# Travelling Salesman Problem with Genetic Algorithm

## Problem

The problem domain of our project is finding the solution of the Travelling Salesman Problem. The context of the travelling salesman problem is to find the shortest distance to travel to all the cities, visiting each city only once. We have tried to build a model which is very generic and can find the optimal solution for the any number of destinations with latitude and longitude given. This algorithm can be used in any real-world scenario whenever anyone wants to find the shortest route to travel though number of cities or destinations.

In this project we are trying to find the shortest route to visit all the state capitals of United States. We have not specified and starting point or the ending point of our route, therefore our algorithm will find the solution which can start from anywhere and end anywhere.

## Algorithm

We have chosen Genetic Algorithm to solve this problem. A genetic algorithm is a heuristic search method inspired by genetic processes in the nature. This method models and uses inheritance, chromosome, selection, mutation, and crossover concepts like in the biology. GAs can find the effective solutions for the TSP problem; however, GAs are more likely to depend on the way the problem is encoded and the crossover and mutations are used.

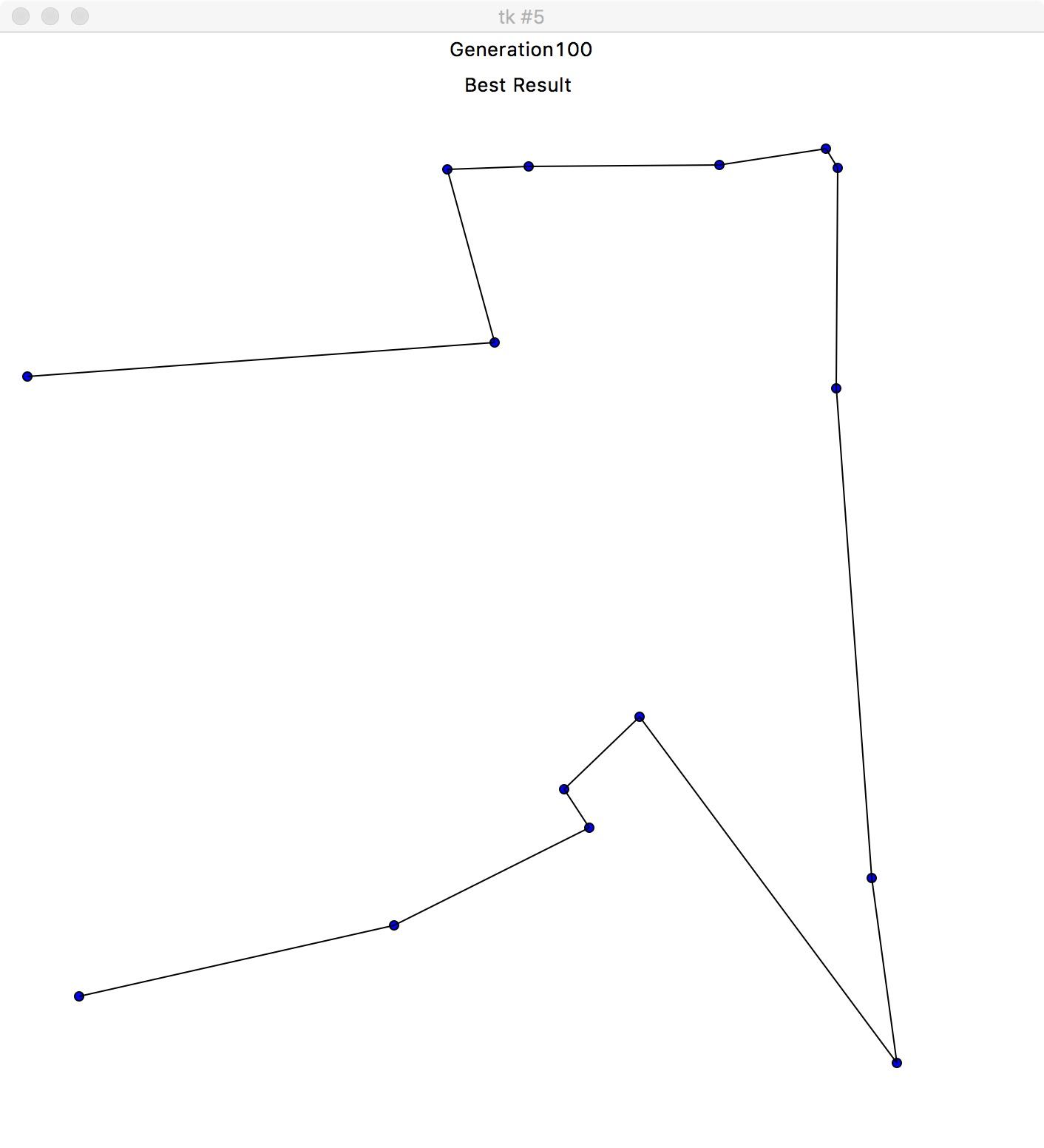
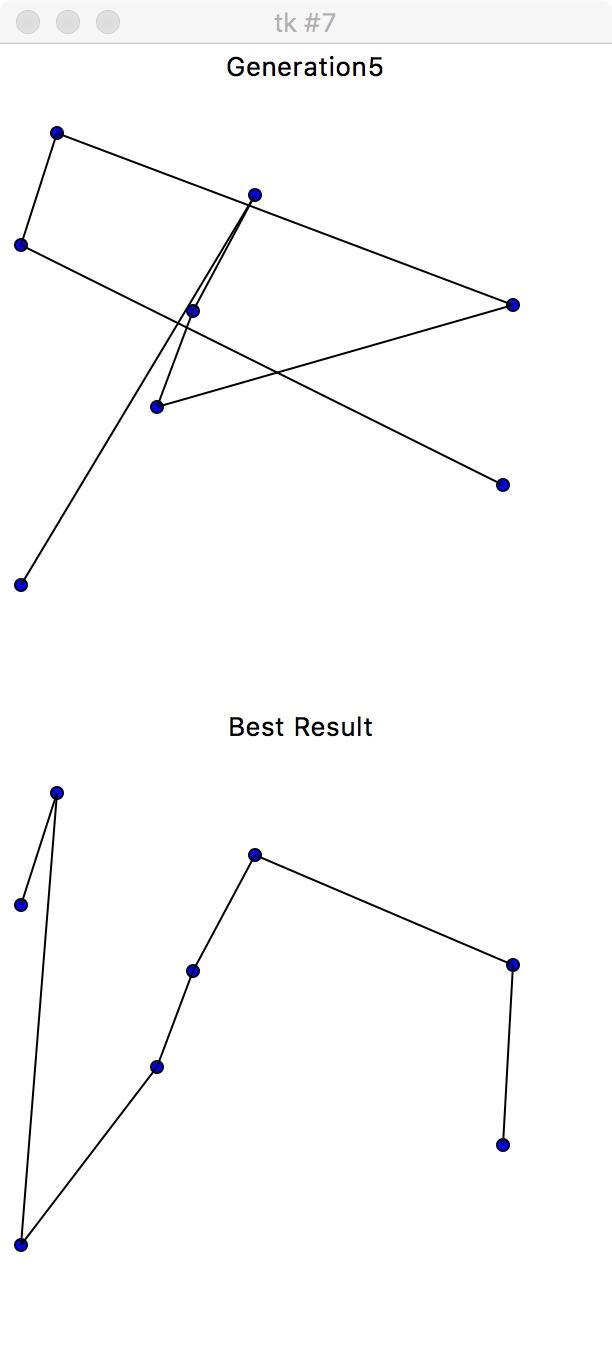
More importantly we wanted to implement genetic algorithm so that we gain a very good understanding of the algorithm, how it works and how to implement it. We choose Travelling Salesman Problem for this because it is very commonly understood problem and we would be able to find a lot of papers based on travelling salesman problem for reference.

## Difficulties Encountered

There were several difficulties we faced while working on this project. Our genetic algorithm was working well with 8 or fewer number of destinations, but as we increased the number of destinations the GA was not giving any good results. First, we solved the problem by giving a mutation rate (Mutation is not performed every time, but a rate is given to specify how often mutation is to be performed). Then again, the GA was only working good for 12 or fewer destinations. We tried increasing the population, but it was only giving slightly better result and taking much longer to process. We solved this problem by switching to Tournament Selection from Roulette Selection. Switching to tournament selection gave us better solution even with higher number of destinations, given enough number of population.

Another difficulty we faced was to find the perfect balance between number of destinations, population, generations, mutation rate and number of competitors in the tournament selection. To overcome this, we had to experiment many times trying out different values for these variables and see how the result changed.

Visualizing the process of GA implementation and progress of our result was very time costly. We had created a visualization where we could see our algorithm working through different orders of destinations and see the progress of the result. But plotting each order made the whole process very long, so we had to remove the progress and plot only the final result to save a lot of time.



The figure on the left shows the visualization we used initially to show the progress as well as the result. The figure on the right shows the visualization we used to save a lot of processing time.

## Performance

Our algorithm is performing pretty good but has a few limitations. Experimenting with different selecting functions and mutations, we gave got to a very good result for the number of destinations as high as 70, but the quality of our result is decreasing as we increase the number of destinations. We might have to have to give a very high number of population and a high number of generations, but the hardware we are working on has limitations. It took 38 minutes to get the result, which was just fine (not very good), with 100 destinations, 50000 population and 100 generations. We are pretty sure that we could get better result with larger number of destination using a very large population, but we would take hours to process with a better hardware and processor. In the table below, we can see how our algorithm performed, what values were given for the parameters and the time taken for each process.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Code | No. of destinations | Population | Generations | Mutation Rate | Mutation Type | Selection Type | Time to run | Results |
| 25 | 5 | 100 | 5 | 1 | Random | RS | 2 | Very Good |
| 26 | 8 | 100 | 5 | 1 | Random | RS | 2 | Not Bad - Could be better |
| 27 | 8 | 100 | 50 | 1 | Random | RS | 2 | Good |
| 28 | 8 | 1000 | 5 | 1 | Random | RS | 2 | Good |
| 29 | 10 | 1000 | 5 | 1 | Random | RS | 2 | Not Bad - Could be better |
| 30 | 10 | 1000 | 50 | 1 | Random | RS | 7 | Fine |
| 31 | 10 | 1000 | 50 | 0.2 | Random | RS | 7 | Good |
| 32 | 10 | 2000 | 50 | 0.2 | Random | RS | 22 | Very Good |
| 33 | 15 | 2000 | 50 | 0.2 | Random | RS | 25 | Pretty Bad |
| 34 | 15 | 2000 | 100 | 0.2 | Random | RS | 24 | Not Bad - Could be better |
| 35 | 15 | 5000 | 100 | 0.2 | Random | RS | 127 | Not Bad - Could be better |
| 36 | 15 | 5000 | 100 | 0.2 | Neighbors | RS | 128 | Not Bad - Could be better |
| 37 | 15 | 10000 | 100 | 0.2 | Neighbors | RS | 483 | Not Bad - Could be better |
| 38 | 15 | 10000 | 100 | 0.2 | Neighbors | TS | 143 | Very Good |
| 39 | 15 | 5000 | 100 | 0.2 | Neighbors | TS | 78 | Very Good |
| 40 | 30 | 5000 | 100 | 0.2 | Neighbors | TS | 88 | Very Good |
| 41 | 50 | 5000 | 100 | 0.2 | Neighbors | TS | 192 | Not Bad - Could be better |
| 42 | 50 | 10000 | 100 | 0.2 | Neighbors | TS | 208 | Very Good |
| 47 | 70 | 20000 | 100 | 0.2 | Neighbors | TS | 561 | Fine |
| 48 | 70 | 30000 | 150 | 0.2 | Neighbors | TS | 1162 | Good |
| 43 | 100 | 10000 | 100 | 0.2 | Neighbors | TS | 328 | Not Bad - Could be better |
| 44 | 100 | 20000 | 100 | 0.2 | Neighbors | TS | 654 | Fine |
| 45 | 100 | 30000 | 200 | 0.2 | Neighbors | TS | 2556 | Fine |
| 46 | 100 | 50000 | 100 | 0.1 | Neighbors | TS | 2279 | Fine |

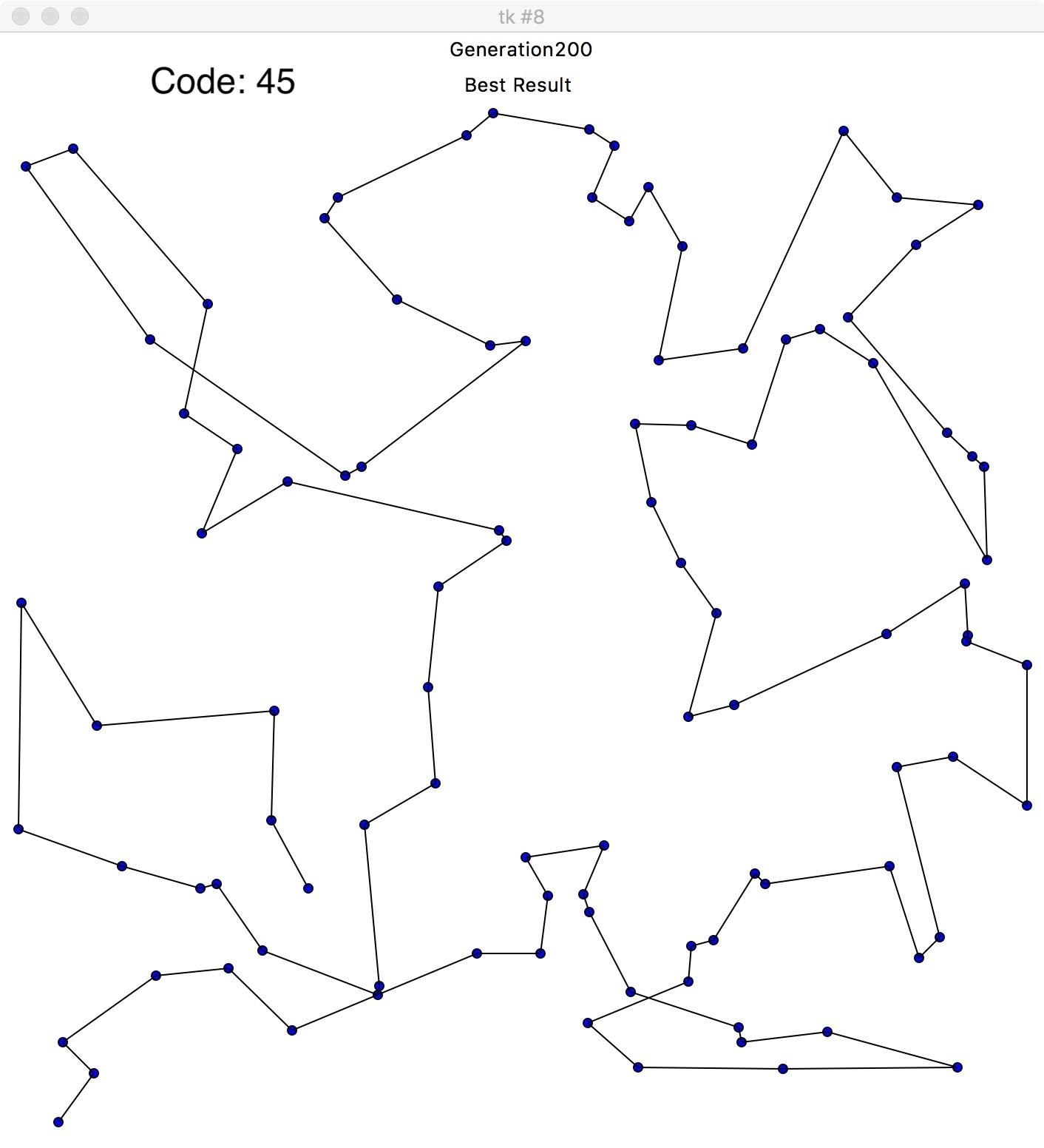
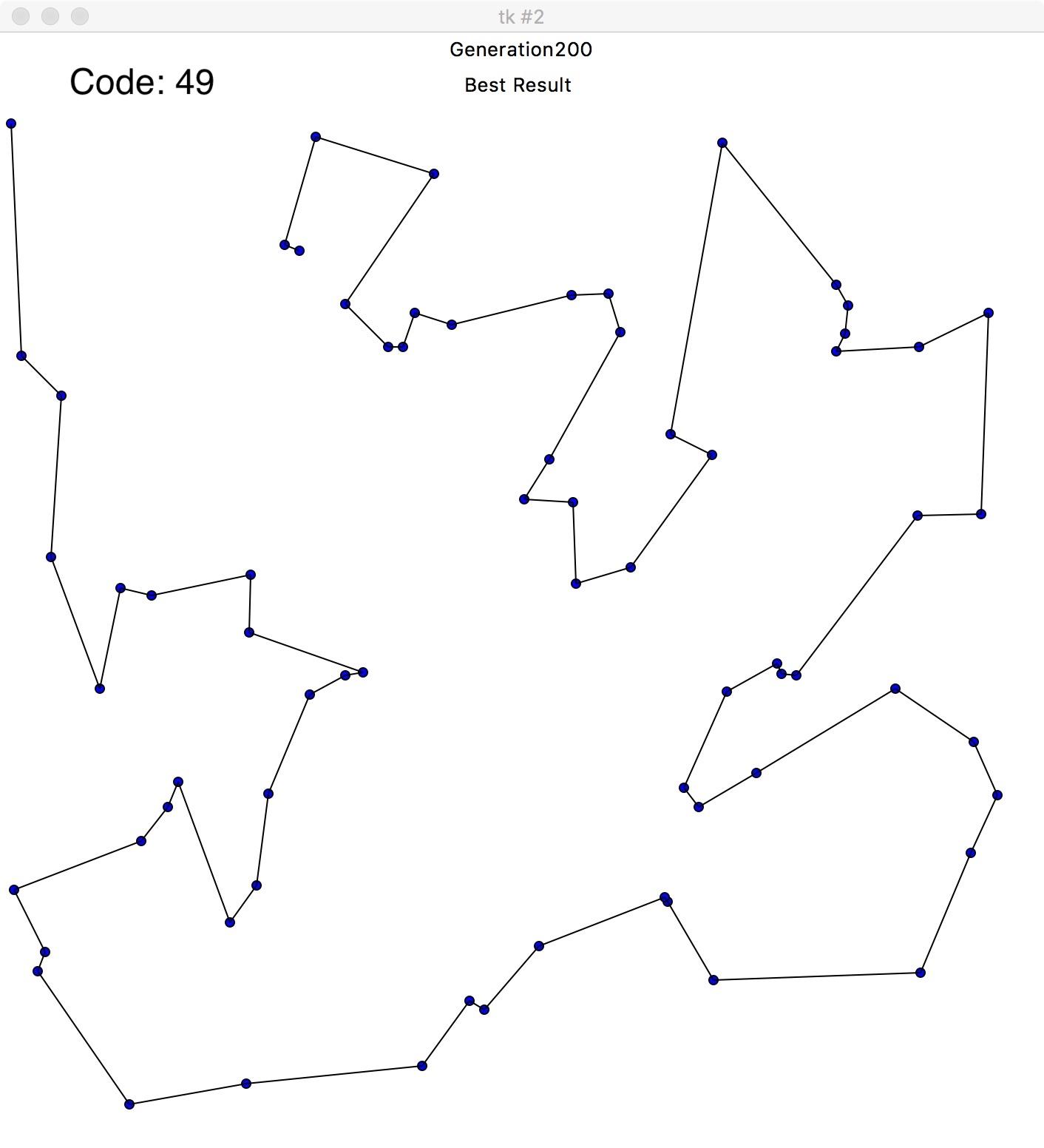
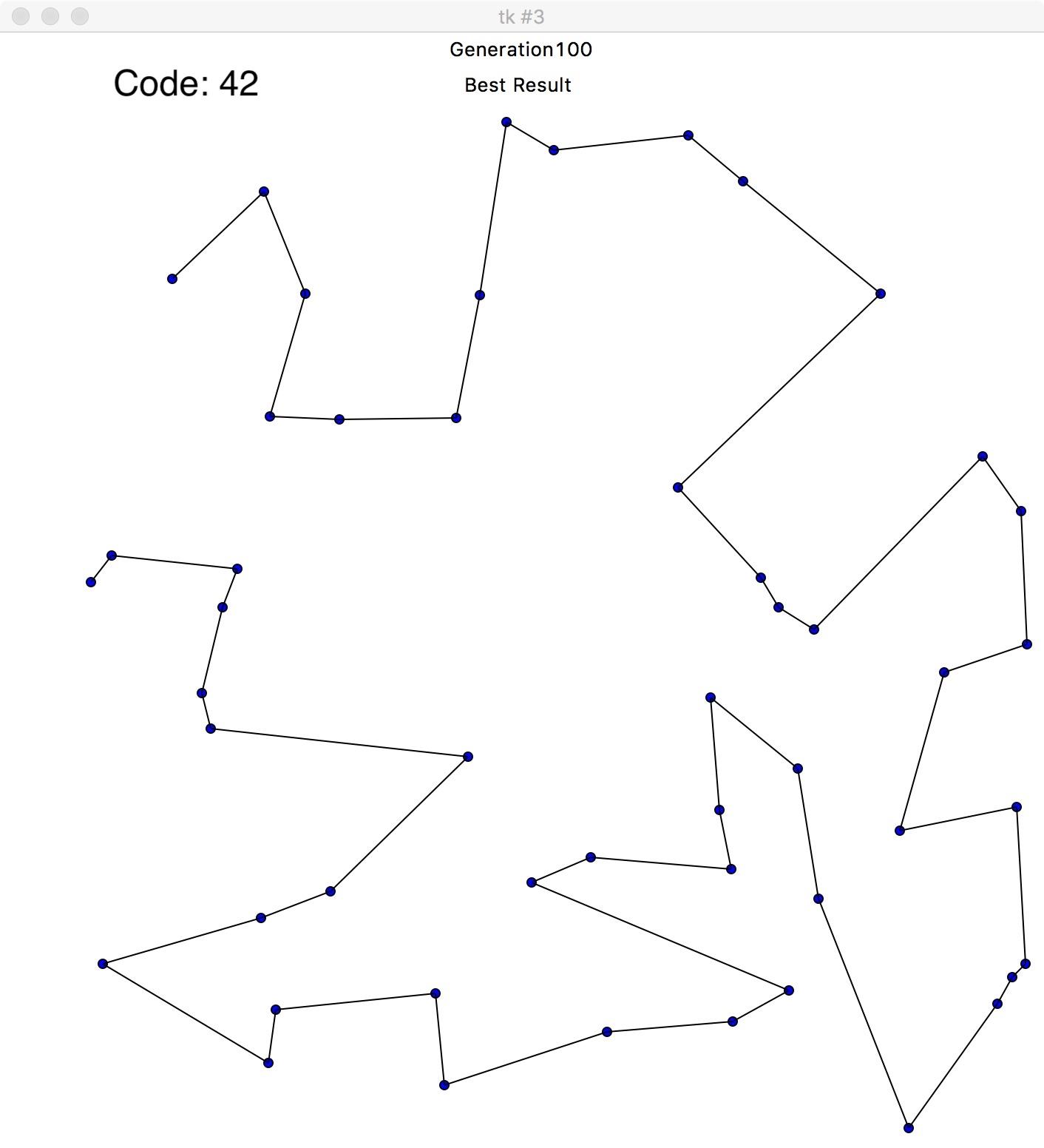
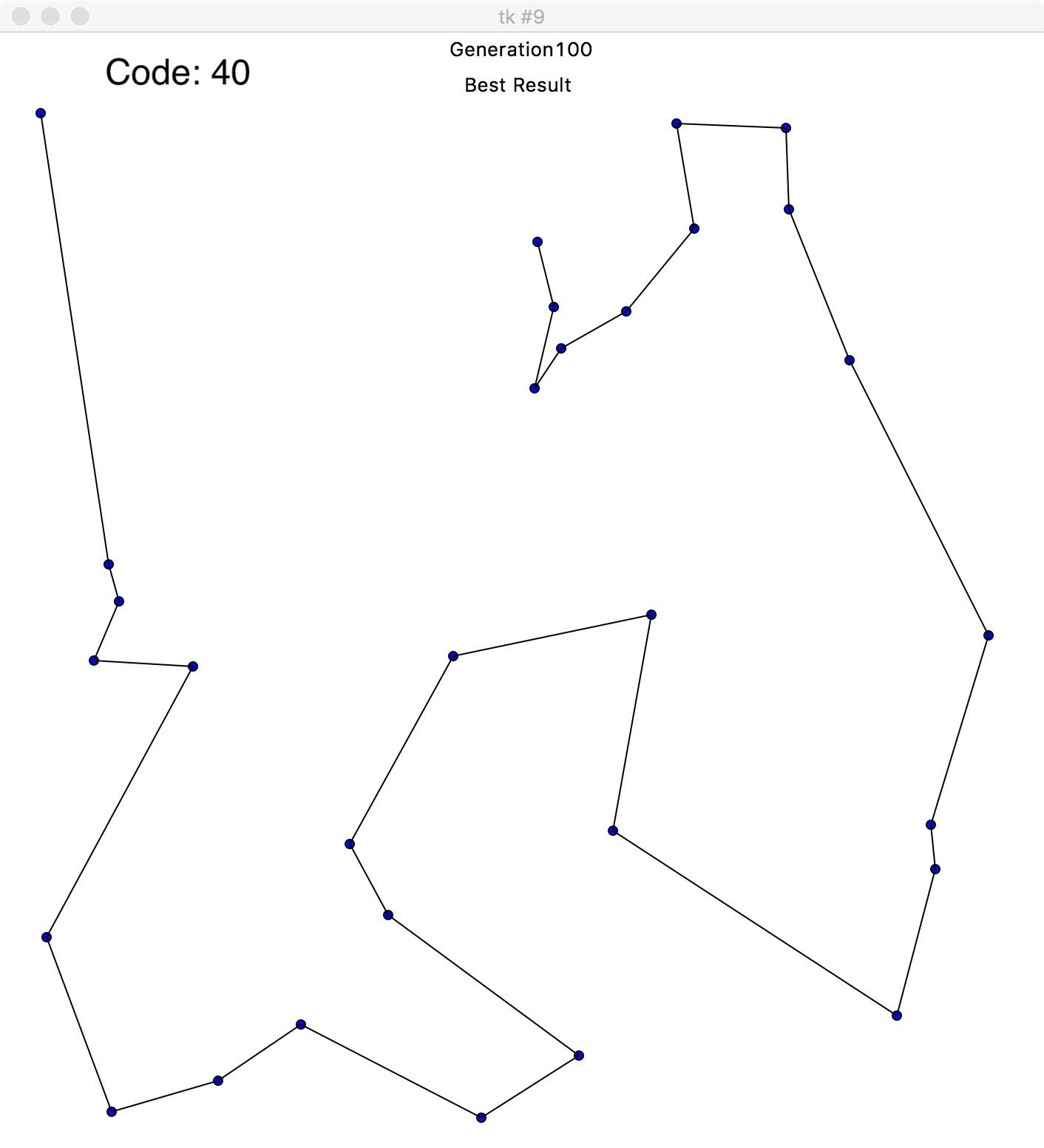
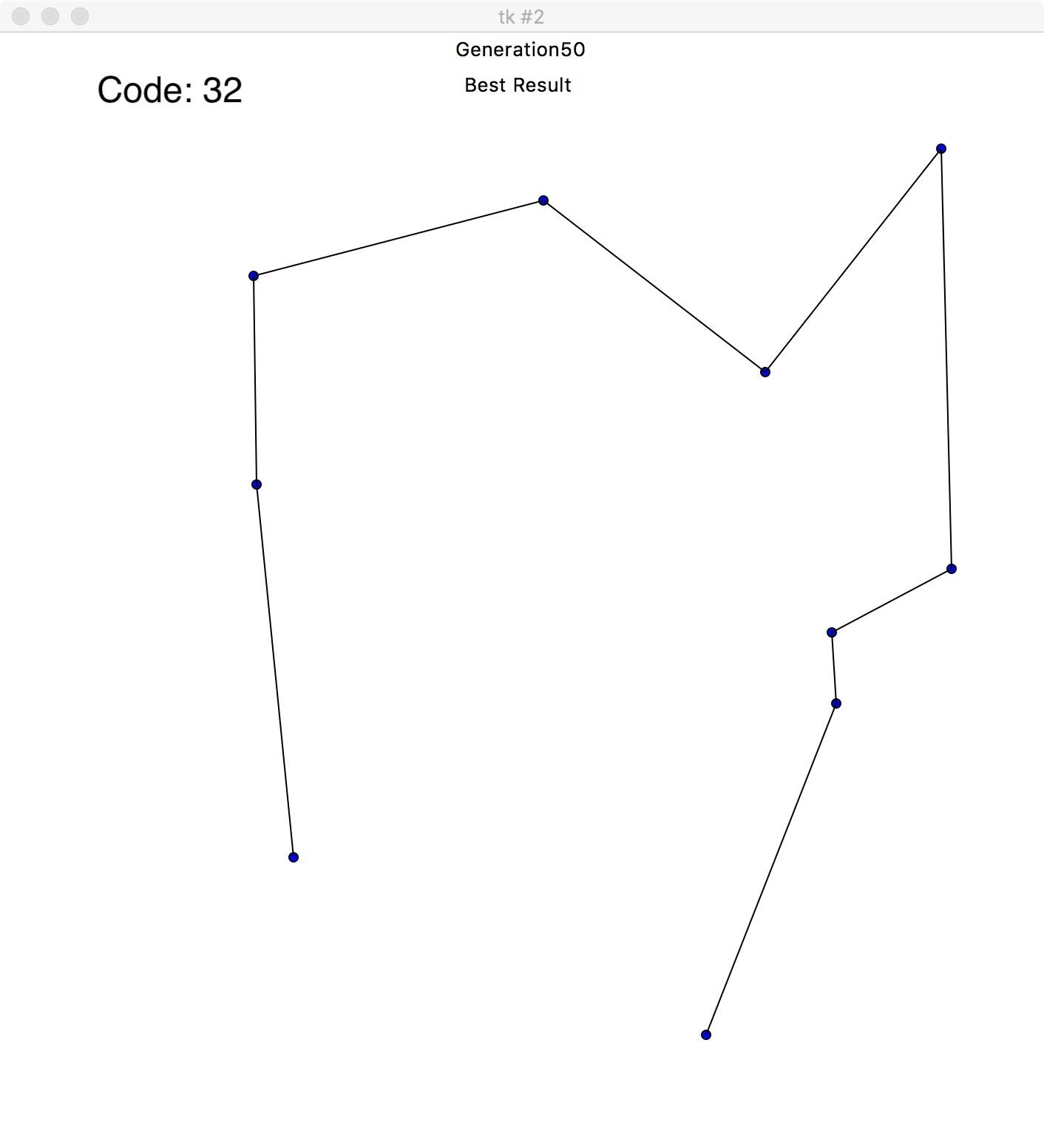
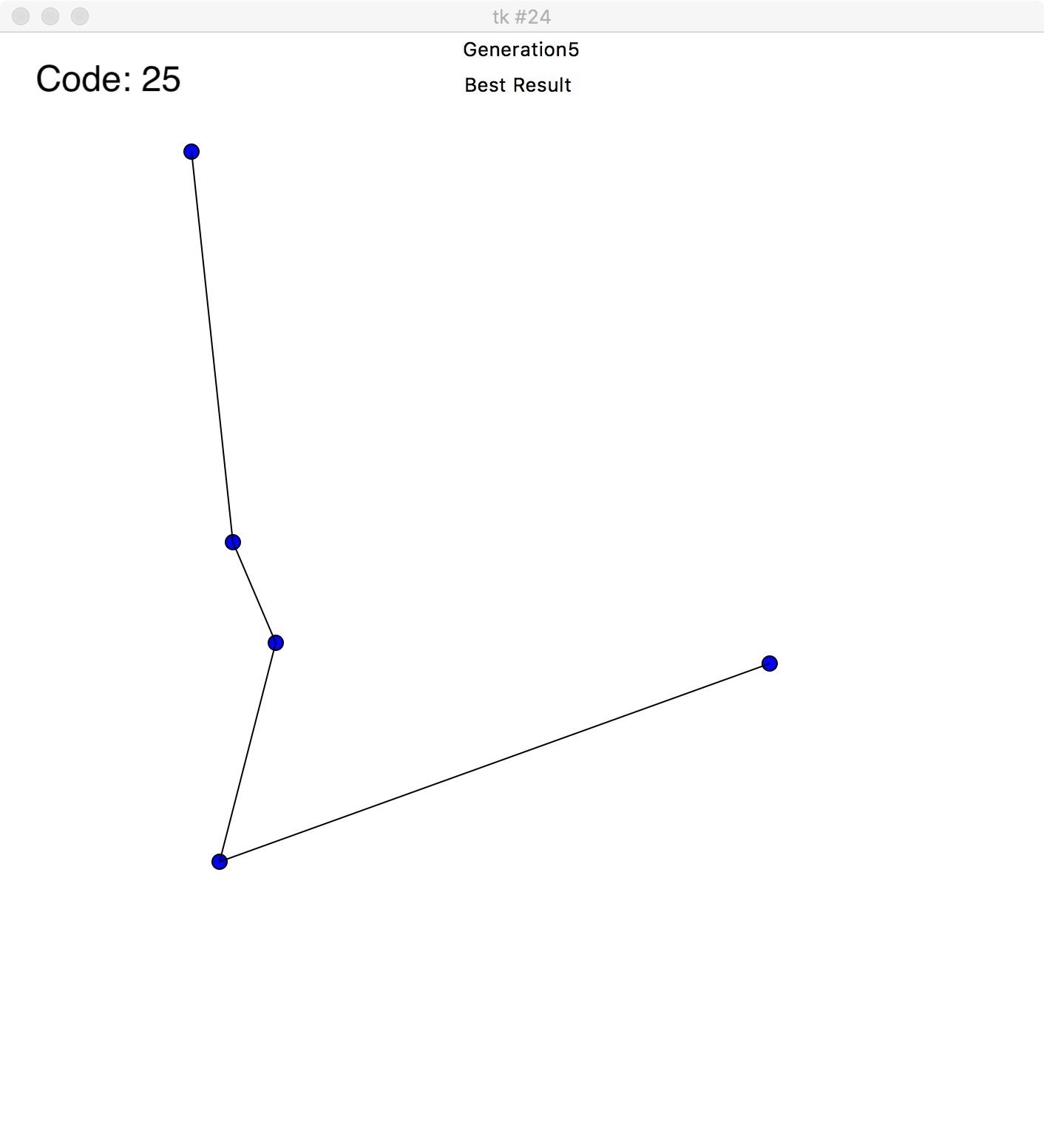
Table 1: Value for different parameters and their results

Abbreviations:

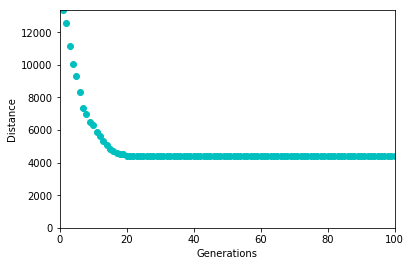
* RS: Roulette Selection
* TS: Tournament Selection
* Random: Randomly Swapped
* Neighbor: Swapped with neighbor

Below are the few examples of the results where we got a good result.

The code in each image refers to the code in the table above, you can see the values of each variables used in the experiment in the above table:

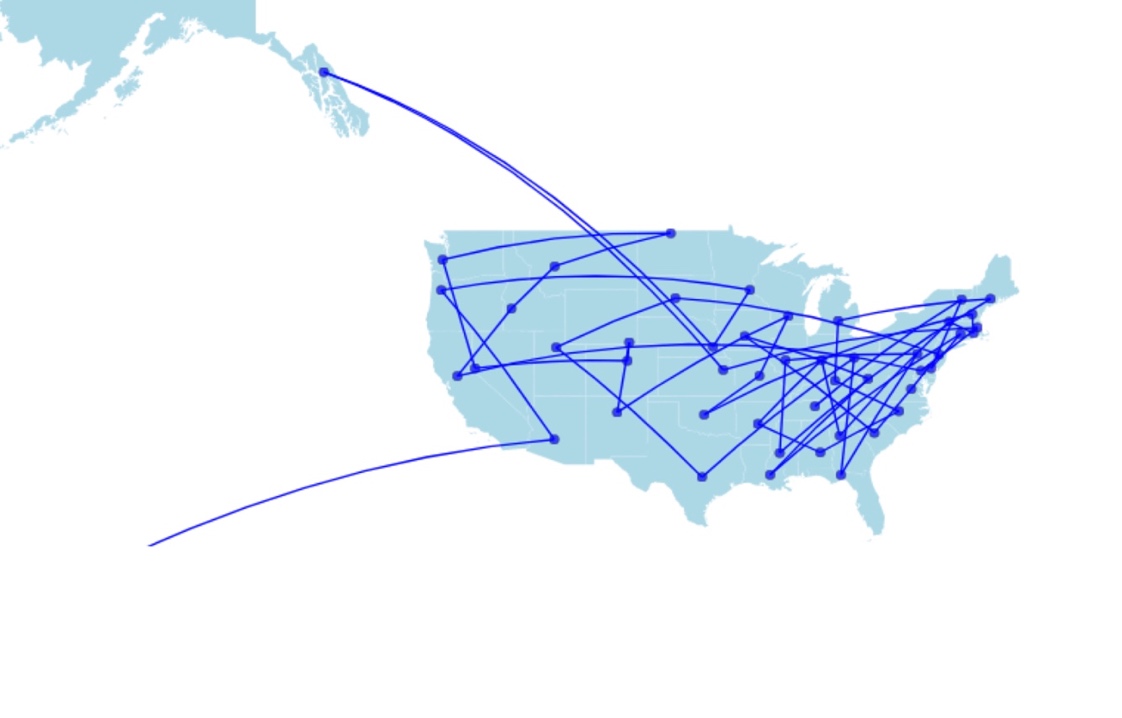


The graph below shows the improvement on the total distance in each generation.

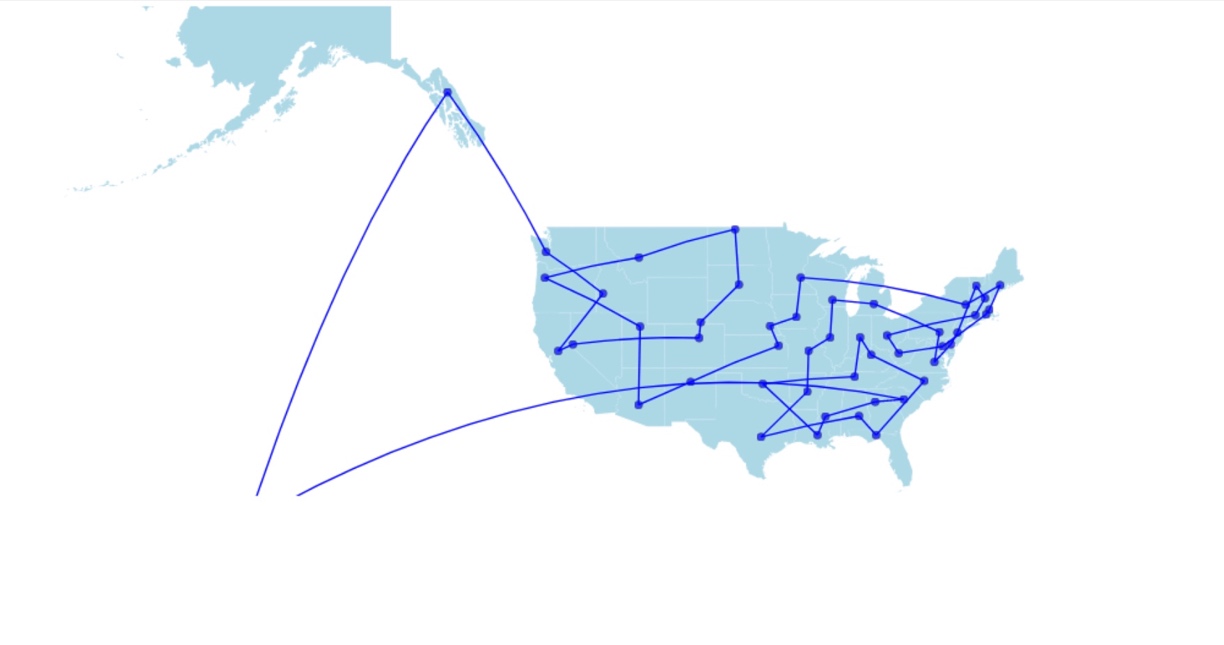


Below is the solution to the finding the shortest route for travelling to all the state capital cities of USA. Here we have 50 destinations with different values for population and generations.

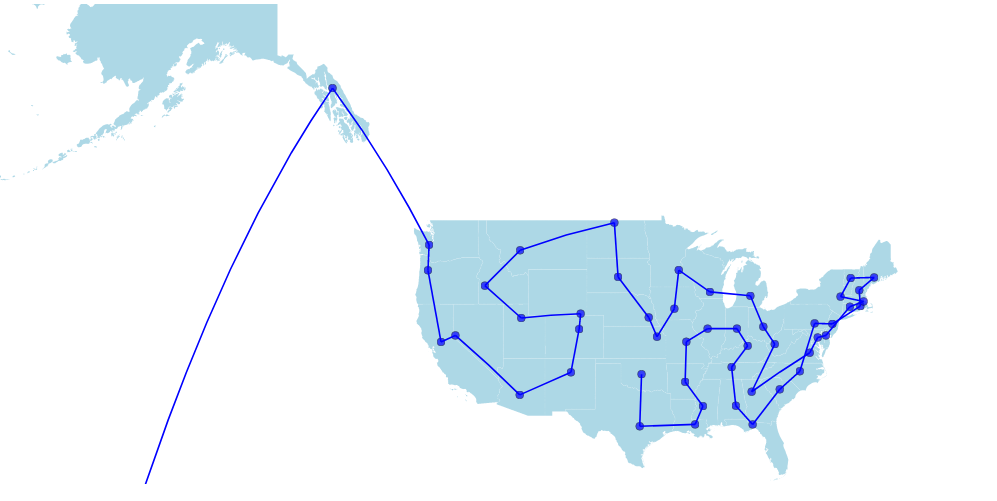
Population: 1000 Number of Generation: 20 Bad Result



Population: 10000 Number of Generation: 100 Better but not satisfactory



Population: 30000 Number of generations: 100 Satisfactory Result



## How would we improve your solution

We wound want to work more with this project and improve the performance of algorithm by merging another algorithm to this model. We are interested to add the 2-opt algorithm in our algorithm. The 2-opt algorithm was proposed for solving the travelling salesman problem. This algorithm searches for the lines(routes) which crosses itself and reorders in such a way that it does not cross. Some of our final results, those of which are not very good results, have routes that cross itself. We could somehow use the 2-opt algorithm to remove the crossing and reorder the gene. This algorithm could be used as a mutation, which is only done for the gene with the highest feature score and could be done in each generation or may be done after a couple of generations, after we have some fare result. We could do different experiments with this algorithm in our genetic algorithm. We would also like to try out other algorithms and test the performance of each algorithm. This could be an experiment where we could find the best algorithm for solving Travelling Salesman Problem.

Another improvement to our algorithm would be to use google API to get the distance, so that we could find the real travelling distance. We could also calculate the according to the traveling time with various means of travelling.

## Alternative Algorithms?

We did not work on any other algorithms for this project, but as already mentioned above, we would like to experiment merging genetic algorithm with 2-opt algorithm to find the optimum solution. We were getting good results with the genetic algorithms with fewer destinations. As we increased our destinations we had to make a few tweaks to our algorithm (like switching to tournament selection, changing mutation rate) and still we were getting good results. But 100 destinations gave us a hard time, and that’s when we started exploring for other algorithms. It’s not that we did not consider other algorithms, it’s just that we did not have enough time to go through another algorithm. But we surely would like to experiment with it in the future.

## Surprising Aspect

The processing time really surprised us. Initially we were visualizing each order and the progress of the process. The plotting of the nodes and the lines were taking too long. So, we decided to remove the part of plotting each genome, and only plotting the final result. The processing time was surprising. With visualization the processing time was 64 secs to get the result for 8 destinations, with 100 populations, and 5 generations. Without visualization the same process took 0.188 Secs. That’s why we switched to only visualizing the final result.

Other Information

* General: Many travelling salesman problem might state that the starting point and the endpoint should be the same; we have not implemented this scenario in our model. We could implement this by making very small changes in our code (adding a static start and end chromosome in each of our gene). We have completed our coding in python and have plotted our result in geospatial graph using javascript. In the initial phase of coding we used tkinter for plotting our results, and after we started getting good results we switched to using the real-world locations and plotting geospatial graph with javascript.
* Population: We took random combinations of the destination indexes and generated the population for our algorithm.
* Selection: We experimented with both Roulette Selection as well as Tournament selection. In our case the tournament selection gave us better results with faster performance.
* Crossover: We were not able to use the single point or 2-point crossover in out genetic algorithm, as in our case each genome should contain each and every nodes or destinations. Using single or 2-point crossover, we would have duplicates in our genomes and may not have all the nodes present in each genome. This is why we used a different crossover. We implemented our crossover by selecting random part of parent-1 and putting it in the beginning of the child in the same order. Then took the values which are not present in the child from parent-2 and put it in the remaining spaces of the child in the same order fond in parent-2.
* Mutation: Initially we applied mutation by swapping two random nodes of each and every genome. For better result not, all genes were mutated but we gave added a mutation rate which specified how often mutation is done. We also experimented with a different kind of mutation where a random node is swapped with its neighbor; we saw a slight improvement in the performance, but not much.
* Geospatial with javascript: We used the D3.js library in javascript for the purpose of visualizing the geospatial data. We took the geojson file of the USA and rendered in the browser using mercator projection. The best order of the cities is exported to the csv file which is taken as the input in javascript. We chose d3.js so that we can make good visualizations and d3.js is very powerful for visualization.We could have added interactive visualizations too but we saw it will take time so we settled on static visualizations.