# Design a Optimized Multimode System integrated Mobility-on-Demand Mass Transit System

## Introduction

The rapid growth of “Mobility on Demand” (MoD) services such as Uber in urban areas is a clear indication of consumer demand for more flexible and convenient transportation services. These services allow customers to access a shared fleet of vehicles by using their cell phones to hail a ride. While these services have the potential to improve urban mobility and lay the groundwork for more scalable and sustainable transportation systems in the future, it is not clear that this is the case in their current form of operating as isolated services. One of the main problems with these types of systems is that the location of the vehicle and the location where the passenger's request for a ride is generated do not overlap, and the vehicle wastes some of the time it takes to get to the passenger's pickup point to pick up the passenger after receiving the order, which is a part of the journey that increases the passenger's waiting time, increases the time the vehicle spends on the road, and does not have a good impact on the passenger, the fleet, or city traffic.

For the MoD system to be truly scalable and sustainable, its average occupancy must be significantly increased. One option is to use the MoD system as a feeder system for Mass Transit service to address the first mile and last mile of public transit trips and increase the use of public transit by the public.

In this paper, we first constructed a model for distributing user demand between the MoD system and public transportation and developed a vehicle scheduling management system while catering to customers traveling by Mod-Transit mode. The results of this system are then compared with the case of pure Mod system travel. We built this system in pursuit of improving vehicle utilization, reducing the time each vehicle travels in an empty state, and looking at what fleet sizes can be mobilized to have the highest overall order-taking rate of the fleet.

## Hybrid Network Construction

## **Bus Network**

I constructed a bus network graph using bus stops as nodes and bus routes as edges. However, I encountered several challenges while building these transit graphs. The main issue was that the geometry point data of the bus stops and the geometry LineString data of the routes were not directly corresponding, making it impossible to simply perform a spatial merge.

To ensure the topology of the edges closely reflects the actual routes, I extracted all the point data from the geometry LineString of each route. Then, I iterated

through the geometry point data of the corresponding bus stops for that route, identifying the closest point on the LineString for each bus stop. Using this, I created a dictionary with the points from the LineString data as keys, facilitating future searches to find the corresponding bus stop for each key.

I built the graph nodes and edges by sequentially extracting points from the LineString data in topological order. If a point corresponded to a bus stop, I updated the point's ID to `{STATION_ID}_{ROUTE}`, where `STATION_ID` is the unique identifier for each bus stop, and `ROUTE` is the unique identifier for the route. This allowed me to differentiate between the same bus stop serving multiple routes.

If a bus stop served more than one route, I created an edge between `{STATION_ID}_{ROUTEA}` and `{STATION_ID}_{ROUTEB}` to represent the transfer connection at that stop. Since the graph nodes were built sequentially based on the topology of the route's LineString geometry, this ensured that all bus stops on the same route were interconnected within the graph and that the edge geometries closely resembled the real-world routes. Additionally, we could calculate the actual distance between stops based on the LineString geometry.

Every edge in the graph has a `travel time` attribute, which is calculated as the edge's real length divided by the average bus speed. Transfer edges also have a nominal `travel time` attribute, which represents the transfer time and is defined as half the frequency of the target bus.

## Rail Network

The construction method for the Rail Network follows the same approach as the Bus Network.

## Road Network

The Road Network was not constructed manually but rather downloaded from OpenStreetMap. The downloaded dataset represents road intersections as nodes and each road segment as an edge. The data quality is excellent, with high precision and a very dense and extensive distribution of nodes and edges. Associating the pickup and drop-off points from our trip data with the corresponding road network nodes aligns well with the real-world taxi-taking behavior, where passengers tend to wait for taxis at intersections.

## Combining Three Networks into a Hybrid Network

This step posed the greatest challenge in constructing the hybrid graph. First, I identified the corresponding node in the road network for each transit stop by finding the nearest node (in terms of geometric distance). Then, I established edges between the nodes in the road network and the transit network using the format {Node at Road Network} —— {Transit Station ID} _{Route}, representing the availability of connections between the road network and the transit network. These edges do not have a geometry but are assigned a travel time attribute. The travel time reflects the waiting time at the transit stop, which is defined as half the frequency of the target transit.
Finally, I merged these three graphs (Bus Network, Rail Network, and Road Network) into a single hybrid graph.

## Trip Data

This project selects Chicago as the study area. The trip data was collected from the Chicago Data Portal. Due to privacy protection measures, the taxi trip data currently available has been anonymized: the pickup and drop-off locations are generalized to the centroid of census tract or community area, and times are rounded to the nearest 15 minutes. This data format can be interpreted as grouping passengers who alight in the same community area under the same hub, thereby categorizing travel demand based on geographic proximity. For this project, I selected trips occurring on a single day between 7:00 AM and 12:00 PM as the dataset.

## Find Shortest Multimode Path for Each Demand

At this stage, for each trip's pickup location and drop-off location, I used the shortest_path function from Python's NetworkX package to compute the weighted shortest path in the multimodal graph, with travel time as the weight. The algorithm employed by NetworkX is Dijkstra’s algorithm.

Dijkstra’s algorithm is a classic algorithm for solving the single-source shortest path problem in graphs with non-negative edge weights. Given a source node and a target node, the algorithm maintains the shortest known distance from the source to all other nodes and iteratively updates paths by greedily selecting the node with the smallest distance. The algorithm ensures that the shortest path to the target node is found by the end of the process.

The core idea of Dijkstra’s algorithm relies on a greedy strategy:

1、Initialize the shortest path distance from the source to all other nodes as infinity (∞), except the source itself, which is set to 0.

2、At each step, select the node with the smallest current shortest distance that has not been finalized, mark its distance as final, and update the shortest path estimates for its neighboring nodes using this finalized distance.
3、Repeat this process until the shortest path distances to all nodes are determined.

Dijkstra's algorithm is feasible because once a node is finalized as having the shortest distance, no better path can be found later due to the non-negative edge weight constraint. This ensures that the path lengths only increase as the algorithm progresses, avoiding the possibility of a "negative cycle" or reducing path lengths through negative weights.

For each trip, the output of the shortest path computation consists of a set of edge IDs and their associated attributes. Based on the edge attribute `mode` (drive, bus, rail), we segment the sequential use of different transportation modes during a single trip, recording the start point, endpoint, and travel time for each mode.
Two adjustments are made during this step:

1. If the travel time for a `drive` segment is less than 5 minutes, the mode for that segment is changed to `walk`.
2. The first time a ride-hailing service is required for the demand is recorded.

When sequentially traversing each segment of the trip using different modes, the travel time for each segment is accumulated. If a segment's mode is `drive`, the accumulated travel time is added to the initially recorded time when the customer first requested a ride. This approach dynamically adjusts the ride-hailing request time based on the trip scenario:
Scenario 1: If the customer uses public transit (e.g., bus or rail) before requesting a ride, the ride-hailing request time is postponed relative to the Mod-only request time.
Scenario 2 (More Complex): If the customer requests a ride multiple times during the trip, such as after using public transit following the first ride, the second ride request time is calculated as the Mod-only request time plus the cumulative travel time of all preceding modes. Additionally, the waiting time during the first ride-hailing service is factored into the later stages of processing.

To improve efficiency, I separately stored the shortest travel times between the starting and ending points of each ride-hailing segment on the road network. This avoids redundant shortest path searches during subsequent operations.

## Set Up Simulation and Optimization Framework

Since ride-hailing demand evolves over time, the vehicle states and unserved passengers are updated every 5 minutes during the simulation. Below is the step-bystep procedure for each simulation iteration:

## Step 1: Identify Active Ride Requests

Each passenger's status is marked as either "waiting" or "completed." Initially, all passengers are marked as "waiting." At each simulation time point, identify all ride requests made up to the current time. These requests form the set of demands that need to be addressed in the current iteration.

## Step 2: Update Vehicle Status

Each vehicle has several attributes:

1、Location: Initially distributed randomly at the drop-off points of orders from 6:45 AM.
2、Available Time: The earliest time the vehicle can serve the next request. Initially set to the simulation start time.
3、Status: Either "busy" or "free." Vehicles are initially set to "free."

Before assigning vehicles to new demands, update the status of all vehicles: Identify "busy" vehicles whose `available time` is earlier than the current simulation time. These vehicles have completed their last trip. Then, Change the status of these vehicles from "busy" to "free," making them available for new assignments.

## Step 3: Direct Assignment of Vehicles to Nearby Demands

For ride requests whose pick-up point matches the location of an available vehicle, directly assign the vehicle to the demand. This step is referred to as direct_assign function. I Perform the following updates for each direct assignment:

1、Update the vehicle's “available time” by adding the travel time of the current trip.
2、Set the vehicle's status to "busy."
3、Update the vehicle's location to the drop-off point of the current trip.
4、Mark the assigned demand's status as "completed."( If a passenger has multiple ride-hailing requests, each request is treated as a separate demand in the simulation.)

## Step 4: Rebalancing and Assigning Remaining Demands

After completing the direct assignment, identify Remaining vehicles that are not yet assigned to a trip and Remaining unassigned demands that still need service. In this part, we use a matrix-based linear programming approach to match vehicles to demands, minimizing total rebalancing time (vehicle relocation time) or passenger waiting time.

The cost for assigning a vehicle to a demand consists of:

1. Travel time between the vehicle's current location and the demand's pick-up point.
2. A penalty cost for unfulfilled demands, set to 1200 seconds (20 minutes). This penalty ensures that demands requiring more than 20 minutes of rebalancing time are treated as "discarded" since the penalty for serving them exceeds the penalty for rejecting them.

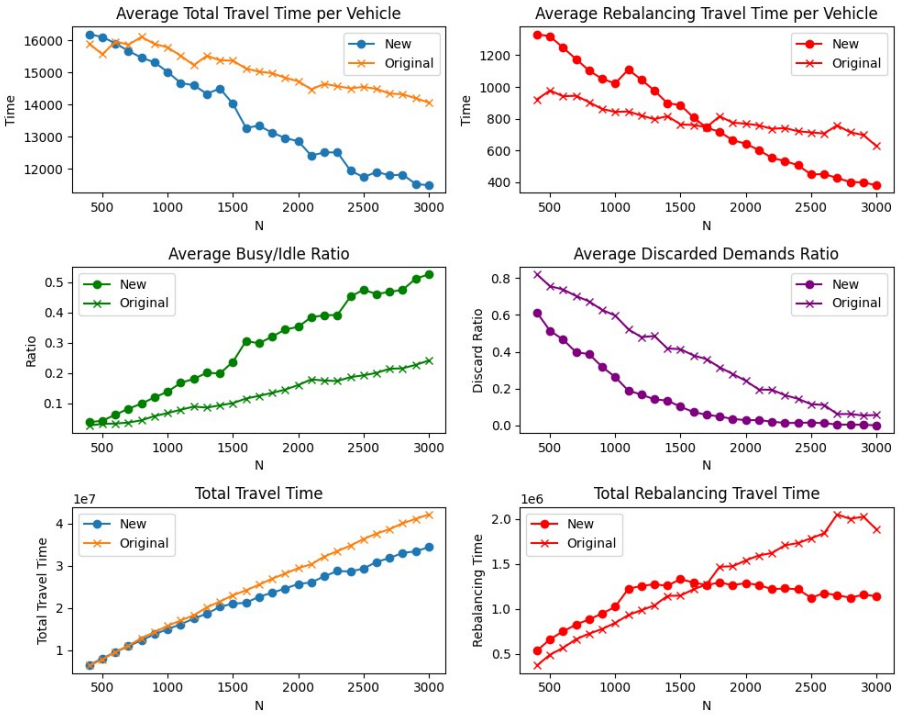
There are two component of decision variable, which are binary variables are defined for:
1. Whether a specific demand is assigned to a specific vehicle.
2. Whether a specific demand is discarded.
Equality Constraints ensure that each demand is either assigned to a vehicle or marked as discarded. Inequality Constraints Ensure that each vehicle serves at most one demand.
After setting, we solve the optimization problem using Python's `linprog` function with the "highs" method. Process the solution to determine the final vehicle-todemand assignments.

## Step 5: Update Vehicle and Demand States

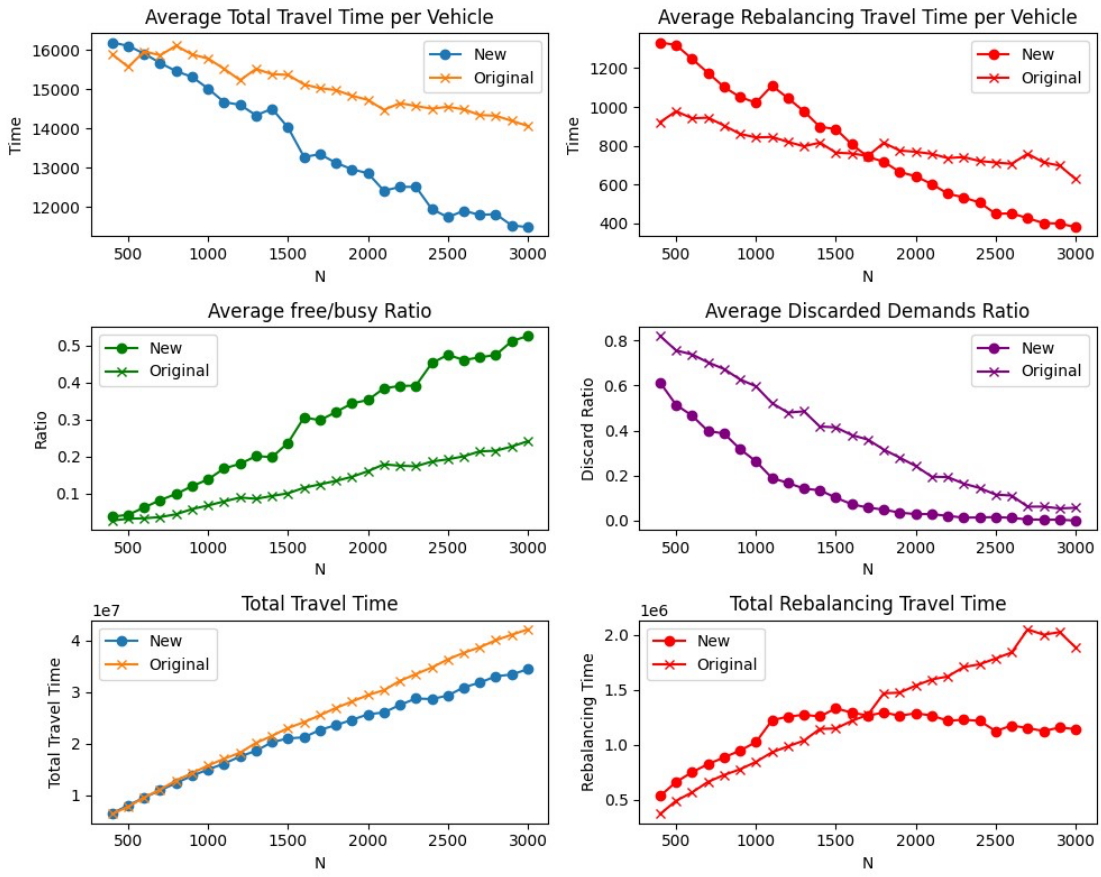
Based on the optimal solution, system update each vehicle's location, `available time` (adding rebalancing and travel time), and status. Mark the status of assigned demands as "completed." For passengers with subsequent ride-hailing requests, adjust the request time for the next trip by adding the waiting time experienced during the current trip.

# Result

To compare the results with the Mod-only taxi trip data, I converted the Mod-only taxi trip data into demand input for the aforementioned simulation and optimization process. The trip times, as well as the pickup and drop-off locations, remained unchanged. I calculated metrics under both scenarios, including Vehicle Total Travel Time, Vehicle Total Waiting Time, Vehicle Average Travel Time, Vehicle Average Waiting Time, Discard Rate, and the number of vehicles assigned to demands versus free cars.




I simulated the model with fleet sizes set to 400, 500, 600, ..., up to 2100. Initially, the performance of the Multimode approach was worse than that of pure taxi trips, but as the fleet size increased, the performance began to surpass the latter. I believe this is due to the fleet size configuration, as I found from Chicago taxi news that the daily active taxis in Chicago are approximately 4000–5000. Therefore, I increased the fleet size for further simulation.




*New*：*Multimode Transportation system*

*Original*：*MoD-only Transportation system*

In terms of Average Rebalancing Travel Time per Vehicle，Multimodal transportation starts at a high value when the number of vehicles is low, but then declines rapidly. There is a clear inflection point where the rate of decline slows down significantly as the number of vehicles increases. This inflection point suggests that as more vehicles are introduced, the system becomes more efficient in assigning rebalancing tasks, eventually reaching a point of diminishing returns. Mod-only mode starts lower but declines more slowly. It maintains a steady decline as the number of vehicles increases, but does not have a significant inflection point like multimodal.

In terms of Total Rebalancing Travel Time，the total multi-modal rebalancing travel time begins to increase sharply as the number of vehicles increases, but the line flattens out as the number of vehicles continues to increase. This suggests that as more vehicles are added to the system, the rebalancing burden across the system stabilizes despite the increase in the number of vehicles. The Mod-only model's line growth is much more stable and does not show any significant slowdown, suggesting that it is less efficient to utilize more vehicles to perform the rebalancing task.

Under the same fleet size, the average total travel time per vehicle in the Mod-only scenario is clearly higher than in the Multimode scenario. In the Multimode setting, each vehicle bears a lighter travel load, implying higher per-vehicle efficiency.

Additionally，Mod-only scenario exhibits a larger total travel time, indicating that, at a system-wide level, vehicles spend more time on the road. This suggests that under Mod-only conditions, meeting similar demand levels or maintaining a certain service standard involves a greater total travel effort, and consequently, more resource investment. It may also introduce additional instability and congestion into the transportation system. Meanwhile, the lower total travel time in the Multimode scenario suggests that, under comparable conditions, the Multimode approach yields a reduced overall travel workload, leading to more efficient resource utilization.

The improved demand satisfaction in the Multimode setting enables serving a greater number of passengers and thus reduces the number of discarded requests, ultimately enhancing overall service quality.

However, as the fleet size grows, the idle-to-busy ratio rises more significantly in the Multimode scenario compared to the Mod-only scenario. This means that with increasing fleet size, the proportion of idle vehicles relative to busy ones becomes greater under Multimode conditions. In contrast, Mod-only operations maintain a relatively low and slowly increasing idle-to-busy ratio, indicating that the proportion of idle vehicles is smaller and resource utilization is more focused. A higher idle-to-busy ratio does not signify increased efficiency; rather, it reveals that in the Multimode scenario, a larger fraction of vehicles remain idle, and thus resources are not being fully utilized.

Reference：

Vakayil, A., Gruel, W., & Samaranayake, S. (2017). *Integrating shared-vehicle mobility-on-demand systems with public transit* (No. 17-05439).