Route Recommendations for Idle Taxi Drivers: Find Me the Shortest Route to a Customer!

Nandani Garg
IIT Madras
Chennai, India
smilenandani@gmail.com

Sayan Ranu
IIT Delhi
New Delhi, India
sayanranu@iitd.ac.in

ABSTRACT

We study the problem of route recommendation to idle taxi drivers such that the distance between the taxi and an *anticipated* customer request is minimized. Minimizing the distance to the next anticipated customer leads to more productivity for the taxi driver and less waiting time for the customer. To anticipate when and where future customer requests are likely to come from and accordingly recommend routes, we develop a route recommendation engine called *MDM: Minimizing Distance through Monte Carlo Tree Search*. In contrast to existing techniques, MDM employs a continuous learning platform where the underlying model to predict future customer requests is dynamically updated. Extensive experiments on real taxi data from New York and San Francisco reveal that MDM is up to 70% better than the state of the art and robust to anomalous events such as concerts, sporting events, etc.

KEYWORDS

Monte Carlo Tree Search; Multi-armed Bandit Learning; Taxi Route Recommendation

ACM Reference Format:
Nandani Garg and Sayan Ranu. 2018. Route Recommendations for Idle Taxi Drivers: Find Me the Shortest Route to a Customer!. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220055

1 INTRODUCTION

With the proliferation of GPS-enabled smartphones, the dynamics of the taxi service industry has changed dramatically. Instead of walking out to the street to find a taxi, we use apps in our smart phones to locate one (Ex. Uber, Lyft). In this highly competitive taxi service industry, minimizing the *wait-time* for customers is important. The wait-time quantifies the time gap between the customer requesting a taxi and the taxi eventually reaching the location of the customer. A taxi service provider can reduce the wait-time of its customers by positioning all *idle* taxis in close proximity to forthcoming taxi requests.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220055

In the current world, once a taxi drops a customer in his/her destination, no further intelligence is provided to the taxi driver to reduce the wait-time for the next customer request. Either the taxi remains stationary or it drives towards a "hot-spot" that typically has high density of customers. Our goal is to develop a data-backed algorithm that recommends routes to drivers during their idle-time such that the distance between them and the next *anticipated* customer request is minimized. The proposed problem is not only important for the existing taxi service industry but also prepares us for the imminent future of driver-less taxis, which have already been introduced for public trials in US and several Asian cities [15], [17]. Since driver-less taxis would be devoid of any human intuition to adapt to changing demand patterns, the proposed problem is a basic necessity.

There are multiple challenges unique to our problem.

1. Dynamic pick-up probabilities: The goal of our technique is to predict a route to an *anticipated* customer request. To anticipate well, we need to know the pick-up probabilities across all regions in a city. An obvious option is to learn the probabilities from historical data. While such an approach is reasonable and has been employed in the past [13, 18], these probabilities are highly dynamic in nature and change with time of the day, the type of the day (such as weekend, weekday, holiday), weather patterns, and events in a city such as concerts, etc [19]. In other words, a static pick-up probability learned from historical data is not robust enough to capture an accurate representation of the situation on ground.

2. Route recommendations: Given the current location of a taxi, we need to identify the shortest route to an anticipated customer. This problem is different from identifying the region with the highest pick-up probability [5, 18] since we need to incorporate the balance between distance travelled and the chances of picking up a customer. A more detailed discussion on this topic is performed in Sec. 2.

3. Competition among taxis: Route recommendations for taxis are different from traditional recommendation engines where the same item can be recommended as many times. For example, in music recommendation, the same song can be recommended to infinite number of users. In our problem, the value of a route reduces every time it is recommended since a customer can be allotted to only one taxi. Thus, the recommendation engine must be aware of the expected number of customers in a route and accordingly ensure that multiple cabs are not allotted to the same anticipated customer.

To overcome the challenges outlined above, we develop a route recommendation algorithm called *MDM (Minimizing Distance through Monte Carlo Tree Search)* to minimize the distance to an anticipated customer for idle taxis. Our problem needs to tackle two issues: (1)

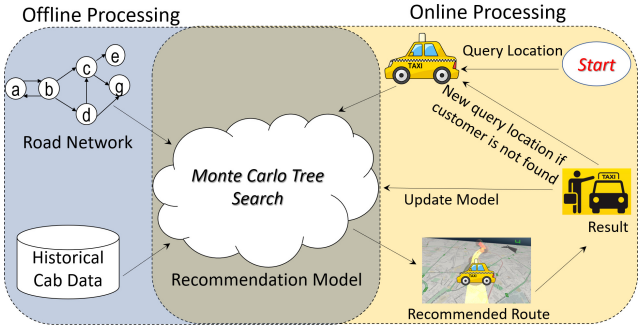


Figure 1: The pipeline of MDM.

a *recommendation engine* that recommends the optimal route and (2) a *sensing component* that allows the recommendation engine to accurately reflect the ground reality. Instead of treating these two components as two independent modules, we integrate them into a single coherent framework by modeling taxi route recommendations as a *Monte Carlo Tree Search* (MCTS) problem [6]. MCTS is a tree search algorithm, where given a state space that is arranged in the form of a tree, MCTS allows us to estimate the most promising *actions* from the current state.

Fig. 1 outlines how MCTS is employed in MDM. Each node in the road network is a state, and there is a directed link from node u to v if a road segment connects u to v. An action corresponds to moving from one node to another through an edge, such as traveling from u to v. The obvious question is therefore, what action should a taxi take given its current location (node) in a road network. We employ the UCT algorithm [6] where each of the possible actions from the current state is modeled as an arm (an outgoing road segment in our problem), and the score of an arm is learned using *Multi-armed bandits* [1]. Intuitively, the score of an arm (road segment) is high if it is part of a short route that typically leads to a customer.

There are two phases: the offline learning phase and the online recommendation phase. In the offline phase, the score of each arm is learned from the historical data. In the online phase, the learned scores are used for route recommendations. More specifically, the location of a taxi is the query node and the arm (outgoing segment) with the highest score is pulled (or recommended). Once the taxi reaches the other end of the recommended segment, it verifies whether a customer is actually found. If not, the recommended segment is negatively *rewarded* to reduce its chances of being recommended again. Furthermore, a new query with the updated current location is generated for the next recommended move. On the other hand, a success in finding a customer sends positive reward for that segment in the recommendation model. Note that different from typical recommendation engines, even in the online phase, the results of the recommendations are fed back into the model and the scores continue to get updated. Consequently, the model never goes stale.

In addition to exploiting the routes with highest score, with a certain probability, the model may choose to *explore* new routes even though it may not look promising according to the model. This exploration aspect allows MDM to discover new "hot" routes that are currently not represented. Overall, through exploration, and the ability to utilize feedback from previous recommendations, MDM captures a true representation of the prevailing conditions.

The main contributions of our work are as follows.

- We formulate the problem of route recommendations to idle taxi drivers, such that the distance to an anticipated customer is minimized.
- We devise a novel algorithm called MDM that employs the unique idea of Monte Carlo Tree Search on road networks. Through this strategy, MDM is able to integrate the sensing and recommendation components into a single coherent framework.
- We perform extensive benchmarking on real datasets from San Francisco and New York. More importantly, we verify the adaptability of MDM to demand anomalies, traffic variations through time of the day, and dense and sparse regions of the city. The results establish MDM to be up to 75% more effective than the state of the art and robust to the highly dynamic characteristics of a road network.

2 RELATED WORK

The problem of recommendation engines for taxis comes under the broad umbrella of trajectory analytics [2, 3, 7–9, 14, 20, 22, 23]. Majority of the existing taxi route recommendation algorithms focus on the time-period where the taxi is occupied by a customer and recommends the optimal route to the destination based on some function of interest[9, 20, 22, 23]. In contrast, our work focuses on the idle time-period of the taxi.

Recently, there has been some work on better utilizing the idle time. These techniques can be grouped into two categories: hotspot prediction and hot-route prediction.

Hotspot Prediction: A hotspot is a region with a high likelihood of finding a customer. Several works recommend *hotspots* to idle taxi drivers [5, 12, 18, 21]. Hotspot recommendation, however, is a different problem than route recommendation. First, there might be multiple routes to the hotspot and the recommendation engine does not specify which route to take. Second, even if an arbitrary route to the hotspot is chosen, only the destination of the route, i.e., the hotspot has a high customer pick-up probability. In route recommendation, the goal is to select the route with the highest customer pick-up probability. Third, in route recommendation, the length of the route is an important consideration, since it is desirable to travel the least distance before finding a customer. Overall, the search space in all these works contains all locations in a city, whereas in our problem, the search space contains all routes originating from the current location of a taxi. Owing to these differences, when hotspot prediction techniques are employed for hot-route prediction, the performance suffers. We substantiate this claim in our empirical evaluation in Sec. 6.

Hot-route prediction: Our problem falls in the hot-route prediction category, where the goal is to select the shortest route to a customer. The work by Qu et al. [13] is the only technique that recommends routes for idle taxi drivers. While this technique can be adopted for our problem, it does not harness all of the capabilities of the modern taxi service industry, and consequently presents significant scope to improve the performance. More critically, this technique does not support dynamic pick-up probabilities, which is

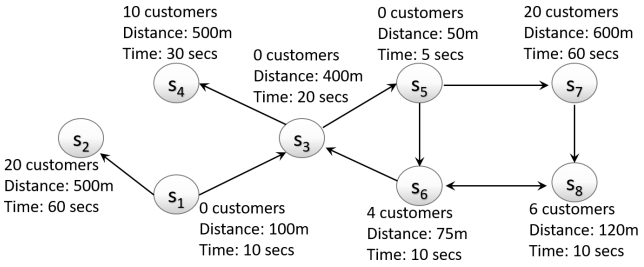


Figure 2: A road segment network

one of the salient features of our work. We substantiate how these weaknesses transfer to inferior performance in Sec. 6.

3 PROBLEM FORMULATION

First, we define the basic entities of a *road network*.

Definition 1 (Intersection). *An intersection is a single point in a road map defined by its latitude, longitude and a unique ID. An intersection is a point where at least two roads meet.*

Definition 2 (Road Segment). *A long road is divided into several road segments by the intersections contained in that road. Each segment s has its own start ID, $s.start$, and end ID, $s.end$, which are essentially the IDs of the starting and the ending intersections of the segment respectively.*

The *length* of a segment s, denoted as $\mathcal{D}(s)$, is the distance between the two intersections it connects. The *expected time* to traverse a segment is denoted using $\mathcal{T}(s)$. The expected time is easily available from mapping services such as Google Maps and Bing Maps. In addition, taxi service providers such as Uber can compute this value from their historical data. We term a segment s_j as *neighbor* of segment s_i if s_j starts at the same intersection where s_i ends. More formally, the neighbors $s_i.neighbors$ of a segment s_i is the set of segments $s_i.neighbors = \{s_j \mid s_j.start = s_i.end\}$.

Definition 3 (Route). *A route R is a sequence of connected road segments $R = \{s_1, s_2, \cdots, s_k\}$ such that $\forall i,\ 1 \leq i \leq k,\ s_i.end = s_{i+1}.start$.*

We use the notation $s \in R$ to denote that segment s is contained in route R. Furthermore, $R.s_i$ denotes the i_{th} segment in R. The distance travelled in a route R, denoted as $\mathcal{D}(R)$, is the total length of its constituent segments, i.e., $\mathcal{D}(R) = \sum_{\forall s \in R} \mathcal{D}(s)$. The expected time to cover a route R, $\mathcal{T}(R) = \sum_{\forall s \in R} \mathcal{T}(s)$, is defined analogously.

Definition 4 (Road segment network). *A road segment network G is a directed graph $G = (V, E)$ where V is the set of all the road segments $V = \{s_k\}$ and E is the set that satisfies $\exists e_{ij} \in E \iff s_j \in s_i.neighbors$.*

Unlike typical road networks, where each node corresponds to a region, such as an intersection, and edges correspond to road segments connecting these regions, in our problem each road segment itself is a node. We adopt this approach since the current location of a taxi is more likely to be a road segment and not a specific region. Similarly, customers may also appear anywhere within a road segment and thus statistical summaries such as the probability of finding a customer in a road segment is more meaningful than the probability of finding a customer at a particular region. This approach of using a road segment network has also been adopted by Qu et al[13].

As outlined in Fig. 1, there are two sources of training data: the road segment network $G(V, E)$ and historical taxi data $\mathbb{H}$. $\mathbb{H}$ is the set of all *customer requests* made in the past.

Definition 5 (Customer Request). *Each customer request C is a spatio-temporal point of the form $C = (s, t)$ where s is the segment the customer is located at and t is the time at which the request is made.*

Example 1. *Figure 2 demonstrates a road segment network where each node represents a segment. In addition to the network, we also store several other statistical information for each segment such as its length, the expected time to cover this segment and the number of customer requests originating from this segment. Note that both the expected travel time and customer requests are time-varying properties and can alternatively be stored as a distribution over time. While we discuss such possibilities, in this example we store a single statistic for simplicity.*

In our problem, the query comes from an idle taxi who is looking for a customer. Therefore, our goal is to recommend the shortest route that would lead to a customer. Note that if the query comes from a taxi at time t, then we need to predict the route to a customer who would request a taxi at time $t + \Delta$ where $\Delta > 0$. Otherwise, the taxi would not be idle and would simply drive to the customer. In other words, we need to select routes based on *anticipated* customer requests. The recommendation problem is therefore defined as follows.

Problem 1 (Route Recommendation). *Let segment q be the query location of a taxi at time t. Given this query, our goal is to recommend the shortest route R^* from q that leads to a customer.*

Quickest Route: In the above formulation, our goal is to find the shortest route to a future customer. An alternative formulation is to recommend the quickest route to a future customer, i.e., minimize $\mathcal{T}(R)$ instead of $\mathcal{D}(R)$. The proposed technique, MDM, can easily adapt to this modification. In essence, both time and distance are nothing but edge weights. By choosing the appropriate edge weight to compute the *score* of a route, MDM can minimize either over distance or over expected time. We use distance since fuel cost is directly proportional to the distance travelled.

4 BASELINE

Since our goal is to find the shortest route to an anticipated customer, we need to balance two aspects of this prediction problem: the probability of finding a customer in a route and the length of the route. To illustrate, consider two routes R_1 and R_2 where $\mathcal{D}(R_1) = 5km$, $\mathcal{D}(R_2) = 1km$. Furthermore, the probabilities of finding a customer at the terminal segment of R_1 and R_2 are $p(R_1) = 0.7$ and $p(R_2) = 0.3$. In this case, even though R_2 is shorter than R_1, its probability of finding a customer is lower. Hence, it is not clear which route should be preferred. The state-of-the-art technique[13] is based on the *maximum-likelihood* approach with some additional optimizations on the modeling. Let d be the largest distance that

a taxi is willing to travel to search for a customer. Beyond d, the taxi prefers to remain stationary and wait for a customer request to arrive. A basic approach for this scenario is to extract all travel routes within distance d from the current location s and choose the one with the highest likelihood of finding a customer.

To formalize, we first define the concept of a *pick-up probability distribution.*

Definition 6 (Pick-up Probability Distribution). *Based on the historical training data, the pick-up probability distribution computes the time-varying distribution of customers across all segments of the network. More specifically, we compute the proportion $p(s,t)$ of customers appearing in some segment s at time t. Since, computing the probability exactly at time t is not practical, a reasonable approach is to divide the historical data into time slots and compute the chances of finding a customer at s in the time slot T in which t lies. Mathematically,*

$$p(s,t) = \frac{|\mathbb{H}_{s,T}|}{|\mathbb{H}|} \tag{1}$$

where $\mathbb{H}$ is the set of all customer requests and $\mathbb{H}_{s,T}$ contains those customer requests originating at segment s in time slot T, i.e., $\mathbb{H}_{s,T} = \{(s,t) \mid (s,t) \in \mathbb{H}, t \in T\}$.

Now, the probability of finding a customer in a given route $R = \{s_1, \cdots, s_k\}$ at time t.

$$p(R,t) = 1 - \prod_{i=1}^{k} (1 - p(s_i,t)) \tag{2}$$

The route R^* is the one with the maximum likelihood of finding a customer.

$$R^* = \arg\max_{R \in \mathbb{R}_d} \{p(R,t)\} \tag{3}$$

where $\mathbb{R}_d = \{R|\ R \text{ is a route}, R.s_1 = s, \mathcal{D}(R) \leq d\}$.

4.1 Limitations

- **Importance of distance:** While the above approach maximizes the likelihood of finding a customer, it is not sensitive to the additional requirement of keeping the length of the route short.
- **Static pick-up probabilities:** Qu et al.[13] assume pick-up probabilities to be static, whereas in reality they change with time, weather conditions, local events, etc. *How do we learn the new pick-up probabilities quickly and adapt the recommended routes?* A good recommendation engine must answer this question.
- **Time slots:** Since pick-up probabilities change with the time of the day, one could improve [13] by dividing the historical data based on time slots with the assumption that the distribution of probabilities are static within each slot. This approach has been employed in hotspot prediction as well [18]. Unfortunately, identifying the optimal partitioning of time slots and the number of partitions are often tricky and irregular. Furthermore the partitioning is done under the assumption that the time-slot partitions behave similarly across days, which is often not true in real life [19].

Powered by these intuitions, we initiate the design of the proposed *MDM* algorithm.

5 MDM: MINIMIZING DISTANCE THROUGH MONTE CARLO TREE SEARCH

There are two key requirements: 1) we should not constrain the search to only among routes of length d, and 2) the underlying recommendation model must remain up-to-date at all times since road conditions are never static. We address these dual needs through multi-armed bandit learning within the Monte Carlo Tree Search (MCTS) framework.

Monte Carlo Tree Search (MCTS) is a method for finding the optimal actions in a given domain by taking random samples in the state space and building a search tree according to the results. In MCTS, the search space is structured in the form of a tree where each node is a state. Two nodes are connected if there exists an action that takes the domain agent from the parent node to the child.

In our problem, each node of the road segment network is a state, and an action corresponds to moving from one node to another through an edge in the road segment network. Given the current location of a taxi, we choose the outgoing segment (edge) that has the highest score. If the chosen edge fails to lead to a customer, we choose an edge from the new location (state) of the taxi and this process continues till a customer is found. Clearly, the efficacy of our method depends on how accurate our score estimates are. The key question at this juncture is therefore: *How do we learn the score of an edge?*

Intuitively, the score of an edge should correspond to the average length of all paths that have started from it and have successfully found a customer. This value can be computed from the historical data. However, customer demands are dynamic and therefore the scores cannot be assumed to be static. To address this need of continuously updating our model and maintain up-to-date score estimates, we employ multi-armed bandit learning.

5.1 Multi-armed Bandit (MAB)

MAB has been studied for decades [16], where given a set of m arms, each associated with an unknown distribution of *rewards*, one repeatedly picks up and plays one of the m arms in each round with the ultimate goal of maximizing the total expected rewards gained in multiple rounds. The rounds unfold in discrete time steps. Only the reward of the arm played is observed and the reward can only be observed after the arm is played. We formalize the above description by introducing the following definitions.

Definition 7. *Expected Reward. In an m-armed bandit problem, each of the m arms has an expected reward when played. We denote the arm played at round r as A_r, and the corresponding reward as W_r. The expected reward of an arbitrary arm a, denoted $Q_r(a)$, is therefore:*

$$Q_r(a) = \mathbb{E}\big[W_r \mid A_r = a\big] \tag{4}$$

We use the notation Q_r^* to denote the expected reward from the optimal arm at round r, i.e.,

$$Q_r^* = \max_{\forall a} \{Q_r(a)\} \tag{5}$$

The gap between the total rewards gained by an algorithm and those gained by playing the optimal arm, is termed as *regret*, and the goal is to minimize its regret.

Definition 8. *Regret. The regret is the expected loss due to not playing the optimal arm. Mathematically, the regret at round r is*

$$Reg(r) = rQ_r^* - \sum_{\forall a} Q_r(a) * c(a) \quad (6)$$

where c(a) is the number of times arm a has been played in the first r rounds.

Minimizing the regret creates the *exploitation-exploration* dilemma. More specifically, the choice of the optimal arm is difficult as the underlying reward distributions are unknown, and potential rewards must be estimated based on past observations. Therefore, one needs to balance the *exploitation* of the arm currently believed to be optimal with the *exploration* of other arms that currently appear suboptimal but may turn out to be superior in the long run. In any specific problem, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps. [16].

5.2 MCTS for Taxi Route Recommendation

Let s be the current location of a taxi. At this stage, the driver needs to take a decision on which neighboring segment to take from $s.neighbors$. The driver may have some estimate on the quality of a neighboring segment based on prior experiences. A rational decision would therefore be to take the segment with the highest estimated quality. Whether this segment is indeed the optimal one however can only be determined in the long run after exploring all other outgoing segments sufficient number of times.

It is easy to see the connection of our problem to MCTS and MAB. Each of the outgoing segments $s' \in s.neighbor$ corresponds to an arm, and the expected reward of s' is inversely proportional to the expected distance to a forthcoming customer. On selecting the best arm (or segment) s', the current location (state) of the taxi changes to s'. At this stage, there are three possibilities.

(1) A customer is found on s'. In this case, the quality of s' is observed and this arm is accordingly rewarded.
(2) The taxi remains idle. Thus, a new MAB is initiated at s' by pulling one of the arms from the set $s'.neighbors$.
(3) s' is a dead-end with no outgoing segments. The route is rewarded (or penalized) accordingly.

To summarize, finding the optimal route corresponds to pulling a series of arms starting from the initial position of a taxi till an arm eventually leads to a customer or meets a dead-end. Consequently, the search space can be visualized as a tree.

Example 2. *Fig. 3(a) presents the search tree from s_5. Each path starting at s_5 in the search tree corresponds to a possible route. Note that we do not allow cycles in any of the routes. The probability of recommending the path $R = \{s_5, s_6, s_3\}$ is the same as the probability of pulling arm s_6 at s_5, not finding a customer at s_6, and then pulling s_3 at s_6, which leads to a customer. Although we show the entire tree in this figure, we do not maintain this tree in memory at any stage. Rather, the tree is explored recursively by starting from the initial current position of a taxi (s_5 in this example). At any stage, only the path corresponding to the pulled arms are stored in memory. Consequently, the memory overhead is low.*

The important question at this juncture is to formulate a policy based on which an arm would be pulled at any particular node of the search tree. Towards that end, we use the *Upper Confidence Bound for Trees (UCT)*[6] algorithm.

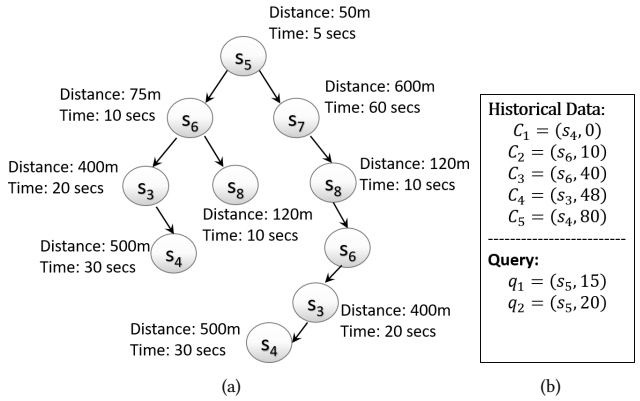


Figure 3: (a) Search tree corresponding to Fig. 2 if s_5 is the current location of the taxi. (b) A sample historical dataset and queries.

5.3 Learning the Recommendation Model

Reward function: We use the historical data $\mathbb{H}$ to estimate the reward for a given segment. Let (q, t) be a query from segment q at time t, where $t_{min} \leq t \leq t_{max}$. t_{min} and t_{max} are the time stamps of the earliest and latest customer requests in $\mathbb{H}$ respectively. After recursively pulling a series of arms (or segments), let s be the current location. The taxi is estimated to reach s at time $t + \mathcal{T}(R)$, where $R = \{q, \cdots, s\}$ is the route taken to reach s from q. The search terminates at s if either s is a dead-end or an *unallocated* customer is found at s. A customer is found, if two conditions are met.

(1) There exists a customer request C from s that originated in the time range $[t, t + \mathcal{T}(R)]$, i.e, $\exists C = (s, t') \in \mathbb{H}$ where $t < t' < t + \mathcal{T}(R)$. The time range ensures two aspects. First, since $t' > t$, it means the taxi driver was not aware of this request when the search query was made. Second, the condition $t' < \mathcal{T}(R)$ ensures that the taxi has not already passed the customer's location by the time the request is made.
(2) Customer C has not already been allocated to some other taxi, i.e., $\mathcal{I}(C) = 0$, where $\mathcal{I}(C)$ is an indicator function defined as follows.

$$\mathcal{I}(C) = \begin{cases} 0, \text{if } C \text{ is unallocated} \\ 1, \text{If } C \text{ is allocated} \end{cases} \quad (7)$$

Initially, all requests are unallocated, i.e., $\forall C \in \mathbb{H}$, $\mathcal{I}(C) = 0$.

If multiple customer requests comply the above two conditions, we allocate the earliest compliant customer request. Depending on how the search terminates, the reward is *back-propagated* to each of the pulled segments in R. More specifically, it is defined as follows.

(1) **Customer is found at s:** The reward of each of the pulled segments $s' \in R$, i.e., all segments in R except q, is the negative of the distance traveled to find a customer. More formally, $\forall s' \in$

$R \setminus \{q\})$, $reward(s') = -\mathcal{D}(R)$. Thus, the higher the distance traveled to find a customer, less is the reward.

(2) **s is a dead-end:** When a route reaches a dead-end, it is regarded as a long route that fails to reach a customer. Thus, $\forall s' \in R \setminus \{q\})$, $reward(s') = -V$, where V is a large constant.

We illustrate the above reward policy with an example.

Example 3. *Consider the training dataset in Fig. 3(b) and the two randomly generated queries. We process q_1 first. Assume, we pull arm s_6 at s_5. There are two customer requests, C_2 and C_3, which originate from s_6. However, none of them can be allocated for q_1 since C_2 originate before q_1, and C_3 originates after q_1 reaches s_6, which is at time stamp $15 + 5 + 10 = 30$. Clearly, s_6 is not a dead-end. Thus, the search must continue. Assume, we pull arm s_3 next where q_1 is expected to reach at time stamp $15+5+10+20 = 50$. C_4 is present at s_3, which is unallocated and is within the time range $[15, 50]$. Therefore, C_4 gets allocated and the search process terminates. Since the distance traveled is $525m$, the pulled arms s_6 and s_3 get rewards of -525 each. Now, let us consider the second query q_2. We once again assume that s_6 is pulled at s_5. Due to the same reasons as in q_1, neither C_2 nor C_3 can be allocated. Next, assume s_3 is pulled again. Here, although C_4 is within the required time range, it does not qualify since it has already been allocated in q_1. Finally, s_4 is pulled and C_5 gets allocated to q_2. The reward for this allocation is -1025 to s_6, s_3 and s_4.*

The expected reward $Q_r(s)$ of a segment s is therefore the *mean* of all rewards observed at s till r rounds of reward allocation.

Example 4. *The expected rewards of s_6, s_3 and s_4 after rounds 1 and 2 are $Q_1(s_6) = Q_1(s_3) - 525$, $Q_2(s_6) = Q_2(s_3) = \frac{-(525+1025)}{2}$, and $Q_1(s_4) = -V$, $Q_2(s_4) = -1025$. Note that the expected reward is set to $-V$ for segments that have not been explored yet. However, the first time an unexplored arm is observed, the -V is not accounted for in the mean computation for expected reward.*

5.3.1 Applying UCT. An obvious approach is to pull the arm with the highest expected reward. This policy is in line with the regret minimization policy outlined in Eq. 6. However, this policy is also susceptible to getting stuck at a local optimum. More specifically, expectations are meaningful only after sufficient number of observations have been made. Thus, in the initial rounds, the arm with the highest expected reward may be of spurious value and consequently lead us towards sub-optimal policies. To mitigate this phenomenon, we employ the *Upper Confidence Bound (UCB)* policy[1].

The key idea of UCB is to infer an *upper confidence bound* on the reward of each arm and instead of the arm with the largest expected reward, we play the arm with the largest upper confidence bound. Initially, all arms should have the same upper confidence bound on the expected reward, which is quite loose since the number of observations made is small. Then as arms are played repeatedly, we gather more observations on these played arms and the tightness of their upper bounds increase. On the other hand, the arms that have not been played (explored) adequately yet would continue to have large (loose) upper confidence bounds, and hence have a high chance of being explored in subsequent rounds. Consequently, the algorithm avoids being trapped in local optimum. In addition, by playing the arm with the largest upper bound in each round, i.e. the largest possible reward, UCB tries to play the optimal arm. Consequently, UCB is a generic framework that does exploration and exploitation simultaneously without compromising on the end goal of minimizing regret. These subtleties are captured using the following upper bound function.

$$A_r = \arg\max_a \left[Q_{r-1}(a) + c\sqrt{\frac{\ln(r-1)}{N_{r-1}(a)}} \right] \tag{8}$$

Here, r denotes the current round and $N_{r-1}(a)$ denotes the number of times arm a has been pulled in the first $r-1$ rounds. c is a positive weight that controls the degree of exploration. At each round r, we select the arm that maximizes the above upper bound. We next explain how this upper bound captures the intended properties.

- The square-root term in Eq. 8 is a measure of the uncertainty or variance in the estimate of a's reward. More importantly, when $N_r(a) \ll r$, it indicates that arm a has not been explored adequately and thus the square-root portion starts to dominate $Q_r(a)$. Consequently, even arms with low rewards have a high chance of being explored.
- Since the numerator of the square-root portion increases logarithmically, the increase in uncertainty gets smaller over rounds. However, it is unbounded. In other words, each arm will be selected at some point of time, but the frequency of selection decreases with more rounds for arms that have low expected reward.

UCT applies the UCB principle on each node of the search tree and pulls the outgoing segment (arm) with the highest UCB value (Eq. 8). When mapped to our problem, starting from the query segment, the outgoing segment with the highest UCB (Eq. 8) is pulled. If the pulled segment is not a terminal node, i.e., a dead-end or a customer is allocated, the exact same process continues recursively.

5.3.2 Policy Learning Algorithm. Alg. 1 presents the pseudocode of learning the expected rewards of all segments in the network. We choose each segment as the query location a large number of times (lines 3-15). The time stamp of the query is randomly generated (line 6). Arms are pulled starting from the query segment based on the UCB policy (line 7). Once a customer is found, the rewards for all segments in the route are back-propagated and therefore updated (lines 8-12). Notice that a segment accumulates rewards not only when it is selected as the query location, but also when it is part of a chosen route. In addition, we also point out that the customer allocation variable $\mathcal{I}(C)$ is re-initialized for each query segment (line 4). Thus, the order in which a segment is chosen to be the query location does not matter.

5.4 Online Recommendation Phase

Alg. 2 presents the pseudocode of our online recommendation algorithm. In this phase, we not only recommend routes, but also update the model based on the success or failure of our recommendations. The basic idea of recommending a route by iteratively pulling a sequence of arms from the query location remains intact (lines 2-4). The only differences from the learning algorithm are:

(1) **Query generation:** The query segment comes from an actual taxi instead of being synthetically generated. Note, at this stage, there is already some estimate of the expected reward, which has been learned in the learning phase.

Algorithm 1 Learning Phase

Initialize expected reward of each segment to 0
$r \leftarrow 0$
for each segment s in network **do**
  Set $\forall C \in \mathbb{H},\ I(C) = 0$
  repeat
    Generate query (s, t) with a random time t in range $[t_{min}, t_{max}]$
    Generate route R by pulling arms based on UCB policy (Eq. 8)
    if R finds a customer C **then**
      $\forall s' \in R \setminus s,\ reward(s') = -\mathcal{D}(R)$
      $I(C) \leftarrow 1$
    else
      $\forall s' \in R \setminus s,\ reward(s') = -V$
    Update expected reward $Q_r(s)$ of each pulled segment in R.
    $r \leftarrow r + 1$
  until a large number of times

Algorithm 2 Recommendation Phase

Require: Query location q
$a \leftarrow q$
while no customer is found at a and a is not a dead-end **do**
  $R \leftarrow R \cup q$
  $a \leftarrow$ pull an arm (outgoing segment from a) based on UCB policy (Eq. 8)
if customer is found at a **then**
  $\forall s' \in R \setminus q,\ reward(s') = -\mathcal{D}(R)$
else
  $\forall s' \in R \setminus q,\ reward(s') = -V$
$r \leftarrow r + 1$ {Number of rounds played including learning phase}
Update expected reward $Q_r(s)$ of each pulled segment in R.

(2) **Customer allocation:** The customer allocation variable $I(C)$ is no longer necessary since the success of a route is determined from live on-road data. More specifically, if the taxi does not find a passenger in the pulled (recommended) segment, then the next outgoing segment is pulled from the current location of the taxi and this process continues. When the taxi eventually finds a customer, the corresponding reward is awarded to all segments through which the taxi traveled. On the other hand, if the taxi stops roaming due to not finding a customer over a long distance, or hits a dead-end, then the traversed segments are rewarded $-V$, where V is a large number.

5.5 Properties of the MDM algorithm

In this section, we summarize the key properties of MDM.

Dynamic Model: In existing techniques for taxi route or hotspot prediction, the recommendation model is built only once from the historical data[13, 18]. In contrast to existing techniques [13, 18], the proposed recommendation model is dynamic and constantly gets updated by appropriately rewarding past recommendations based on their success or failure in finding customers.

Load-balancing among multiple drivers: Due to the dynamic model, MDM can adapt to competition among multiple drivers. More specifically, if there is a hot-route, then static models recommend the same route to all drivers even after the route gets exhausted[18]. On the other hand, the state-of-the-art technique in this space[13], does not recommend two consecutive routes to the same region to avoid exhausting a region. This approach runs the risk of the other extreme; even when there is a genuine need of many taxis, such as after a concert or soccer match, enough taxis would not be recommended due to the forced requirement of route diversification. In essence, an effective recommendation technique should balance the need to diversify automatically. MDM is able to do that by utilizing the feedback from past recommendations. If a hot-spot begins to form, routes to this region would begin to gather high rewards which in turn would lead to further enrichment of this region with more taxis. On the other hand, if a hot-spot gets exhausted, the low rewards would feed in to the model and alternative routes would be recommended.

Balancing route length with probability: As discussed earlier, the goal is to recommend a route that is short and has high probability of finding a customer. The basic approaches discussed earlier over-compensates on either length (Best-next-hop) or pick-up probability distribution (Maximum likelihood). The proposed approach finds a balance between both the approaches. The reward is inversely proportional to length. Furthermore, if a route does not find a customer often, then the reward is extremely low ($-V$). This design ensures that recommended routes are both short and likely to find customers.

Independent of time-slots: Since customer demands change with time, recommendations model often rely on partitioning the historical data into time-slots so that time-specific models can be built[18]. As discussed earlier in Sec. 4.1, this approach makes several assumptions that may not hold true in real road-networks. By employing a dynamic recommendation model, MDM is not handicapped by the need to partition data into time slots and thereby producing a more robust and practical recommendation platform. We substantiate this claim further in Sec. 6 through empirical studies on real taxi datasets.

6 EXPERIMENTS

In this section, we perform extensive experiments on real taxi datasets and establish that: 1) MDM is up to 70% more effective than the state-of-the-art technique[13], and 2) MDM is robust to dynamic road conditions such as anomalous events and aperiodic taxi demands.

6.1 Experimental Setup

All experiments are implemented in Java and performed on a machine with Intel(R) Xeon(R) 2.5GHz CPU E5-2609 with 512 GB RAM running Ubuntu 14.04.

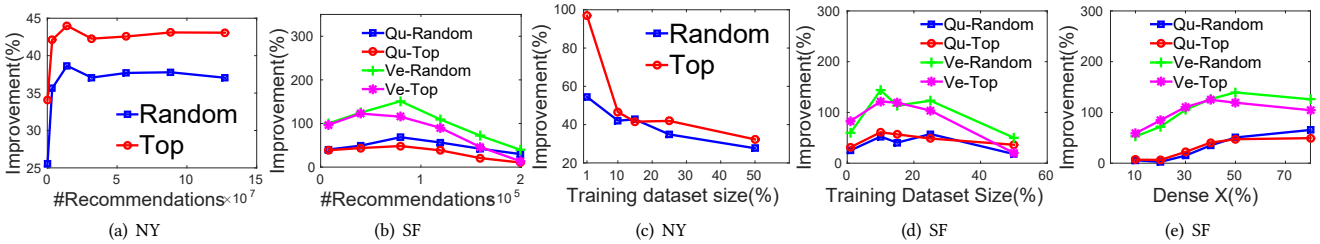
6.1.1 Datasets. We use real taxi datasets from New York and San Francisco. Table 1 summarizes the various characteristics of these datasets. The road networks of both New York and San Francisco have been obtained from OpenStreetMap[10].

New York [4]: This dataset contains records of yellow and green taxis in the city of New York. Each record includes fields capturing pick-up and drop-off dates/times, locations and trip distances. The New York dataset is the largest taxi dataset that is publicly available. From this dataset, we extract all pick-ups from January, 2010 to March, 2010 and construct our dataset.

San Francisco [11]: The dataset has been collected over a duration of one month through the taxi-spotting project [11] and

Table 1: Dataset Statistics

Dataset Type	# of Pickups	# Edges	# Nodes (Segments)
San Francisco (SF)	165164	7963	3527
New York (NY)	42947007	141372	61298



(a) NY (b) SF (c) NY (d) SF (e) SF

Figure 4: Percentage improvement of MDM against (a-b) the number of recommendation queries and (c-d) training data size. (e) Comparison of MDM, *Qu*, and *Ve* in dense regions.

contains the trajectories and taxi pick-up and drop-off information from taxis in San Francisco.

6.1.2 Baselines. We compare MDM with the state-of-the-art techniques for hot-route prediction as well as hotspot prediction.

Qu et al. [13]: The technique proposed by [13] is the state-of-the-art technique for route recommendation to idle taxis. We refer to this technique as *Qu*.

Verma et al. [18]: This work is the state-of-the-art technique for hotspot prediction. Verma et al. divides a city into zones and recommends zones where the taxi should go to find a customer. For route prediction, we consider each segment as a zone and the route to the segment is considered as the recommended route. We refer to this technique as *Ve*. Unlike MDM and *Qu*, *Ve* not only needs the pick-up and drop-off locations, but also the trajectories of the taxis during their idle time. Since trajectories are available only on the SF dataset, we are unable to compare with *Ve* in NY.

6.1.3 Evaluation framework. To evaluate the performance of the benchmarked techniques, we partition each dataset into two portions: the training set and the testing set. The recommendation models are learned on the training set and verified on the testing set. For verification, we select a random query segment and time stamp t and feed it to the recommendation engine of each of the techniques. The recommended route R successfully finds a customer if a customer exists in the testing set in R in the time range $[t, t + \mathcal{T}(R)]$. When a customer is eventually found, we remove it from the testing set to ensure the same customer request is not allocated to multiple taxis.

The training set is constructed using two strategies. In the first strategy, we randomly select $X\%$ of all taxi drivers. All customer pick-ups serviced by these selected taxi drivers and their corresponding driving routes form the training set. X guides the size of the training set. We call this the *Random* training set. The second strategy is inspired by [13]. Here, each taxi driver is sorted based on the number of customer pick-ups they have on record. The top-X drivers with the highest customer pick-ups are selected and their customer pick-ups and driving routes form the training set. The idea is to utilize the knowledge of experienced drivers in the training set. We call this the *Top* training set.

6.1.4 Evaluation Metric: We compare the performance of MDM with *Qu* and *Ve* using the *improvement(%)* metric. Intuitively, it measures the percentage by which MDM is better than the competing technique (and vice-versa). Mathematically, it is defined as follows. For a given query, let d_{MDM} and d_M be the distances traveled by the taxi on the routes recommended by MDM and method M respectively till a customer is found. M is either *Qu* or *Ve*. The improvement of MDM over method M is computed as

$$Improvement(\%) = \begin{cases} \frac{d_M - d_{MDM}}{d_{MDM}} \times 100 \text{ if } d_{MDM} < d_M \\ -1 \times \frac{d_{MDM} - d_M}{d_M} \times 100 \text{ otherwise} \end{cases}$$

A negative value indicates that the route recommended by the competing algorithm M is shorter in length and a positive improvement indicates the opposite.

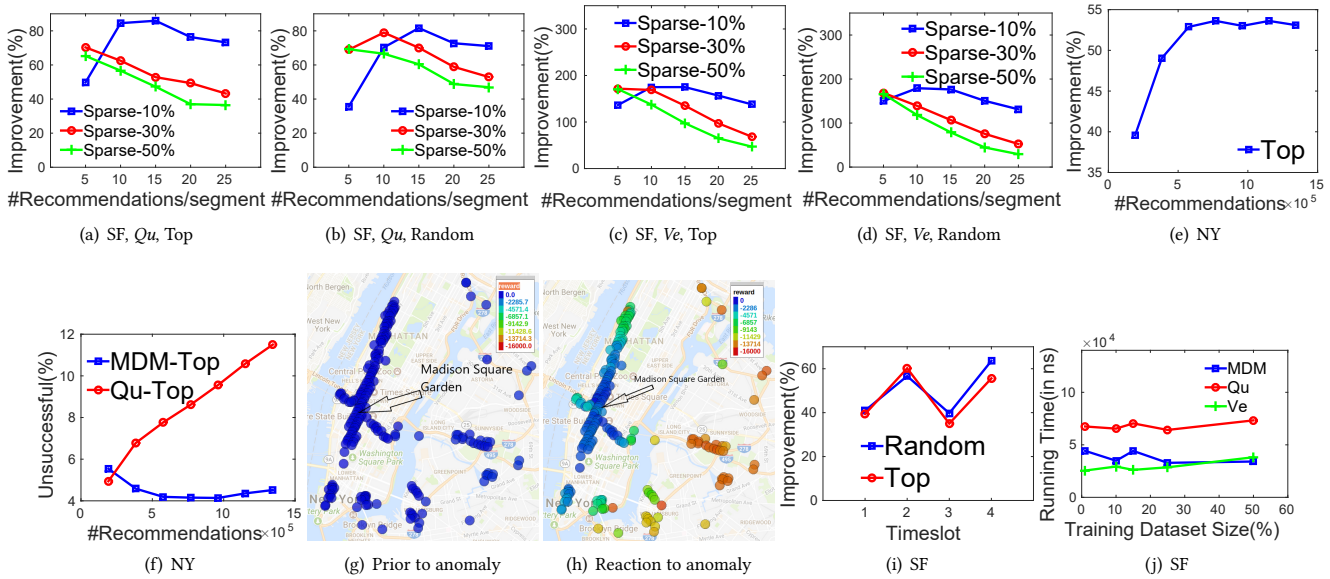
6.1.5 Parameter Values. Unless specifically mentioned, the default amount of historical data used for training is set to 10% of all taxi trajectories in New York and San Francisco. All parameters values for *Qu* and *Ve* are set based on the values suggested in their respective works[13, 18]. The exploration parameter c (Eq. 8) in MDM is set to 6. V is set to $3km$.

6.2 Recommendation Quality

First, we study how the quality of recommendations vary as we increase the number of recommendation queries. As more recommendations queries are made, the recommendation problem becomes harder since only a limited number of customers are available in the testing set. Figs. 4(a)-4(b) present the improvements in NY and SF. Several observations can be made from these results.

First, across both datasets and training dataset types (Top and Random), the performance of MDM is better as indicated by a positive Improvement value. In SF, the performance gap is as high as up to 70% for *Qu* and 175% for *Ve*. MDM is able to achieve this improved performance due to employing a dynamic recommendation model. As more recommendations are made, the pick-up probabilities change and hence predictions based on static models suffer.

Second, we notice in Figs. 4(a)-4(b) that the performance gap of MDM with *Qu* and *Ve* initially increases and then it either saturates or goes down. This trend is a consequence of the fact that initially the customer density is high and thus, it is easier for a recommended route to find a customer. However, as the number of unallocated customers decreases, the recommendation task gets harder and this is where MDM achieves the highest performance gap. Beyond that, when almost all customers have been allocated, all techniques find it difficult and thus, the performance gap decreases.



(a) SF, *Qu*, Top (b) SF, *Qu*, Random (c) SF, *Ve*, Top (d) SF, *Ve*, Random (e) NY

(f) NY (g) Prior to anomaly (h) Reaction to anomaly (i) SF (j) SF

Figure 5: Performance improvement of MDM in sparse regions against (a-b) *Qu*, and (c-d) *Ve*. Subfigures (a) and (c) uses training data from the top-10% most successful drivers and (b), (d) uses random 10% drivers for training. (e-f) Reaction to anomalous event in Madison Square Garden. (g-h) Depicts the change in reward values as a reaction to an anomalous event. (i) Variation in performance across time of the day. (j) Comparison of querying times of MDM, *Qu*, and *Ve*.

Finally, the results on SF clearly indicates that hot-route prediction techniques are superior to hot-spot detection techniques for the proposed problem.

Next, we study the performance variation with increase in training dataset size. Figs 4(c)-4(d) present the results. Across both NY and SF, we observe that the improvement is higher when training sets are small. As the training set increases, the improvement of MDM decreases. Note that this result does not indicate that MDM performs worse with increase in training size. Rather, it only shows that *Qu* and *Ve* close the performance gap with MDM to a certain extent. In MDM, even the recommendations made during test phase is used to update the model. Consequently, a small amount of training data does not hurt. On the other hand, *Qu* and *Ve* need large volumes of training data to learn effective recommendation policies and match up to the performance of MDM.

Cross-validation: We also performed 10-fold cross validation in NY and SF, where the improvements over *Qu* are 33% and 25% in NY and SF respectively. We did not compare against *Ve*, since both *Qu* and MDM are significantly better in performance.

6.3 Robustness to Road Network Conditions

In this section, we benchmark the robustness of MDM against various road work properties.

Impact of customer density: The experiment against number of recommendations indicated that when customer density is low, performance improvement of MDM is more profound. We next verify the impact of customer density directly.

In Fig. 4(e), we plot the improvement in performance against varying neighborhood densities. More specifically, each segment is sorted in descending order based on the number of customer pick-ups in that location. A segment is in Dense-X% if it is among the top X% of the sorted list. In addition, all segments within 1KM from a Dense-X% node are also classified as Dense-X% since they lie in the neighborhood of a dense segment. Fig. 4(e) shows the improvement if we only consider segments within Dense-X%. As density decreases, we observe that the competitive advantage of MDM increases. This trend matches our intuition and explanation provided earlier.

We further substantiate the ability of MDM to do well in difficult situations by computing the improvement in customer-sparse neighborhoods. In these experiments, we choose the query regions from only sparse neighborhoods of various intensities. For example, Sparse-10% contains only top-10% sparsest neighborhoods. Figs. 5(a)-5(b) and Figs. 5(c)-5(d) present the results against *Qu* and *Ve* respectively on both kinds of training datasets. In the x-axis we plot the number of query recommendations per segment, instead of total number of recommendations since the number of segments in a region varies. For example, the number of segments in Sparse-10% is much smaller than in Sparse-50%. As clearly visible, the sparser the region, the higher is the improvement provided by MDM. MDM is up to 80% better than *Qu* and 190% better than *Ve*. This result further substantiates our claim that MDM is more robust to difficult situations in terms of customer availability.

Adapting to anomalies: A good route recommendation engine should learn from anomalous events and alter its recommendations accordingly. For example, when a concert ends, large numbers of customers become suddenly available and models learned from historical data do not accurately reflect the ground reality. In the next experiment, we evaluate how MDM and *Qu* react to such

anomalous events. We omit *Ve* from the next set of experiments due to much weaker performance.

To construct an anomalous event, we first train both MDM and *Qu* on the standard historical data. In the test phase, however, we inject synthetic data to create an anomalous event. More specifically, we choose Madison Square Garden *(MSG)* as the place of a concert. To model the scenario following a concert, we make large number of customers available in the 1KM radius around MSG. Thus, if a taxi is recommended a route in the neighborhood of MSG, with a high likelihood, it finds a customer.

Fig. 5(e) presents the results against the number of recommendation queries from only those taxis that are within 3KM from MSG. Initially, the improvement is around 40%, which eventually increases to 55%. This trend is expected. *Qu* relies only on historical data and has no scope of adapting to anomalies. In MDM, initial recommendation routes are based on historical data. However, routes towards MSG slowly start to gain high rewards since recommendations in that direction would find customers with high likelihood. Thus, MDM adapts and provides recommendations customized to the anomalous event. Figs. 5(g) and 5(h) bring out this aspect, where we plot a heat map of the reward values on nodes around MSG. The color spectrum ranges from blue (high rewards) to red (low rewards). Before the start of the anomalous event, the rewards values are homogeneous in that region (Manhattan). However, following the anomalous event, the reward values around MSG are significantly higher than places further away.

This adaptability of MDM is further supported by Fig. 5(f). We call a recommended route "unsuccessful" if the taxi fails to find a customer even after traveling for 3KM. We plot the percentage of unsuccessful recommendations in Fig. 5(f) against the number of recommendation queries. As more recommendations are made, the performance of MDM improves in terms of number of unsuccessful recommendations. On the other hand, *Qu* follows an exact opposite trend. This experiment clearly brings out the key distinguishing feature of MDM, wherein the model never goes stale.

Adapting to time-slots: How does the performance of MDM vary against various times of the day? We next answer this question. First, we partition the dataset into four different timeslots: 1. $12AM-8AM$, 2. $8AM-12PM$, 3. $12PM-5PM$, 4. $5PM-11{:}59PM$. Generally slot 1 and 3 are non-peak hours, whereas 2 and 4 are peak hours. As shown in Fig. 5(i), regardless of the time of the day, MDM has a performance improvement in the range of 35% to 65%.

Querying time: Fig. 5(j) analyzes the online query recommendation time. MDM and *Ve* are close to 1.5 times faster than *Qu*. Nonetheless, all three techniques have fast running times and can be employed to deliver results in real-time.

7 CONCLUSION

In this paper, we establish that it is possible to effectively predict the shortest route to an anticipated customer. The proposed algorithm called *MDM: Minimizing Distance through Monte Carlo Tree Search* achieves this by employing a continuous-learning platform. MDM structures the search space of taxi routes in the form of a search tree, wherein each outgoing segment from a node acts as an arm. Through this unique strategy, MDM obtains up to 70% better performance than the state of the art technique on real road networks and taxi datasets from New York and San Francisco. Furthermore, MDM is robust enough to absorb the dynamic nature of road networks, such as aperiodic and anomalous customer demands, without compromising on the quality. In addition, MDM takes us a step closer towards the future of driver-less taxis.

8 ACKNOWLEDGEMENTS

We acknowledge the help of Animesh Singh and Hasit Bhatt in helping us construct the datasets for this project.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.
[2] Prithu Banerjee, Sayan Ranu, and Sriram Raghavan. 2014. Inferring uncertain trajectories from partial observations. In *ICDM*. 30–39.
[3] Prithu Banerjee, Pranali Yawalkar, and Sayan Ranu. 2016. Mantra: a scalable approach to mining temporally anomalous sub-trajectories. In *SIGKDD*. 1415–1424.
[4] Dan Donovan, Brian; Work. 2016. New York City Taxi Trip Data (2010-2013). https://doi.org/10.13012/J8PN93H8. (2016).
[5] Yong Ge, Hui Xiong, Alexander Tuzhilin, Keli Xiao, Marco Gruteser, and Michael Pazzani. 2010. An energy-efficient mobile recommender system. In *SIGKDD*. 899–908.
[6] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.
[7] Vinay Kolar, Sayan Ranu, Anand Prabhu Subramainan, Yedendra Shrinivasan, Aditya Telang, Ravi Kokku, and Sriram Raghavan. 2014. People In Motion: Spatio-temporal Analytics on Call Detail Records. In *COMSNETS*. 1–4.
[8] Shubhadip Mitra, Sayan Ranu, Vinay Kolar, Aditya Telang, Arnab Bhattacharya, Ravi Kokku, and Sriram Raghavan. 2015. Trajectory aware macro-cell planning for mobile users. In *INFOCOM*. 792–800.
[9] Gabor Nagy and Said Salhi. 2005. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European journal of operational research* 162, 1 (2005), 126–141.
[10] OpenStreetMap contributors. 2017. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org. (2017).
[11] Michal Piorkowski, Natasa Sarafijanovoc-Djukic, and Matthias Grossglauser. 2009. A Parsimonious Model of Mobile Partitioned Networks with Clustering. In *COMSNETS*.
[12] Jason W Powell, Yan Huang, Favyen Bastani, and Minhe Ji. 2011. Towards reducing taxicab cruising time using spatio-temporal profitability maps. In *International Symposium on Spatial and Temporal Databases*. Springer, 242–260.
[13] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. 2014. A cost-effective recommender system for taxi drivers. In *SIGKDD*. 45–54.
[14] Sayan Ranu, P Deepak, Aditya D Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*. 999–1010.
[15] Reuters. 2016. Uber debuts self-driving vehicles in landmark Pittsburgh trial. https://www.reuters.com/article/us-uber-autonomous/uber-debuts-self-driving-vehicles-in-landmark-pittsburgh-trial-idUSKCN11K12Y. (2016).
[16] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[17] The-Straits-Times. 2016. World's first driverless taxi trial kicks off in Singapore. http://www.straitstimes.com/singapore/transport/worlds-first-driverless-taxi-trial-kicks-off-in-singapore. (2016).
[18] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, and Hoong Chuin Lau. 2017. Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues. In *ICAPS*, Vol. 27. 409–417.
[19] Huimin Wen, Jianping Sun, and Xi Zhang. 2014. Study on Traffic Congestion Patterns of Large City in China Taking Beijing as an Example. *Procedia - Social and Behavioral Sciences* 138 (2014), 482–491.
[20] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL*. 99–108.
[21] Jing Yuan, Yu Zheng, Liuhang Zhang, Xing Xie, and Guangzhong Sun. 2011. Where to find my next passenger. In *13th international conference on Ubiquitous computing*. 109–118.
[22] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. 2011. Urban computing with taxicabs. In *13th international conference on Ubiquitous computing*. 89–98.
[23] Yu Zheng, Jing Yuan, Wenlei Xie, Xing Xie, and Guangzhong Sun. 2010. Drive smartly as a taxi driver. In *7th international conference on autonomic & trusted computing*. 484–486.