Moving BlueBikes

Pravalika Putalapattu

Abstract

Bike sharing systems are becoming increasingly popular in urban environments, offering a convenient and eco-friendly mode of transportation. However, efficiently managing bike sharing operations presents significant challenges, such as balancing bike supply and demand, optimizing routes for bike redistribution, and maximizing overall system performance. This project aims to address these challenges using Gurobi optimization and graph search algorithms.

The project begins by collecting data on bike demand, station capacities, travel times between stations, and other relevant factors for the Boston BlueBikes system. A Gurobi model is then constructed to represent the BlueBikes system. The model incorporates constraints on bike limits at stations, travel times between stations, bike demand, and system-wide time limits. Additionally, it includes an objective function that considers factors such as bike movements, demand satisfaction, and station capacities. The model can be adapted and customized to accommodate specific operational requirements and system characteristics. Real-world data from BlueBikes is used to validate and evaluate the model's performance.

The outcomes of this project can provide valuable insights and practical recommendations for bike sharing operators, city planners, and policymakers to enhance the effectiveness and sustainability of bike sharing operations. By leveraging mathematical modeling and optimization techniques, this project aims to contribute to the advancement of bike sharing systems and their integration into urban transportation networks.

Keywords: Linear Optimization, Data Analytics, Transportation and Infrastructure

1 Introduction

Bike-sharing systems have gained widespread popularity as a sustainable and flexible mode of urban transportation. These systems offer users the convenience of renting bicycles for short-term use, providing an eco-friendly alternative to traditional commuting methods. However, managing bike-sharing systems efficiently presents significant challenges that necessitate effective strategies and optimization techniques.

Optimizing the management of bike-sharing systems is crucial to ensure the availability of bikes at high-demand stations, facilitate efficient bike redistribution, and enhance overall customer satisfaction. This project aims to help BlueBikes manage their bike-sharing system by creating an algorithm that helps BlueBikes reduce the number of stations with low numbers of bikes and docks. The algorithm offers BlueBikes truck drivers an optimal route to transport bikes between stations. I will use the Gurobi Linear Optimizer to solve this problem.

The outcomes of this project hold immense value for bike-sharing operators, urban planners, and policymakers. By optimizing bike allocation, minimizing operational costs, and improving system accessibility, cities can promote sustainable transportation, alleviate traffic congestion, and create greener and more livable urban environments. This project works toward Sustainable Development Goals 9 and 11: Industry, Innovation, and Infrastructure; Sustainable Cities and Communities. It also indirectly helps with Goal 13, Climate Action.

In the following sections, we will explore the specific aspects of bike-sharing system management, the challenges involved, and the proposed approach utilizing Gurobi optimization to effectively address these issues. Through a combination of mathematical modeling, data analysis, and optimization techniques, this project strives to make significant contributions to the field of bike-sharing system management and sustainable urban transportation.

2 Methods

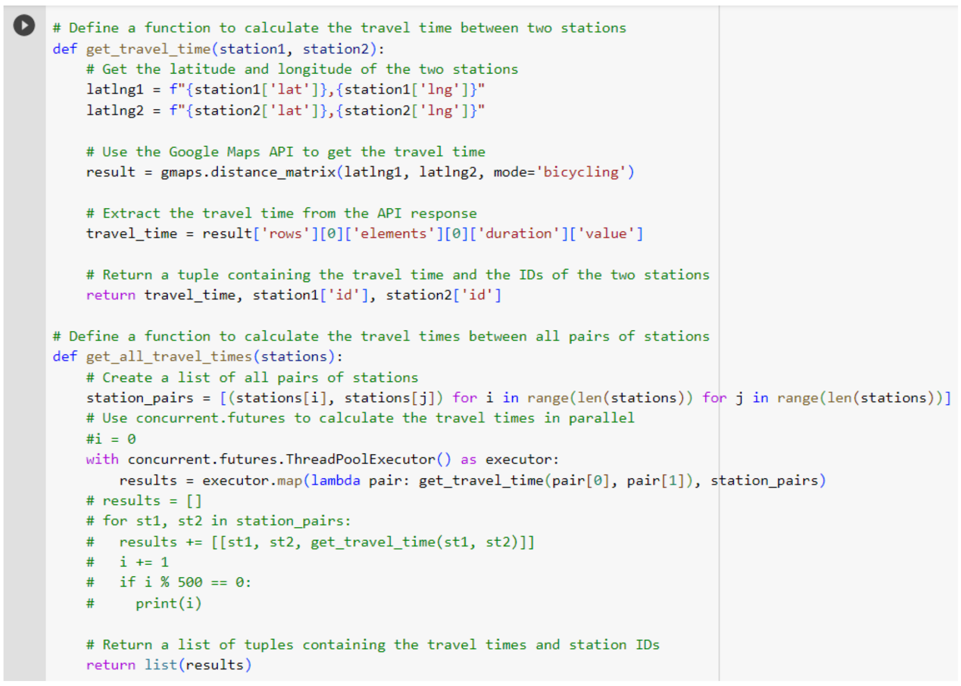
The algorithm to solve the problem operates as follows:

1. Data Collection: Find the coordinates of each BlueBikes station. Calculate the time it takes to travel between each pair of stations.
2. Data Collection: Find the BlueBikes stations that are low on bikes or docks, as well as how popular these stations are.
3. Model: Construct a linear optimization model to find the best path between stations.

2.1 Data Collection

The data collection consisted of two major components: the station locations aspect and the station identification aspect.

In order to find the coordinates of each station, I used the BlueBikes publicly available station dataset, which included a name, ID, longitude, and latitude for each station. There were 450 stations in the dataset, which meant that I needed to compute the time it took to travel between around 200,000 pairs of stations. This was not feasible by hand, so I used the Google Maps API, which included a module that calculated the driving distance and time between pairs of coordinates. When I ran the code to calculate these distances, the program took an incredibly long time, so I decided to use parallel computing to speed up the process:



```python
# Define a function to calculate the travel time between two stations
def get_travel_time(station1, station2):
    # Get the latitude and longitude of the two stations
    latlng1 = f"{station1['lat']},{station1['lng']}"
    latlng2 = f"{station2['lat']},{station2['lng']}"

    # Use the Google Maps API to get the travel time
    result = gmaps.distance_matrix(latlng1, latlng2, mode='bicycling')

    # Extract the travel time from the API response
    travel_time = result['rows'][0]['elements'][0]['duration']['value']

    # Return a tuple containing the travel time and the IDs of the two stations
    return travel_time, station1['id'], station2['id']

# Define a function to calculate the travel times between all pairs of stations
def get_all_travel_times(stations):
    # Create a list of all pairs of stations
    station_pairs = [(stations[i], stations[j]) for i in range(len(stations)) for j in range(len(stations))]
    # Use concurrent.futures to calculate the travel times in parallel
    #i = 0
    with concurrent.futures.ThreadPoolExecutor() as executor:
        results = executor.map(lambda pair: get_travel_time(pair[0], pair[1]), station_pairs)
    # results = []
    # for st1, st2 in station_pairs:
    #   results += [[st1, st2, get_travel_time(st1, st2)]]
    #   i += 1
    #   if i % 500 == 0:
    #     print(i)

    # Return a list of tuples containing the travel times and station IDs
    return list(results)
```

I then needed to identify the stations with low bikes and docks. The list of stations with low bikes and docks is constantly changing over time due to factors such as traffic, weather, and time of day. I created this list on a relatively sunny Tuesday evening by counting the number of red and yellow stations on the BlueBikes app by hand. I first counted these stations normally to find the stations that were low on bikes, then checked out a bike and counted again to see which stations were low on docks.

While there does exist a user-created BlueBikes API on Github, the API was offline at this time. In the future, I hope that I could either create an API or use one that becomes available to count these stations automatically.

I also assigned a multiplier value to each station based on how popular it was. Although I could not calculate popularity so precisely, I aimed to approximate this value by checking for certain buzzwords in the name of each station. For example, if the name of the station contained the words "T" or "MBTA", it is clear that the bike station is near a train station. Thus, it should be weighted as more important than other stations in our model. I checked for the following words: ["College", "Uni", "MIT", "Harvard", " T", "MBTA", " Sq", "City Hall", "Post Office"].

The data collection was performed in Python.

2.2 Model

To address the optimization problem at hand, a Gurobi model was developed and implemented using the following parameters, variables, constraints, and objective function.

Parameters:

- time_limit: the maximum amount of time the path could take. Initialized to 3600 and 7200 seconds for two different trials.
- bike_limit: the maximum number of bikes that can fit in the BlueBikes trucks. I assumed that BlueBikes would use two trucks that can each fit 25 bikes, so the limit was set to 50.

Variables:

- path[i, j]: Binary variable representing whether a bike-sharing station i is visited at time step j. Its value is 1 if the station is visited, and 0 otherwise.
- bikes[j]: Integer variable representing the number of bikes taken or removed from a station at time step j.

Constraints:

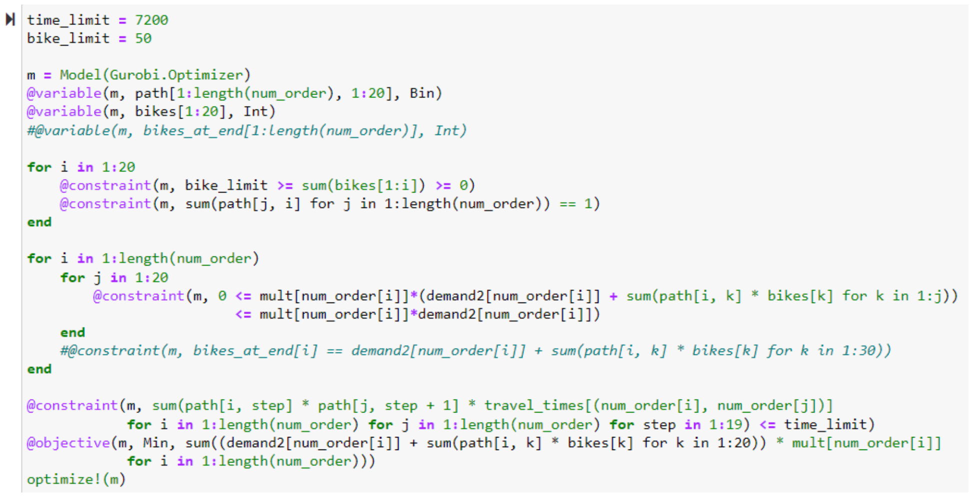
- Bike Limit Constraint: $\textbf{bike\_limit} >= \sum(\textbf{bikes}[1:i]) >= 0$: Ensures that the total number of bikes does not exceed the specified bike limit and remains non-negative.
- Station Visit Constraint: $\sum(\textbf{path}[:, j]) == 1$: Ensures that exactly one station is visited at each time step.
- Availability Constraint: For each station, at no point in the path does it have a negative number of bikes or docks.

Objective Function: The objective of the model is to minimize the following expression:

$$\sum_i \left((\textbf{demand}[\textbf{num\_order}[i]] + \sum_k (\textbf{path}[i,k] * \textbf{bikes}[k])) * \textbf{mult}[\textbf{num\_order}[i]] \right)$$

This objective function calculates the absolute difference between the demand for bikes at each station and the actual number of bikes available, multiplied by a multiplier factor. The aim is to minimize this sum, indicating a closer alignment between bike availability and demand.

This model was implemented in Julia, as shown below.

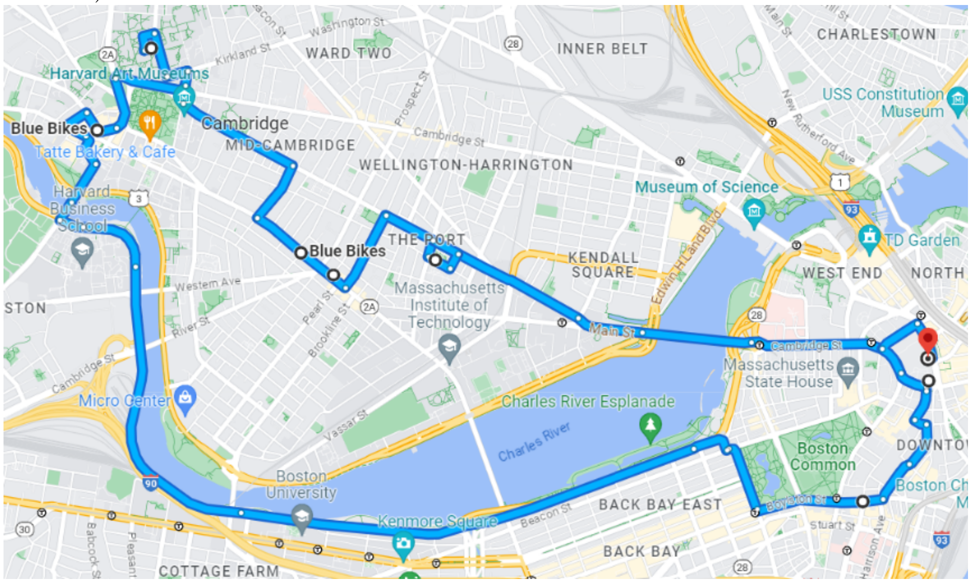


```
time_limit = 7200
bike_limit = 50

m = Model(Gurobi.Optimizer)
@variable(m, path[1:length(num_order), 1:20], Bin)
@variable(m, bikes[1:20], Int)
#@variable(m, bikes_at_end[1:length(num_order)], Int)

for i in 1:20
    @constraint(m, bike_limit >= sum(bikes[1:i]) >= 0)
    @constraint(m, sum(path[j, i] for j in 1:length(num_order)) == 1)
end

for i in 1:length(num_order)
    for j in 1:20
        @constraint(m, 0 <= mult[num_order[i]]*(demand2[num_order[i]] + sum(path[i, k] * bikes[k] for k in 1:j))
                           <= mult[num_order[i]]*demand2[num_order[i]])
    end
    #@constraint(m, bikes_at_end[i] == demand2[num_order[i]] + sum(path[i, k] * bikes[k] for k in 1:30))
end

@constraint(m, sum(path[i, step] * path[j, step + 1] * travel_times[(num_order[i], num_order[j])]
            for i in 1:length(num_order) for j in 1:length(num_order) for step in 1:19) <= time_limit)
@objective(m, Min, sum((demand2[num_order[i]] + sum(path[i, k] * bikes[k] for k in 1:20)) * mult[num_order[i]]
            for i in 1:length(num_order)))
optimize!(m)
```

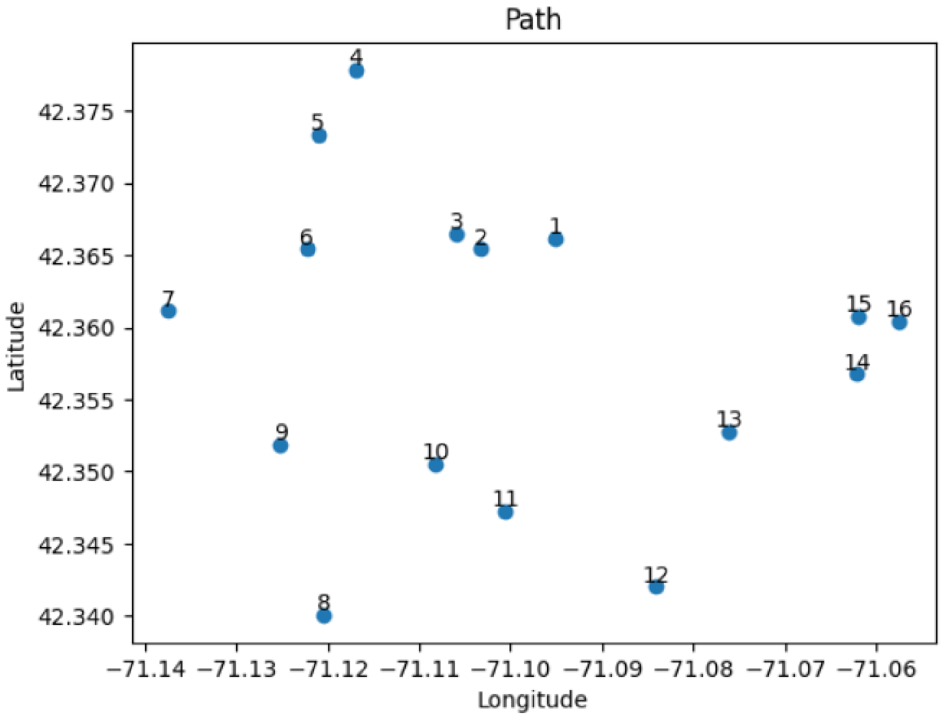
3 Results

The implemented Gurobi model successfully solved the optimization problem for Blue-Bikes, providing valuable insights into the optimal path and its impact on demand reduction. Two scenarios were examined: a 1-hour time limit and a 2-hour time limit. The obtained results demonstrate the effectiveness of the model in delivering bikes between stations.

For the 1-hour time limit, the model identified an optimal path that reduced the demand for bikes by 67 (i.e., 67 bikes were moved from low-dock stations to low-bike stations).



In the case of the 2-hour time limit, the model achieved even greater demand reduction, with a total of 104 bikes moving to meet demand across the system. I had some difficulty displaying this path due to Google Maps' constraint with the number of points it could display in a route. I have included the stations in a scatterplot below:



By strategically determining the optimal path and reallocating bikes, BlueBikes can better cater to user demand, enhance user experience, and improve the overall sustainability of the transportation network. The achieved results demonstrate the value of optimization models for the United Nations' Sustainable Development Goals, particularly Goal 11: Sustainable Cities and Communities. By optimizing the utilization of bike-sharing resources, this approach promotes efficient transportation systems, reduces traffic congestion, and supports environmentally friendly modes of transportation.

4 Next Steps

In order to move forward and get my solution implemented by the BlueBikes company, I would have to work on two major tasks: analyzing the finances and streamline implementation.

4.1 Analyzing Finances

Implementing this solution could have a wide array of benefits for the BlueBikes company:

- Reduced Opportunity Costs: Bike demand imbalances can lead to missed revenue opportunities. If a station frequently runs out of bikes or has excess bikes, potential riders may choose alternative modes of transportation. By reducing demand imbalances through optimized redistribution, BlueBikes can capture more ridership and generate additional revenue. According to the BlueBikes station data, most rides are taken by single-use customers, so the company has a lot to gain by making sure bikes and docks are always available.
- Customer Satisfaction and Retention: A well-balanced bike availability across stations enhances the overall customer experience. With the optimized system in place, BlueBikes can provide riders with a higher likelihood of finding available bikes at their desired locations, reducing waiting times, and increasing customer satisfaction. With this solution, BlueBikes increases their capacity to turn single-use riders into customers with subscriptions.

However, it would also incur the following costs:

- Materials Costs: Implementing this system would require the use of one or more trucks, which introduce costs such as gas and drivers' wages.
- Training and Workforce: Introducing a new bike redistribution system may require training staff members on how to operate and utilize the system effectively. Training costs can include conducting training sessions, preparing training materials, and allocating staff resources for training purposes.
- Monitoring and Evaluation: Continuous monitoring and evaluation of the system's performance are essential to ensure its effectiveness and make necessary adjustments. This may involve dedicating resources to monitor key performance indicators, analyze data, and assess the impact of the solution. The associated costs include personnel time, data analysis tools, and reporting mechanisms.

In order to convince BlueBikes to implement this solution, I would need to prove that the benefits outweigh the costs. While this would be extremely challenging, due to the ridership being largely single-use riders, the company has a lot to gain in terms of revenue. As public transportation is an important issue, I also hope that this project could be partially funded by the Massachusetts or Boston transportation departments.

4.2 Streamlining Implementation

For this solution to be practically useful to the BlueBikes company and Boston residents, it would have to be implemented regularly, multiple times a day. Thus, there are several parts of my process that would have to be modified to be faster and easier to perform.

4.2.1 Data Collection

I would not need to compute station locations or travel times repeatedly. However, I would need to identify which stations have low bike and dock counts repeatedly throughout the day, which would require the use of a BlueBikes API. I would also need to include a better way to calculate the importance of each station, perhaps in a way that changes over time.

4.2.2 Modeling

Although the Gurobi Optimizer is powerful and accurate, it takes an incredibly long time to run, so the model must be modified. One approach could be to run the optimizer on multiple threads or cores. Another approach could be to use a BFS or DFS to find the path, which would be far simpler, but also far easier to parallelize.

5 References

1. Motivate International, Inc. (2022). *Bluebikes System Data.* Blue Bikes Boston. https://www.bluebikes.com/system-data
2. P, R. (2018). *Raghavp96/bluebikedata: A website, API, and database for Blue Bike Data.* GitHub. https://github.com/raghavp96/bluebikedata