Spring 2019-20

MT21104

Genetic Algorithms in Engineering Process Modelling

Travel Time Optimization Problem using Genetic Algorithm

Submitted by:

Suyash Namdeo: 17CE10059

Harsh Vardhan: 17CE10022

Contents:

Note: The whole code files are uploaded in my GitHub repo:
https://github.com/YashNamdeo/Genetic_project_MT21104

Motivation:

For our course project, we are interested in solving the travel time optimization problem for taxi vehicles as we are highly interested in **Intelligent Transportation System**. This problem is very relevant given the rapid and significant evolution that is happening in urban transportation. The emergence of Internet-connected smartphones has allowed us to plan and optimize our daily commute. Today, ride-sharing companies like Uber and Lyft are using such technologies to gather large amounts of data and improve the efficiency of transportation systems, laying the ground for a more connected and centrally controlled transportation structure. This problem is related to vehicle routing - the optimization of each vehicle actions to maximize the system efficiency and throughput. By optimizing the time travel, we can decide which vehicle to assign to each ride request. The vehicle routing problem is a combinatorial optimization problem where the number of feasible solutions for the problem increases exponentially with the number of customers to be serviced. As such, it is relevant to our class in a sense that we can use evolutionary / nature-inspired algorithms to generate possible solutions.

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 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 the travel time optimization problem for taxi vehicles is tackled. 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 machine learning is used to forecast travel times between every pair of pickup and drop-off locations. Then evolutionary algorithms, namely ant colony and genetic are used, to find the best travel itinerary for the vehicles in the dataset.

Data Preprocessing:

Simple data pre-processing techniques such as eliminating duplicates, checking latitude and longitude bounds, removing obvious outliers, converting data fields to sensible units, etc. are done to get a clean dataset for the experiments.

Dataset Description:

The data that we worked 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 2016 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
- **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
- **trip_duration** — duration of the trip in seconds (also the target variable)

Data Visualization:

A big component of this 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.

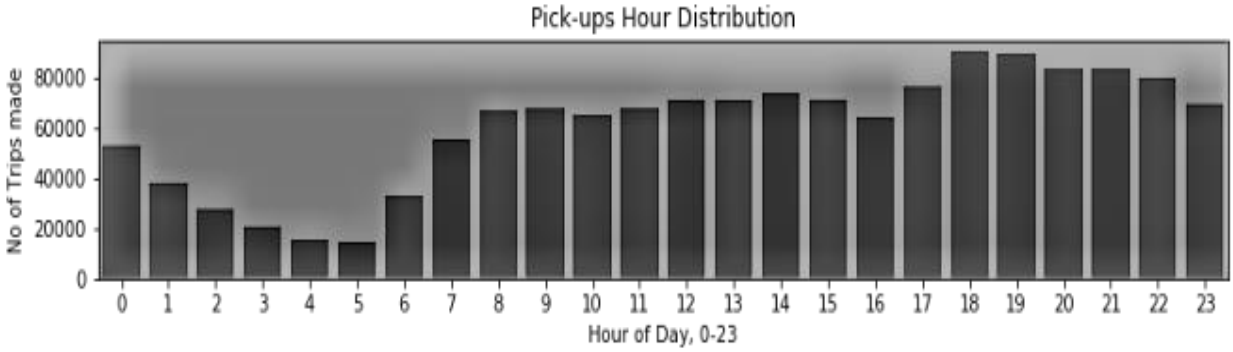


Fig 1

Fig 1 is the pickup time distribution by hours of the day. 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.

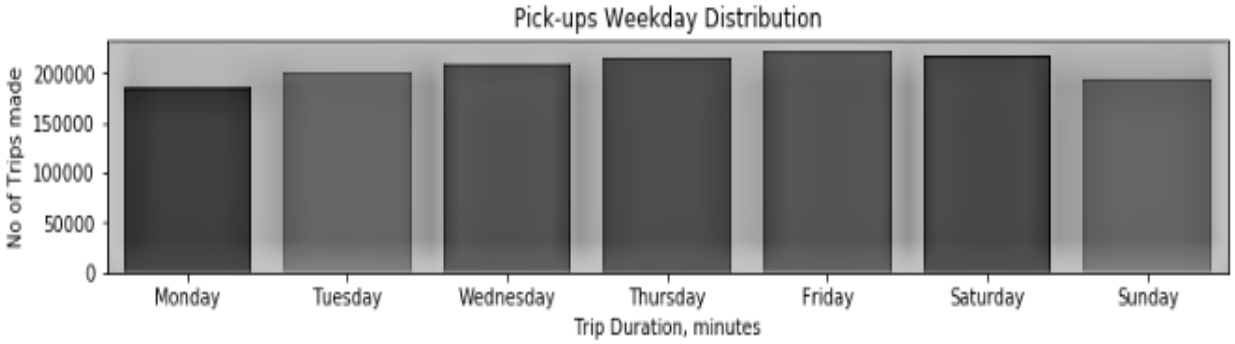


Fig 2

Fig 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.

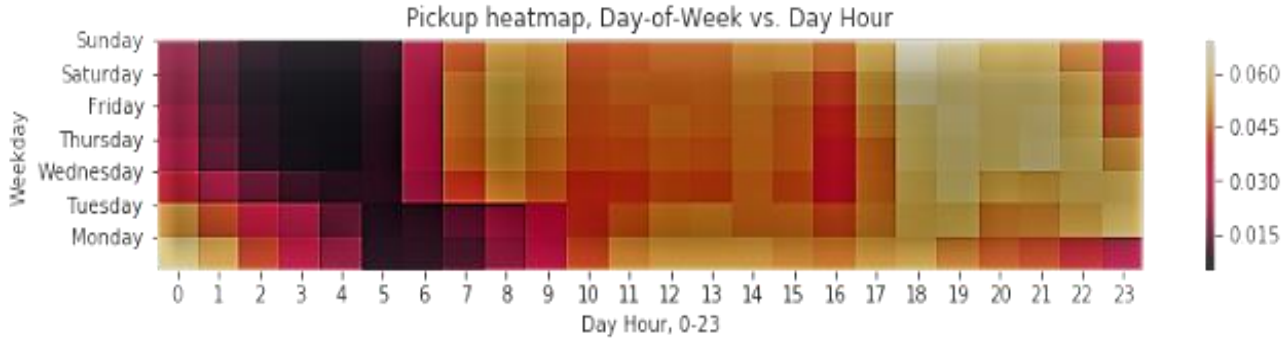


Fig 3

Fig 3 is the pickup time heat-map for the results from Fig 1 and Fig 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 important feature is **Trip Duration**.

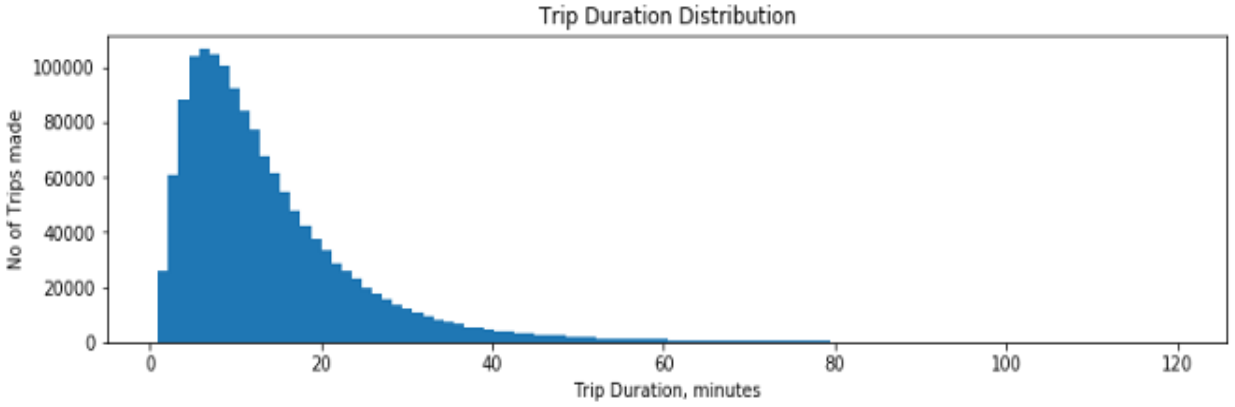


Fig 4

Fig 4 is the trip duration distribution by minutes. I 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.

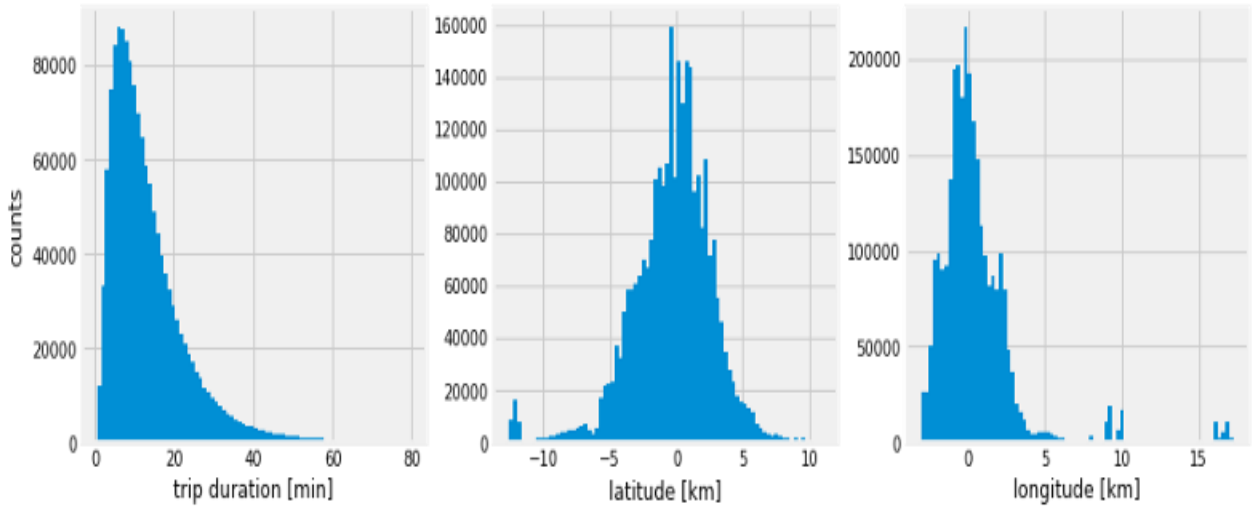


Fig 5

Fig 5 displays the comparisons of trip duration distribution by minutes versus by latitudes and longitudes. The distribution by longitude is also quite left-skewed, with a peak roughly at 1 kilometer. The distribution by latitude is more normally distributed, with a peak at 0 kilometers.

Machine Learning:

The next step is to build a machine learning model to predict travel times between pickup and drop-off locations. With the 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, Gradient boosted trees could be an appropriate model for this data, which can easily capture nonlinear relationships, accommodate for complexity, and handle categorical features. The input features that were used are: **passenger_count, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude,** and **store_and_fwd_flag**. The output target is **trip_duration**.

This model is implemented 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. Root mean squared logarithmic error is used as the evaluation metric, as it reduces error magnitude. The final parameters for the XGBoost model are: *{fbooster = gbtree; objective = linear; learning _rate = 0.05; max _depth = 14; subsample = 0.9; colsample _bytree = 0.7; colsample _bylevel = 0.7; silent = 1; feval = rmsleg}*.

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**.

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 I was to evaluate every possible path, where n represents the number of locations, it would have the following running time:

O(n) = [(1/2) (n — 1)!]

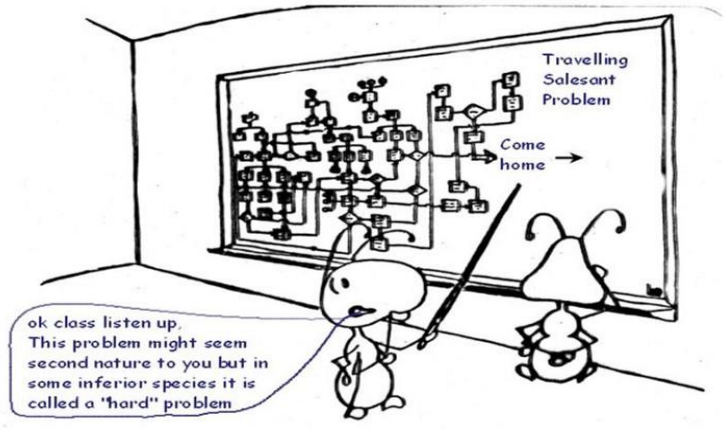
Given this running time, the solution space quickly becomes intractable once more locations are added to the problem definition. Therefore the use of meta-heuristics to search the solution space is employed directly. Rather than going through each possible path, possible paths are generated and search them in a guided stochastic manner.

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 for possible paths. Ants leave pheromones on their travel path, depending on the path quality. Since the aim is to minimize the travel cost, hence **pheromone quality tau** is used as an equivalent to lower travel times. Each edge in the complete graph will have a corresponding inverse **travel time cost eta** and **pheromone tau**. 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:

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

Ant Colony Optimization

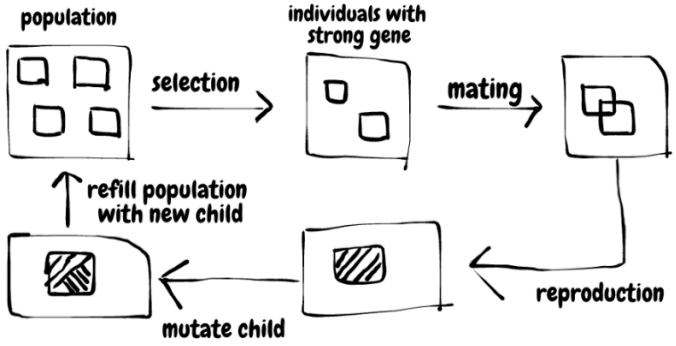


Prepared by:
Ahmad Elshamli, Daniel Asmar, Fadi Elmasri

Where **tau** represents an n by n pheromone matrix and **eta** represents an n by n inverse travel time cost matrix. All values in the pheromone matrix are first initialized to $1/n^2$. It is found that initializing the pheromone matrix with 1's had little to no impact on the results. The corresponding exponents **alpha** and **beta** are used to control the influence of the pheromones and travel time cost in the probability calculation between two nodes. In the denominator, there is a summation of these values where **h** represents all possible locations that are available for the 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 this solution, each pheromone edge is updated by first multiplying it by **p** (the residual coefficient). This represents the rate at which the pheromones "evaporate" or decrease their influence. We then add to the pheromone edge **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.

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 the parent for the next generation
6. *Repeat step 2*

An initial population is created by randomly generating n number of paths. The 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, a different path is randomly picked and mutated. The 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]
```

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

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

After creating the mutant, crossover is initiated by iterating over each index in both the current and mutant path and select a location based on the crossover rate c:

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

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

Offspring Path:
[14, 12, 13, 0, 7, 8, 14, 11, 1, 1, 5, 4, 2, 6, 12]
```

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

```
Offspring path with duplicates removed:
[14, 12, 13, 0, 7, 8, 11, 1, 5, 4, 2, 6]

Missing locations:
[3, 9, 10]

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

Once the final offspring path is obtained, its fitness is evaluated 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.

Results:

In the trial runs of each optimization algorithm, it is 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.

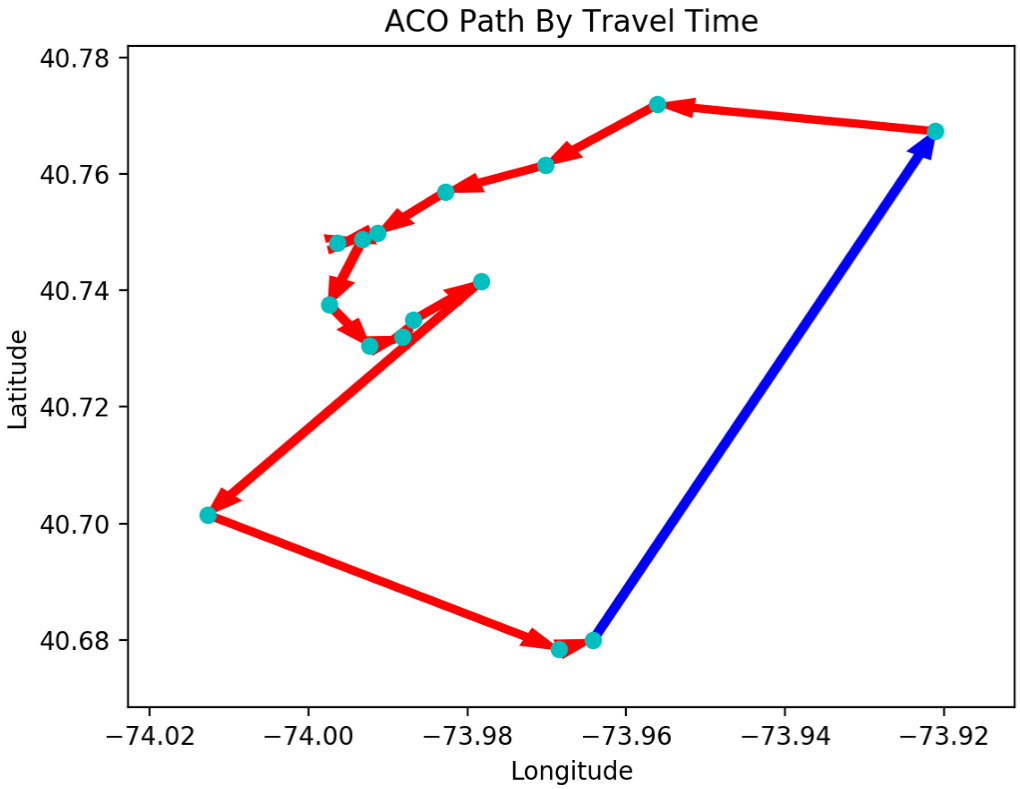
ACO Results:

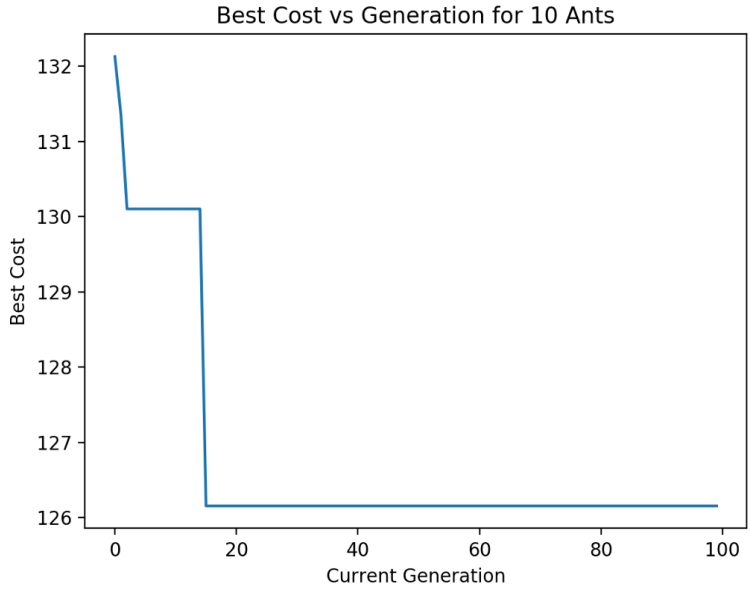
For the trials of Ant Colony Optimization, the following values are used: *alpha = 1.0, beta = 10.0, p = 0.5, and q = 10.0* while the number of ants and generations g are varied.

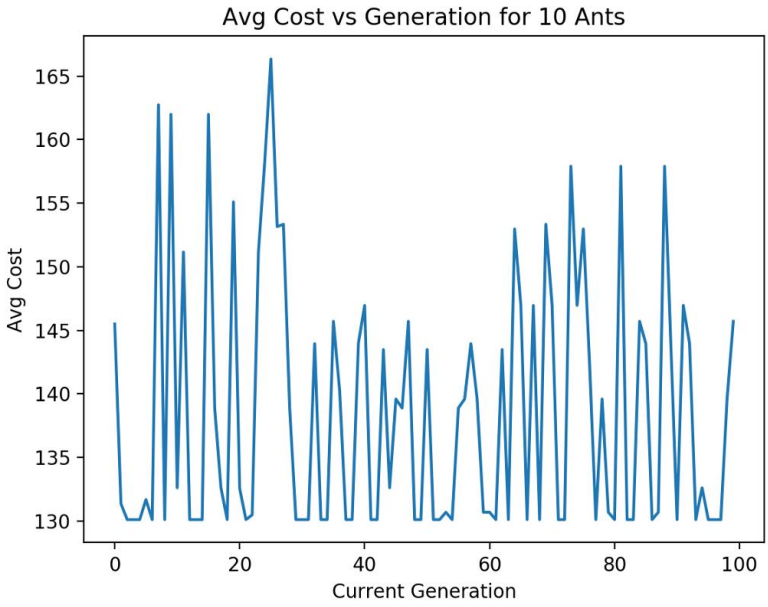
Table 1: Ant Colony Results

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 minimum. It continued to explore other possible solutions. This is evident by its wide variation in mean cost.





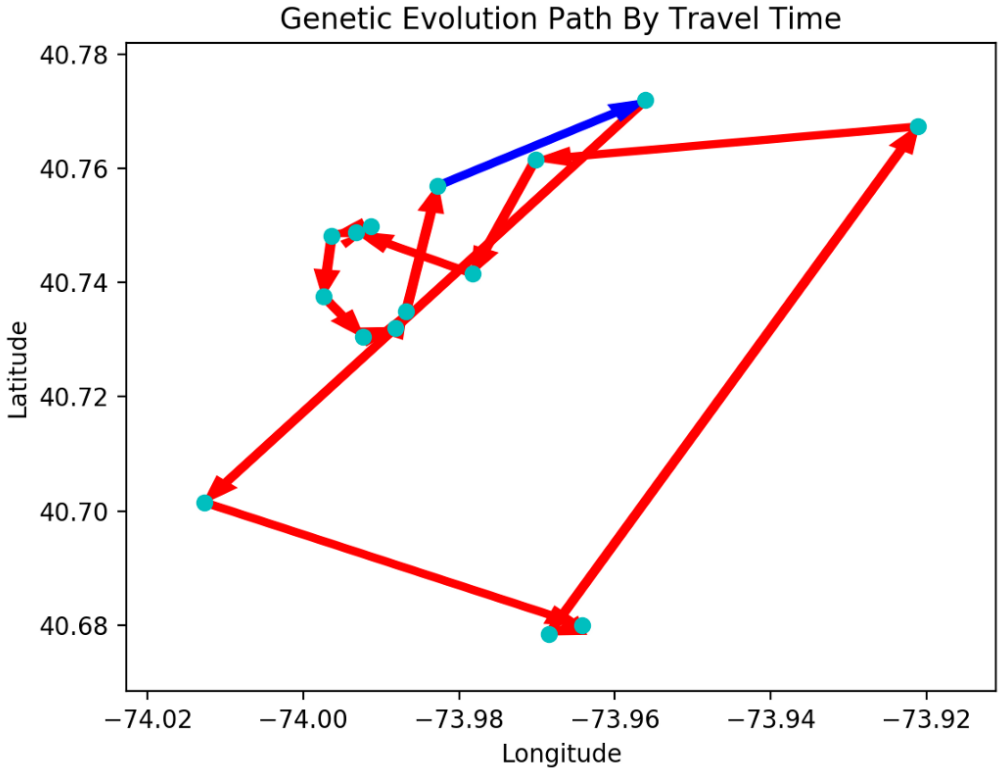


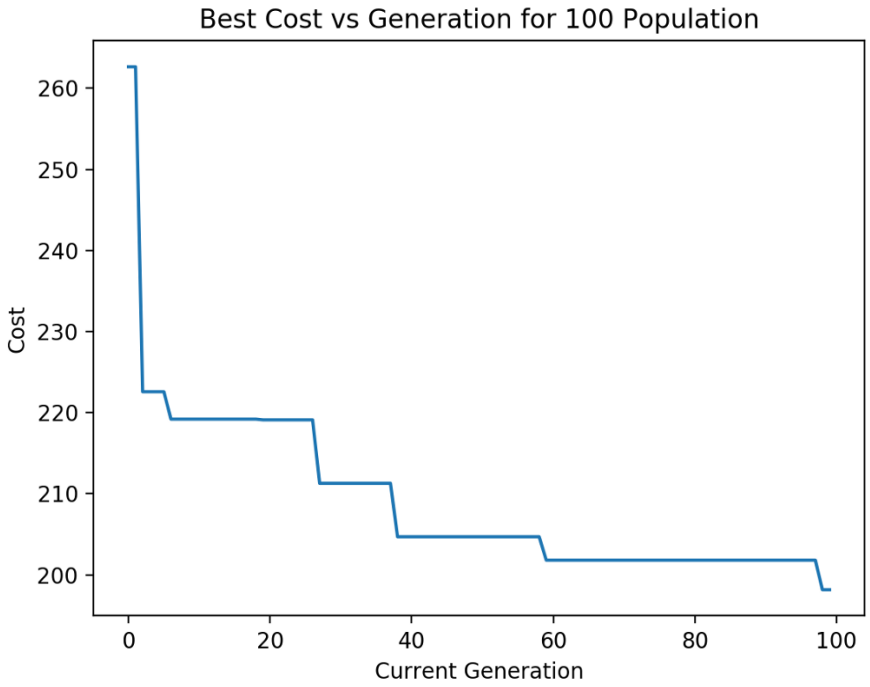
GE Results:

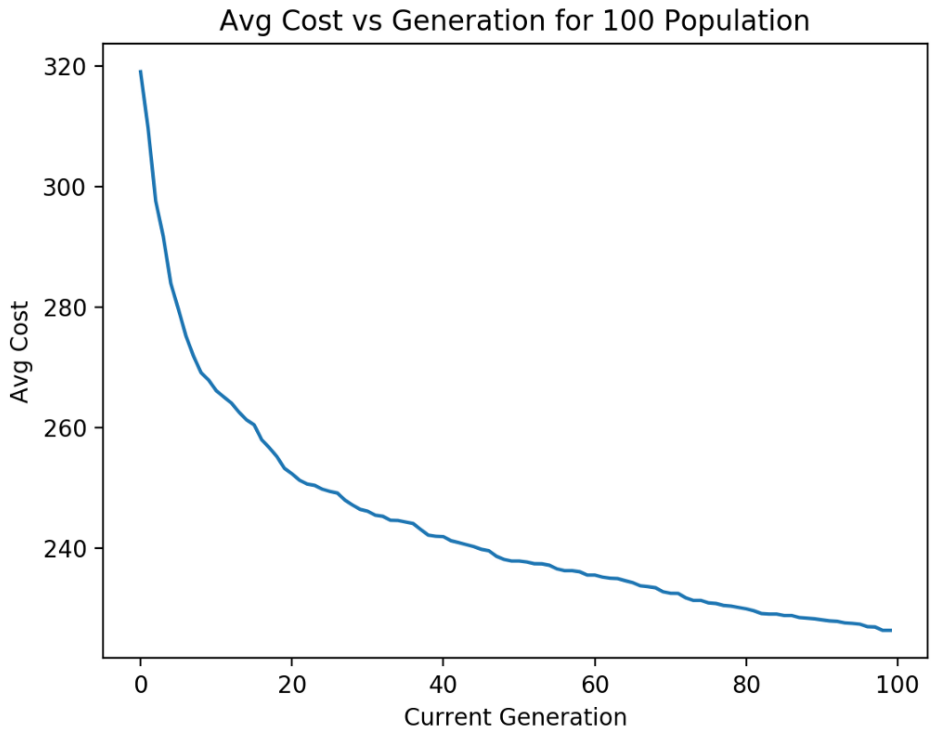
For the trials of Genetic Evolution Optimization, the following values are used: *mutation rate m= 0.5, crossover rate c = 0.7* while the size of population p and the number of generations g are varied. From the results, it can be seen the more generations and the larger the population, the better the 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




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


Avg Cost vs Generation for 100 Population
Avg Cost
320
300
280
260
240
0
20
40
60
80
100
Current Generation

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 used for **population p**, **mutation rate m**, or **crossover rate s** in the Genetic Evolution program, it was unable to find paths with costs as low as ACO. Multiple factors could cause this.

First, consider how the solution paths are generated in both algorithms. In the ant colony, the **travel time cost n_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 the selection phase do we need to check whether or not the offspring has a better value.

This would explain why the 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 mutation and crossover operations are implemented. 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 the ant colony. It shows that the algorithm did not get stuck at a local minimum. 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 can be seen with that curve is a decreasing slope that looks soon to level off.

Conclusion:

While the traveling salesman problem itself may be old, its application to modern day services like Uber and Lyft reminds me 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 project indicate, it can sure that ant colony optimization truly was made to solve this kind of problem.

Note: You can view the complete code for this project from my GitHub repo: https://github.com/YashNamdeo/Genetic_project_MT21104

References:

https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

https://en.wikipedia.org/wiki/Travelling_salesman_problem

https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

https://towardsdatascience.com/data-visualization-for-machine-learning-and-data-science-a45178970be7

https://towardsdatascience.com/travel-time-optimization-with-machine-learning-and-genetic-algorithm-71b40a3a4c2