Itinerary optimization for multi locations travel using Machine Learning (XGBoost) and Genetic Algorithm

Md Touhid Hossain
PhD student, Civil and Environmental Engineering
University of Wisconsin-Milwaukee

Table of Contents

I. Introduction

'Itinerary optimization' is the process of finding the best travel path for visiting multiple destinations (Sykes, 2022). Route planning and scheduling vehicles for any transportation network is very important in the area of transportation planning and engineering (Yu, 2014). Generally, we consider the shortest route as the best. But in real world an optimal route can be found depends on different road/travel path conditions. Each road comes with traffic, bridges, tunnels, tolls, speed limits and many other factors that affect the suitability of traveling on that road. So, we can say that the shortest path isn't always the best. Depending on what a business needs, the optimal route might be the one of the following:

- Route taking the least time.
- Route using the smallest number of vehicles.
- Route costing the least money.
- Route minimizing fuel use or carbon emissions.
- Route covering the shortest distance.

In this project, a program is developed to find an optimal itinerary for any multi locations travel using a Machine Learning (XGBoost) method and Genetic Algorithm. In following sections of this report, I will try to explain the motivation, data selection and processing, model selection, and finally implementation along with conclusions and future works.

II. Motivation

Route optimization is a vastly studied research topic in modern intelligent transportation system. While doing the literature review, I found a lot of mentionable works for route planning and

optimization. Here I am going to mention the most important ones which inspired me to do this project. Such as, the authors in (Xiao, 2017) proposed an optimal path planning and real-time forecasting method using ant colony algorithm. Another work by (Angskun & Angskun, 2009) proposes an e-tourism system emphasized on a travel planning optimization under the energy and time constraints in accordance with the tourist requirements. Finally, I found the idea 'Traveling salesman problem', and the idea to solve it using randomized bias genetic algorithm in (Gupta, 2017). This inspired me to use Genetic Algorithm for itinerary optimization while traveling multiple locations.

III. Objective and Execution Steps

The objective of this project is to find the optimal route based on the first criteria i.e., route that takes the least time. The tasks will be done in two steps:

- Prediction of travel time using Machine Learning (XGBoost) Model
- Optimize route selection using Genetic Algorithm (GA)

The figure below shows a flow diagram of steps in this project.

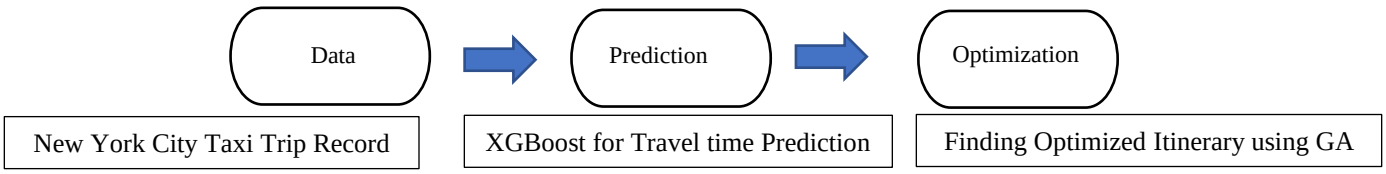

Figure 1: A flow diagram showing stepwise of work done.

IV. Data Selection and Processing

For itinerary optimization, we need historical travel data for many multiple locations travel. But most of the commercial organizations do not share their data publicly. While searching for publicly available data, I found that there is a publicly available dataset provided by New York City known as TLC Trip Record Data (*TLC Trip Record Data - TLC*, 2016). Also, I found a Kaggle dataset (*New York City Taxi Trip Duration*, 2016) of the same taxi trip data that was cleaned and need few pre processing steps before using it in our prediction model. So, I used that dataset in this project. I will give a brief description of the dataset and preprocessing done before going into the main model to predict travel time.

New York City Taxi Data

The dataset selected is a taxi trip record for the year 2016 with the following 11 data fields in it.

Data fields:

a) id - a unique identifier for each trip

b) vendor_id - a code indicating the provider associated with the trip record

c) pickup_datetime - date and time when the meter was engaged

d) dropoff_datetime - date and time when the meter was disengaged

e) passenger_count - the number of passengers in the vehicle (driver entered value)

f) pickup_longitude - the longitude where the meter was engaged

g) pickup_latitude - the latitude where the meter was engaged

h) dropoff_longitude - the longitude where the meter was disengaged

i) dropoff_latitude - the latitude where the meter was disengaged

j) 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; N=not a store and forward trip

k) trip_duration - duration of the trip in seconds

At first, I used ArcGIS to visualize the spread of the data and detect any abnormality in it, which is shown in figure 2 below.

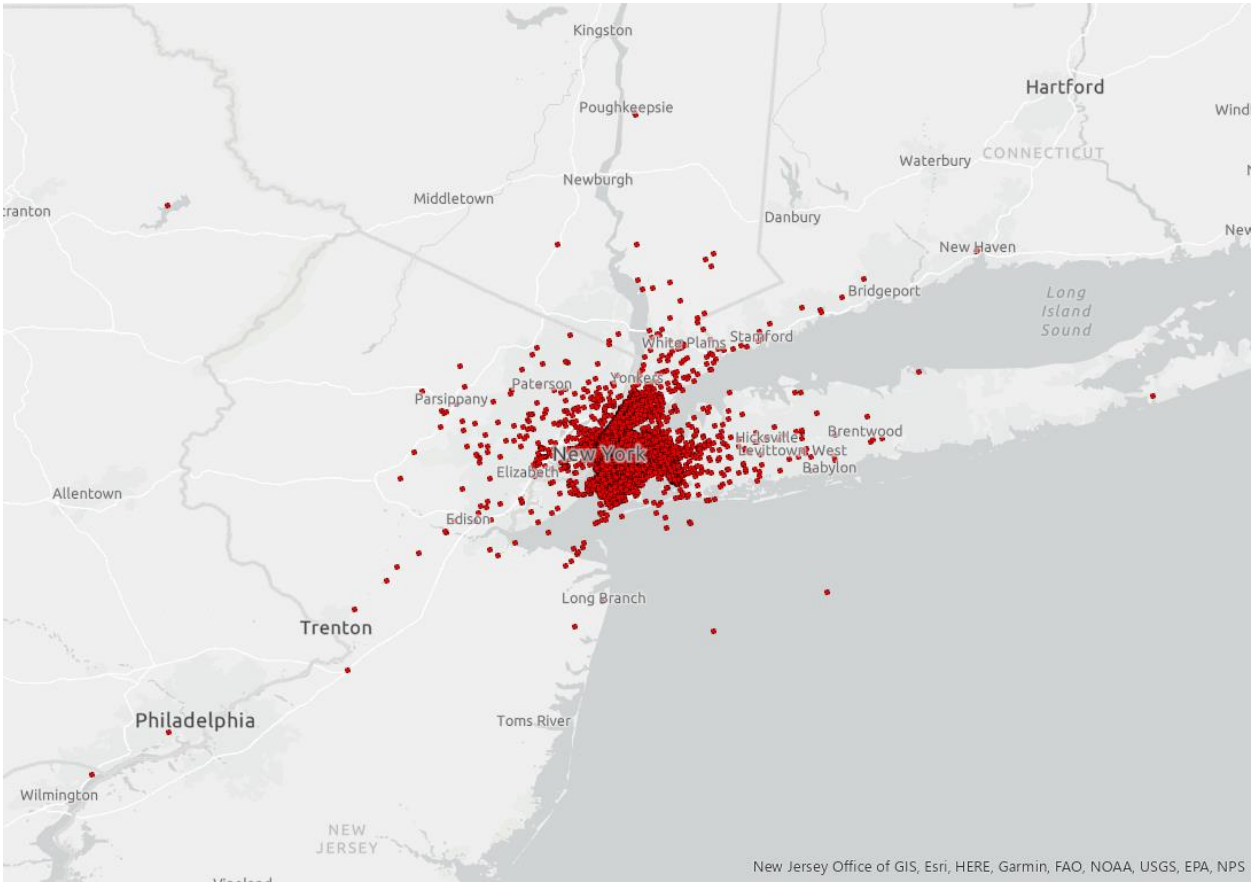

Figure 2: Visualizing spatial data in ArcGIS for outlier and anomaly detection.

Figure 2 shows that there are few outliers in the dataset. These outliers show that some drop off locations are in Canada, in the Pacific Ocean, or on the Ellis Island (Statue of Liberty), where cars simply don't go. But as those are very few, I kept them in the as removing them is a very difficult task. Also, the machine learning model we chose can deal very well with few outliers.

Data Processing

At first, the categorical data field store_and_fwd_flag is converted to a numerical data (1 for Y and 0 for N). From the location's coordinates which are given in the dataset, I calculated the Manhattan distances between each pair of points which is used as trip distance variable in the travel time prediction model. Also, I found the longitude and latitude differences to get a sense of direction (East to West, North to South). Next, the data field picup_datetime is converted into 5 different data fields, such as, pickup_month, pickup_weekday, pickup_hour, and pickup_minute. Finally trip duration is converted to minutes from seconds for easier interpretation. All these are done using python package panda and the following figure shows a screenshot with a part of the converted data fields.

trip_duration	pickup_month	pickup_day	pickup_weekday	pickup_hour	pickup_minute	latitude_difference	longitude_difference	trip_distance
8	3	14	0	17	24	-0.002335	0.017525	1.372146
11	6	12	6	0	43	-0.007412	-0.019066	1.829440
35	1	19	1	11	35	-0.053852	-0.026306	5.538397
7	4	6	2	19	32	-0.013252	-0.002228	1.069567
7	3	26	5	13	30	-0.010689	0.000130	0.747485

Figure 3: Screenshot of converted data fields before feeding into the Machine Learning prediction Model.

V. The prediction Model

The first step of the project is to predict a travel time between any two locations. For this task, a low variance prediction model with capability of handling categorical features was necessary. Choosing the right model is very important as a great deal of success of optimizing the itinerary depends on it. While searching properties of different prediction model, I found that in case of structured or tabular datasets on classification and regression predictive modeling problems, XGBoost perform better than most other (Brownlee, 2016). Keeping these factors in mind, I chose the tree-based model, Extreme Gradient Boosting (XGBoost) as the prediction model. How our model is trained, and the parameters are selected play a vital role for our optimization. These are discussed below.

Extreme Gradient Boosting (XGBoost) Parameters

Before running XGBoost, we must set three types of parameters: general parameters, booster parameters and task parameters (*XGBoost Parameters — Xgboost 1.6.1 Documentation*, 2022). **General parameters** relate to which booster we are using to do boosting, commonly tree or linear model. **Booster parameters** depend on which booster you have chosen. **Learning task parameters** decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks. The figure at the right is a screenshot of the parameters defined in my program. My objective function is linear regression, and I chose the others as default except evaluation metrics 'feval' as RMSLE (Root Mean Square Logarithmic Error).

```
#XGBoost parameters
params = {
    'booster':            'gbtree',
    'objective':          'reg:linear',
    'learning_rate':      0.05,
    'max_depth':          14,
    'subsample':          0.9,
    'colsample_bytree':   0.7,
    'colsample_bylevel':  0.7,
    'silent':             1,
    'feval':              'rmsle'
}
```

Figure 4: A screenshot of the parameters chosen for XGBoost during coding

Training the Model

After selection of the parameters, the XGBoost model is trained. The data was split into training and testing set for the training purpose. Also, I used evaluation da to evaluate the model during training. After training the model it was saved and later loaded to the code of Genetic Algorithm to predict travel time. I plotted the feature score of the model after training. The figure 5 below shows the feature score of the different features. It can be said that the features are evenly significant in the model, which is a good characteristic of a regression model. I also found that the mean absolute error between the prediction and actual value of travel was 4.8 minutes which can be acceptable in this case.

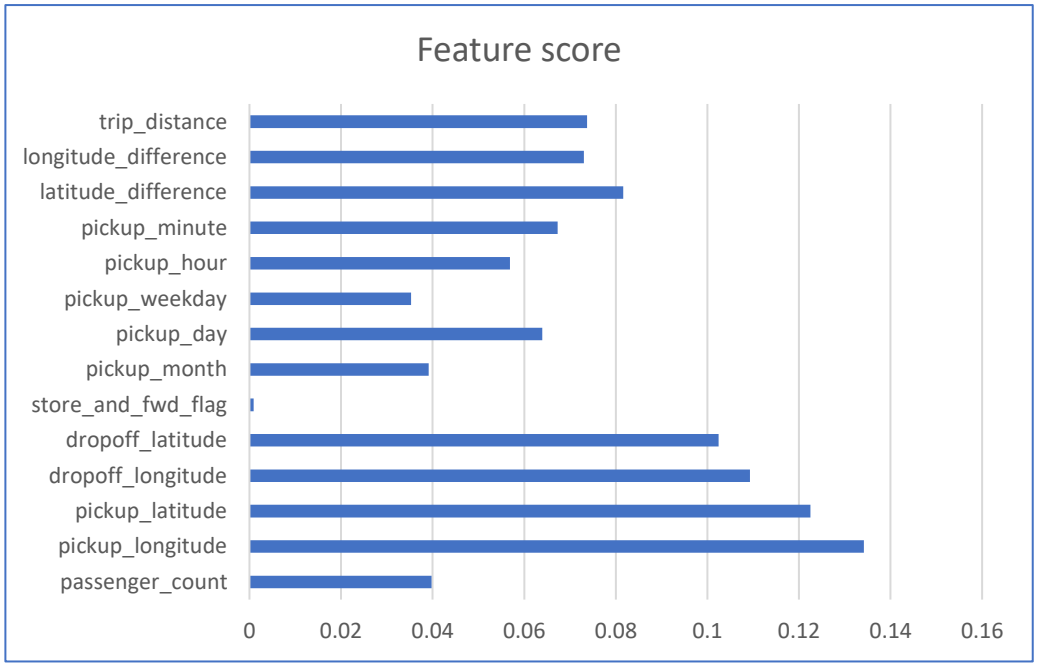

Figure 5: Feature score of different variables used in the XGBoost regression model.

VI. Itinerary Optimization using Genetic Algorithm

The concept of survival for the fittest is used in genetic algorithm (GA) approach for optimization (Goldberg, 1988). The GA inspired from the processes of evolution and so it can be termed as

an evolutionary algorithm. This process makes the strongest elements stronger while the weakest elements are eliminated. The solution of an optimization problem using the GA methodology involves a stochastic search of the solution space. A detail discussion on steps of GA can be found in (Murray-Smith, 2019). Here, I will try to give a brief overview as below.

Initial Population: At the beginning, an initial population is generated, and these are decoded to obtain the corresponding parameters. These parameter values are then introduced into the system model. A simulation is run, and results are obtained for each set of parameters within the population, using a measure of performance based on a cost function. When the cost values are all found, they are sorted into ascending order. finally, the smallest cost values are chosen as the best and are then subjected to operations involving reproduction, crossover and mutation.

Reproduction: The reproduction procedure involves retaining the best candidates for the next population. The other candidates in the population are replaced by new candidates formed through processes of crossover and mutation.

Crossover: Crossover is a process in which two candidates from the current generation (parent) engage in a procedure in which some characteristics from one parent are interchanged with characteristics from the corresponding positions in the other. This process produces two new candidates (offspring) and the procedure is repeated until there are sufficient offspring.

Mutation: Mutation involves selection, on a random basis, of a certain number of the candidates in the current population and random alterations are then made to their values. This provides a random element within the GA search process so that more of the search space is considered. Once the candidates have been changed to form the new population they have to be evaluated, as were

the previous generation. The whole procedure is then repeated for a predefined number of iterations (generations) to produce a final solution.

As the initial population of the genetic algorithm in this project, I chose seven random locations (L1, L2, L3, L4, L5, L6, L7 respectively) from the dataset. The co-ordinates of the locations are shown in the figure 6 below, which was found using ArcGIS software.

Figure 6: Randomly selected seven locations (1 to 7 are location ID) on ArcGIS.

The goal is to find the most optimal travel path or itinerary while traveling to all these locations. So, I developed my code using python packages for genetic algorithm. At first, the genetic algorithm creates a possible path by randomly guessing. Then next generation of multiple combination of paths are formed from the initial guess.

Suppose that the n locations are listed in a random order L1 → L2 → · · · → Ln in a given chromosome of the GA. Since we are trying to minimize the total distance for completing a travel loop, the fitness function will be the total travel time T as in the following equation (1)

$$T = \sum_{k=1}^{n} t[L(k), L(k+1)] \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Here, t is the time of travel between any pair of locations of a given list. I will try to discuss how the optimization is being done in following parts. Figure 7 is the screenshot of the program done for finding travel time between any two pair of locations.

```
def travel_time_between_points(point1_id, point2_id, hour, date, passenger_count = 1,
                               store_and_fwd_flag = 0, pickup_minute = 0):
    """
    Given two points, this calculates travel between them based on a XGBoost predictive model
    """

    model_data = {'passenger_count': passenger_count,
                  'pickup_longitude' : point1_id[1],
                  'pickup_latitude' : point1_id[0],
                  'dropoff_longitude' : point2_id[1],
                  'dropoff_latitude' : point2_id[0],
                  'store_and_fwd_flag' : store_and_fwd_flag,
                  'pickup_month' : my_date.month,
                  'pickup_day' : my_date.day,
                  'pickup_weekday' : my_date.weekday(),
                  'pickup_hour': hour,
                  'pickup_minute' : pickup_minute,
                  'latitude_difference' : point2_id[0] - point1_id[0],
                  'longitude_difference' : point2_id[1] - point1_id[1],
                  'trip_distance' : 0.621371 * 6371 * (abs(2 * np.arctan2(np.sqrt(np.square(np.sin((abs(point2_id[0] - point1
                                    np.sqrt(1-(np.square(np.sin((abs(point2_id[0] - point1_id[0]) * np.pi / 180) / 2)))))) + \
                                       abs(2 * np.arctan2(np.sqrt(np.square(np.sin((abs(point2_id[1] - point1_id[1]) * np.pi /
                                    np.sqrt(1-(np.square(np.sin((abs(point2_id[1] - point1_id[1]) * np.pi / 180) / 2)))))))
                 }

    df = pd.DataFrame([model_data], columns=model_data.keys())

    pred = np.exp(loaded_model.predict(xgb.DMatrix(df))) - 1

    return pred[0]

coordinates = test_locations
print(coordinates)
```

Figure 7: Screenshot of the function developed that use the XGBoost train model to calculate travel time t between a given pair of locations

From figure 7, we can see that all the different data attributes are taken in the model and then using the prediction power og XGBoost model developed, and with the location's latitude and longitude, the program is finding travel time. When the travel time is predicted, this is used by the genetic algorithm to calculate the total time for completing one loop in all seven locations. This total time is used as the fitness score for the GA. Below is the screenshot of the code which is developed to find the fitness score in Genetic Algorithm

```
def fitness_score(guess):
    """
    Loops through the points in the guesses order and calculates
    how much distance the path would take to complete a loop.
    Lower is better.
    """
    score = 0
    for ix, point_id in enumerate(guess[:-1]):
        score += travel_time_between_points(coordinates[point_id], coordinates[guess[ix+1]], 11, my_date)
    return score

def check_fitness(guesses):
    """
    Goes through every guess and calculates the fitness score.
    Returns a list of tuples: (guess, fitness_score)
    """
    fitness_indicator = []
    for guess in guesses:
        fitness_indicator.append((guess, fitness_score(guess)))
    return fitness_indicator

print(check_fitness(test_generation))
```

```
[(['L6', 'L5', 'L2', 'L7', 'L3', 'L1', 'L4', 'L6'], 109.67423677444458), (['L2', 'L3', 'L7', 'L5', 'L4', 'L1', 'L6', 'L2'],
122.93982410430908), (['L6', 'L2', 'L7', 'L1', 'L4', 'L3', 'L5', 'L6'], 125.08935642242432), (['L7', 'L2', 'L3', 'L5', 'L1',
'L4', 'L6', 'L7'], 85.55466842651367), (['L2', 'L5', 'L3', 'L6', 'L1', 'L7', 'L4', 'L2'], 102.80579376220703), (['L3', 'L1',
'L7', 'L5', 'L6', 'L2', 'L4', 'L3'], 129.72163581848145), (['L1', 'L2', 'L6', 'L7', 'L3', 'L5', 'L4', 'L1'], 98.914847850799
56), (['L1', 'L6', 'L5', 'L2', 'L3', 'L7', 'L4', 'L1'], 101.58901357650757), (['L6', 'L7', 'L3', 'L1', 'L5', 'L4', 'L2', 'L
6'], 95.36197471618652), (['L2', 'L5', 'L7', 'L4', 'L3', 'L6', 'L1', 'L2'], 89.0522928237915)]
```

Figure 8: Screenshot of the function developed to calculate fitness score

Based on the fitness score after each generation of different combination of guess by the genetic algorithm, they are sorted according to fitness score to find the most suitable or optimal travel path. Also, the fittest ones take part in next generation breeding. The sorting and breeding are done by the following function shown in the screenshot of Figure 9. In Genetic algorithm, for each time of

new generation creation, the fittest combinations are chosen and as this thing happen time and again, the fitness score of combination improves over time.

```
#test_generation = create_generation(list(test_locations.keys()), population=10)
def get_breeders_from_generation(guesses, take_best_N=10, take_random_N=5, verbose=True, mutation_rate=0.1):
    """
    This sets up the breeding group for the next generation. You have
    to be very careful how many breeders you take, otherwise your
    population can explode. These two, plus the "number of children per couple"
    in the make_children function must be tuned to avoid exponential growth or decline!
    """
    # First, get the top guesses from last time
    fit_scores = check_fitness(guesses)
    sorted_guesses = sorted(fit_scores, key=lambda x: x[1])         # sorts so lowest is first, which we want
    new_generation = [x[0] for x in sorted_guesses[:take_best_N]]
    best_guess = new_generation[0]

    if verbose:
        # If we want to see what the best current guess is!
        print(best_guess)

    # Second, get some random ones for genetic diversity
    for _ in range(take_random_N):
        ix = np.random.randint(len(guesses))
        new_generation.append(guesses[ix])

    # No mutations here since the order really matters.
    # If we wanted to, we could add a "swapping" mutation,
    # but in practice it doesn't seem to be necessary

    np.random.shuffle(new_generation)
    return new_generation, best_guess
```

Figure 9: Screenshot of the function developed to sort the fittest for new generation breeding in

Finally, I want to show the function that gives the optimal order of traveling at the end of my developed program. The figure 10 below shows the coding done in python. For a predefined maximum no of generations, the evolution happens and after each iteration, the fitness score improves. And after certain generations of evolution, I found that the combination of travel locations is stable in their order. Also, fitness score is constant. I took that as my optimal travel itinerary.

```
def evolve_to_solve(current_generation, max_generations, take_best_N, take_random_N,
                    mutation_rate, children_per_couple, print_every_n_generations, verbose=False):
    """
    Takes in a generation of guesses then evolves them over time using our breeding rules.
    Continue this for "max_generations" times.
    Inputs:
    current_generation: The first generation of guesses
    max_generations: how many generations to complete
    take_best_N: how many of the top performers get selected to breed
    take_random_N: how many random guesses get brought in to keep genetic diversity
    mutation_rate: How often to mutate (currently unused)
    children_per_couple: how many children per breeding pair
    print_every_n_geneartions: how often to print in verbose mode
    verbose: Show printouts of progress
    Returns:
    fitness_tracking: a list of the fitness score at each generations
    best_guess: the best_guess at the end of evolution
    """
    fitness_tracking = []
    for i in range(max_generations):
        if verbose and not i % print_every_n_generations and i>0:
            print("Generation %i: "%i, end='')
            #print(len(current_generation))
            print("Current Best Score: ", fitness_tracking[-1])
            is_verbose = True
        else:
            is_verbose = False
        breeders, best_guess = get_breeders_from_generation(current_generation,
                                                            take_best_N=take_best_N, take_random_N=take_random_N,
                                                            verbose=is_verbose, mutation_rate=mutation_rate)
        fitness_tracking.append(fitness_score(best_guess))
        current_generation = make_children(breeders, children_per_couple=children_per_couple)

    return fitness_tracking, best_guess

current_generation = create_generation(list(test_locations.keys()),population=500)
fitness_tracking, best_guess = evolve_to_solve(current_generation, 100, 150, 70, 0.5, 3, 5, verbose=True)
```

Figure10: Screenshot of the function developed for finding optimal travel path through evolution

VII. Final Optimized Itinerary

After choosing seven locations and running the genetic algorithm, I found an optimal travel itinerary as follows. Please note due to the randomness of selection criteria of GA, each time when we will run the code, there can be different order of the locations as the optimal travel itinerary. For one of my cases, I got the following travel order as the optimal path:

L6> L7> L4> L1>L5>L2>L3>L6

And the total time for completing the loop starting at L6 and then ending at L6 location was found 59 minutes. Figure 11 below shows the order in ArcGIS map.

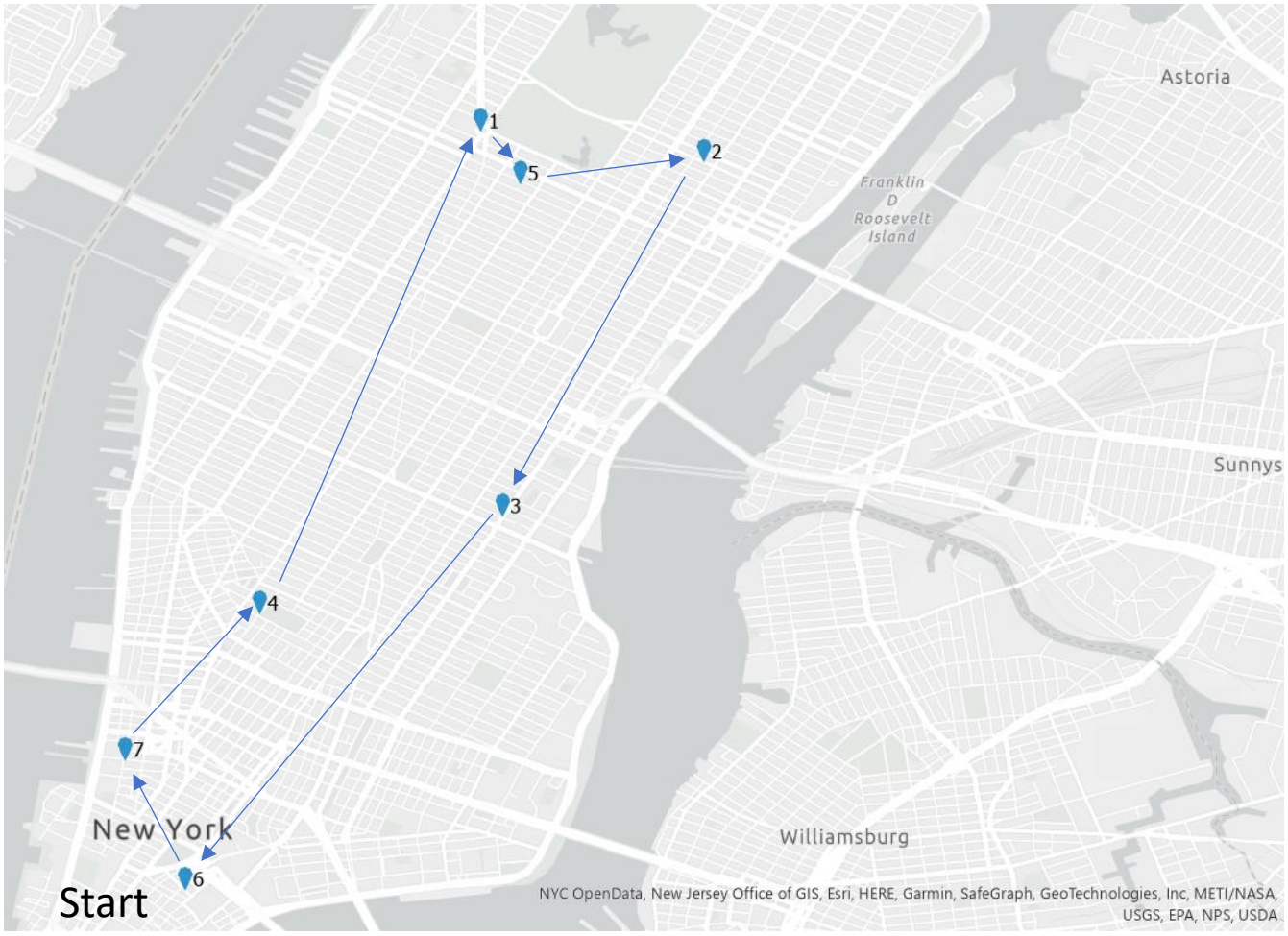

Figure 11: Travel order for the selected seven locations in NY city.

VIII. Conclusion

Now a days, genetic algorithms are used by large companies to optimize schedules and design products that range from large aircraft to tiny computer chips to medicines(*Practical Genetic Algorithms, 2nd Edition | Wiley*, 2004). The advancement in parallel computing has made genetic algorithms even more appealing because it can alleviate their main drawback: the lack of speed. This allows for more iterations and more total runs, which increases both the chances of finding

better solutions and the certainty that the solutions found are the optimal ones. Overall, the future of genetic algorithms in optimization is very promising.

In conclusion, I found a demo solution with seven random locations, which can be implemented in large scale for hundreds of locations. The solution is not the exact route planner—it suggests a visit order. In future, it is possible to incorporate Google Maps API and plan out the exact path between each pair of points. Also, weather data, time of the day and other factors affecting travel time can be incorporated in the model. This will require a different approach to constructing the input space altogether. So, it can be said that this is only a starting point for a full scaled route planning project.

References

Angskun, T., & Angskun, J. (2009). A Travel Planning Optimization under Energy and Time Constraints. *2009 International Conference on Information and Multimedia Technology*, 131–134. https://doi.org/10.1109/ICIMT.2009.86

Brownlee, J. (2016, August 16). A Gentle Introduction to XGBoost for Applied Machine Learning. *Machine Learning Mastery*. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Goldberg, D. (1988). *Genetic Algorithms in Search Optimization and Machine Learning*. https://doi.org/10.5860/choice.27-0936

Gupta, I. K. (2017). Randomized bias genetic algorithm to solve traveling salesman problem. *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–6. https://doi.org/10.1109/ICCCNT.2017.8204127

Murray-Smith, D. J. (n.d.). *Genetic Algorithm—An overview | ScienceDirect Topics*. Retrieved May 17, 2022, from https://www.sciencedirect.com/topics/engineering/genetic-algorithm

Practical Genetic Algorithms, 2nd Edition | Wiley. (n.d.). Wiley.Com. Retrieved May 19, 2022, from https://www.wiley.com/en-us/Practical+Genetic+Algorithms%2C+2nd+Edition-p-9780471455653

New York City Taxi Trip Duration. (n.d.). Retrieved May 17, 2022, from https://kaggle.com/competitions/nyc-taxi-trip-duration

Sykes, P. (n.d.). *What Is Route Optimization?* Retrieved May 17, 2022, from https://blog.routific.com/what-is-route-optimization

TLC Trip Record Data—TLC. (n.d.). Retrieved May 17, 2022, from https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

XGBoost Parameters—Xgboost 1.6.1 documentation. (n.d.). Retrieved May 18, 2022, from https://xgboost.readthedocs.io/en/stable/parameter.html

Xiao, S. (2017). Optimal travel path planning and real time forecast system based on ant colony algorithm. *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, 2223–2226. https://doi.org/10.1109/IAEAC.2017.8054413

Yu, H. (n.d.). *Optimization of vehicle routing and scheduling with travel time variability—Application in winter road maintenance*. 154.