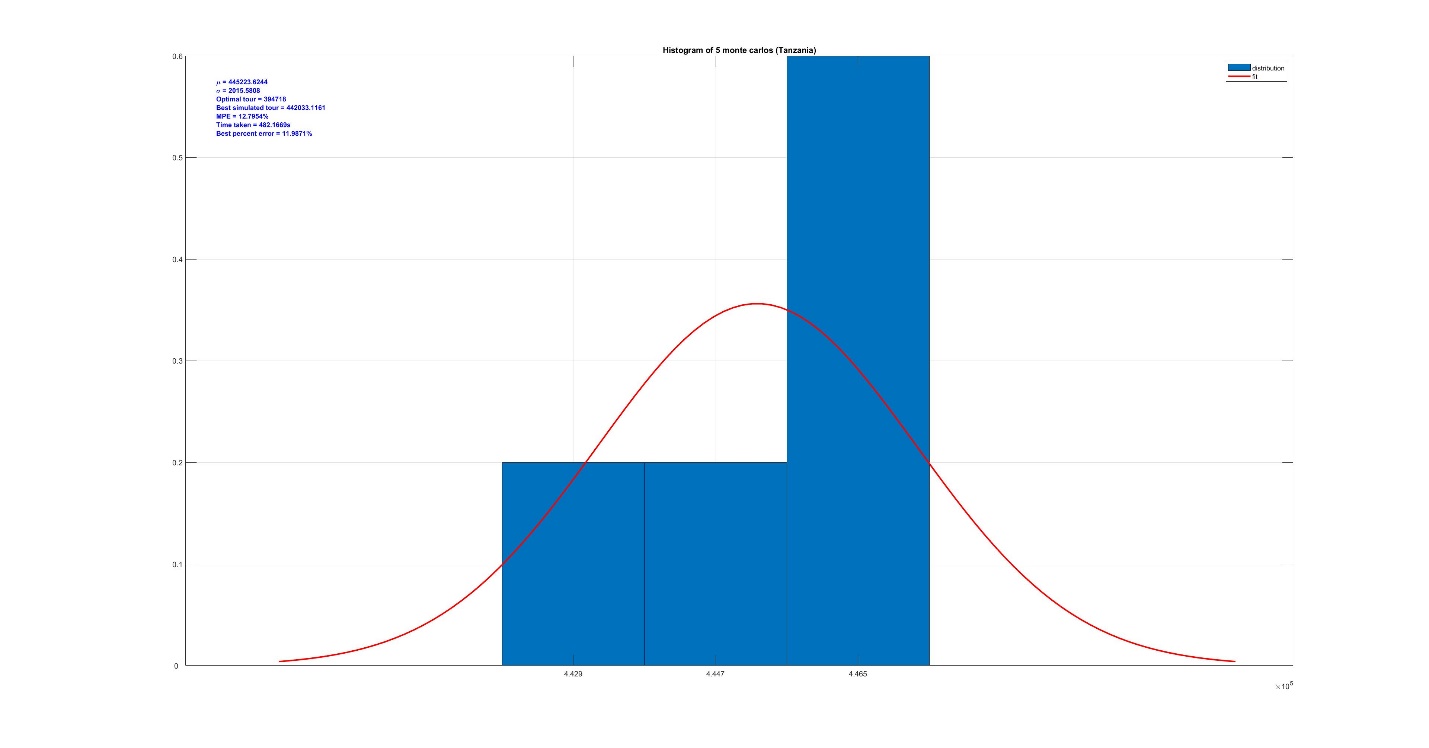
*Canada: 4663 cities, optimal length of 1290319*

**

*Canada Performance: Mean percent error of 13.371%, best simulated cost error of 12.1669%*

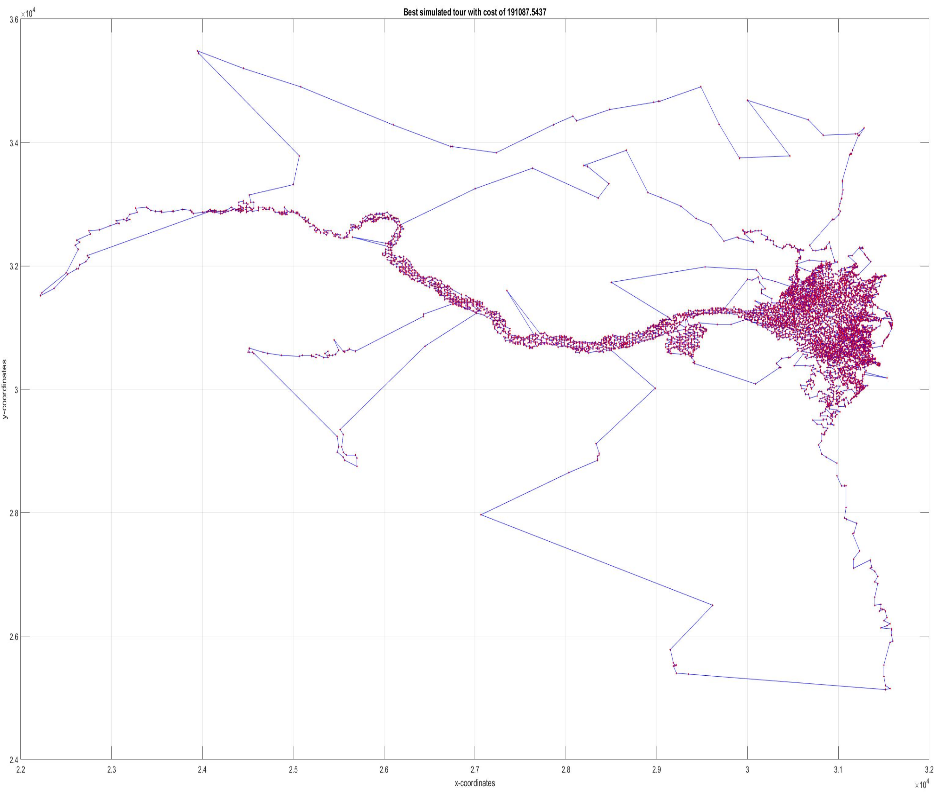
**

*Tanzania: 6117 cities, optimal length of 394718*

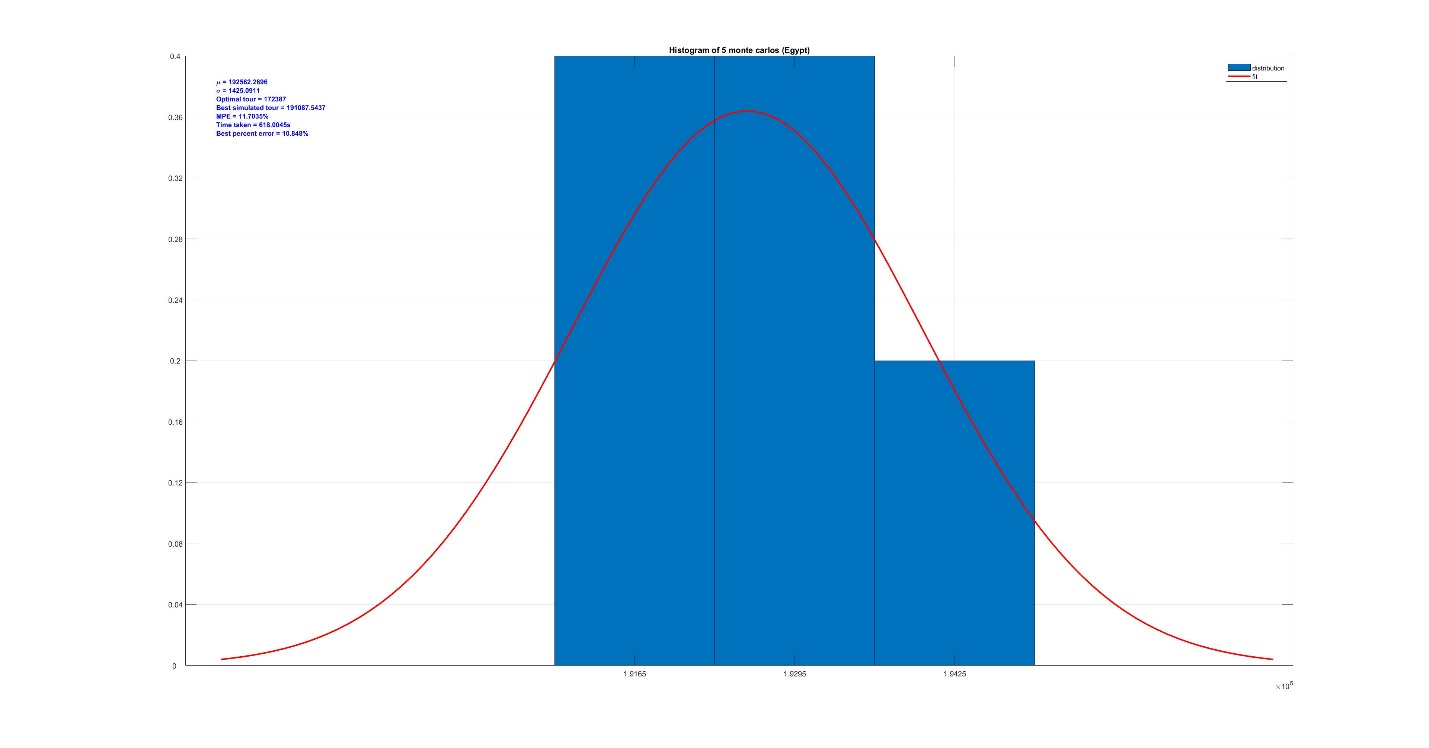
**

*F*

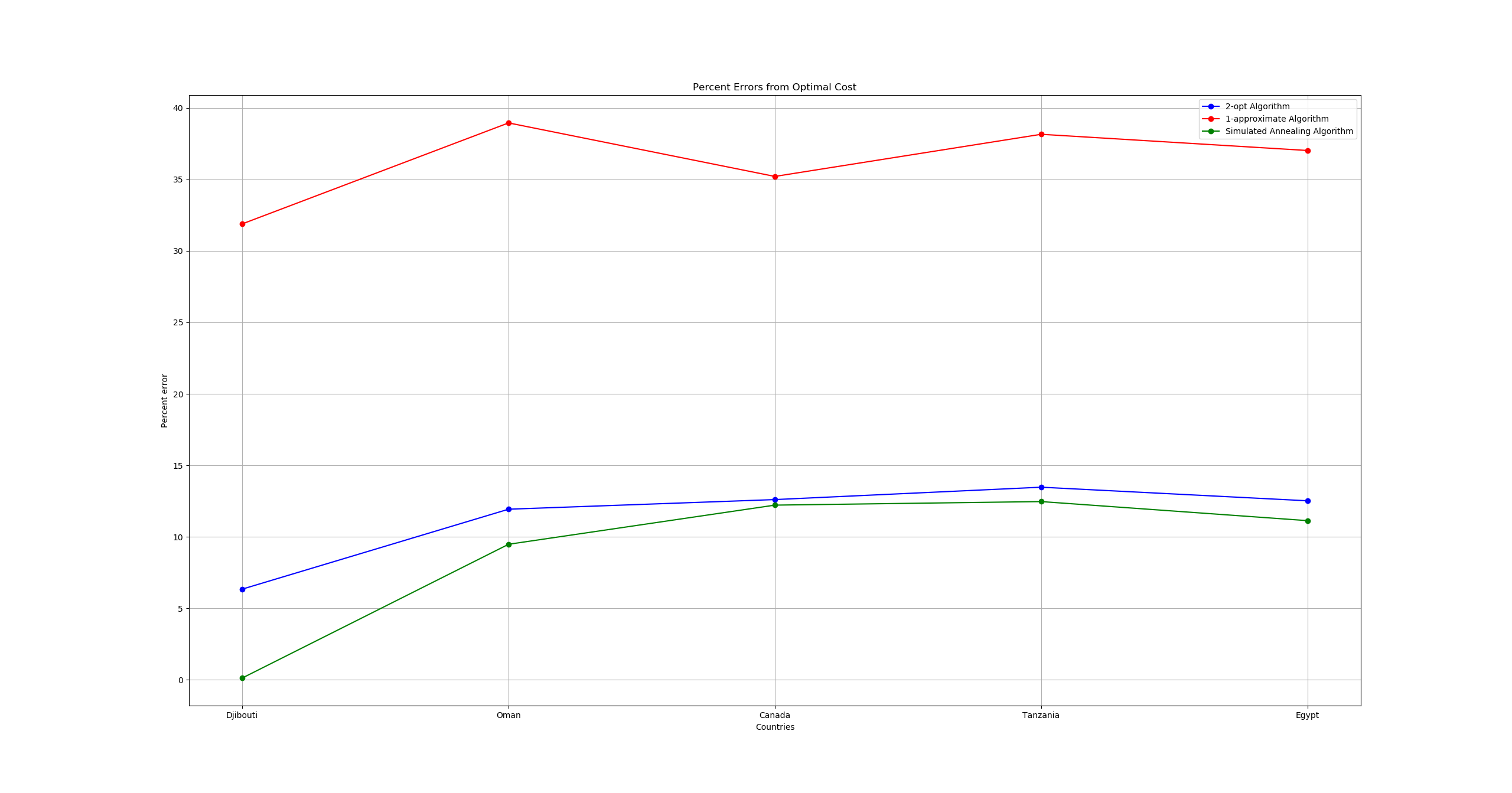
*Tanzania Performance: Mean percent error of 12.7954%, best simulated cost error of 11.9871%*

**

*Egypt: 7146 cities, optimal length of 172387*

**

*Egypt Performance: Mean percent error of 11.7035%, best simulated cost error of 10.848%*

As depicted above, the method of simulated annealing seems to hover at around 10-12% error over 5 Monte Carlo runs for the larger datasets. When comparing the simulated annealing algorithm (green) with 2-opt (blue) and 1-approximate (red), we see that it performs better for all countries:

2-opt and simulated annealing do indeed perform better than the 1-approximate algorithm in all TSP datasets. However, simulated annealing may have performed better than 2-opt by a slight margin due to the fact that it can escape from premature convergence to a local optimum through exploration. When keeping the tour of the smallest cost over each iteration, there is a higher probability that simulated annealing can converge closer to the global optimum solution. As for the 2-opt algorithm, the initialization of an arbitrary tour plays a huge role in where the algorithm will converge. For our implementation, we decided to generate a Hamiltonian cycle at random. Thus, each run of 2-opt, like simulated annealing, will give a different solution. However, 2-opt is looser in predictability in that every run generates a random tour, whereas simulated annealing is almost guaranteed to maintain consistent performance over runs.

In conclusion, we plan on implementing a better initializer in the future, perhaps with K-nearest neighbors or K-means clustering methods, in order to obtain a good, deterministic solution for the 2-opt algorithm.