Travel Time Optimization Problem via Ant Colony and Genetic Evolution

Rishika Drona

Abstract

Using a collection of New York City locations represented as longitude and latitude coordinates, we search the solution space to find the most efficient traveling salesman path. By creating a regression model to solve the travel time cost between two locations, we optimize the traveling salesman path using two biologically inspired meta-heuristics: ant colony and genetic evolution. Our experimentation shows that ant colony optimization performs better 30% better than genetic evolution.

1 Introduction

Urban transportation is going through a rapid and significant evolution. Since the birth of the Internet and smartphones, we have become increasingly connected and are able to plan and optimize our daily commute. Along with that, large amounts of data are gathered and used to improve the efficiency of existing transportation systems. Real-time ride-sharing companies such as Uber and Lyft are using this data to revolutionize the taxi industry, laying the ground for a more connected and centrally controlled transportation structure, and building innovative systems like car-pooling.

In this project, we tackle the travel time optimization problem for taxi vehicles. This can be framed as the *Traveling Salesman Problem*, a well-known computer science problem. The objective is to find the shortest route that visits a set of locations. For this problem, optimization techniques are required to intelligently search the solution space and find near-optimal solutions. More specifically, we first use machine learning to forecast travel times between every pair of pickup and dropoff locations. Then we use evolutionary algorithms, namely ant colony and genetic, to find the best travel itinerary for the vehicles in the dataset.

2 Problem Description

2.1 Dataset

The data that we will work with is the NYC Taxi and Limousine Commission Trip Record data, which can be accessed at this URL: *https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page*. We have the monthly data in the year 2021 for Yellow Taxi, Green Taxi, and For-Hire Vehicle. The dataset has close to 1.5 million trip records with the following 11 attributes:

- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record
- **pickup_datetime** - date and time when the meter was engaged

- **dropoff_datetime** - date and time when the meter was disengaged
- **passenger_count** - the number of passengers in the vehicle (driver entered value)
- **pickup_longitude** - the longitude where the meter was engaged
- **pickup_latitude** - the latitude where the meter was engaged
- **dropoff_longitude** - the longitude where the meter was disengaged
- **dropoff_latitude** - the latitude where the meter was disengaged
- **store_and_fwd_flag** - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server: Y = store and forward and N = not a store and forward trip
- **trip_duration** - duration of the trip in seconds, target variable

We conducted simple data pre-processing techniques such as eliminating duplicates, checking latitude and longtitude bounds, removing obvious outliers, converting data fields to sensible units etc. to get a clean dataset for our experiment.

2.2 Data Visualization

A big component of our project is to visualize the data and figure out the best features to be used for the machine learning model at the next step. **Pickup Time** is a feature that we first looked at:

- Figure 1 is the pickup time distribution by hours of the day. As can be seen, the most popular pickup hours are between 6 and 10 PM with more than $70,000$ trips made. On the other hand, the least active pickup hours are from 2 to 6 AM with less than $35,000$ trips.
- Figure 2 is the pickup time distribution by weekdays in a week. It seems like Friday is the most popular day to hail a taxi with close to $220,000$ trips made, while Sunday is on the other end of the spectrum with approximately $165,000$ trips.
- Figure 3 is the pickup time heatmap for the results from Figure 1 and Figure 2. The most active pickup times are between 6 to 9 PM from Thursday to Sunday. The least active pickup times are between 2 to 5 PM from Wednesday to Sunday.

The next feature that we examined is **Trip Duration**:

- Figure 4 is the trip duration distribution by minutes. We observed that the distribution is heavily left-skewed, with more than 75% of trips between 3 and 12 minutes. More than $10,000$ trips last between 5 to 7 minutes.
- Figure 5 displays the comparisons of trip duration distribution by minutes versus by latitudes and longtitudes. The distribution by longtitudes is also quite left-skewed, with the peak roughly at 1 kilometers. The distribution by latitudes is more normally distributed, with the peak at 0 kilometers.
- Figure 6 is the trip duration heatmap by minutes on hours in a day versus weekdays in a week. The longest trips occur frequently between 2 to 4 PM from Wednesday to Friday. The shortest trips happen between 8 to 9 AM on Monday and Tuesday.

We are also interested in **Pickup** and **Dropoff** Locations:

- We plotted the spatial density of pickup and dropoff locations, as shown in Figure 7. It looks a bit as an image you can see from space, with Manhattan and the two airports (LGA and JFK) "lighting up" clearly.
- We went even further to show typical trips on a map in Figure 8. The magenta circles are pickup locations, the green circles are dropoff locations, and the cyan arrow lines are the length of the trip. We can see an uneven distribution of pickup and dropoff locations, as there are much more dropoff locations outside of Manhattan (Queens and Brooklyn).

2.3 Machine Learning

The next step of our pipeline is to build a machine learning model to predict travel times between pickup and dropoff locations. With our data and objective, a simple linear regression won't be powerful enough. With a few random outliers in a huge dataset and possibly a number of categorical features, we decided to go with gradient boosted trees, which can easily capture nonlinear relationships, accommodate for complexity, and handle categorical features. The input features that we used are: **passenger_count**, **pickup_longitude**, **pickup_latitude**, **dropoff_longitude**, **dropoff_latitude**, and **store_and_fwd_flag**. The output target is **trip_duration**.

We implemented this model via XGBoost - an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. This model is easy to set up, but is difficult to fine-tune and interpret. We used the root mean squared logarithmic error as the evaluation metric, as it reduces error magnitude. With this metric, we ran different hyper-parameters choice and compare the results. The final parameters we used for our XGBoost model are: $\{booster = gbtree, objective = linear, learning\_rate = 0.05, max\_depth = 14, subsample = 0.9, colsample\_bytree = 0.7, colsample\_bylevel = 0.7, silent = 1, feval = rmsle\}$

The mean absolute error we achieved is 4.83. The XGBoost model is saved as a pickle file and would then be used as input for the next step: optimization.

3 Proposed Solutions

Because this problem involves minimizing travel time while also visiting every location, it is analogous to the Traveling Salesman Problem (TSP). The Traveling Salesman Problem is a combinatorial NP hard optimization problem. This means the problem does not have a polynomial time solution for it. The possible solutions grow at a *factorial* rate the more locations are added. A brute force approach to solving this problem would involve checking every possible combination of locations. If we were to evaluate every possible path, where n represents the number of locations, it would have the following running time:

$$O(n) = \frac{1}{2}(n-1)! \tag{1}$$

Given this running time, the solution space quickly becomes intractable once we start adding more locations to our problem definition. Therefore, we employ the use of meta-heuristics to search the solution space directly. Rather than going through each possible path, we will generate possible paths and search them in a guided stochastic manner.

3.1 Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm is a biologically inspired meta-heuristic that searches the solution space in a way that emulates the way ants search possible paths. Ants leave *pheromones* on their travel path, depending on the path quality. Since we are trying to minimize our travel cost, we will have our pheromone quality $\tau \propto$ lower travel times. Each edge in our complete graph will have a corresponding inverse travel time cost η and pheromone τ.

With each generation, a set number of ants are randomly placed at starting points in the graph. Each ant then builds a solution by traversing the graph. It does so by choosing the next location with a weighted probability. The equation to calculate the probability of going from location i to location j is:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{h \in E}[\tau_{ih}]^{\alpha}[\eta_{ih}]^{\beta}} \tag{2}$$

Where τ represents an $n \times n$ pheromone matrix, and η represents an $n \times n$ inverse travel time cost matrix. All values in the pheromone matrix are first initialized to $\frac{1}{n^2}$. We found that initializing the pheromone matrix with 1's had little to no impact on our results. The corresponding exponents α and β are used to control the influence of the pheromones and travel time cost in the probability

calculation between two nodes. In the denominator, we have a summation of these values where $h \in$ represents all possible locations which are available for our ant to visit. These are the nodes that particular ant has not visited yet. These terms are all added together to calculate the weighted probability of traveling from node i to node j.

After each ant generates their corresponding solution, the pheromone matrix is updated. How this is done depends on the strategy employed. In our solution, we update each pheromone edge by first multiplying it by ρ, or the residual coefficient. This represents the rate at which the pheromones "evaporate" or decrease their influence. We then add to the pheromone edge $\frac{q}{C}$ where q represents the pheromone intensity and C represents the total cost of the generated path. Because the travel time cost is in the denominator, paths with greater travel times will have a lower probability of being picked. This process is repeated for g number of times, where g represents the number of generations.

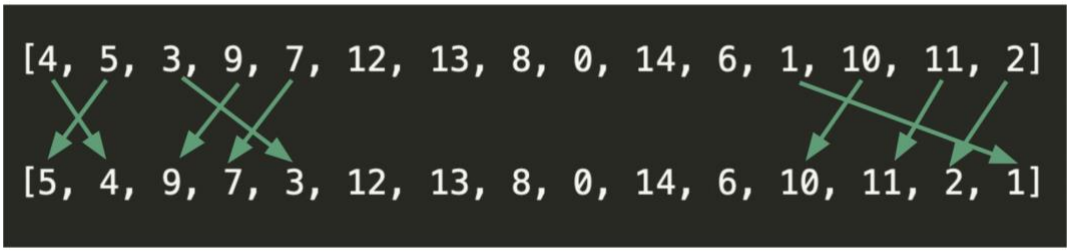
3.2 Genetic Evolution

Genetic Evolution (GE) is another biologically inspired meta-heuristic that takes its steps from the process of evolution. It makes use of mutation, crossover, and selection functions over successive generations. These generations make up a population of solutions. This optimization algorithm follows the following steps:

1. Create a population of routes
2. Mutate
3. Crossover
4. Determine the fitness (travel time)
5. Select parent for next generation
6. *Repeat from step 2*

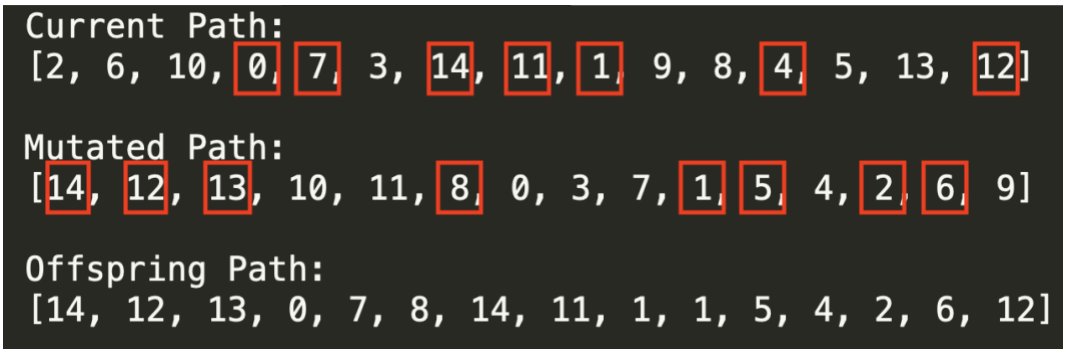
An initial population is created by randomly generating n number of paths. Our mutation and crossover operations are based on mutation rate m and crossover rate c. Both of these values are between 0 and 1 inclusively. When iterating over each path in the current population, we randomly pick a different path and mutate it. Mutation is done by iterating over every location in the path and randomly swapping it with the previous index's location. This random swap is weighted by the mutation factor m.

```
Original Path:
[4, 5, 3, 9, 7, 12, 13, 8, 0, 14, 6, 1, 10, 11, 2]
```

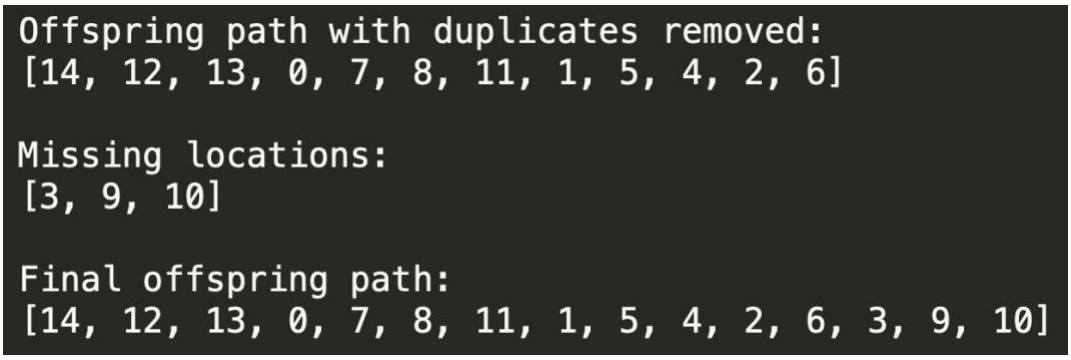



```
Mutated Path:
[5, 4, 9, 7, 3, 12, 13, 8, 0, 14, 6, 10, 11, 2, 1]
```

After we have created our mutant, we then begin crossover. We iterate over each index in both our current and mutant path and select a location based on the crossover rate c:



Order matters in travel paths. It is not enough for us to end our crossover operation here. As shown in the figure above, the locations 1 and 12 appears more than once. Because of this, we need to remove duplicates. We then find the missing locations, shuffle them, and append them to the path:



Once we have the final offspring path, we can evaluate its fitness compared to the current path's. If it has a lower travel time, it is then chosen as the parent for the next generation, replacing the current route. This process is repeated for g number of times, where g represents the number of generations.

4 Experimental Results

In our trial runs of each optimization algorithm we found that Ant Colony Optimization performed better than Genetic Evolution. This is to be expected, as the Ant Colony Optimization algorithm was specifically designed to solve the Traveling Salesman problem. We used $n = 15$ or fifteen different locations for each trial with both algorithms.

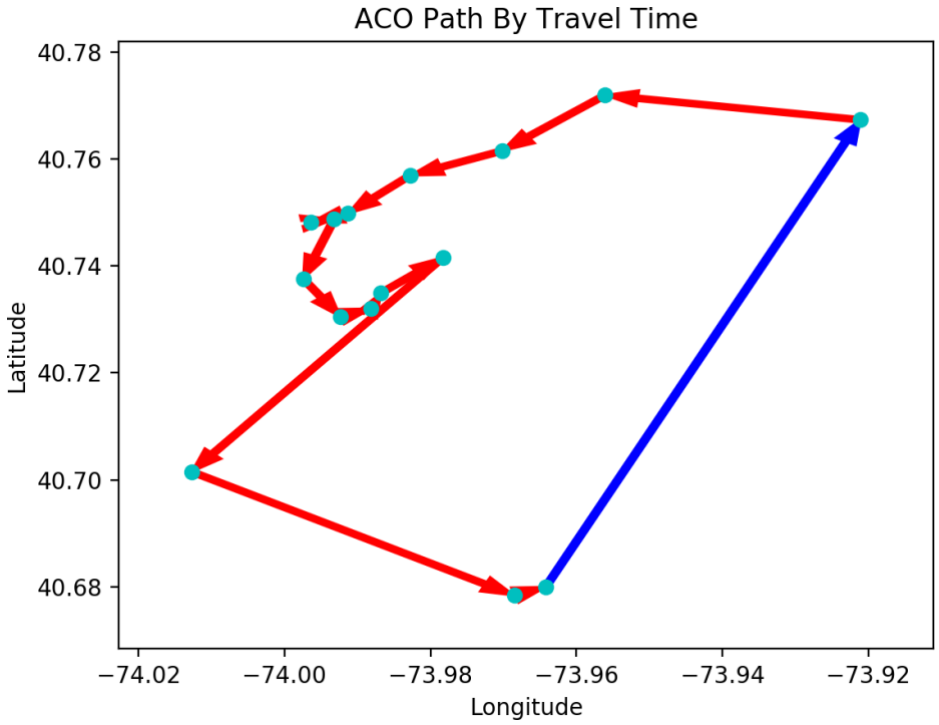
4.1 Ant Colony Optimization Results

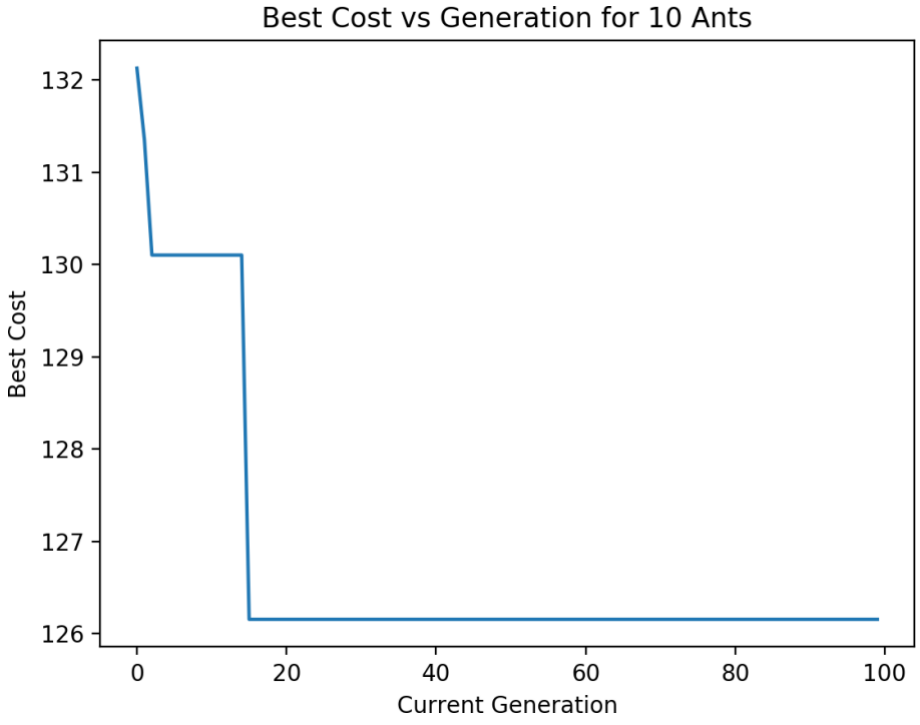
For the trials of Ant Colony Optimization, we used the following values: $\alpha = 1.0$, $\beta = 10.0$, $\rho = 0.5$, and $q = 10.0$ while we varied the number of ants and generations g.

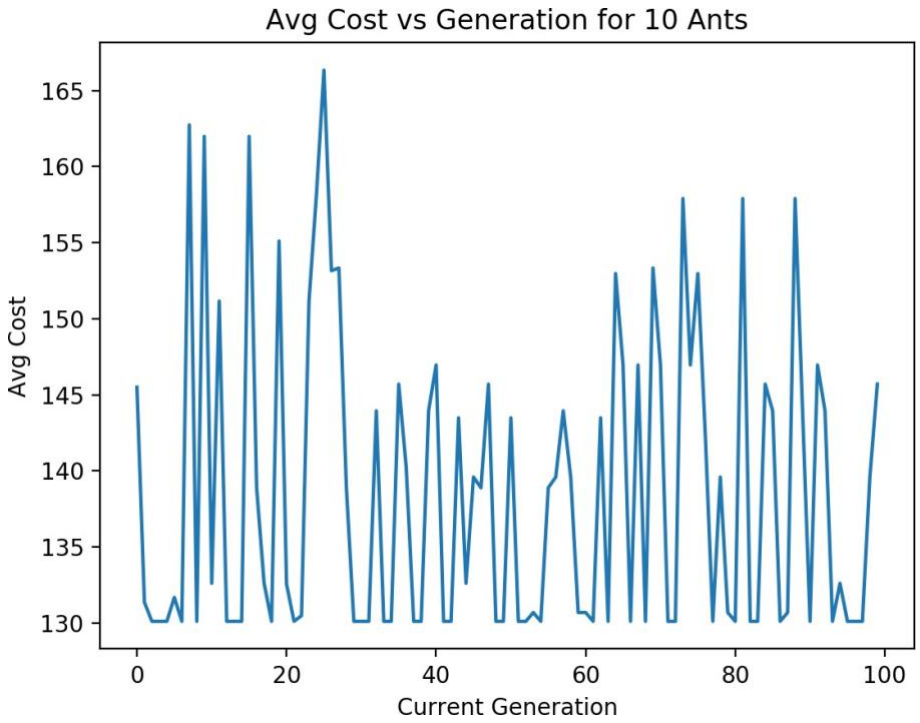
Table 1: Ant Colony Results

Ants	g	Mean Cost	SD Cost
10	100	126.51	0.88
10	200	128.31	0.81
10	300	**126.18**	0.33
10	400	129.53	0.74
10	500	126.19	0.60
20	100	128.01	**0.14**
20	200	127.44	1.63
20	300	126.49	1.04
20	400	126.27	0.60
20	500	126.29	0.58

The graphs below were produced with 10 ants and $g = 100$. As you can see from the average cost per generation graph, ACO was not stuck at a local minima. It continued to explore other possible solutions. This is evident by its wide variation in mean cost.





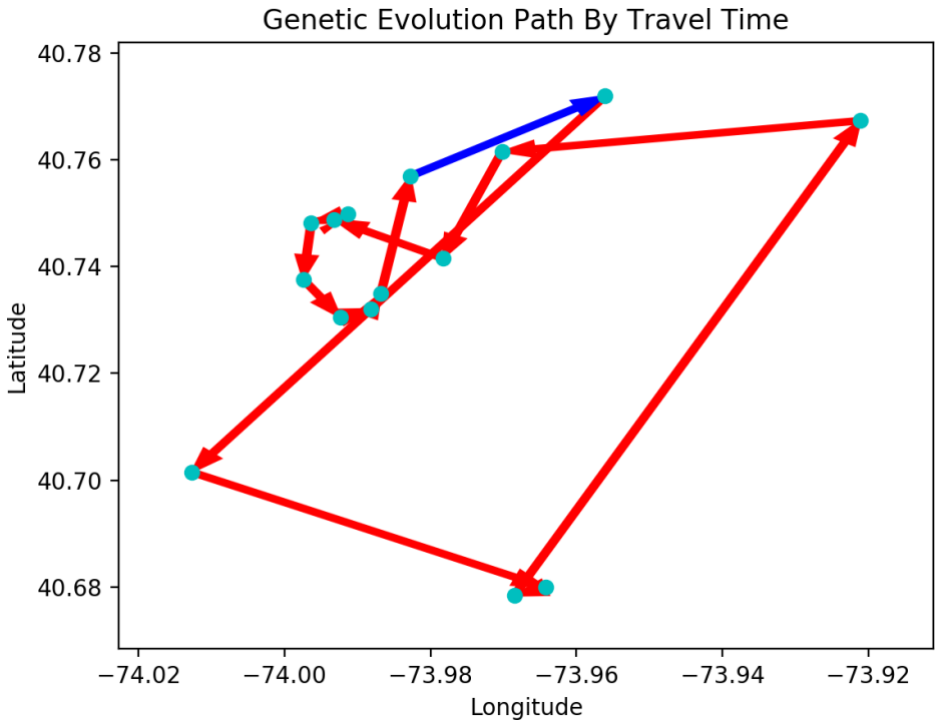


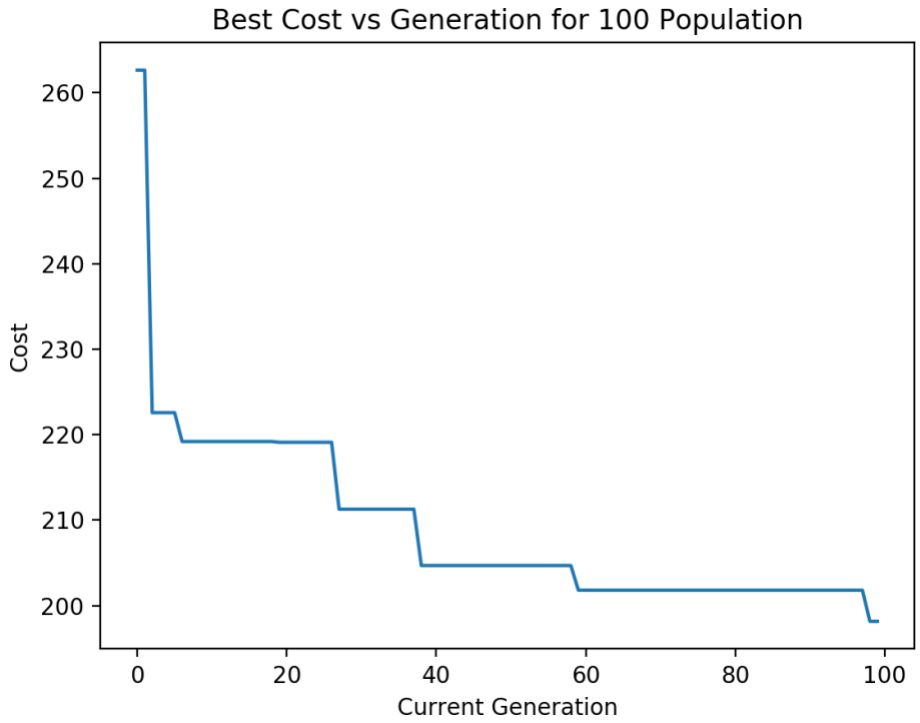
4.2 Genetic Evolution Optimization Results

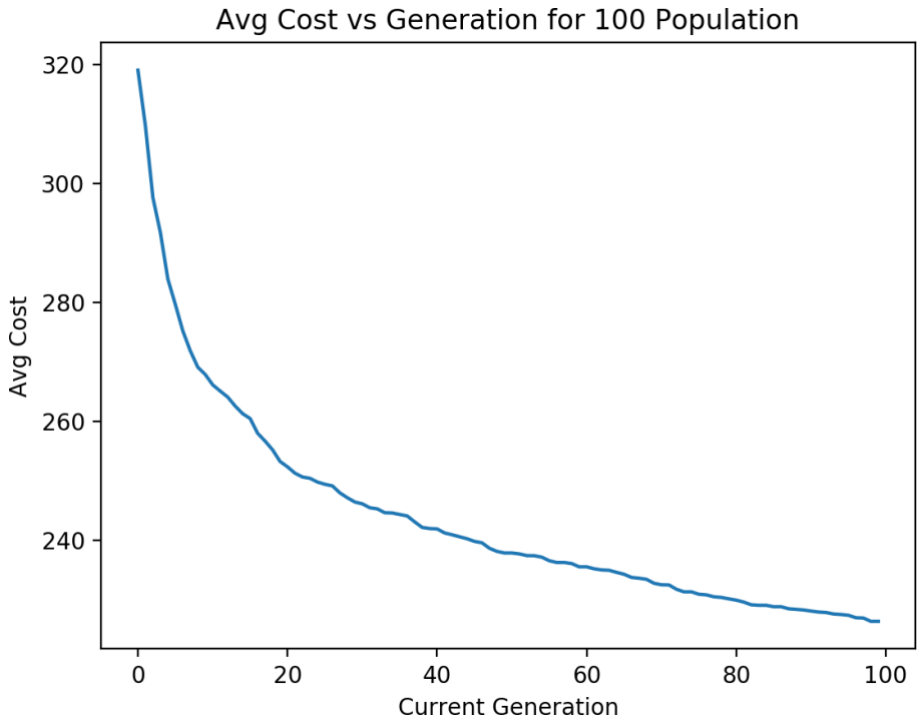
For the trials of Genetic Evolution Optimization, we used the following values: mutation rate $m = 0.5$, crossover rate $c = 0.7$ while we varied the size of population p and the number of generations g. From the results we can see the more generations and the larger the population, the better the solution the algorithm finds. However, compared to Ant Colony Optimization, this solution is still inferior.

Table 2: Genetic Evolution Results

p	g	Mean Cost	SD Cost
100	100	210.62	14.29
100	200	194.48	11.04
100	300	199.62	12.01
100	400	196.95	**8.98**
100	500	190.39	14.48
100	100	207.88	11.48
200	200	197.59	9.69
200	300	191.99	9.98
200	400	187.17	12.27
200	500	**184.14**	14.66


Genetic Evolution Path By Travel Time
40.78
40.76
40.74
40.72
40.70
40.68
Latitude
−74.02
−74.00
−73.98
−73.96
−73.94
−73.92
Longitude


Best Cost vs Generation for 100 Population
260
250
240
230
220
210
200
Cost
0
20
40
60
80
100
Current Generation



5 Discussion

It is clear from the experimental results that Ant Colony Optimization is the better algorithm for optimizing our problem, which is analogous to the Traveling Salesman Problem. Regardless of which parameters we used for population p, mutation rate m, or crossover rate s in our Genetic Evolution program, it was unable to find paths with costs as low as ACO. We believe multiple factors caused this.

First, consider how the solution paths are generated in both algorithms. In ant colony, the travel time cost η_{ij} is factored into *each decision* an ant makes. This is because the inverse travel time matrix is used in the probability calculation for determining which location an ant should visit next. Genetic evolution only evaluates cost after it has applied its mutation and crossover operation. The travel cost has no impact on this stochastic component of genetic evolution. Mutations and crossovers are done randomly and, although influenced by crossover and mutation rate, does not consider whether these changes give us a lower value. Only at selection do we check whether or not the offspring has a better value.

This would explain why ant colony not only performs better, but also why it does so in such a marked manner, such that its first generation produces a result more than a hundred minutes faster than genetic evolution. Another factor to consider is how we implemented our mutation and crossover operations. There may be better ways to implement them which also takes into account the cost of the path before selection.

It is interesting to see the average cost of each generation for ant colony. It shows us that the algorithm *did not* get stuck at a local minima. Rather, it was still searching for various other paths, none of which were better than what it found. Since the ants' decisions are based on a weighted probability, this makes sense. Comparing this to genetic evolution's average cost graph tells us a story about how that algorithm is likely to converge. The trend we see with that curve is a decreasing slope that looks soon to level off.

6 Conclusion

While the traveling salesman problem itself may be old, its application to modern day services like Uber and Lyft reminds us of its relevance and ugly factorial running time. With the recent advancements of biologically inspired optimization algorithms, we can still find feasible solutions without having a polynomial time solution.

There is future work to be done for genetic evolution optimization in this problem domain. The challenge will be in finding a more effective way to factor in the path cost into the mutation and crossover operations. But as the results of this paper indicate, we affirm that ant colony optimization truly was made to solve this kind of problem.

References

[1] Bell, John E. & McMullen, Patrick R. (2004) Ant colony optimization techniques for the vehicle routing problem. In *Advanced Engineering Informatics 18*, pp. 41–48. Elsevier.

[2] Bertsimas, Dimitris & Jaillet, Patrick & Martin, Sebastien. (2018) Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications. Accepted for publication in *Operations Research*. Operations Research Center, Massachusetts Institute of Technology. Cambridge, MA.

[3] Jia, Yongzheng & Xu, Wei & Liu, Xue. (2016) An Optimization Framework For Online Ridesharing Markets. *arXiv preprint arXiv:1612.03797.*

[4] Lin, Yeqian & Li, Wenquan & Qiu, Feng & Xu, He. (2012) Research on Optimization of Vehicle Routing Problem for Ridesharing Taxi. 8th International Conference on Traffic and Transportation Studies, Changsha, China, August 1–3, 2012. *Procedia - Social and Behavioral Sciences* **43**: 494-502.

[5] Ma, Changxi & He, Ruichun & Zhang, Wei (2018) Path optimization of taxi carpooling. PLoS ONE 13(8): e0203221 *https://doi.org/10.1371/journal.pone.0203221*

[6] Xiong, Naixue & Wu, Wenliang & Wu, Chunxue (2017) An Improved Routing Optimization Algorithm Based on Travelling Salesman Problem for Social Networks. *Sustainability* 9, 985. *doi:10.3390/su9060985*

Appendix

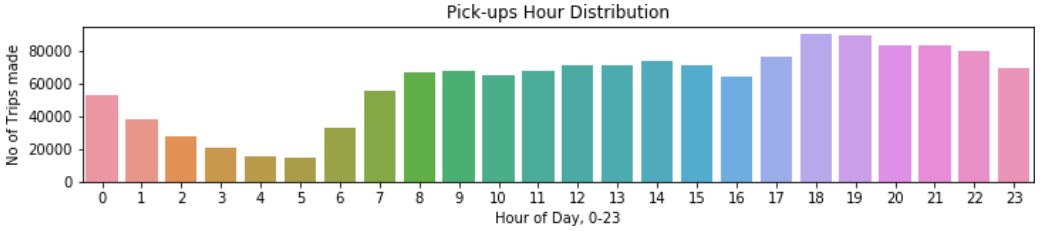


Figure 1: Pickups Hour Distribution

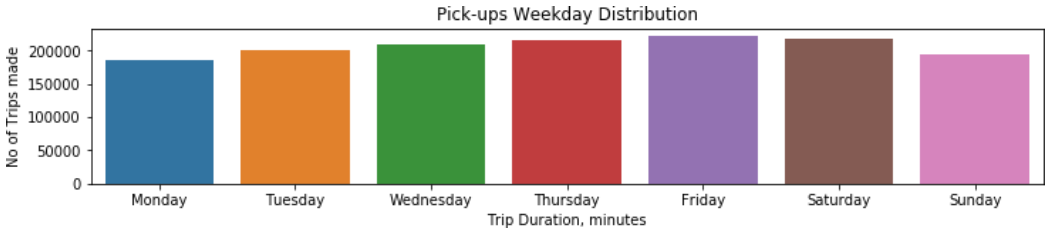


Figure 2: Pickups Weekday Distribution

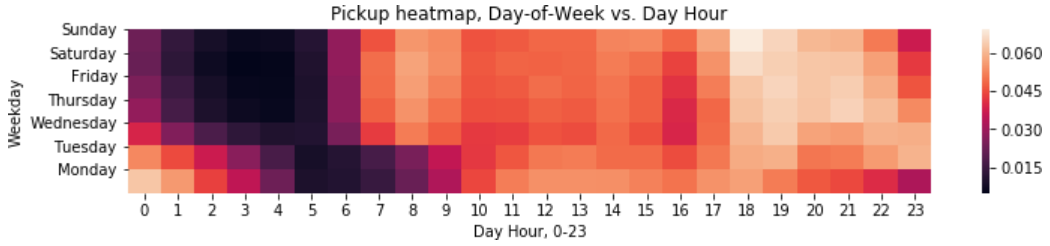


Figure 3: Pickups Heatmap Hours Versus Weekdays

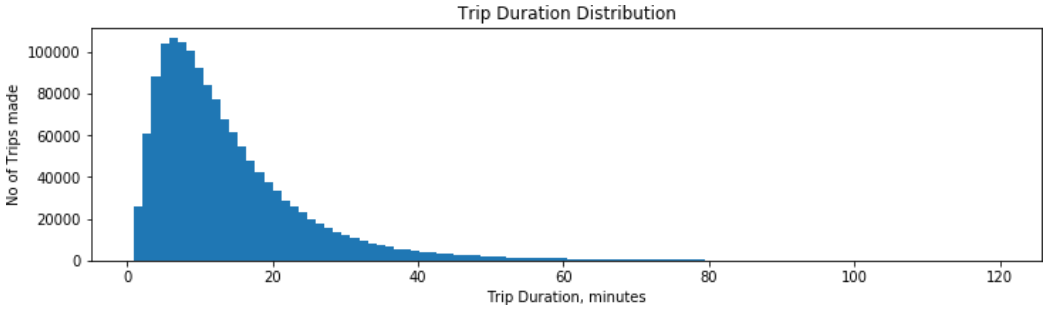


Figure 4: Trip Duration Distribution By Minutes

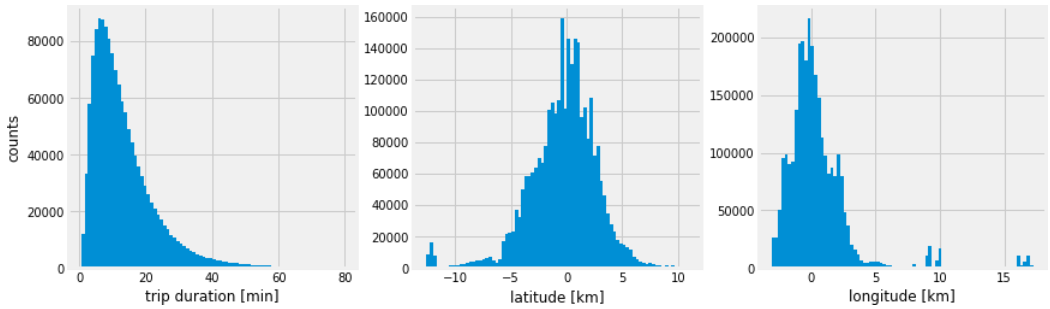


Figure 5: Trip Duration Histogram By Minutes, Latitudes, and Longtitudes

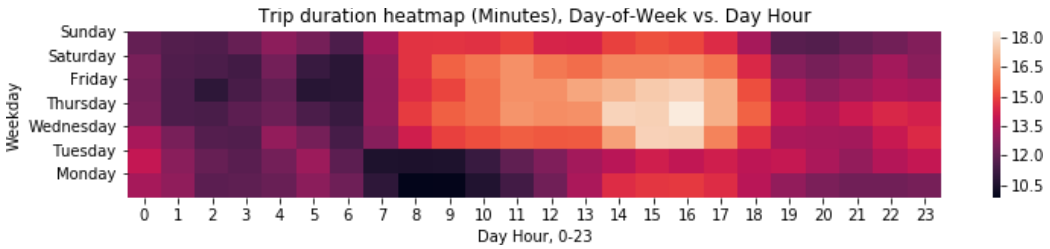


Figure 6: Trip Duration Heatmap in Minutes Hours Versus Weekdays

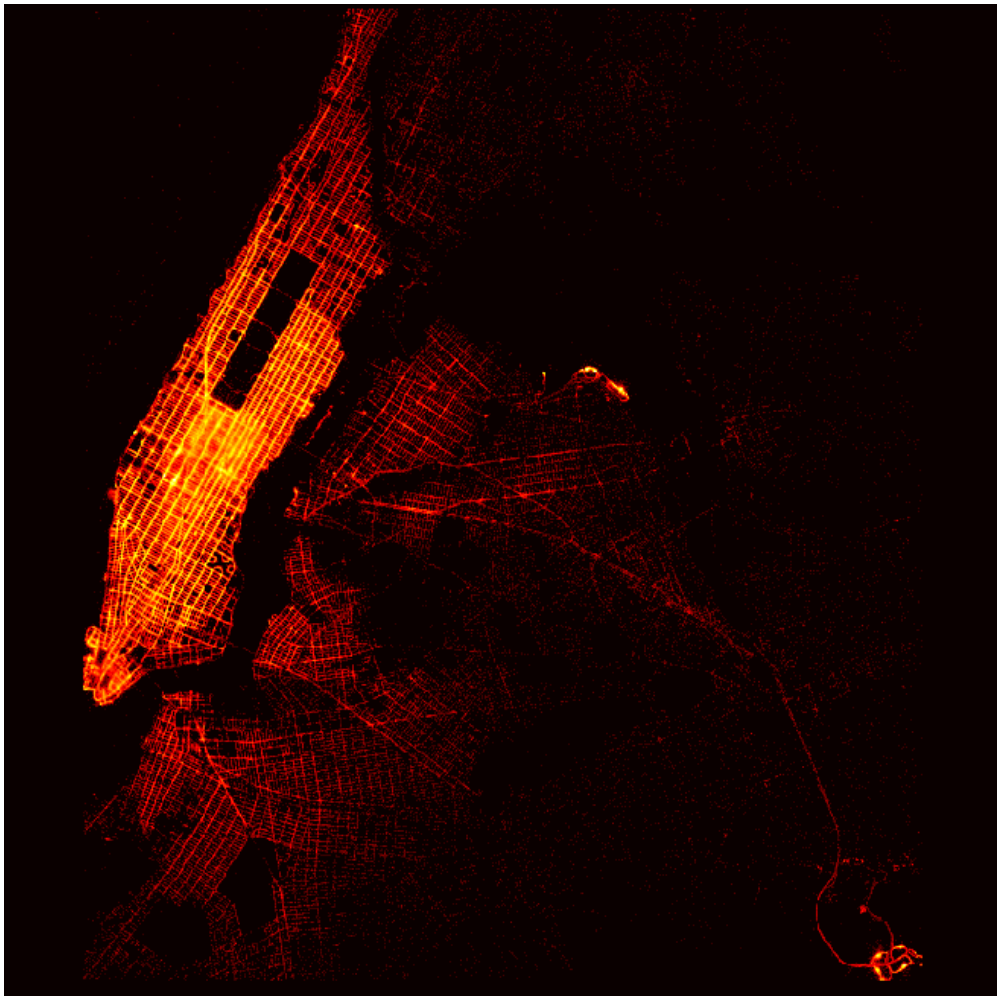


Figure 7: Spatial density plot of the pickup and dropoff locations

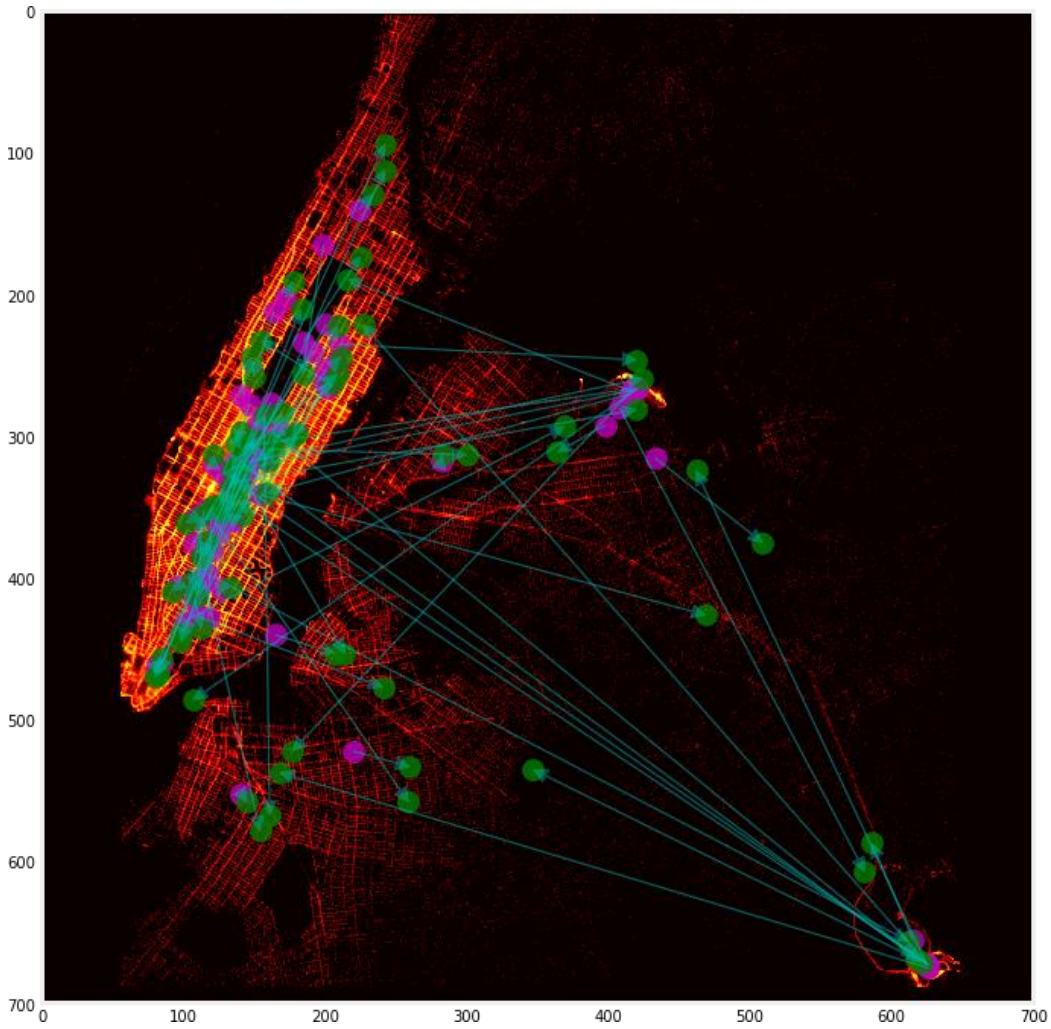


Figure 8: Typical trips on the New York City Map