BMGT 831 Network Optimization Project Report

Smart Routing for Uber Contract Driver

Team name: **G**ala**XY** Team member: Mingze **G**ao, Yang **X**ie, Zhiyuan **Y**ang

Abstract: With the development of smart phones, smart apps, and internet of things, Uber now is revolutionizing the delivery of what we conventionally think of as taxi service. More and more people join Uber as the contract drivers to gain some revenue. Therefore, a smart strategy, e.g. empty car prepositioning and customer rejection, becomes more and more important for these contract drivers to maximize the profits. In this project, we proposed 2 heuristic prepositioning policies and 2 heuristic customer rejection policies based on the prior knowledge of the city maps and the competitors' behaviors. We analyze these policies through comparison of the maximum amount of revenue. We implement our algorithms in Matlab, and optimize code to make running time to be within few seconds.

Keywords: Uber contract driver, heuristic, routing, maximize revenue

I Introduction

Uber was founded as "UberCab" by Travis Kalanick and Garrett Camp in 2009 and the corresponding smartphone app was released the following June. It develops, markets and operates the Uber mobile app, which allows consumers with smartphones to submit a trip request which is then routed to Uber drivers who use their own cars. As of April 12, 2016, the service is available in over 60 countries and 404 cities worldwide [1]. The dramatic development of Uber inspires several other companies, which have copied Uber's business model, leading a trend that has come to be referred to as "Uberification". Any private drivers can join Uber as the contract driver after passing some necessary checks.

In this project, we play a role of contract driver in Uber system. Our goal is to maximize the revenue under the competition of other contract drivers. We simply the problem in to following:

> *Each group will have control of one car on a road network. The car is able to collect money by moving people (or possibly food or other goods) from one location to another location across the network. There are also k competitor cars on the road. The goal is to maximize revenue by prepositioning the car to maximize capture of demand, and make decisions about which customers to pick-up.*

In order to maximize the revenue, we need to set up a smart strategy/policy to solve the following two problems:

1) The empty-car policy: if the car is empty (no customers), what is the best routing strategy?
2) The pick-up policy: given a customer request nearby, how to determine whether to pick up the customer or reject the request.

Based on these two problems, we have develop several heuristic policies. The reasons why we say "heuristic" instead of "optimal" is because:

1) Suppose there is a function f defined revenue value, for the city with larger map, there will be numerous unknown parameters inside f. So it is very impractical to solve argmax f under current computer's ability.
2) We only know the prior knowledge of the map, e.g. the prior probabilities of the customers' requests occurring. It is impossible to predict the future.

The rest of report is organized as: 1) in section II, we do the literature review of related works; 2) in section III, we analyze the input information and introduce our preposition policy; 3) in section IV, we propose several rejection policies; 4) in section V, we first summarize the decision flow as a role of driver, and then we analyze the results of corresponding algorithms and provide our intuitive explanation; 5) in section VI, we conclude the project.

II Literature Review

In recent years, car routing problem has becomes a hot topic in operation research field, especially for truck routing. Lots of smart routing algorithms have been well designed to solve all kinds of real-world issues.

B. Chakraborty et.al. [2] proposed an efficient algorithm to solve the route planning in navigation system based on Genetic Algorithm (GA). Given a set of origin-destination pairs, there could be multiples feasible routes for a driver. Intuitively the best routes is the shortest path, which can be easily solved through Dijkstra or Bellman-Ford algorithms. However, the shortest path may not be best routes from the point of view of the driver's satisfaction, e.g. the driver requires shortest time. By applying GA, [2] can efficiently find the alternate non-0verlapping routes to satisfy the driver's requirement. This approach can be implemented into the Uber's routing problem in order to provide the best routes for the contract drivers.

Yeqian Lin et.al. [3] proposed a model for vehicle routing problem for ride-sharing taxi. Their model can improve transportation efficiency of taxi and mitigate urban traffic congestion. They take both customers and taxi drivers into account, and optimize the following objectives: minimization of operating cost, maximization of customer satisfaction, and travel mileage, waiting time.

Jorge Nunes et.al. [4] presents a case study of a taxi drive company. Their algorithm is trying to improve the efficiency of picking up customers in order to save time and fuel. They reconstruct the problem and transform it into the typical travelling salesman problem (TSP) where the goal is, given a set of cities and roads, to find the best route by which to visit every city and return home. The proposed algorithm is based on GA.

[5] takes the survey of vehicle routing problems. They classifies routing problems from the perspective of information quality and evolution, and present a comprehensive review of applications and solution methods for dynamic vehicle routing problems.

III Prepositioning Policy

The prepositioning policy is to solve the first problem in section I. When the car is empty, what can we do in order to maximize the revenue? But before introducing the prepositioning policy, we first list our analysis of the input data files.

A. Input Information Analysis

In the input files, we have the following parameters:

N: the node set denoted $N = n1, n2, n3, \ldots, n_N$

***G*:** street network; $G_{ij} = t$ implies that it requires t minutes to traverse from n_i to n_j

$\boldsymbol{P_i}$**:** probability that a customer appears at node *i*

$\boldsymbol{D_{ij}}$: probability that the customer at node *i* requests a ride to node *j*

$\boldsymbol{M_{ij}}$**:** income for a ride from node *i* to node *j*

To make it easy to understand, we visualize the data using Problem 2 Instance 1 as an example. The figure is shown in fig. 1.

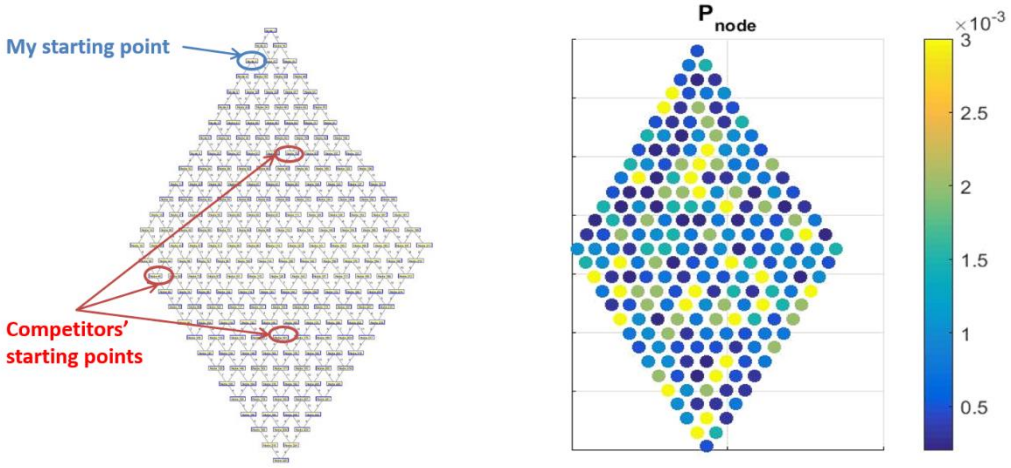

Fig. 1. Problem 2 Instance 1 visualization

The left figure shows the topology of the map. It is a diamond shape grid. We also mark the starting position of our car and three competitor cars, because the starting position is very important when deciding the next move. We will explain this in the later. The figure on the right shows the corresponding prior probabilities of the nodes in the map. From the figure, we can observe that:

1) The prior probabilities P_{ij} is relatively small, making the customers' requests very sparse. This is reasonable since it is close to the real world case. Besides, the sparse requests challenges the effectiveness of our proposed strategy. Think of that if the probabilities are very high, the customers' requests will be very intense. Our preposition policy and rejection policy might be unnecessary.
2) The probabilities P_{ij} looks randomly distributed. The node with high probability may have neighbors with very low probability. We will calculate the Revenue Expectation to reform this map and smooth the difference between the node and its neighbors.

B. Revenue Expectation Calculation

1. Single node Expectation calculation

Our prepositioning policy are built on whether the current node will provide good enough revenue. We evaluate every node by calculating the revenue expectation. The equation we use to initialize the expectation is following:

$$E_i^0 = \sum_{j=1}^{n} P_i D_{ij} M_{ij} \tag{1}$$

where all the items in the right side has been explained in above subsection. E_i^0 indicates the revenue expectation of node i in 0^{th} transition. We cannot directly calculate the expectation of every node because all these nodes are interfered with each other, which means the expectation of one node depends on the expectations of all the rest nodes. So solve the dependency, we iteratively calculate the expectation step by step. The next step depends on the previous calculated expectation. Fig.2 shows the procedure of calculation.

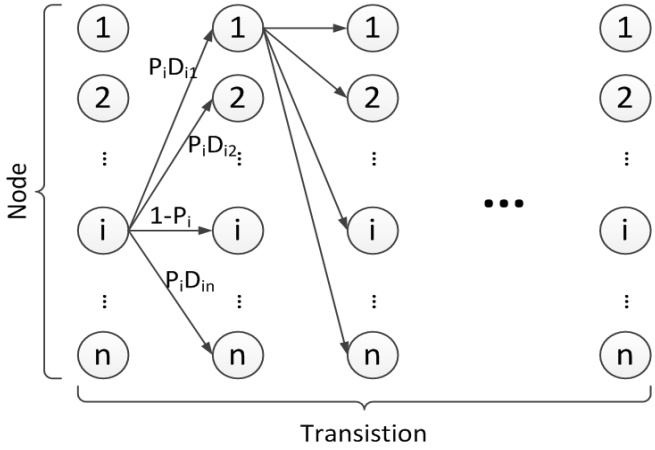

Fig.2 Revenue Expectation calculate transitions

The procedure in fig. 2 can also be written in the following equation:

$$E_i^m = \sum_{j=1}^{n} P_i D_{ij} (M_{ij} + E_j^{m-1}) \tag{2}$$

Where E_i^m indicates the expectation of m^{th} procedure. Obviously E_i^m will converge eventually. Based on our experimental result in Matlab, E_i^m will converge when m = 5.

C. Accumulated Expectation Calculation

In above subsection, we discussed the revenue expectation of a single node. However, if our car is in node i, we can also pick up the customers nearby me, if I am closer than the competitor. To address this issue, we use the accumulated expectation to indicate how "good" this node is. We define a radius r, as shown in fig. 3. Given a node i, the accumulated expectation will be the summation of all the nodes that the distance from these nodes to i is less than r.

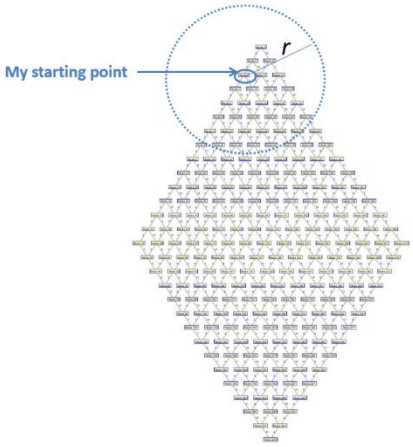

Fig. 3. Accumulated Expectation with radius r.

We use the result of single node expectation, the equation is shown below:

$$E_i^{m,r} = \sum_{\forall j, dist(i,j) \leq R_{exp}} E_j^m \tag{3}$$

The radius r is defined as R_{exp} in the equation. In this problem, we set the value of parameters as:

$$m = 5, R_{exp} = 2 * \overline{edge(i,j)}$$

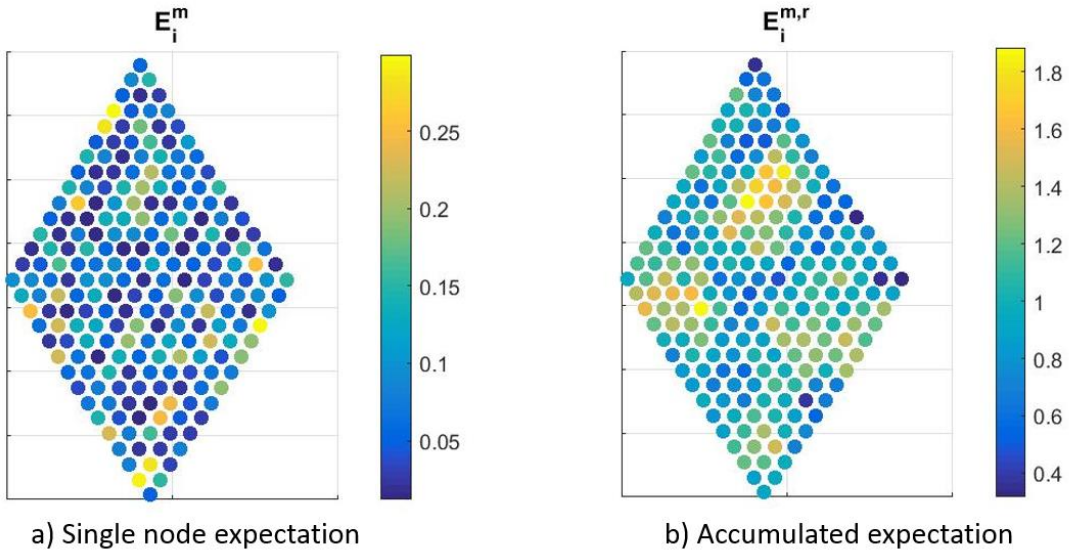

Fig. 4. Expectation comparison

Fig. 4 shows the expectation result comparison. The left figure shows the single node expectation calculated in equation (2). The right figure shows the accumulated expectation calculated in equation (3). We can see that compared with single node expectation, accumulated

expectation is more like a smoothing process. The expectations within some region in b) are very close. This is similar to the real world case. For example, cabs like to hang around some places such as airports, bus or train stations, because these places have more customers and the customers like to travel further.

D. Empty Car Prepositioning Policy

In this subsection, we proposed two preposition policies when the car is empty. The first policy is called neighbor move, and the second policy is named radius move. The neighbor move is to move to the neighbor node whose accumulated expectation is highest amongst these neighbors. The radius move is to move to the node whose accumulated expectation is highest amongst all the nodes within distance R. The move path is the shortest path. Fig. 5 shows a motivation example of these two policies.

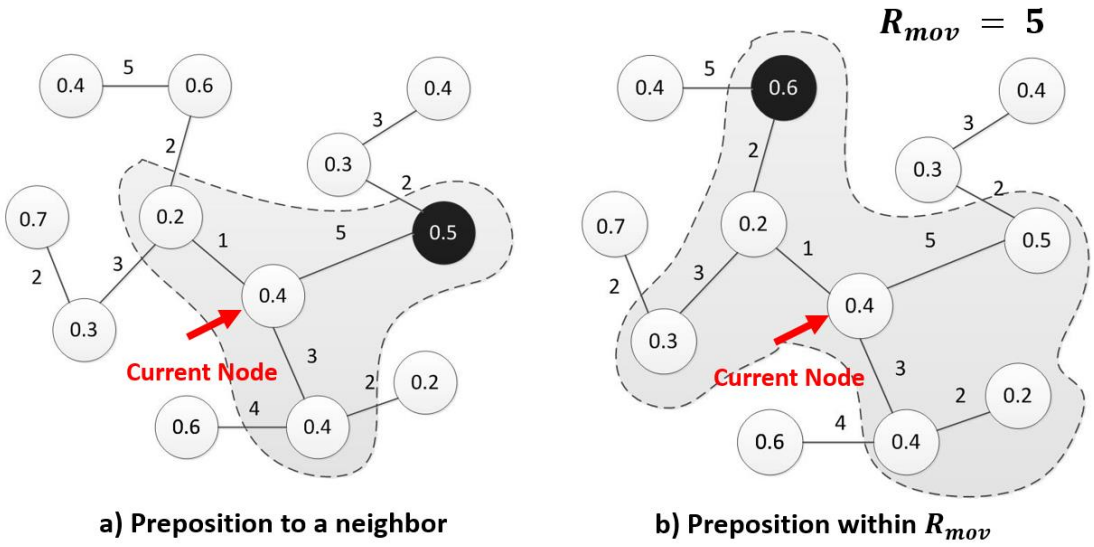

Fig. 5. Prepositioning policy

In fig. 5, the gray region is the set of candidate nodes for next move. We have marked the current node, and the black node is the target node to move to. In a), we adopt neighbor move, the highest expectation within gray region is 0.5. So the next move is from current node to the node with 0.5 expectation. b) shows the radius move. We define the radius value $R_{mov} = 5$, so the node is a candidate node if shortest distance to current node is less than 5. Apparently the next move is to the node with 0.6 expectation.

Both two policies have some drawbacks. We use fig. 5 to explain the weakness of each policy.

1) The neighbor move may stack in to some local optimal node.
   Considering fig. 5 a), the next move is to 0.5. However, after move to 0.5, all the neighbors around 0.5 are less then itself. So based on the policy, we will stack at 0.5 if no customer shows up. Therefore, we will perform like the competitor cars, who stop when empty.
2) The radius move will go through some nodes with very low expectation.
   Considering fig. 5 b), the next move is 0.6 shown in black. Since our move policy obeys the shortest path, we need to go through 0.2 to reach 0.6. However, 0.2 is a very low

expectation, which means we can only earn very few amount of money before we reach 0.6. If we set R_{mov} to be large, and the nodes lays on the shortest paths are "bad" nodes, we will definitely lose the chance of earning more money.

One may ask a question that: if there already exist competitor cars in the black node, shall we still move? Because the competitor cars are always closer than us if the customers show up in the black node. The answer is yes. The competitor cars does not affect our policy. To illustrate this answer, we consider two scenarios:

1) Before we reach the black node, a customer shows up and picked up by the competitor car.
   In this case, when we reach the black node, its expectation does not change. Because all the parameters are the same. For example, the probability of customer's request maintains the same. It is independent with what happened in previous time.
2) When we reach the black node, there are some competitor cars in the node already.
   Apparently we don't need to worry about this case, since our car has higher priority than the competitor cars. If a customer request shows up, we can take it first.

IV Rejection policy

In section III, we solve the question of "How to route my car when it's empty?". In this section, we will answer the second question in section II: "How to decide whether to accept or reject a customer?". Similar with the preposition policy, we proposed two rejection policies.

A. Distance Rejection Policy

Reject the request if the customer is too far away from me. We define a rejection radius $R_{rej} = \alpha \times d$, where d is longest shortest path between any two nodes in the graph. α is the scaling factor.

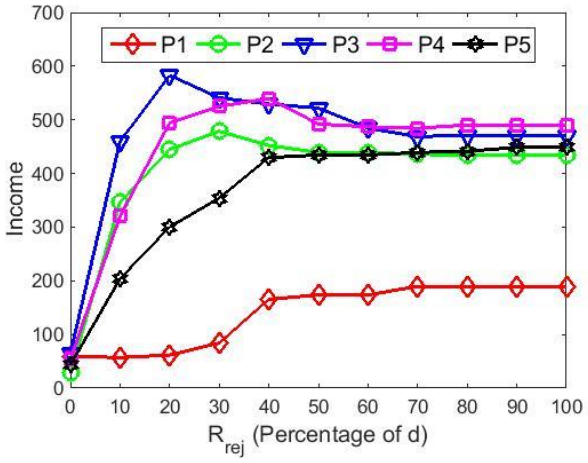

Fig. 6 rejection radius scaling.

This rejection policy is very straightforward. Here we show the results when scaling the factor from 10% to 100% in fig. 6. In the figure, P1 to P5 represent different graphs. From the figure we can observe that:

1) If the rejection radius is very small, we will gain less revenue.
2) If the rejection radius is larger enough (e.g. $\alpha > 60\%$), the revenue becomes stable.
3) In most cases, we can gain more revenue if the rejection radius is within the range of (20% ~ 40%).

These three observation obeys the intuition. For 1), if the rejection radius is small, our car will be empty in most time, losing chances to earn money. For 2), if the rejection radius is too large, we need to travel through very long distance to pick up the customer, therefore wasting chances for "good" customers nearby.

B. Expectation Rejection Policy

Our second rejection policy is decided by the revenue expectation. If the revenue expectation for not picking up the customer is larger than the money I can earn from the order, reject it.

The revenue expectation (I_{exp}) is defined in equation (4):

$$I_{exp} = \sum_{i \in N'} [1 - (1 - p_i)^{t - t_i}] \times \overline{I_i} \tag{4}$$

All the terms in the equation are defined as:

t : time from my position to the customer

t_i: time from my position to node i

$\overline{I_i}$: average income if I take a customer at node i

N' : the set of nodes that within t distance from me

p_i : the probability that a customer shows up at node i

Based on the second rejection policy, the decision is made not by the pickup distance but the revenue expectation. This is more realistic since our goal is to maximize the revenue not to minimize the travel distance. In the first rejection policy, one bad case could be that we travel relatively long distance to pick the customer, but the customer only need a very short drive. However, we will reject the customer like this case based on the second policy, because we judge the customer by revenue instead of distance.

V Decision Flow and Policy Comparison

Fig. 6 shows the state transition graph of our proposed smart strategy. There are three states: move for prepositioning, move for task, and wait for customer. On the directed edges, we list the transition conditions. The two moves are very easy to understand. The particular state is "wait" state. The prepositioning policy is trying to lead us to the "better" node. However, as we mentioned in section III, we could stuck in some local optimal based on the prepositioning policy. In this case, we need to wait for the new customer appears.

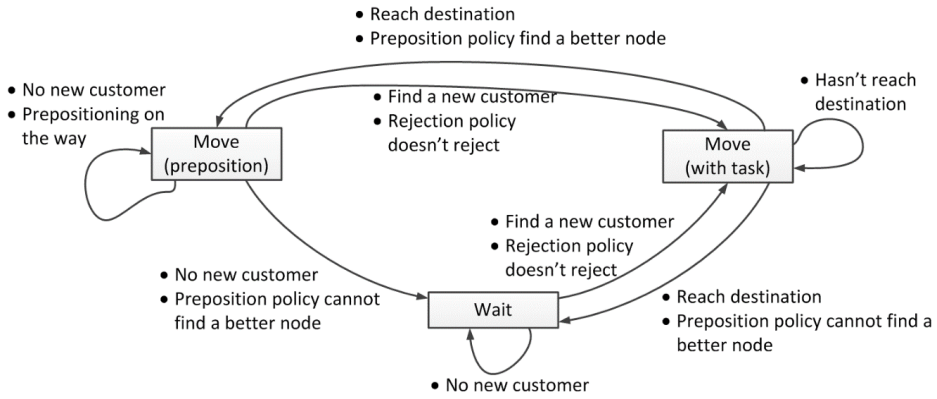

Fig. 6. State transition graph of proposed smart strategy

Next, we compare the prepositioning policies and rejection policies by analyzing the corresponding results. We implement these algorithms in Matlab 2014b. The experiment shows that the running time of our code is within a few seconds.

A. Comparison of Prepositioning Policies

To compare the prepositioning policies, we use second rejection policy to reject bad customers. The result is shown in fig. 7. $R_{mov} = \beta \times r_{mov} = \beta \times \overline{edge(\iota, \jmath)}$, which is the radius for next prepositioning move. We average results from all the given instances. The red dash line indicates the baseline income. The baseline is to perform like a competitor car, which is stops when empty and moves when there is customer. The blue dash line is for the neighbor move policy. The neighbor move always goes to its neighbors, so it is not affected by R_{mov}. The black folding line shows how the revenue varies with the increase of R_{mov}. From the figure, we can conclude that:

1) When the car is empty, move is much better than stay.
2) The neighbor move and radius move have very similar performance. But in general, radius move is slightly better, since it is more possible to reach the global optimal instead of stuck in local optimal.

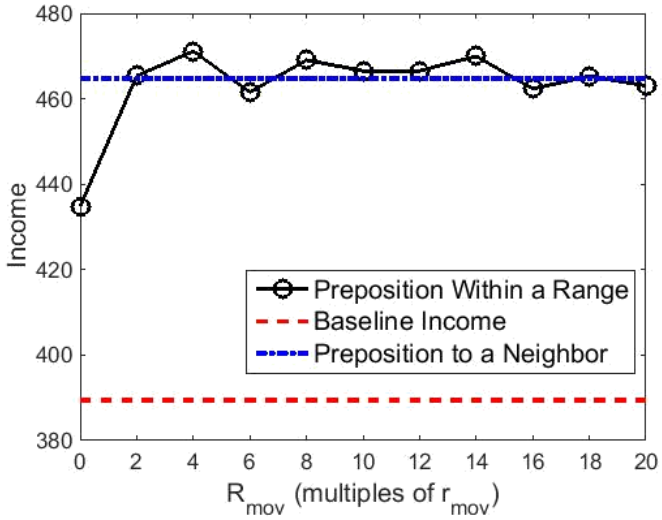

Fig. 7. Comparison of prepositioning policies

B. Comparison of Rejection Policies

To compare the rejection policies, we adopt neighbor move as the prepositioning policy when car is empty. In figure 8, $R_{rej} = \alpha \times d$, d is the graph diameter (the longest shortest distance between any two nodes). We average the results of all given instances. Similar like the figure 7, the red dash line indicates the baseline strategy (like competitor car). The blue dash line is for expectation rejection policy. Clearly it is non-related to distance R_{rej} so it is the straight line. The black curve shows how the revenue varies with the distance goes up. From the figure, we can observe that:

1) Rejection will increase the revenue.
2) For distance rejection policy, the revenue is like a concave function. The optimal occurs when the distance is in middle range.
3) Expectation rejection is overwhelmingly better than distance rejection, which is the same as we analyzed in section IV.

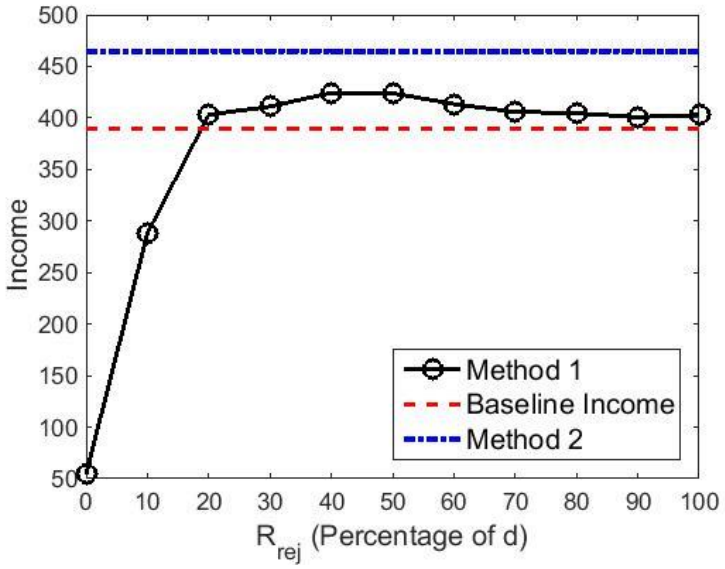

Fig. 8. Comparison of rejection policies

VI Results

In this section, we present the simulation results for all the instances of the five graphs. We adopt the neighbor move policy to preposition my car when it is empty and the second method to reject the bad customers. The results of our policy are also compared to the baseline for each graph.

Figure 9 shows the total income for my car and other competitors for each instance in each graph when my car is adopting our policy. Figure 10 illustrates the average improvement of our policy compared to the baseline for all the graphs. According to the figure, our policy can achieve up to 31% improvement compared to the baseline.

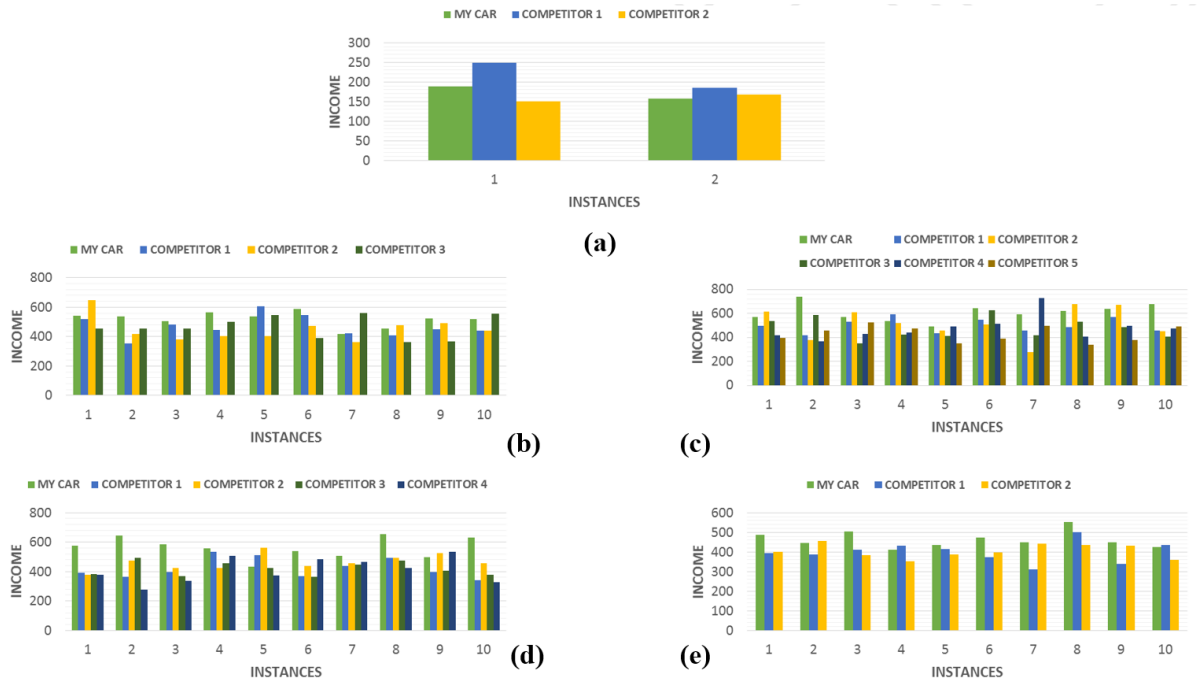

Fig. 9. Total income for my car and other competitors for (a) Graph 1, (b) Graph 2, (c) Graph 3, (d) Graph 4 and (e) Graph 5

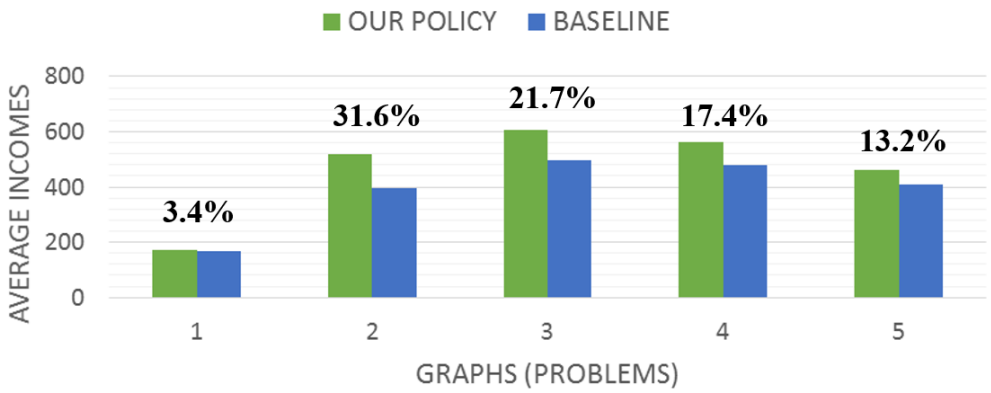

Fig. 10. Average income of my car when our policy is adopted and for the baseline case (numbers above the bars represent the improvement of our policy)

VII Conclusion

In this project, we simulate the behavior of contract driver in Uber. To maximize the revenue, we analyze the problem and proposed two prepositioning policies and two rejection policies. Based on these policies, we designed a state transition graph to indicate the smart strategy as the driver. We analyze and explain each policy through experimental results.

Reference

[1] https://en.wikipedia.org/wiki/Uber_(company)

[2] B. Chakraborty, T. Maeda and G. Chakraborty, "Multiobjective route selection for car navigation system using genetic algorithm," *Proceedings of the 2005 IEEE Midnight-Summer Workshop on Soft Computing in Industrial Applications, 2005. SMCia/05.*, 2005, pp. 190-195.

[3] Research on Optimization of Vehicle Routing Problem for Ride-sharing Taxi, Yeqian Lin, Wenquan Li, Feng Qiu, He Xu. School of Transportation, Southeast University, No. 2 Si Pai Lou, Nanjing 210096, P. R. China

[4] Jorge Nunes, Luís Matos, and António Trigo. 2011. Taxi pick-ups route optimization using genetic algorithms. In *Proceedings of the 10th international conference on Adaptive and natural computing algorithms - Volume Part I* (ICANNGA'11), Andrej Dobnikar, Uroš Lotrič, and Branko Šter (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 410-419.

[5] V. Pillac, M. Gendreau, C. Guéret, A.L. Medaglia "*A review of dynamic vehicle routing problems*" Technical Report 2011-62, CIRRELT (2011)