IE-537
PROJECT REPORT
SHARED-TAXI ROUTING

1. Abstract

With the rise of shared-taxi services in cities worldwide, several studies have focused on the dual objective of improving the shared-taxi experience for the customer and enhancing profitability for the cab owning companies. Matching riders with cabs that are closest to their pickup points, while ensuring that the cabs do not deviate much from their prior planned route seems to be the general scheme followed to achieve these goals. This study focuses on the routing aspect of shared taxis, where the objective is to find a route that has the maximum probability of picking up a co-passenger while minimizing the total distance covered from source to destination. A pickup density map is created based on a dataset showing historical data of cab pickups in New York City. Dijkstra's algorithm is used to find the ideal path through this network, while maintaining a trade-off between the potential profit obtained from picking up an additional passenger and the extra fuel consumption cost for taking long detours.

2. Introduction

Ride sharing has been gaining popularity in the past five years in urban areas, due to the services provided by cab companies like Uber, Lyft and Via. Riders are provided incentives to switch to ride sharing as opposed to traditional cab services by being offered cheaper rides. Another tactic used to woo environmentally conscious riders is to market the service as a green initiative. With the rapid rise of automobiles on the roads, ride sharing provides a means to decongest roads and reduce the emission of pollutants into the atmosphere. By clubbing together people headed in the same direction, everyone gets to their destination at a cheaper fare, while the cab companies would earn more profits owing to multiple passenger pickups while using up fuel for a route only once.

The riders of shared-taxis are advised though, that if they are constrained by time, they would be better off availing a regular cab service. This happens because, in the case of ride sharing, multiple pickup and drop off points mean that the cab's route is no longer a single source to single destination routing problem. Requests for pickups arrive dynamically, which means that the cab's route must be updated dynamically. Also, owing to the fact that in most cases, all passengers in a shared-taxi need not be headed towards the very same destination, some degree of detouring is inevitable. What this translates to then, is that riders would normally take more time to reach their destination using a shared-taxi than if they had opted for the traditional cab service. This is a trade-off that they should be willing to make for paying less.

One situation where ride-sharing affects cab companies negatively is when a rider avails the service and gets to their destination paying the discounted fare, while being the sole passenger for the entire trip. This could happen during such times of the day when there is not much demand for cabs. It could also happen if the route chosen is through scantily populated areas, which do not have many people availing cab services. The first issue could be avoided by studying the hourly demand pattern in a city and disabling the ride-sharing option during the times of day when the demand is bleak. The latter issue is being addressed in this study. It is based on the realization that shared taxis could be routed along streets where there is a high probability of picking up a co-passenger. This is an issue unique to shared-taxis because:

1) traditional cabs do not need to pick up more passengers until the current one is dropped

2) detours are allowed as it is assumed that the riders of shared taxis are not constrained by time

By routing shared-taxis in this way, it is more likely for them to have multiple passengers per trip, which would translate to higher profits for the cab companies. But this needs to be done keeping in mind that long detours could incur higher fuel costs. In other words, in anticipation of picking up an additional passenger, we cannot afford to have a significantly large deviation from the shortest distance path, as the profit that would be made from such a pickup would get offset by a large expenditure on fuel.

This study proposes an approach to address this issue and carries out analyses using data available through a publicly available dataset. The remainder of the report is organized as follows: Section 3 examines the literature for previous studies carried out on routing and improving the operations of shared taxis. Section 4 discusses the data source used for this study. Section 5 highlights the assumptions made and Section 6 compares possible modeling options and describes the proposed model in depth. Sections 7 and 8 show the analysis and results for this study and Section 9 discusses possible leads for improving the existing study. Finally, Section 10 has concluding remarks.

3. Literature Review

There have been several studies in the past based on improving the routing and operations of shared taxis. Some of the relevant ones are mentioned below:

Wei et al. [1] worked on the problem of improving the global level of service (LOS) for shared-taxi systems by making use of a reinforced learning method. They realized that focusing only on the current trip's waiting time and detour distance could impact the overall number of pickups and would ultimately give suboptimal results. They tested their proposed model on large-scale networks, confirming an improved LOS and reduced passenger request rejection rates.

Jung et al. [2] worked on improving shared-taxi algorithms with the objective of minimizing passenger route detours. They compared three algorithms- Nearest vehicle dispatch, Insertion heuristic and Hybrid simulated annealing- and ran simulations for two different taxi operation strategies. The hybrid approach, which was in fact a combination of simulated annealing and insertion heuristic, was able to maximize system efficiency and get better results than the other algorithms applied independently.

Li et al. [3] studied the potential advantages and disadvantages of combining people and goods in the same taxi network. Two models were proposed for such a combined system- the Share-a-ride problem (SARP) and the Freight insertion problem (FIP). The corresponding MILP formulations were presented and the static and dynamic cases were analyzed on a dataset containing coordinates and time-stamps for taxis in San Francisco. It was seen that SARP could be solved to optimality only in case of smaller instances, owing to its high computational complexity. FIP, on the other hand, proved to be computationally efficient and enabled them to perform an extensive numerical study. It was concluded that, in addition to the taxi-sharing systems, a traditional freight service must also be available to satiate all requests.

Sanders and Schultes [4] attempted to obtain large speedups for vehicle routing on road networks by making use of preprocessed data and running point-to-point queries. They worked on improving their previous study by reducing the time required and the size of data obtained during the preprocessing phase and making querying faster. Results showed that the new algorithm gave an order of magnitude reduction in processing and query time, and a smaller search space.

Masoud and Jayakrishnan [5] proposed a real-time ride-matching algorithm to find an optimal route plan in a flexible ridesharing system. The objective was to maximize the number of riders served, minimize transfers and waiting times, while ensuring that users ride with co-passengers that they prefer to ride with. The spatiotemporal ellipsoid method (STEM) was used to obtain a time-expanded feasible network, and the concept of peer-to-peer ride exchange was used to increase the number of successful matches in the system.

Lin et al. [6] proposed a route optimization model for a static, multiple-vehicle dial-a-ride problem with time windows, with the objective of minimizing operating cost and maximizing customer satisfaction. The probabilistic method of simulated annealing was used to find the global minimum of the cost function. The proposed method was tested on a sample network with a stipulated number of requests per origin-destination pair and a designated pickup time window. Results showed that there was a 19% saving in mileage and 66% increase in the available taxis.

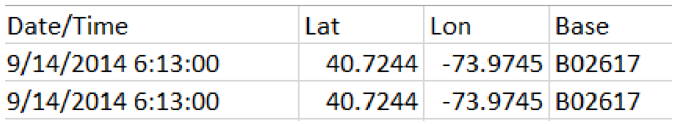
Rong et al. [7] used historical data to understand the best taxi seeking strategy with the intention of improving the revenue efficiency of taxi drivers. The passenger seeking process was modeled as a Markov Decision Process and was solved using a dynamic programming approach. Based on passenger trip data and the current taxi status, the next movement of each driver was predicted. Implementation of this solution showed a considerable rise in the drivers' revenue efficiency.

Ding et al. [8] created a system called HUNTS, based on historical and online GPS data, to generate hunting trajectory recommendations for improving the profits earned by taxi drivers. The problem of finding a connected trajectory with high profit within a given time period was solved using a novel technique called trajectory sewing. Their results showed that the suggested models were better and more efficient when compared to the traditional places of interest recommendations.

4. Dataset

4.1 Data source

The data used to carry out this study is from a dataset titled ‘Uber Pickups in New York City’ [9], which was downloaded from Kaggle. It was obtained by FiveThirtyEight from the NYC Taxi & Limousine Commission by submitting a Freedom of Information law request. While the original dataset has about 20 files, this study is exclusively based on the file ‘uber-raw-data-sep14.csv’. A snip from this file is as shown:



Date/Time	Lat	Lon	Base
9/14/2014 6:13:00	40.7244	-73.9745	B02617
9/14/2014 6:13:00	40.7244	-73.9745	B02617

That is, it has information about the date and time, the latitude and longitude of the pickup and the TLC base company code affiliated with the pickup. As this study is concerned with the location of pickups, only the latitude and longitude information will be used.

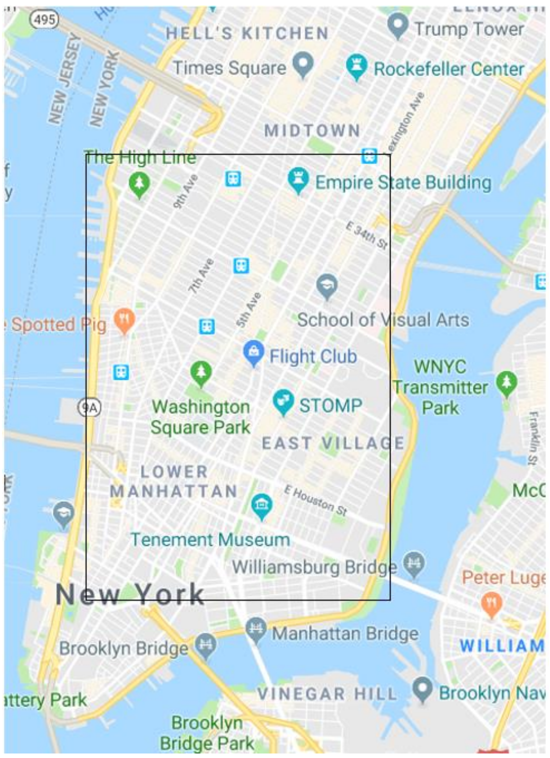
4.2 Data cleaning and refinement

The file chosen above has over a million data points for the month of September 2014, spread out over the whole of New York city. To simplify the analysis, only a specific section of Manhattan is considered for this study, which is defined by a range of latitudes and longitudes. The range chosen is:

Latitudes: (40.7115, 40.7530 N)

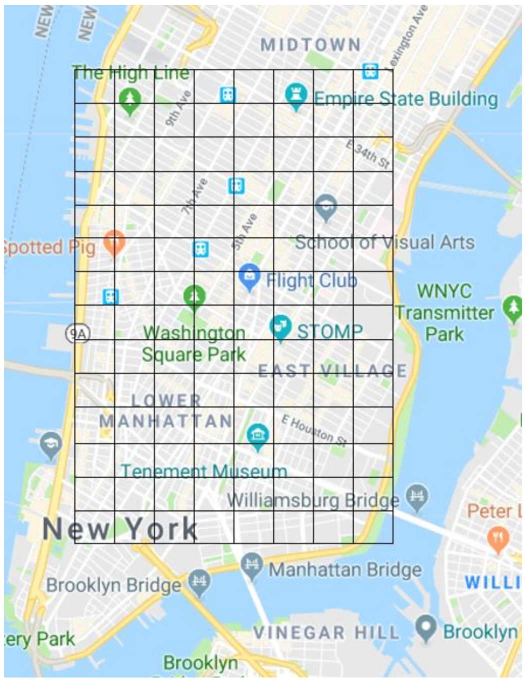
Longitudes: (-73.9750, -74.0115 E)

This results in a rectangular area as shown below, which will be the focus of this study:



Source: Google Maps

Doing this also reduces the number of data points to be considered, reducing the count to 394,153 pickups. Next, in order to make sense of the data and to derive insights, it is decided to group the pickup points based on location. The most elegant way to do this seemed to be by dividing the rectangular area into a grid with grid lines coinciding with equally spaced latitude and longitude lines. The same is shown below:



A 14 X 8 grid is used for this. The numbers 14 and 8 were chosen by trial-and-error, maintaining a trade-off between computational complexity and network resolution. Increasing the number of rows/columns would help in achieving greater resolution in the graph as each square would cover fewer blocks/streets. On the other hand, decreasing this number would reduce the network size, make computation easier.

Finally, in order to make the grid structure usable for this study, the pickup density for each square is calculated based on the dataset described above. They are also labelled as shown below:

n	0	2487	4163	1439	8426	5363	5449	6775
m	0	3065	3921	1348	5829	5113	4553	3725
l	126	4009	2354	2979	7399	6493	6310	3901
k	4	5371	5641	4113	6906	5559	4619	1881
j	4	10729	4762	4194	7025	6973	3265	1073
i	75	4470	4839	3750	5062	5157	2717	1684
h	160	4563	6988	5257	5240	4570	2081	1912
g	497	3394	5693	2449	4904	4315	2999	1963
f	172	4889	5595	4266	6808	4068	3759	1623
e	1605	5075	6460	9020	6691	2443	3386	1908
d	2077	3372	7249	6925	4430	6496	2944	929
c	1775	4978	5143	3203	3294	3429	990	223
b	1338	4648	1392	1561	1926	909	340	81
a	437	2294	171	532	678	252	60	224
	a	b	c	d	e	f	g	h

The numbers shown in the individual squares represent the number of pickups that happened in the area of Manhattan represented by those squares. For e.g. cell 'cd' had 3203 pickups whereas cell 'kb' had 5371 pickups made by an Uber cab during the period of study.

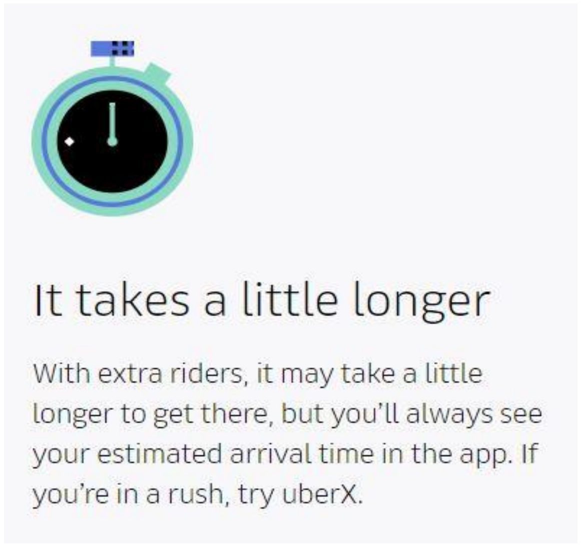
Now that the basic structure is in place, the objective of the study may approximately be re-stated in terms of this grid: "Given a source cell x_1y_1 and a destination cell x_2y_2, find the path through the grid that passes through higher valued cells as far as possible, while attempting to reduce deviations from the straight-line path between source and destination."

5. Assumptions

While every attempt has been made to keep the study as realistic as possible, it is impossible to model everything as they are in the real world. To make up for these shortcomings, and for the remainder of this study to be meaningful, certain assumptions have been made:

- While the dataset being used is for pickups made by Uber as a whole, it is assumed that the relative proportion of pickups for each cell remains the same for shared taxis.

- More pickups from a cell in the past is an indicator of higher probability of picking up a rider from that cell in the future. In other words, there is a higher probability of picking up a rider when traversing through cell ‘kb’ when compared to cell ‘cd’.
- The source and destination cells are not very close in the grid. In case they are only separated by a cell or two, it would not make sense to take a detour in anticipation of picking a new rider. Consider ‘ac’ as source and ‘ae’ as destination. There might be a tendency to detour along ‘bc-bd-be’, owing to their higher pickup probabilities. But considering the closeness of the source and destination, a straight-line path seems to be the most reasonable choice in this case.
- Riders take the ride sharing option because they want to travel in the most economical way and are hence willing to reach their destination later than when they would have, had they taken a traditional cab. This is an assumption which Uber itself makes, as is evident from the following disclaimer on its advertisement for Uber Pool.



It takes a little longer

With extra riders, it may take a little longer to get there, but you’ll always see your estimated arrival time in the app. If you’re in a rush, try uberX.

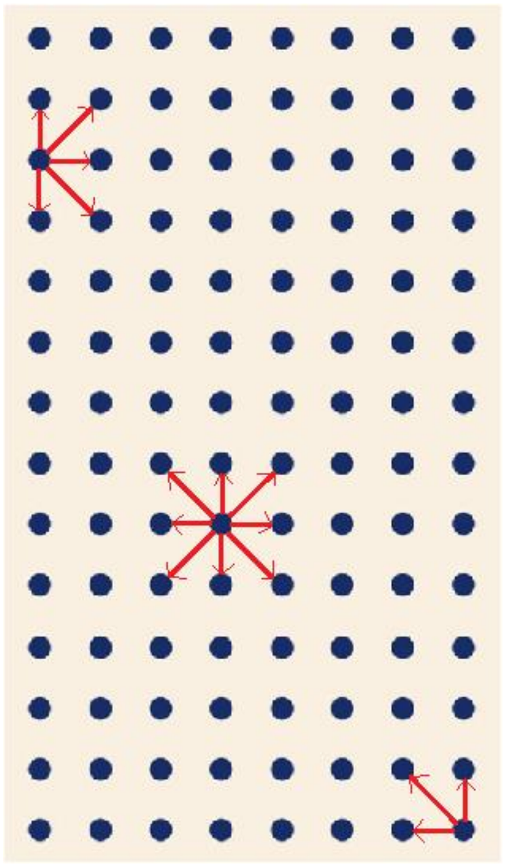
Source: https://www.uber.com/ride/uberpool/

- Co-passengers picked up on the way to the destination are traveling in the same direction. As most ride sharing algorithms match passengers according to this criterion, this study will assume this as a given. Moreover, as each square in the grid is large enough to cover a handful of streets/blocks, it may be reasonable to assume that all passengers in a trip have the same destination square on the map.

6. Modelling and Method Selection

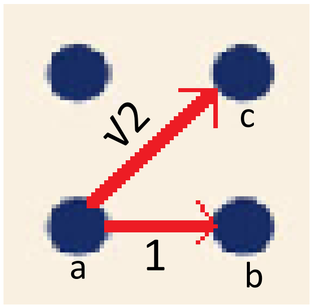
6.1 Network preparation

From the grid structure created in Section 4, it can be seen that converting to a set of nodes would enable us to model the problem as a shortest path problem through a network of nodes and edges. For this purpose, it is decided to replace each square in the grid with a node having a capacity equal to the pickup density represented by that square. Also, every node will be connected to all of its neighboring nodes through edges, that allow flow in both directions. The updated structure is shown below:



A network structure is obtained having 14 X 8 = 112 nodes. As shown by the red arrows, a typical edge node will be connected to five of its neighbors, an internal node will be connected to eight of its neighbors, and a typical corner node will be connected to three of its neighbors. In reality, this is the case for all 112 nodes in the graph, but this has not been explicitly shown to keep the figure simple.

Now, due to the symmetric grid-like structure of nodes, distances between neighboring nodes can be assumed as shown:



The distance between adjacent nodes in the same row or column is assumed to be 1 unit. So, edge a-b will have length 1 unit. By simple geometry, adjacent nodes that are not in the same row and column would be separated by √2 units. Therefore, edge a-c will have length √2 units.

6.2 ILP Formulation

With the knowledge that the problem at hand can be regarding as an application of the shortest path problem, we can try formulating the same as an integer linear program:

$$
\begin{aligned}
\min \sum_{u,v\in A} c_{uv}x_{uv} & \\
\text{s.t} \sum_{v\in V^+(s)} x_{sv} - \sum_{v\in V^-(s)} x_{vs} &= 1 \\
\sum_{v\in V^+(u)} x_{uv} - \sum_{v\in V^-(u)} x_{vu} &= 0 \quad \text{for each } u \in V \setminus \{s,t\} \\
\sum_{v\in V^+(t)} x_{tv} - \sum_{v\in V^-(t)} x_{vt} &= -1 \\
x_{uv} &\geq 0
\end{aligned}
$$

Source: https://math.stackexchange.com/questions/1595886/linear-programming-and-shortest-path

In the typical application of the shortest path problem, c_{uv} would represent the weight of edge uv. In this case, c would have to be a function of the distance between nodes, as well as the node capacities. This will be examined in depth later.

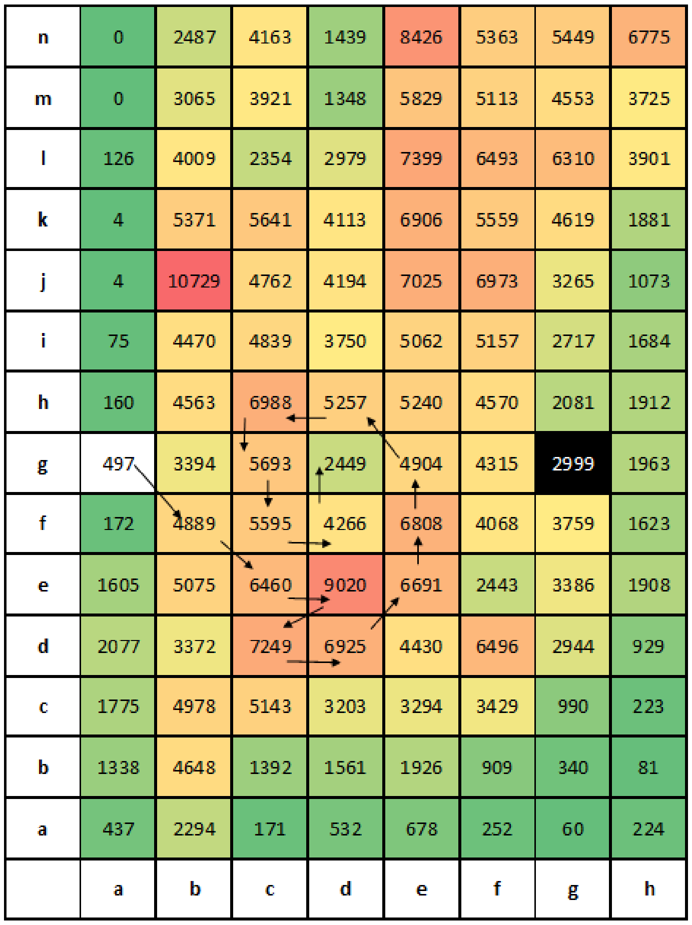
The ILP formulation is guaranteed to give an optimal result, and the number of constraints increases linearly with the number of nodes. However, it is seen that for larger problems, ILPs tend to be computationally complex, and hence turn out be quite slow. In comparison, using other strategies could save us time and would in fact, be necessary considering the real-time nature of vehicle routing. So, we will abandon this approach for now.

6.3 Greedy heuristics

As a greedy approach to solving this problem, one could consider the following strategy:

"From a given node, move to its unvisited neighbor that has the highest capacity. For each such movement, increase the capacity of the destination by a fixed amount so that eventually the destination becomes increasingly desirable. At some point, we would arrive at the destination."

Let us now apply this strategy for a sample origin-destination pair of 'ga' -> 'gg':



n	0	2487	4163	1439	8426	5363	5449	6775
m	0	3065	3921	1348	5829	5113	4553	3725
l	126	4009	2354	2979	7399	6493	6310	3901
k	4	5371	5641	4113	6906	5559	4619	1881
j	4	10729	4762	4194	7025	6973	3265	1073
i	75	4470	4839	3750	5062	5157	2717	1684
h	160	4563	6988	5257	5240	4570	2081	1912
g	497	3394	5693	2449	4904	4315	2999	1963
f	172	4889	5595	4266	6808	4068	3759	1623
e	1605	5075	6460	9020	6691	2443	3386	1908
d	2077	3372	7249	6925	4430	6496	2944	929
c	1775	4978	5143	3203	3294	3429	990	223
b	1338	4648	1392	1561	1926	909	340	81
a	437	2294	171	532	678	252	60	224
	a	b	c	d	e	f	g	h

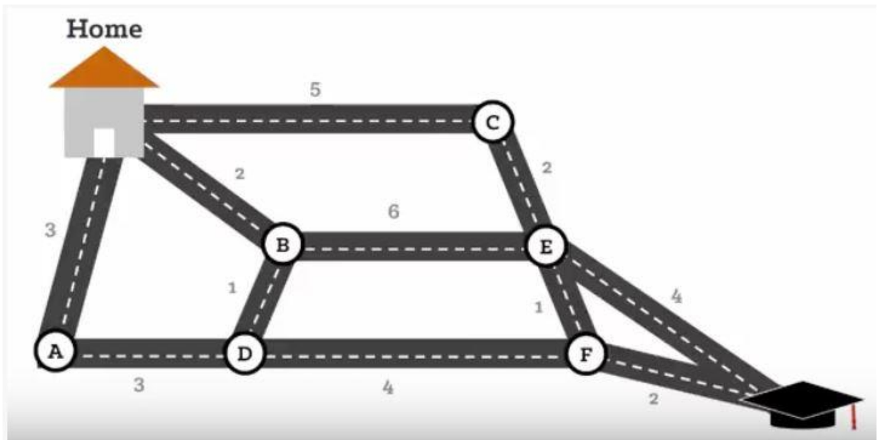
The white node indicates the source and the black node indicates the destination. Applying the greedy strategy gives us the path represented by black arrows, starting from node 'ga' and terminating abruptly at node 'gd'. We see that this approach fails to produce even a feasible solution. The fact that the destination capacity was being increased for each movement along the path did not help as we never reached any neighboring node of the destination node 'gg'.

There does not appear to be any other obvious greedy heuristic to solve this problem, and hence we abandon this approach as well.

6.4 Dijkstra's algorithm

Dijkstra's algorithm is a commonly used algorithm for finding the shortest path between nodes in a graph. It is named after its creator Edsger Dijkstra, who was a Dutch computer scientist. The algorithm computes the shortest path from a given source node to all other nodes in the graph. The problem of finding the shortest path up to a desired destination node could then be regarded as a special case of this approach. Dijkstra's algorithm can be applied to graphs that have directed or undirected edges, but the edge weights must be non-negative.

A typical application of Dijkstra's algorithm for finding the shortest path is shown below:



Source: https://brilliant.org/wiki/dijkstras-short-path-finder/

Given a road network as a graph, individual streets are the edges and street intersections constitute nodes of the graph. The edges have non-negative weights which could be a measure of the distance between nodes that they connect, the travel time, or the cost incurred in travelling along that edge. Upon specifying the source and destination nodes, the algorithm calculates the path along the graph, such that the sum of edge weights along which traversal occurs is minimized. If the edge weights represent distance, this translates to finding the shortest path between source and destination.

The pseudocode used to perform Dijkstra's algorithm is shown below:

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:            // Initialization
        dist[v] ← INFINITY                 // Unknown distance from source to v
        prev[v] ← UNDEFINED                // Previous node in optimal path from source
        add v to Q                         // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                       // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]   // Node with the least distance
                                                   // will be selected first
        remove u from Q

        for each neighbor v of u:          // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:              // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

While the time complexity of the algorithm depends on how it is implemented, the original version of the algorithm runs in $O(n^2)$ time. It is also known that using binary heap with priority queue implementation or Fibonacci heap could reduce the run time to O(E log n) and O(E + n.log n) respectively [10][11].

6.5 Applying Dijkstra's algorithm to this study

What makes the network obtained in Section 6.1 different from the typical application of Dijkstra's algorithm seen above, is the presence of node capacities in addition to edge weights. It is desired to find a path that not only reduces the distance, currently indicated as edge weights, but also passes through nodes that have a high capacity. This may be achieved by consolidating these node capacities into the edge weights, so that the resulting network would become suitable to the application of Dijkstra's algorithm.

As stated in the objective of this study, we plan to minimize the distance covered and maximize the sum of pickup densities of the nodes through which traversal occurs. This is indicative of a direct relationship between edge weight and distance, and an inverse relationship between edge weight and pickup density. Then, the simplest equation that consolidates these two relations would be:

$$\text{Edge weight} = \text{distance} / \text{pickup density}$$

In more specific terms, if we consider moving from node a to b, the edge weight a-b will be given by:

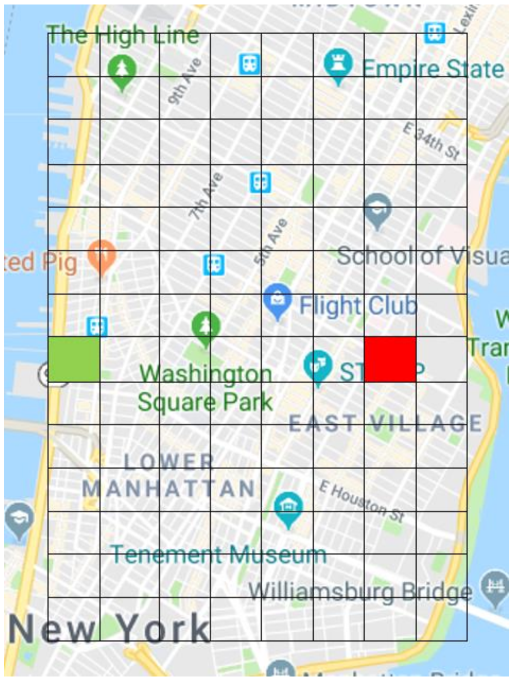
$$\text{Edge weight }(a,b) = \text{distance }(a, b) / \text{pickup density at node } b \tag{1}$$

This computation is done for all edges in the network, giving us the edge weights required for the application of Dijkstra's algorithm. For the special case of the pickup density being equal to 0, the edge weight is explicitly set to be equal to the distance.

7. Analysis

An implementation of Dijkstra's algorithm in Python, obtained from GitHub [12] is used to perform experiments for this study. The node connections based on the grid of nodes and edge weights are fed into the code, in order to get as output the desired path that balances distance and population density. The analysis is carried out in the PyCharm Community Edition IDE on a system having 16GB RAM and an Intel Core i-7-7500U processor.

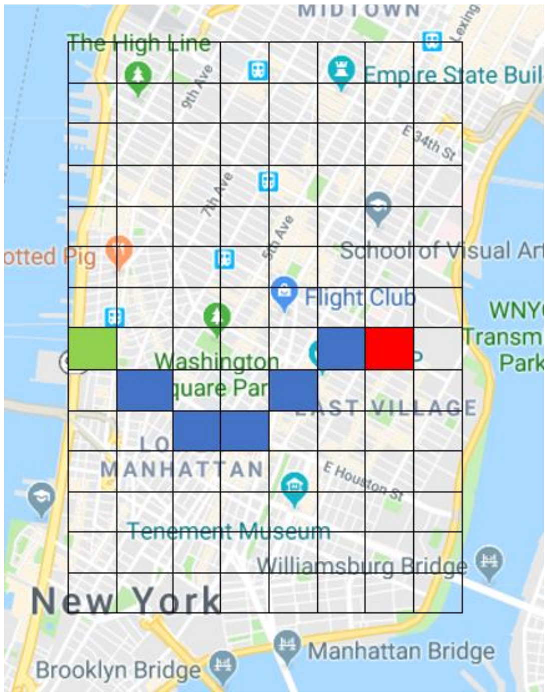
As a simple starting case, the task of moving from node 'ga' to node 'gg' is considered.



The source and destination nodes are shaded green and red respectively, in the grid. The code allows us to feed in these two parameters, and outputs the desired path. In this case, Python outputs the following:

```
('ga', 'fb', 'ec', 'ed', 'fe', 'gf', 'gg')
```

This indicates the path to be followed from ‘ga’ to ‘gg’. That is, from ‘ga’ move to node ‘fb’, then move to node ‘ec’ and so on until node ‘gg’ is reached. This is shown visually in the map below:



Thus, the path shown by the blue squares is the required path from source to destination that has the highest probability of picking up a co-passenger while ensuring that the distance travelled is minimum.

Now, it is not very difficult to see that despite incorporating distance into the edge weight, we still have a significant detour from the intuitive straight-line path between ‘ga’ and ‘gg’. In reality, if there is a considerably large detour, the cost spent on fuel for traveling the extra distance might actually outweigh the profit made by picking up an extra rider. In essence, we might have been better off doing a simple shortest path analysis on the network.

One way to improve this situation is to realize how the current way of modelling the edge weights gives a somewhat equal importance to distance between nodes and the pickup density. With this rigid formulation, it is not possible to give one factor more importance than the other. It might then make sense to modify this formulation to allow for some flexibility that would enable users to specify the relative importance of either factor as per their specific need.

For instance, a cab company may decide that, during particular hours of the day, it would be more profitable to take the shortest path between two points as the chances of picking up a co-passenger are scarce. Or they might feel that detours could in fact be profitable, but the detour distance needs to be restricted to within a certain factor of the shortest distance path between the two points.

In order to allow for such flexibility in the formulation, it is decided to modify the former formula for edge weight as follows:

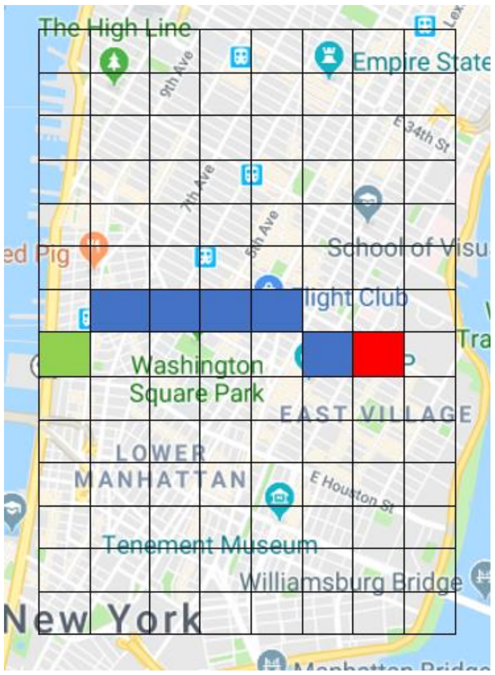
$$\text{New Edge weight (a, b)} = \text{distance (a, b)} / (\text{pickup density at node b})^k \qquad (2)$$

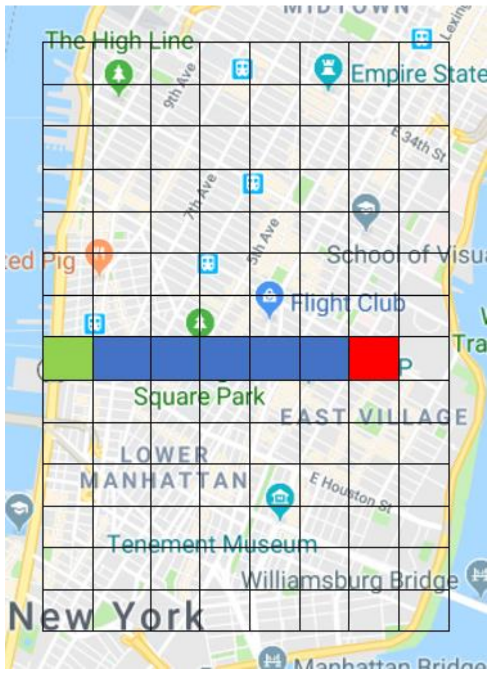
By introducing the parameter k, we are now able to change the relative importance of the pickup density with respect to the distance between nodes. A higher value of k would signify greater importance for the pickup density and would hence allow for longer detours. A lower k value would reduce the importance or effect of the pickup density with respect to the distance. This would restrict detours between the source and destination. In the limiting case where $k \rightarrow 0$,

$$\text{New edge weight} \rightarrow \text{distance} / (\text{pickup density at node b})^0 \rightarrow \text{distance}$$

In other words, for k = 0, the new edge weights would simply be equal to the distances between nodes, and the pickup densities would be avoided altogether. The path thus found would be the shortest path between the source and the destination.

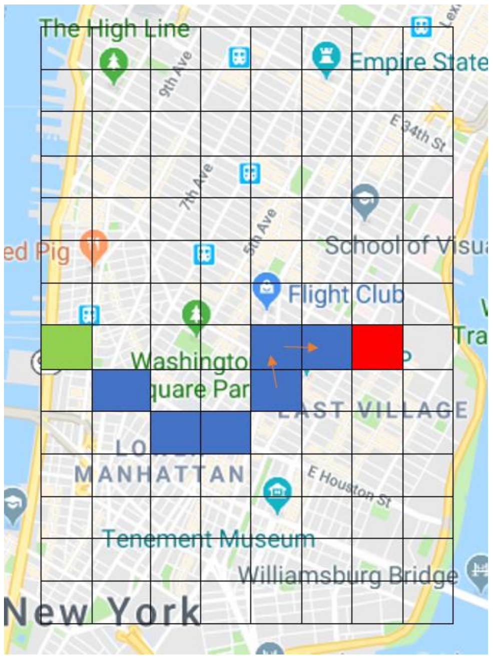
It may be noted that the earlier formula (1) was in fact a special case of formula (2) with k = 1. The value of k may now be manipulated for the sample origin destination pair tried earlier:

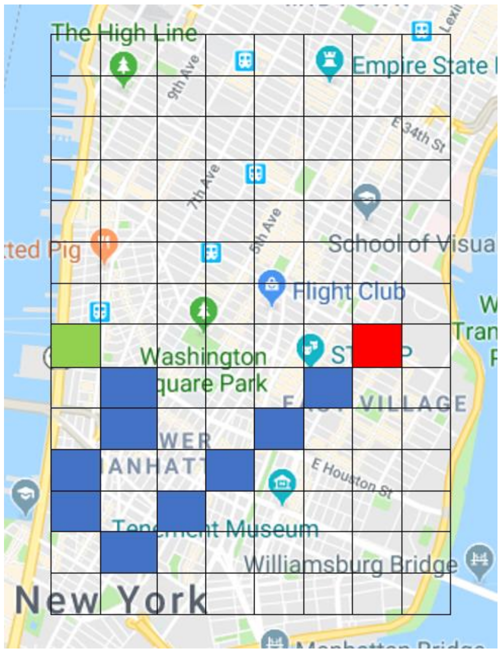

k = 0.6, ('ga', 'hb', 'hc', 'hd', 'he', 'gf', 'gg')

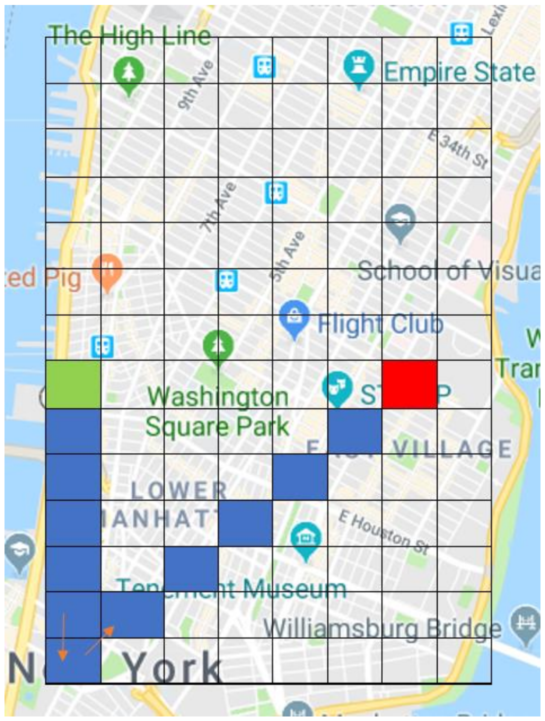

k = 0, ('ga', 'gb', 'gc', 'gd', 'ge', 'gf', 'gg')

The earlier case represented k = 1. Upon reducing k to 0.6, it is seen that a new path is suggested, which is shorter than the original. There is a detour nevertheless and is a consequence of a reduced albeit non-zero importance given to the pickup density. Finally, the limiting case of k = 0 shows a straight-line path between source and destination, which again was expected since edge weights equal distances in this case.

We could also try increasing the value beyond 1. The following are plots for the same origin-destination pair with k-values 10, 100 and 500 respectively:


k = 10


k = 100



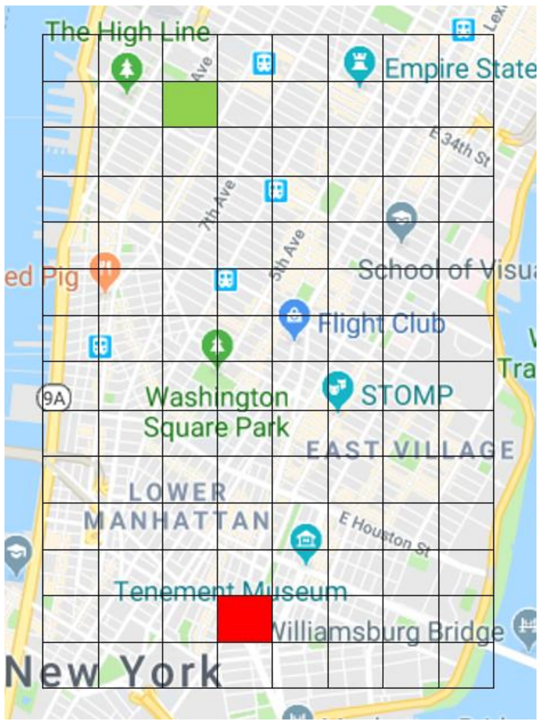
k = 500

From these plots, it is seen that as the value of k is increased, the length of the path from the source to the destination keeps increasing. This agrees with our expectation that an increasing importance is being placed on the pickup density, which in turn is causing larger detours. Also, for k = 1000 (plot not shown), the same path as k = 500 is obtained. In fact, it is experimentally found out that the same path is obtained for all values of k >=145. This means that for k values above this threshold, the longest path passing through the high pickup density nodes has been determined, and no longer path will be determined.

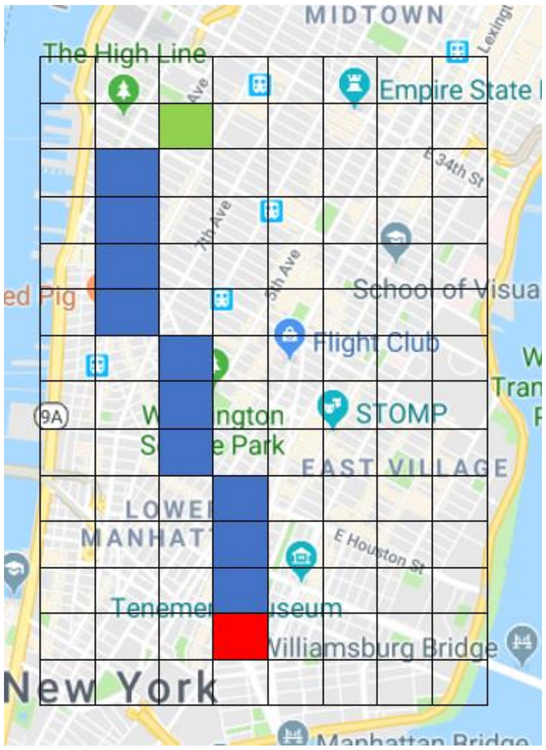
The solutions above are obtained at extremely small runtimes, the highest being for the k = 500 case, which on average turned out to be 0.003s.

We may thus achieve a trade-off between taking a detour and traveling through nodes with high pickup density values, by setting the value of k accordingly.

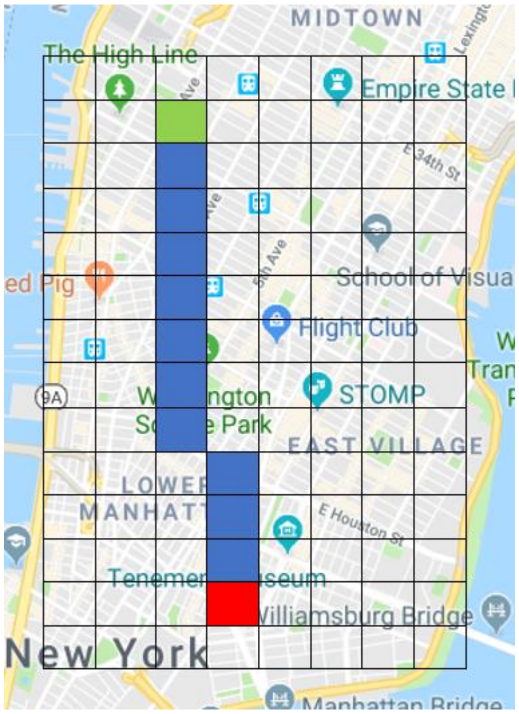
The same procedure is now repeated for a different origin-destination pair to validate our findings. We plan to find a route from ‘mc’ to ‘bd’:



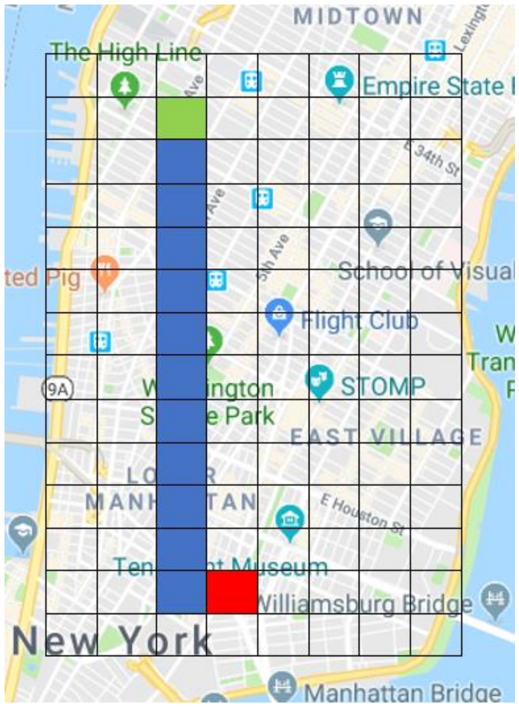
We look at the plot for k = 1



We can compare this with the plots for k = 0.5 and k = 0:

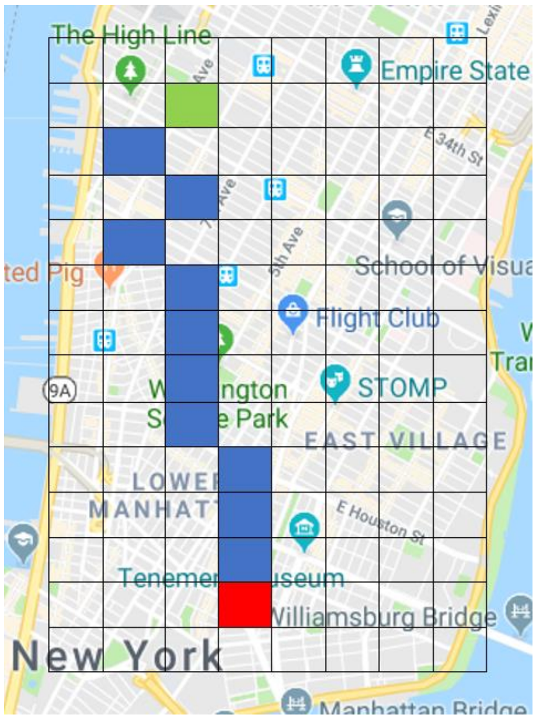


k = 0.5

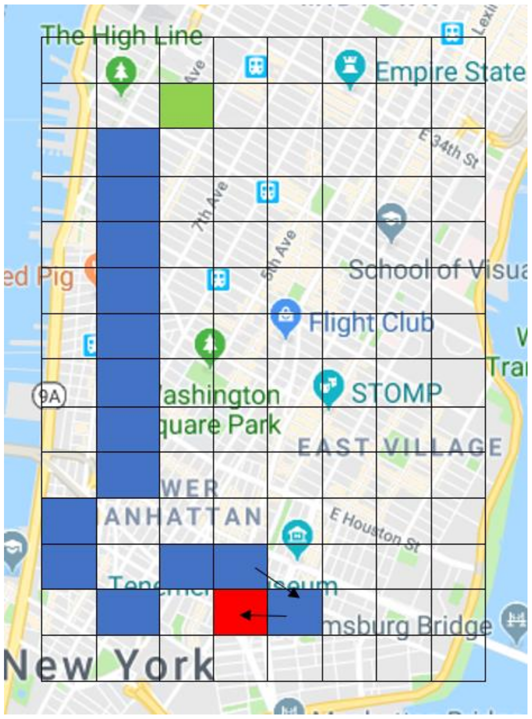


k = 0

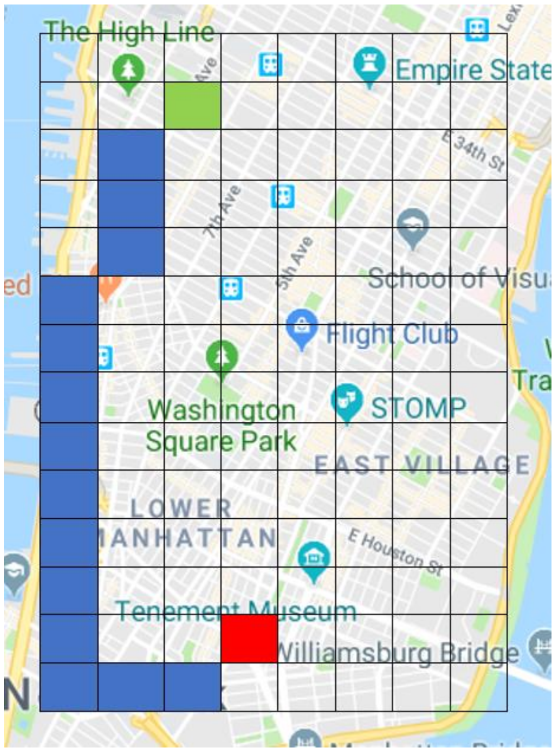
From the three plots, we can see that a trend similar to the previous origin-destination pair is seen. For k = 1, there is a reasonable amount of detour, which reduces slightly for k = 0.5 and finally approximates a straight-line path for k = 0. Next, we look at k values greater than 1:

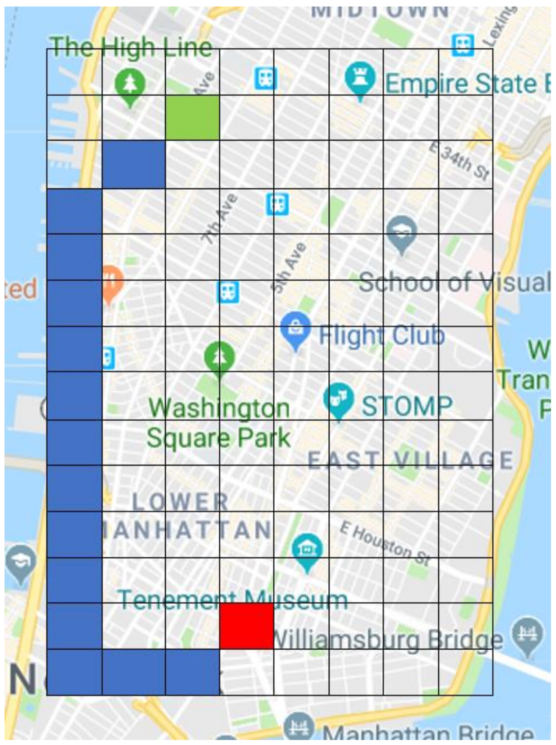


k = 10



k = 100


k = 500


k = 1000

Again, the same trend of increasing path length with an increase in the value of k is observed. In some cases, there may not strictly be an increase in path length in terms of the number of intermediate nodes, as seen for the paths generated for k-values of 100, 500 and 1000. But, what matters is that that the deviation from the straight-line path (k = 0) increases progressively. It is experimentally found that for k-values >= 538, the same path is obtained each time.

Here again, the solutions are computed extremely fast, the highest time taken being for the k = 1000 case, which on average turned out to be 0.005s.

Thus, we may conclude that in both cases, there exists a value of k above which the same path is outputted. This path then represents the longest path from source to destination that passes through the high density pickup points while maintaining a balance with the distance covered.

8. Results

For the two cases seen above, we can make the following general remarks:

- If k is set to 0, the shortest path between the source and destination is determined.
- As the value of k is increased, deviation from the straight-line or the shortest distance path increases progressively
- There exists a saturation value of k, which is dependent on the origin-destination pair, above which the path determined remains the same, irrespective of the k-value.

Path	k saturation value
ga -> gg	145
mc -> bd	538

Looking at the range of acceptable k-values that cause a change in the determined path and the amount of detour that occurs for each setting, a reasonable value of k that provides a good trade-off between the detour distance and the potential benefit of picking up a new passenger seems to be 1. It could be argued, looking at the two plots for k = 1, that this setting itself allows for a reasonable detour distance and anything higher could potentially outweigh the profit that would have been made by serving another rider.

However, it is important to note that the actual value of k must be decided on a case-by-case basis, depending on how much relative importance one wishes to accord to the two conflicting factors. Also, as seen in both cases, path calculation occurs very fast and hence, this approach may be considered for real-time use.

9. Scope for improvement

While this study looks at routing of shared taxis based on two factors- path length and passenger pickup probability- at a high level, it may be improved in the following ways:

- As this study is based on dividing the city map into a grid, and each grid covers a sizeable area of the city, the finer details of the exact street/intersection of pickup and drop off are lost. The same can be extended to the output path- this study only suggests the general *direction* of moving from source to destination, rather than the specific road/street to take. Going to a finer level of detail would entail deriving street- or block- specific values for the pickup density- which would be cumbersome- but certainly doable.
- This study makes use of the dataset only for the location information of the pickups. Given that the time and date information are available too, one could think of creating a dynamic, time-of-the-day and day-of-the-week dependent pickup density map, as opposed to the static pickup density map used in this study. This would then enable users to set different k values for different time periods. For e.g. during hours when the pickups are generally low, we could prefer to avoid large detours as there would be a lower chance of picking up an extra rider. So, we could set k to a low value for such hours, compared to what we would have set during rush hours.
- This study assumes that any co-passenger picked up along the way would also be traveling towards the same destination as the original passenger for whom the output path was found. The way modern cab routing algorithms club users is by ensuring that they are traveling in the same direction, but not necessarily towards the same destination. An improvement over this study could take this fact into account, and give a provision to add multiple destinations, and subsequently update the output path to accommodate the new destination nodes.

10. Conclusions

In this study, the routing of shared taxis with the objective of maximizing the probability of picking up a co-passenger and minimizing the total distance was considered. A dataset showing Uber pickups in the city of New York was used to create a pickup density grid, which was superimposed over the original map to obtain a heat map of pickups. After consolidating pickup densities and distances into the edge weights, Dijkstra's algorithm was applied to obtain the desired path. Two origin-destination pairs were considered, and it was concluded that adjusting the parameter k would allow for modifying the relative importance of one factor with respect to the other. On one end of the spectrum, a k value of 0 would fetch us the shortest path between the source and destination. At the other end, it was experimentally found that path determination would saturate at a particular value of k which was dependent on the origin-destination pair considered. Also, this path would indicate the longest path that passed through the high density pickup points while maintaining a balance with the distance covered. In conclusion, it was realized that k needs to be adjusted on a case-by-case basis, and its appropriate value could be determined based on the trade-off one is willing to make between fuel expenditure for detouring and the potential profit one expects from picking up a new rider. Some possible leads for improving the present study include modelling at a more granular level and having a dynamic, time-dependent pickup density grid.

11. References

[1] Wei, Chong & Wang, Yinhu & Yan, Xuedong & Shao, Chunfu. (2017). Look-ahead Insertion Policy for A Shared-taxi System Based on Reinforcement Learning. IEEE Access. PP. 1-1. 10.1109/ACCESS.2017.2769666.

[2] Jung, Jaeyoung & Jayakrishnan, R & Young Park, Ji. (2015). Dynamic Shared-Taxi Dispatch Algorithm with Hybrid Simulated Annealing. Computer-Aided Civil and Infrastructure Engineering. 31. 10.1111/mice.12157.

[3] Li, B & Krushinsky, Dmitry & A. Reijers, Hajo & Van Woensel, Tom. (2015). The Share-a-Ride Problem: People and Parcels Sharing Taxis. European Journal of Operational Research. 238. 10.1016/j.ejor.2014.03.003.

[4] Sanders, Peter & Schultes, Dominik. (2006). Engineering Highway Hierarchies. ACM J. Exp. Algorithms. 17. 804-816. 10.1007/11841036_71.

[5] Masoud, N., & Jayakrishnan, R. (2017). A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system. Transportation Research Part B: Methodological, 106, 218-236. DOI: 10.1016/j.trb.2017.10.006

[6] Lin, Yeqian & Li, Wenquan & Qiu, Feng & Xu, He. (2012). Research on Optimization of Vehicle Routing Problem for Ride-sharing Taxi. Procedia - Social and Behavioral Sciences. 43. 494–502. 10.1016/j.sbspro.2012.04.122.

[7] Rong, H., Zhou, X., Yang, C., Shafiq, M.Z., & Liu, A.X. (2016). The Rich and the Poor: A Markov Decision Process Approach to Optimizing Taxi Driver Revenue Efficiency. CIKM.

[8] Y. Ding, S. Liu, J. Pu and L. M. Ni, "HUNTS: A Trajectory Recommendation System for Effective and Efficient Hunting of Taxi Passengers," 2013 IEEE 14th International Conference on Mobile Data Management, Milan, 2013, pp. 107-116.

[9] https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city

[10] https://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/

[11] https://www.quora.com/What-is-the-complexity-of-Dijkstras-algorithm

[12] https://gist.github.com/kachayev/5990802