Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues

Tanvi Verma, Pradeep Varakantham, Sarit Kraus[†] and Hoong Chuin Lau
School of Information Systems, Singapore Management University, Singapore
[†]Department of Computer Science, Bar-Ilan University, Israel
tanviverma.2015@phdis.smu.edu.sg, pradeepv@smu.edu.sg, sarit@cs.biu.ac.il, hclau@smu.edu.sg

Abstract

Taxis (which include cars working with car aggregation systems such as Uber, Grab, Lyft etc.) have become a critical component in the urban transportation. While most research and applications in the context of taxis have focused on improving performance from a customer perspective, in this paper, we focus on improving performance from a taxi driver perspective. Higher revenues for taxi drivers can help bring more drivers into the system thereby improving availability for customers in dense urban cities.

Typically, when there is no customer on board, taxi drivers will cruise around to find customers either directly (on the street) or indirectly (due to a request from a nearby customer on phone or on aggregation systems). For such cruising taxis, we develop a Reinforcement Learning (RL) based system to learn from real trajectory logs of drivers to advise them on the right locations to find customers which maximize their revenue. There are multiple translational challenges involved in building this RL system based on real data, such as annotating the activities (e.g., roaming, going to a taxi stand, etc.) observed in trajectory logs, identifying the right features for a state, action space and evaluating against real driver performance observed in the dataset. We also provide a dynamic abstraction mechanism to improve the basic learning mechanism. Finally, we provide a thorough evaluation on a real world data set from a developed Asian city and demonstrate that an RL based system can provide significant benefits to the drivers.

1 Introduction

Taxis are an important part of urban transportation. With the rapid development of information technology, sensing and networking technologies are widely used in transportation systems. Each taxi's status and its Global Positioning System (GPS) location can be collected in real time. In fact, relying on these advances, aggregation systems such as Uber, Grab, Lyft etc. have been able to activate more cars that act like taxis thereby significantly improved customer experience by reducing wait times and increasing availability. In this paper, we focus on further improving performance (in terms of revenues earned) from a driver's perspective by using current and past movement trajectories and trips.

Recently, driver-less taxis (Reuters 2016; Straitstimes 2016) have been introduced for public trials in the US and several Asian cities. The vision is to have self-driving taxi fleets. This also serves as another motivation for pursuing this research from the perspective of taxis, as there would be no human intuition to continuously adapt to changing demand patterns.

Generally taxis roam around when they do not have a customer on board and this is referred to as "cruising". A cruising taxi can potentially find customers either directly (on streets) or indirectly (due to being in close proximity to customers who put in a call/request to taxi companies or taxi aggregation systems). In both cases, it is imperative for the taxi to be in the "right" location at the "right" time to reduce cruising time and increase revenue. Our focus in this paper is on developing a Reinforcement Learning (RL) approach that will provide guidance to cruising taxi drivers on the "right" locations to be at different times of the day on different days of the week so as to maximize long-term revenue.

RL is an ideal approach for this problem because: (a) Maximizing the long term revenue by finding customers during cruising requires making a sequence of decisions (ex: wait in current zone for 5 minutes, if no customer found, wait for 5 more minutes and if you cannot still find a customer move to zone B) ; (b) Reinforcements are well defined, i.e., the sum of revenue earned from a customer (if customer on board) and cost from travelling between locations; (c) Customer demand is uncertain and RL approaches can capture such uncertainty quite well; and finally (d) Because of its learning focus, RL can adapt to any changes in demand patterns.

However, employing an RL approach that learns from trajectory data of taxis requires the presence of well defined state and action spaces. Furthermore, those state and action spaces should be populated directly from reading the data. Because of these requirements, there are multiple translational challenges involved in applying RL for this problem at city scale:

- Typically in spatio-temporal problems, an abstraction (grouping) of locations and time is employed as the state space. For the problem of interest in this paper, such an abstraction can work for representing the state space; however, the "goodness" of the selected abstraction is not easy to ascertain. The effectiveness of abstraction is dependent on value of learned policy and learned policy is dependent on abstraction.
- Another challenge is with respect to understanding actions of drivers from the data, specifically when they are cruising. There can be multiple data points that are com-

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

bined to retrieve one action and in many cases there can be multiple interpretations to drivers' action during "cruising", such as wandering around in circles, going to a taxi stand, going to a specific street, etc.

- RL relies on learning not just from single decisions but from a sequence of decisions. Therefore, the data needs to be annotated appropriately (with states, actions and reinforcements) and then condensed into learning episodes.
- The final challenge is evaluation. While performing human experiments with actual drivers would be the ideal case, due to the capital intensive nature of such experiments, it is not feasible to consider many drivers. Therefore, we provide a detailed simulator that allows performance comparison of actual drivers (using human intuition) and our agent that uses RL.

This paper addresses the above mentioned challenges by making the following contributions:

- We provide an annotation procedure that annotates the trajectory data with the decision taken by the taxi driver. This procedure also compactly represents annotated data as an activity graph for each cruising trajectory.
- We first employ a Monte Carlo RL method to learn from the activity graphs by computing a static abstraction obtained by using clustering.
- To account for the cyclic dependence between state abstraction and learned policy, we provide an iterative abstraction approach that continually improves abstraction based on the learned information.
- Finally, we propose an evaluation method and use real data set to evaluate our policy. We also compare our policy with an experimental heuristic policy to represent simple policies that can be employed by taxi drivers.

Experimentally, we found that RL is effective and except in one time interval (evening peak hour), the average revenue earned by the learned policy is better than the top 10 percentile revenue among all drivers. For some time intervals the agent performance is better than top 1 percentile revenue among all drivers. Though we employ the revenue maximization objective, we show that taxi utilization also increases significantly.

2 Related Work

There are two categories of research relevant to the work presented in the paper: (a) Taxi guidance; and (b) Reinforcement Learning.

Taxi Guidance Taxi guidance has been studied extensively in the literature. Qu *et al.* (Qu et al. 2014) employs pick-up probability to recommend a driving route to driver for profit maximization. Similarly, Hou *et al.* (Hou et al. 2013) suggests cruising route to vacant taxis such that the overall vacancy time is minimized. (Yuan et al. 2011b) use driver's experience to find parking spots for a cruising taxi. Yuan *et al.* (Yuan et al. 2011a) employ taxi trajectories to learn traffic patterns and estimate travel time. (Powell et al. 2011) recommends locations for taxi drivers by constructing a spatio-temporal profitability map of surrounding regions of the driver and computing potential profit using historical data.

While all these approaches are closely relevant, there are multiple key differentiating aspects to this paper:

- Unlike many of the existing approaches which rely on heuristics (e.g., low profitability regions, low vacancy rates) to **myopically** optimize revenue, we provide a formal learning approach that considers long term revenue.
- Taxi driver behaviour is inherently represented within the RL approach, so any preferences (with respect to areas) are inherently captured.
- Unlike existing approaches, RL approach relies on past experiences and not just on aggregate statistics from the data.

Wang *et el.* (Wang and Lampert 2014) also employed an RL approach to improve taxi revenue but from only taxi trip data (and not from taxi trajectory data). Due to learning from trip data, they are forced to make many approximations on movement between trip end and trip start events. In addition, our abstraction method is a key differentiator.

Reinforcement Learning RL can be broadly divided into two classes, model-based learning and model-free learning. Model-based methods require a model of transition probabilities and the reward function to compute values of states. There are many existing works which deal with learning transition and reward models (Schneider 1997; Chakraborty and Stone 2011; Hester and Stone 2009). However, as we obtain samples of experience from the dataset, a model-free learning is more suitable. Temporal Difference (Tesauro 1995) and Monte Carlo (Sutton and Barto 1998) methods are well known model-free learning methods. We use Monte Carlo method to estimate our state-action values.

The problem of dynamically abstracting states has been studied by a number of researchers. Li *et al.* (Li, Walsh, and Littman 2006) provides a list of abstraction techniques and classify them into five different types of abstraction. Bulitko *et al.* (Bulitko, Sturtevant, and Kazakevich 2005) build a hierarchy of abstraction levels, explores at each level and repairs the abstraction during exploration. Andre and Russell (Andre and Russell 2002) provide a method of safe state abstraction while maintaining hierarchical optimality. Mannor *et al.*(Mannor et al. 2004) employ a clustering algorithm to cluster the state space based on computing utility of merging two neighbors. Jong and Stone (Jong and Stone 2005) encapsulate states based on policy irrelevance, states with same optimal action are aggregated.

All the approaches are closely relevant and our iterative dynamic abstraction approach builds on these approaches. There are two key differences: (a) In all the above-mentioned works, action space remains constant. However, in our problem setting, our action space is correlated to the state space, as actions correspond to "moving to a certain location or state". When state space is altered during abstraction, action space also changes. (b) Secondly, our abstraction approach employs multiple iterations to modify abstractions based on learned information.

Recently, DeepRL has been popular because of its success in playing games(Mnih et al. 2013; Silver et al. 2016). Such approaches are extremely interesting and provide extremely good results. However, there are two key issues with respect to applying DeepRL for taxi problem. While this may change in the future, at present, it seems that Deep RL is ideally suited for environments where tens of millions of learning episodes can be obtained on demand. Unfortunately in the taxi case, we do not have access to a simulator that can generate many learning episodes quickly and instead has to follow demand patterns. Also, deep learning is ideally suited for problem domains where state space has hundreds of features (like with image pixels). In our case, the number of features within the state space is too small (in the order of 10 even if we include other aspects such as revenue earned so far and other drivers) for deep learning on state space to be useful.

3 Taxi Dataset

We consider a taxi dataset from a major company in Singapore. Apart from trip[1] information, the major component of the data is the movement logs. Each log entry captures the following information:

$$\Big\langle \text{Latitude, Longitude, Taxi ID, Driver ID, Taxi Status} \Big\rangle$$

Latitude and longitude provide the GPS coordinates. Since multiple drivers can drive a single taxi (typically one person drives the morning shift and one person drives the evening shift), we have two IDs (Taxi ID and Driver ID) to uniquely determine the log entry. The log entry also contains different states of taxis - free (meter off, actively looking for next passenger), busy (not accepting bookings), POB(Passenger On Board) and off-line.

When there is no customer on board a taxi, there is a log entry for that taxi every 30 seconds. On the other hand if there is a customer on board a taxi, then there is a log entry for that taxi every 1 minute. We have this distinction because there is more important information to be captured about a cruising taxi than a hired taxi. With more than 20000 unique drivers, this dataset provides a wealth of information about cruising taxis.

4 From Taxi Dataset to Driver Activity Graphs

Before we can learn using RL, we need a representation of the decisions taken by taxi drivers during cruising. Since dataset only contains log entries, we have to annotate groups of log entries as high level decisions (e.g., going to a certain location). In this section, we describe details on converting from log entries in data to high level activities.

A ”cruising trajectory” starts when a taxi goes to ”free” state and ends when it goes to a ”non-free” state (passenger on board, busy, break, off-line, on call etc.). We mine cruising trajectories of drivers from the dataset and annotate the trajectories with the decisions made during the course of trajectory. To explain the details, let us consider the example trajectory shown in Figure 1, where a taxi driver's cruising trajectory started at A and ended at E.

[1]A trip corresponds to movement of taxi from a source to destination with customer on board.

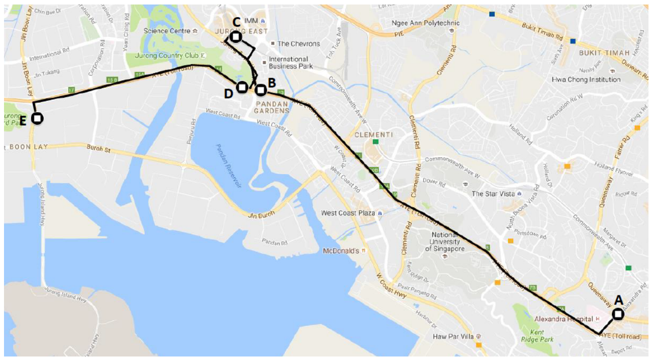

Figure 1: A cruising trajectory which starts at A and terminates at E. B, C and D are intermediate decision coordinates.

Initially, we start out by assuming that the driver made the decision to go to E at A itself. Intuitively, if the driver had made a decision to go to E at A, then he/she would have chosen a route that is close[2] to the shortest path distance between the two points. In this case, E is not close to the shortest path distance. So we identify the point on the cruising trajectory which is close to the shortest path distance to E. This point is D. We then evaluate if the driver could have made the decision to go to D at A itself. If not, we identify the point where the driver decided to go to D, which in this case happens to be C. We repeat the computation and the final trajectory is A, B, C, D, E.

As can be noted, this process requires extensive shortest path computations between different points. In order to perform this efficiently, we had to create special data structures to pre-store shortest path information between points. A typical cruising trajectory contains 50-100 coordinates in the real data.

$$\begin{tikzcd} \boxed{A} \arrow[r, "d_1"] & \boxed{B} \arrow[r, "d_2"] & \boxed{C} \arrow[r, "d_3"] & \boxed{D} \arrow[r, "d_4"] & \boxed{E} \end{tikzcd}$$

Figure 2: Activity graph for the cruising trajectory displayed in Figure 1. d_1, d_2, d_3 and d_4 are intermediate distances travelled. Nodes contain information about decision time epoch and GPS coordinate of the node. Terminating node E additionally stores trip information started there, if any.

Once the data is annotated, we then convert each cruising trajectory into an activity graph to get a summary view of driver activities. The activity graph can be viewed as a directed graph with decision coordinates as nodes. Distance travelled between the coordinates is treated as weight of the edge between them. The terminating node of the activity graph also contains information about revenue earned. If the trajectory ended with getting a trip, the revenue earned is equivalent to the fare of the trip minus the cost of travel for

[2]We allow for a 30% gap from shortest path.

the trip. The revenue earned is treated as zero if the trajectory ended without finding a passenger (taxi state changing to break, busy etc.).

5 RL for Taxi Driver Guidance

We now formulate the problem of assisting a cruising taxi as an RL problem. In this RL representation, we capture spatio-temporal aspects in the state space. Specifically, state is given as follows:

$$\langle \text{day-of-week, zone, time-interval} \rangle \tag{1}$$

We divide the entire map of Singapore into several zones and zone division procedures are described in later parts of this section. Based on traffic intensity, time is divided into 6 time intervals: 0-6 hours, 6-9 hours, 9-12 hours, 12-17 hours, 17-20 hours and 20-24 hours. If there are n zones, there are n actions available to a cruising taxi, i.e., stay in the current zone or move to remaining $n-1$ zones.

5.1 Episodes

For RL to be applied, we convert activity graphs into episodes. Each node in the activity graph represents a state and the subsequent node represents the action taken. We use the zone structure of the map and spatio-temporal information present in each node to convert activity graphs into a series of state-action pairs. The last node of the activity graph is always considered as terminal state of the episode. The cost of travel between nodes is determined by applying a fixed cost per km to the weight of the edge. If the cruising trajectory ends with finding a passenger, a positive reward (equivalent to the fare of the trip minus cost to travel the trip) is awarded.

Equation 2 represents an episode for activity graph shown in Figure 2. S_x is the state and Z_x is the zone of node X, S_{term} is the terminal state.

$$(S_a, Z_b) \xrightarrow{c_1} (S_b, Z_c) \xrightarrow{c_2} (S_c, Z_d) \xrightarrow{c_3} (S_d, Z_e) \xrightarrow{c_4, R} S_{term} \tag{2}$$

$$R = \text{fare of trip} - \text{cost to travel the trip}$$
$$c_i = d_i * \text{travelling cost per km}$$

We learn Q values of state-action pairs from episodes.

5.2 Monte Carlo Estimation of Q Values

Monte Carlo (MC) method is a way of solving the reinforcement learning problem based on averaging sample returns. We use first-visit MC method to estimate the value of a state-action (s,a) pair (Sutton and Barto 1998). Algorithm 1 provides the detailed algorithm to compute Q-values and the best action in each state.

Return of (s,a) pair ($Ret(s,a)$) in an episode is the cumulative reward accumulated till the end of the episode. For example in episode mentioned in equation 2, $Ret(S_c, Z_d)$ is $(R - c_4 - c_3)$. $Q(s,a)$, the value of (s,a) pair, is estimated as the average of the returns following the first time that the state s was visited and action a was taken in each episode. Value of s is defined as $\max_a Q(s,a)$. Line 9 computes the return for each (s,a) pair over all episodes and line 13 computes the average Q value for every (s,a) pair.

During learning, there might be a few (s,a) pairs which are visited rarely. Estimation of $Q(s,a)$ will not be accurate if very few number of episodes are used to estimate the value. To avoid such inaccuracies, we introduce a variable *min-count* and estimate values for only those (s,a) pairs which have been visited in atleast *min-count* number of episodes. $Count(s,a)$ is the total number of training episodes in which (s,a) was visited. Policy $\pi(s)$ maps state s to its optimal action. $\mathcal{S}$ is the set of states and $\mathcal{A}$ is set of actions. $\mathcal{S}_{learned}$ is the set of states for which we could learn optimal policy.

Algorithm 1 MC state-action value estimation

```
 1: Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}$
 2:     $Q(s,a) \leftarrow 0$
 3:     $Count(s,a) \leftarrow 0$
 4:     $Ret(s,a) \leftarrow$ empty list
 5:     $\mathcal{S}_{learned} \leftarrow$ empty set
 6: for every episode in training episodes do
 7:     for each $(s,a)$ pair in the episode do
 8:         $G \leftarrow$ return after first occurrence of $(s,a)$
 9:         $Ret(s,a) \leftarrow Ret(s,a) + G$
10:         $Count(s,a) \leftarrow Count(s,a) + 1$
11: for all $s \in \mathcal{S}, a \in \mathcal{A}$ do
12:     if $Count(s,a) \geq$ min-count then
13:         $Q(s,a) \leftarrow \frac{Ret(s,a)}{Count(s,a)}$
14:         if $s$ not in $\mathcal{S}_{learned}$ then
15:             add $s$ to $\mathcal{S}_{learned}$
16: for all $s \in \mathcal{S}_{learned}$ do
17:     $\pi(s) \leftarrow \underset{a}{argmax} Q(s,a)$
```

5.3 Zone Structure

In our RL model, states rely on the zone of the taxi and actions correspond to the zone the driver should move to if he/she does not find a customer. Therefore, to learn effectively from the data, we need to have the "right" set of zones. "Right" in the previous statement refers to having a set of zones which yield high Q-values while having enough data points for each (s,a) pair. More specifically, if zones are too big, it would increase uncertainty in outcome for actions (as taxi can be anywhere in a big zone). Similarly, if zones are too small, we may not have sufficient training data to learn something meaningful. We explore ways to find a balance between uncertainty and granularity. In this section we propose two ways to learn zone structure of the map:

Static Zones In this method, we learn zone structure by merging zones to neighbour zones based on the training data available in each zone. A zone z maps to a state s and vice versa if z is the zone of s as mentioned in equation 1. An episode is said to be relevant to z if there is any state s in the state-action pairs of the episode, which maps to z . We start with a large number of uniformly distributed zones and check how many relevant episodes are present in each zone, if the number is less than *min-count*, we merge the zone to its nearest zone. We repeat merging zones till each zone has sufficient data to learn from. In our experiment we started with 500 zones and the final zone structure had 111 zones.

Dynamic Zones We now describe an iterative method to learn zone structure dynamically based on Q-values of a state. As environment dynamics are different for different time-intervals, we have different zone structures for different combinations of *time-interval* and *day-of-the-week* to improve the performance. In this method, we fix *time-interval* and *day-of-the-week* so that each zone maps to a unique state and a unique action. We learn separate zone structures for different combinations of *time-interval* and *day-of-the-week*.

At a high level, at each iteration of this method, we learn Q-values for the current zone structure based on a good part of the data (about a month of data in our case). Then, based on insights explained later in this section, we decide whether certain low valued zones needs to be split into smaller zones. Once certain zones are split, we learn Q-values for the new set of zones from another part of the data. We then again check if certain zones can be split. This process is continued until our data is exhausted.

$Q(s,a)$ is expected revenue earned till the end of cruising if action a was taken in state s. Suppose action a is optimal action for a state which maps to a large zone z. As zone is large the uncertainty in outcome of taking a is also high. If the large zone is split into smaller zones, it is possible that a is not optimal any more for states mapping to smaller zones as the uncertainty is reduced. To decrease the uncertainty in outcome of optimal action, we split larger zones into smaller zones if smaller zones have adequate data and if it results in increasing the overall value of the bigger zone.

Suppose s and a are state and action that map to zone z. If z is split, it affects Q-values of s as well as Q-values of all the other states s' in which action a was taken. We term these other states as *incoming states*. Let z_1 and z_2 are new smaller zones. s_1, s_2 are states and a_1, a_2 are actions which map to new zones. a' represents rest of the actions.

Theorem 1 *The value of an incoming state s' either increases or remains same after the zone split.*

Proof 1 *Suppose as per algorithm 1, $Ret(s',a) = x$ and $Count(s',a) = n$. After split all the (s',a) pair will either map to (s',a_1) or (s',a_2). Let, $Ret(s',a_1) = x_1$, $Ret(s',a_2) = x_2$, $Count(s',a_1) = n_1$ and $Count(s',a_2) = n_2$. We can see that*

$$x = x_1 + x_2, \quad n = n_1 + n_2$$

$$Q(s',a) = \frac{x}{n}, \quad Q(s',a_1) = \frac{x_1}{n_1}, \quad Q(s',a_2) = \frac{x_2}{n_2}$$

There are two possibilities for state-action values of the new actions

1. $Q(s',a_1) = Q(s',a_2)$

$$\boxed{\begin{aligned} Q(s,' a_1) = Q(s',a_2) &\Rightarrow \frac{x_1}{n_1} = \frac{x_2}{n_2} \Rightarrow \frac{x_1}{n_1} = \frac{x - x_1}{n - n_1} \\ \Rightarrow \frac{x_1}{n_1}(n - n_1) + x_1 = x &\Rightarrow \frac{x_1}{n_1}(n - n_1 + n_1) = x \\ \Rightarrow \frac{x_1}{n_1} = \frac{x}{n} &\Rightarrow Q(s_1,a) = Q(s,a) \end{aligned}}$$

$$\textbf{\textit{Therefore:}}\ Q(s',a_1) = Q(s',a) = Q(s',a_2) \qquad (3)$$

2. $Q(s',a_1) \neq Q(s',a_2)$

Without loss of generality, let us assume $Q(s',a_1) > Q(s',a_2)$

$$\boxed{\begin{array}{l|l} Q(s',a_1) > Q(s',a_2) & Q(s',a_1) > Q(s',a_2) \\ \Rightarrow \frac{x_1}{n_1} > \frac{x_2}{n_2} & \Rightarrow \frac{x_1}{n_1} > \frac{x_2}{n_2} \\ \Rightarrow \frac{x_1}{n_1} > \frac{x - x_1}{n - n_1} & \Rightarrow \frac{x - x_2}{n - n_2} > \frac{x_2}{n_2} \\ \Rightarrow \frac{x_1}{n_1}(n - n_1) + x_1 > x & \Rightarrow x > \frac{x_2}{n_2}(n - n_2) + x_2 \\ \Rightarrow \frac{x_1}{n_1}(n - n_1 + n_1) > x & \Rightarrow x > \frac{x_2}{n_2}(n - n_2 + n_2) \\ \Rightarrow \frac{x_1}{n_1} > \frac{x}{n} & \Rightarrow \frac{x}{n} > \frac{x_2}{n_2} \\ \Rightarrow Q(s',a_1) > Q(s',a) & \Rightarrow Q(s',a) > Q(s',a_2) \end{array}}$$

$$\textbf{\textit{Therefore:}}\ Q(s',a_1) > Q(s',a) > Q(s',a_2) \qquad (4)$$

If a was the optimal action of state s' before the split, a_1 becomes the new optimal action and its value increases to $Q(s',a_1)$. If any other action a' was the optimal action then $max(Q(s',a'), Q(s',a_1))$ becomes the new value of s'. Hence, the value of s' either remains same or increases after the split. ■

Thus we do not worry about the values of incoming states and only consider the value of s to decide if it is good to split.

To learn zone structure dynamically, instead of constructing episodes, we use activity graph to construct a list of $<$ *start-point, end-point, return* $>$ tuples. Tuple $< A, B, ret >$ can be read as "at point A, it was decided to move to point B and *ret* was the cumulative reward accumulated till the end of the activity graph". A tuple can be easily mapped to an (s,a) pair by determining zones of *start-point* (maps to state) and *end-point* (maps to action). Tuple $< A, B, ret >$ maps to zone z if point A is in zone z. We construct a tuple list ($TList_z$) for each zone. The Q-values of a state can be easily estimated by mapping tuples to (s,a) pairs and averaging the corresponding returns.

We use k-means clustering algorithm to split a zone into two zones. *start-point* of all the tuples present in $TList_z$ are divided into two clusters. The tuple list of parent zone can easily be divided into tuple lists of children zones by simply checking if *start-point* of a tuple maps to z_1 or z_2. As our objective is to increase the granularity at the same time maintaining meaningful learning, we split the zone if overall value of parent state increases after split and optimal action of children state are different than the optimal action of parent state. To avoid a zone structure which is too dense, we define a threshold value of minimum zone size. We split only if children zones have sufficient training data and they are larger than the threshold value.

Algorithm 2 provides the pseudo code for learning zone structure dynamically. We start with dividing the map of Singapore into four large uniform zones and then split the zones repeatedly until further split is not possible. If tuples are from N months of data, T_n represents group of tuples which are from n^{th} month.

As after split there are new zone centres, it is possible that tuples which are mapped to a nearby zone now maps to new

Algorithm 2 Dynamic zoning

```
1: Preprocessing - Construct $T_n$ from activity graphs
2: Initialize zone-structure with 4 large uniform zones
3: for $n \in N$ do
4:     for each $tuple \in T_n$ do
5:         Append the tuple to appropriate $TList_z$
6:     repeat
7:         $converged \leftarrow true$
8:         sort zone-structure in descending order of size of zones
9:         for $z \in$ zone-structure do
10:            if WorthSplitting($z$) then
11:                $converged \leftarrow false$
12:                Split $z$ into $z_1$ and $z_2$
13:                Re-align any affected tuple
14:                Update zone-structure
15:    until converged
```

zones. We construct a list of affected tuples and realign them after we have split a zone. Algorithm 3 provides steps to

Algorithm 3 WorthSplitting(z)

```
1: Divide z into $z_1$ and $z_2$ using k-means clustering
2: if size of children zones $\leq$ min-size then
3:     return false
4: Construct tuple lists and Q-values for children zones
5: if $\underset{a}{max}Q(s,a) \geq \underset{a}{max}Q(s_1,a) + \underset{a}{max}Q(s_2,a)$ then
6:     return false
7: if $\underset{a}{argmax}Q(s,a) == \underset{a}{argmax}Q(s_1,a)$
   and $\underset{a}{argmax}Q(s,a) == \underset{a}{argmax}Q(s_2,a)$ then
8:     return false
9: return true
```

decide if it is favourable to split a zone. Table 1 displays the learned number of zones for each time-interval on weekdays and weekends.

Table 1: Number of Dynamic Zones

Day	0-6	6-9	9-12	12-17	17-20	20-24
Weekdays	54	48	51	53	86	75
Weekends	63	58	66	64	97	86

6 Experiments

We now describe the set up and results to compare the performance of our RL agent with actual drivers.

6.1 Evaluation Method

In this section "driver" means real world taxi drivers for whom we have historical data and "agent" means our learning agent which follows the learned policy. To evaluate the quality of policies learned by our approaches, we compare average revenue earned by our learning agent with the top percentile revenue of drivers. We also compare against revenue earned by greedy heuristics typically employed by drivers during cruising. For our experiments, we simulate the agent movements on real data. Since, we only advise one driver, we can assume that rest of the data about other drivers' movements does not change. Table 2 describes the terms and notation used in this section.

Simulation of Agent Movements A key aspect of the agent movement simulation is assigning the available trips to the agent while considering competition from active drivers. To accurately simulate the agent movements according to the real data, we look at the trip data and trajectories of all active drivers during a given date and time-interval. We find the relevant available trips (non pre-booked trips) that originated from each state during that date and time. Revenue earned, duration and distance also stored for each trip.

When the agent visits state s at time t, we try to assign an available trip which originated from that state. As the agent is competing with other drivers present in the state at that time, we compute an assignment probability (p^{st}_{assign}) with which a trip can be assigned to the agent. The probability can be computed as the number of trips available divided by the number of cruising drivers present in the state at that time. To compute p^{st}_{assign}, we consider durations of γ minutes. We maintain a list of trips available in every γ minute interval and the number of cruising drivers available in the corresponding interval. This way we have multiple assignment probabilities for a single state, and a trip is assigned based on when (time of day, t) the agent visits the state.

Table 2: Notations

Notation	Description	Value
travelling-cost	travelling cost per km	15 c / km
cruising-cost	travelling cost per minute	10 c / min
δ	decision interval	2 minutes
γ	duration used to compute p^{st}_{assign}	2 minutes
min-count	min number of training data needed to learn Q-values	10
min-size	min width of a zone	500m
p^{st}_{assign}	taxi assignment probability	from data
p_{stay}	probability to stay in current zone	0.5

A second aspect of the agent movement simulation is identifying the cost while "cruising". To estimate the travel time between zones, we compute the average time taken to travel between zones based on trip information present in the data. We maintain a list of average travel time between zones for every hour of the day. This information is used when the agent cruises from one zone to another zone.

Driver revenue Driver's earning is computed from the trip data. It is difficult to estimate the exact cruising distance of our agent. Hence for fair comparison, instead of applying a cost of travel per km, we apply cost of travel per cruising minute. To estimate cruising cost of drivers, we compute time duration for which the driver was not hired in the time-interval. Then a *cruising-cost* per minute is applied for this duration. Thus, a driver's revenue in a time interval is com-

Table 3: Weekdays (revenues in SGD)

Strategy	**0-6 hours**	**6-9 hours**	**9-12 hours**	**12-17 hours**	**17-20 hours**	**20-24 hours**
Average of top 1% drivers	143.26	91.86	70.81	107.75	97.09	126.97
Average of top 5% drivers	112.82	77.85	60.09	91.97	82.52	110.27
Average of top 10% drivers	97.09	69.71	54.26	83.13	74.89	101.55
Average of top 20% drivers	78.66	59.71	47.24	72.20	65.59	90.97
Heuristic	147.4	72.42	52.7	85.13	66.3	93.59
Static zone	164.77	84.98	57.71	93.32	74.76	100.99
Dynamic zone	186.52	88.37	58.3	91.43	75.25	110.95

Table 4: Weekends (revenues in SGD)

Strategy	**0-6 hours**	**6-9 hours**	**9-12 hours**	**12-17 hours**	**17-20 hours**	**20-24 hours**
Average of top 1% drivers	188.21	76.35	76.68	117.69	102.29	136.93
Average of top 5% drivers	161.27	63.74	66.22	102.44	89.31	121.35
Average of top 10% drivers	145.95	56.79	60.35	93.41	82.07	112.67
Average of top 20% drivers	126.69	48.15	52.94	81.86	72.89	102.01
Heuristic	175.92	60.48	55.43	95.23	70.31	99.94
Static zone	189.35	69.67	59.91	108.13	79.24	104.81
Dynamic zone	195.35	74.37	69.77	111.54	79.75	114.96

puted as the revenue from all the driver trips in the time interval minus cost of travelling all trip distances minus cost of all cruising.

Heuristic strategy Another benchmark we employ to evaluate performance of our learning approach is a heuristic strategy that is typically employed by drivers. When a cruising driver is in a locality, he generally takes one of two options - stay in the current locality or move to a nearby locality. When the agent follows the heuristic strategy, it stays in the current zone with a probability equal to p_{stay} and with the remaining probability moves to a nearby zone. Since $p_{stay} = 0.5$ worked the best, we employ this strategy.

Agent revenue We compute the agent's revenue for each time-interval for a given date. The actual time (t) of the day is also maintained. t is initialized with a start time of the interval. The evaluation ends when the value of t reaches the end of the time-interval. We observe the top earning drivers of the given date and time-interval and find their GPS location at the start of the interval. The corresponding state of the GPS location is used to initialize the agent's starting state. Following are the steps employed to compute the agent's revenue:
(1) We use p^{st}_{assign} to assign a trip originated from state s. One random trip is assigned from the list of available trips.
(2) If a trip was assigned to the agent, its next state becomes end zone of the trip and time t is updated. The fare of the trip (in the data) is considered as the revenue of the agent.
(3) If trip was not assigned, the agent waits for *decision-interval* (δ) minutes in the current zone before again taking decision based on learned policy.
(4) After taking action, the agent moves to the suggested zone and t is updated based on precomputed travel time between zones for the given hour of the day.
(5) After the agent reaches the next state s', it tries to assign a trip to the agent based on the $p^{s't}_{assign}$.
(6) If the action was to stay in the same zone, a trip assignment is attempted after δ minutes. The assignment strategy remains same as explained in step 1.
(7) Steps 1-6 are repeated until t is equal to the end time of the time-interval.
(8) If a policy is not available for state s, the agent takes action based on heuristic strategy.
(9) The agent's revenue is equivalent to the fare of all the assigned trips during the time interval minus cost of travelling associated with all the trips minus cost of cruising during the time interval.

Algorithm 4 Agent's Revenue

```
1: Initialize the agent's initial state $s$
2:     $t_{start}$ = start time of the time-interval
3:     $t_{end}$ = end time of the time-interval
4:     $t \leftarrow t_{start}, rev \leftarrow 0$
5:     $fare \leftarrow 0, cost \leftarrow 0, cruising\_time \leftarrow 0$
6: while $t \leq t_{end}$ do
7:     assign trip with $p^{st}_{assign}$ probability
8:     if trip was assigned then
9:         $s \leftarrow$ end state of trip
10:        $fare \leftarrow fare +$ trip fare
11:        $cost \leftarrow cost +$ cost of travelling the trip distance
12:        $t \leftarrow t +$ trip duration
13:    else
14:        $t \leftarrow t + \delta$
15:        if $s \in \mathcal{S}_{learned}$ then
16:            use learned strategy
17:        else
18:            use heuristic strategy
19:        go to advised zone
20:        $t \leftarrow t +$ time to travel to advised zone
21:        $cruising\_time \leftarrow cruising\_time +$ time to travel to
           the advised zone
22: $rev \leftarrow fare - cost - cruising\_time * cruising\_cost$
23: return $rev$
```

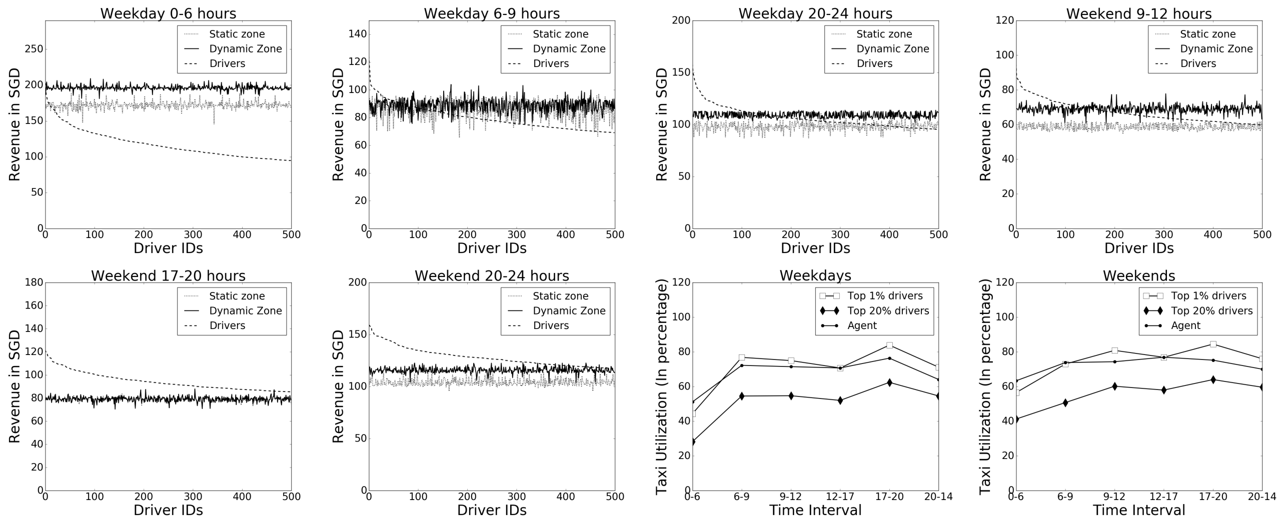

Figure 3: Revenue comparison for various time-intervals and taxi utilization comparison for weekdays and weekends

We use approximately 2 million episodes extracted from around 1% of movement trajectories over a period of 8 months to learn a policy for our agent. Total number of GPS coordinates present in experimental dataset were 197 million and there were total 84 million decision points.

6.2 Experimental Results

To evaluate the learning, we selected one month (not used for learning) and compared average agent revenue against average of top percentile revenues earned by drivers during that month. There were 20 weekdays and 10 weekends in the evaluation month. We find starting state of top 500 drivers in each time interval and use those states as initial state of the agent for evaluation. For each initial state, the revenue is averaged over 1000 executions. Hence for a given time interval and day, the agent revenue is averaged over 500,000 executions (500 different initial states * 1000 executions).

It should be noted that we are comparing average revenue of the agent against average of top percentile revenues of the drivers. Since we take best cases for the drivers (in terms of always finding customers) and average case for the agent, this provides a huge advantage to the driver revenues. This advantage is provided so as to offset any errors in cruising costs (which typically have a minor impact). Also, if the agent performs on par with top 10 percentile revenues, then this comparison allows us to claim extremely good performance for the agent.

Tables 3 and 4 present the evaluation results for weekdays and weekends respectively. We see that early morning hours when drivers mostly roam to find passengers, the agent performs much better than the other drivers. During morning time interval on weekdays, the agent's revenue is 14.7% higher for static zones and 30% higher for dynamic zones. Apart from peak-hour in the evening, the agent always fares better than top 10 percentile revenue of drivers. We believe the slightly lower performance during 17-20 hours time-interval is due to traffic congestion. The agent always follows the shortest distance route while cruising, and there might be longer routes which take less time. As time taken to travel is taken from the real data, the agent might be wasting time while following the shortest route during peak hours.

To get a better sense of the revenue earned by the agent with other drivers, we select a weekday and a weekend from the test data set. For the selected days we generate a list of top 500 drivers during each interval in terms of their revenues. We then find the corresponding starting state of drivers from their GPS logs. We use these starting states as the initial state of the agent during our evaluation. Figure 3 compares revenues of samples of an individual agent (a sample corresponds to one instantiation of trip allocation to the agent) with the top 500 drivers over different time intervals. The X-axis represents driver IDs of the best 500 drivers or 500 samples for the agent.

We also evaluate utilization of the agent, which is computed as the percentage of time the agent is occupied during a given time-interval. Figure 3 compares utilization on weekdays and weekend. We observe that though maximizing taxi utilization is not the learning objective, the agent's taxi utilization is always high (always better than 20 percentile and slightly worse than 1 percentile value of driver utilizations).

7 Conclusion and Future Work

In this paper, we show that an RL agent, with no knowledge of the environment or taxi demand scenario, is capable of obtaining revenue which is comparable to (and in some cases higher than) revenue earned by top 10 percentile of drivers. One fundamental assumption is that there is one single learning agent in the environment and the rest of the drivers in the environment do not learn. We recognize that the environment dynamics would be drastically different if there are multiple learning agents. Hence, an immediate future direction of this work is to explore when there are cooperative or selfish agents who are also learning.

8 Acknowledgements

This research project is partially supported by National Research Foundation Singapore under its Corp Lab @ University scheme and Fujitsu Limited. Sarit Kraus was supported in part by the the Israeli Ministry of Science grant No. 3-13564.

References

Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, 119–125.

Bulitko, V.; Sturtevant, N.; and Kazakevich, M. 2005. Speeding up learning in real-time search via automatic state abstraction. In *AAAI*, volume 214, 1349–1354.

Chakraborty, D., and Stone, P. 2011. Structure learning in ergodic factored mdps without knowledge of the transition function's in-degree. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 737–744.

Hester, T., and Stone, P. 2009. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 717–724. International Foundation for Autonomous Agents and Multiagent Systems.

Hou, Y.; Li, X.; Zhao, Y.; Jia, X.; Sadek, A. W.; Hulme, K.; and Qiao, C. 2013. Towards efficient vacant taxis cruising guidance. In *2013 IEEE Global Communications Conference (GLOBECOM)*, 54–59. IEEE.

Jong, N. K., and Stone, P. 2005. State abstraction discovery from irrelevant state variables. In *IJCAI*, 752–757. Citeseer.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *ISAIM*.

Mannor, S.; Menache, I.; Hoze, A.; and Klein, U. 2004. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, 71. ACM.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Powell, J. W.; Huang, Y.; Bastani, F.; and Ji, M. 2011. Towards reducing taxicab cruising time using spatio-temporal profitability maps. In *Advances in Spatial and Temporal Databases*. Springer. 242–260.

Qu, M.; Zhu, H.; Liu, J.; Liu, G.; and Xiong, H. 2014. A cost-effective recommender system for taxi drivers. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 45–54. ACM.

Reuters. 2016. Uber debuts self-driving vehicles in landmark pittsburgh trial. *Reuters*, 14 September 2016. Available: http://www.reuters.com/article/us-uber-autonomous-idUSKCN11K12Y [Last accessed: November 2016].

Schneider, J. G. 1997. Exploiting model uncertainty estimates for safe dynamic control learning. *Advances in neural information processing systems* 1047–1053.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Straitstimes. 2016. World's first driverless taxi trial kicks off in singapore. *The Straits Times*, 26 August 2016. Available: http://www.straitstimes.com/singapore/transport/worlds-first-driverless-taxi-trial-kicks-off-in-singapore [Last accessed: November 2016].

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tesauro, G. 1995. Temporal difference learning and td-gammon. *Communications of the ACM* 38(3):58–68.

Wang, J., and Lampert, B. 2014. Improving taxi revenue with reinforcement learning.

Yuan, J.; Zheng, Y.; Xie, X.; and Sun, G. 2011a. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 316–324. ACM.

Yuan, J.; Zheng, Y.; Zhang, L.; Xie, X.; and Sun, G. 2011b. Where to find my next passenger. In *Proceedings of the 13th international conference on Ubiquitous computing*, 109–118. ACM.