Tutorial: (Ruining Dong, Wednesday 6:15pm - 8:15pm)

APPLIED DATA SCIENCE:
BUILDING A GAME PLAYING AGENT TO MODEL TAXI DRIVERS

Akira Takihara Wang
The University of Melbourne
913391

Calvin Zhuoqun Huang
The University of Melbourne
908525

Chaoyi Chen
The University of Melbourne
825874

Yong See Foo
The University of Melbourne
910978

October 22, 2019

1 Introduction

In New York City (NYC), the iconic Yellow Taxi rides illustrate the life of New Yorkers - a busy city buzzing with citizens by day with a vivid nightlife to match it. In the golden time of Yellow Taxis, it was synonymous to New York like Broadway and the Empire State Building - but it has now faded from popularity and is no longer the ride of choice. Today, thanks to the sudden rise of popular ride-sharing services such as Uber, Lyft, Via, Juno and Gett, the competition has become a fierce playing ground for the transportation business.

During this time, the New York Taxi and Limousine Commission (TLC) released a staggering dataset [11] with over 1.5 billion records of Taxi and For-Hire vehicle trips over numerous years. Although the data used for this work is exclusive to 2015-2017, the data still holds over 280 million instances - indicating a computationally expensive problem even with powerful computers. Hence, additional tools and methods to tackle these limitations were needed, particularly methods such as serialization using feather [17], optimized languages for big data such as Apache Arrow [3] and multiprocessing via the Python library [5].

So, the question arises naturally: *"How can a taxi driver maximize their earnings, despite the growing competition in New York?"*. In the previous phases of our investigations, we found that Manhattan and the Airports were the "best" areas, that is, they had the highest frequency of trips (a crude approximation to taxi demand) and highest revenue generated consistently. We also draw from other existing works which have addressed this problem from different perspectives. One such approach is to build a taxi recommendation system to reduce the idle times of taxis [18][19]; whilst another popular approach is to model the taxi cruising process as a Markov Decision Process (MDP) [9][15]. The former approach reduces the optimization problem to a network flow problem to assign trips to taxis. However, the analyses rely on external datasets which contain taxi IDs which is not available to us. Due to the limitation of data availability, this approach was not possible. The latter approach uses MDP to maximize the earnings of taxi drivers and will serve as the basis of one of our approaches. MDP is capable of modelling taxi trips as movements between spatial-temporal states, providing an effective solution to the optimization problem at hand.

Since this work aims to deploy some form of a real-time recommendation solution, several tree-search based machine learning algorithms became unfeasible due to the high complexity and costly computation. Although more dynamic approaches such as Reinforcement Learning exists, the difficulty in training and interpreting its decisions make it difficult to work with. Conversely, a plethora of commonly used techniques from Game Theory can be applied to this problem if we abstract the decisions of a taxi driver into a game-playing agent.

Explicitly, an Opening Book [7] is a powerful technique which refers to a database of "opening moves" for a given game (e.g. Chess, Shogi, Go). Agents which utilize Opening Books are significantly boosted inefficiency as it eliminates the need to search through computationally expensive decisions. For example, the first ten moves of a Chess game is quite open-ended, which would be practically impossible to compute in real-time - a necessity for our problem. But, if an agent can create an Opening

Book, it will be able to return instantaneous decisions and save precious computational time and look ahead. Because of this effective property, we implement a similar strategy to Opening Book, which is called Book Learning.

2 Problem Overview

In this section, we formulate the problem into a game with discrete states, define the subproblems required to be solved for this game, and construct a generic definition of an agent in this space.

As the problem *"How can a taxi driver maximize their earnings, despite the growing competition New York?"* is vague, we abstract this "goal" as maximizing the earnings of a taxi driver in a full working week within NYC. For this problem, a full working week is defined as 6 shifts, each with a maximum passenger transportation time of 10 hours. Taxi drivers must also have a minimum of an 8 hour break between two consecutive shifts. With the problem now formally defined, we represent a game to be $g_i \in G$, where G is the collection of all possible games.

2.1 Game State

Throughout the instance of a game g_i, a taxi driver belongs to some particular state s at any given point of time. A *state* is composed of a location on the game board and a discretized time component.

To reduce the complexity of the game, the space of NYC is represented as a 86×141 2D grid, where only 3508 cells make up NYC. This drastically reduces the complexity of the game, which is easier to deal with compared to the complex road network of New York. Taxi cruising is modelled by travelling between adjacent grid cells, rather than by travelling on roads.

The time component is divided into bins with a base unit of minutes, where all game decisions are made on a per-minute basis. The time and space are then combined to define the states of the game. The above definition can be written down more formally for the game g_i in the table below:

Variable		Cardinality
Location	$l \in L$	$\lvert L\rvert = 3508$
Time	$t \in T(i)$	$\lvert T(i)\rvert = 7 \times 24 \times 60 \approx 10080$
State	$s(l,t) \in \mathbb{S}$	$\lvert\mathbb{S}\rvert = 3508 \times 10080 \approx 3.5 \times 10^7$

In the following sections, we will use l_s, t_s to denote the location and time at a given state s.

2.2 Agent Actions

We proceed by abstracting the taxi driver as an agent in this game. In this problem, a taxi driver is either:

- searching for passengers,
- transporting passengers,
- resting between shifts, or
- choosing a shift time and starting location.

An agent only has to make decisions in the first and last scenarios. When searching for passengers, an *action* can be defined as (l', z), where l' is a location the agent moves to, and

$$z = \begin{cases} 0 & \text{if the taxi is unavailable for trips} \\ 1 & \text{if the taxi is for hire} \end{cases}.$$

Given that the agent is currently at state $s(l,t)$, its set of possible actions is $A_s = (N(l) \cup \{l\}) \times \{0,1\}$, where $N(l)$ is the set of locations adjacent to l. We define the full set of actions as $A = S \times \{0,1\}$, and it is straightforward to see that A_s is a subset of A. Following this formal definition of an action, we define a *policy function* $\pi \in \Pi$, where $\Pi : S \to A$ maps all pickup states $s \in S$ to one of the possible actions $a \in A_s$.

The other situation to consider is the choice of shift times and starting locations. As there are 6 shifts, the agent has to select 6 starting times $T_{\text{start}} = \{t_{\text{start}}^{(i)}\}_{i=1}^{6}$ and 6 starting locations $L_{\text{start}} = \{l_{\text{start}}^{(i)}\}_{i=1}^{6}$.

2.3 Earnings

With the definitions declared above, we can now formalize the game goal as the *maximization of earnings* for a taxi driver. In our definition, a trip includes:

- an initial location l and time t,
- a ending location l' and time t', and
- an income of r.

Suppose that during shift i, an agent moved through the sequence of states $w^{(i)} = (s_0^{(i)}, s_1^{(i)}, \dots s_m^{(i)})$, where $s_0^{(i)} = (l_{\text{start}}^{(i)}, t_{\text{start}}^{(i)})$. Let $r_j^{(i)}$ denote the income generated by moving from $s_{j-1}^{(i)}$ to $s_j^{(i)}$ for $j = 1, 2, \dots m^{(i)}$. If there is no trip between the two states, then $r_j^{(i)} = 0$. The movements of the agent across a given week can be summarised as $w = \{w^{(i)}\}_{i=1}^{6}$. The total earnings across a week can then be expressed as:

$$q(w) = \sum_{i=1}^{6} \sum_{j=1}^{m^{(i)}} r_j^{(i)}.$$

2.4 Goal of This Work

The goal for this problem is to invent an agent $A = (\pi, L_{\text{start}}, T_{\text{start}})$, such that the expected value of $q(w)$ is maximized. Particularly, we seek to find an agent A^* such that

$$A^* = \arg\max_A \mathbb{E}_G[\mathbb{E}_{W(A)} q(w)],$$

where $W(A)$ is a random variable capturing the values of w (as defined in the previous section).

3 Methodology

Following the Problem Overview, we can now describe our approach to this problem. We break the problem down into two major components:

1. Given the shift times T_{start}, infer the optimal policy π and starting locations L_{start}.
2. Infer the optimal shift times T_{start}.

We begin our analysis by first preprocessing our datasets. The preprocessing steps include cleansing the dataset to only contain accurate records, and feature engineering additional attributes for convenience.

We then benchmark a provided Random Walk Agent as a baseline for performance. Consequently, a preliminary Pseudo Greedy Agent which makes locally optimal choices is developed. This is formulated by moving to regions that are known to have high taxi demand as determined from previous work. Since the Pseudo Greedy Agent has no practical heuristic once inside the aforementioned regions, we refine the agent by introducing an additional heuristic based on trip counts. This improved agent is named the Advanced Greedy Agent. The trip counts heuristic will also be used to find an ideal starting location.

The downsides of the greedy agents is that they are only capable of making locally optimal choices. Therefore, we also develop a Markov Model which is designed to maximize the total earnings of an entire shift instead, directly addressing our goal. This is to be achieved by utilizing dynamic programming techniques.

The last component to developing the agents is to find the optimal starting times of the shift. This is done by performing a grid search to find the starting times that correspond with the highest expected earnings of an agent.

For evaluation, we simulate each of the developed agents and compare their respective distributions of earnings. The agent with the highest expected earnings is to be selected as the final agent for the game. Lastly, we interpret the decision making process of our agents by visualizing the proposed movements between cells on a map of NYC.

4 Datasets

The datasets applied in this work are non-trivial as they are of high dimensionality, with a size of 35GB+ of data (when stored in `csv` format). In addition, the datasets come in several categories, all of which require different preprocessing and analysis. These analyses will be outlined in the following subsections.

4.1 Description

There are precisely three categories of data used in the work: the game board representation, the historical taxi trips, and external data for additional information.

4.1.1 Game Board Representation

The representation is captured inside a `GeoJSON` file, which projects a grid overlaid on the NYC area and is provided as part of problem. Although the actual cells are obtained by overlaying a 87×141 rectangular grid, the game grid itself only comprises of 3508 cells to approximate the geography of NYC. These cells were chosen in a way such that the connectivity of the board is not compromised. The information provided by the `GeoJSON` file for each cell includes

- an ID,
- the coordinates of its four vertices, and
- a list of adjacent cells (neighbours).

4.1.2 Taxi Trips

The raw taxi trips data is distributed by the NYC Taxi & Limousine Commission [11], and is publicly available on the Internet. Notably, the dataset covers all years between 2009 to the present time, including a variety of vehicles. These include the Yellow Taxis (Medallion), the Green Taxis (Street Hail Livery) and For-Hire Vehicles (FHVs). The Yellow Taxis are covered for the whole time period; the Green Taxis were introduced in August 2013; and the FHV's were introduced in 2015. It should be noted that the original data prior to July 2015 include GPS coordinates (latitude, longitude); whilst the post-2015 data stores locations as taxi zones.

In this work, a modified dataset combining both the Yellow and Green taxi data between mid 2015 to mid 2017 is utilized. Particularly, the pick-up and drop-off locations are mapped to cell coordinates, as provided by the Game board Representation data.

4.1.3 Other External Data

External data is also collected to improve the flexibility and robustness of the models. Here, any data that is not publicly available via TLC or provided as part of the work is considered to be external. In particular, we used weather data collected from Iowa Environmental Mesonet[2], licensed by Department of Agronomy, Iowa State University. In this weather dataset, daily weather information including 24 hours of precipitation, 24 hours of snowfall and snowfall depth observations are collected and recorded.

We have also analysed some additional datasets that may have some potential impact on traffic conditions, such as data for active projects under construction [13], emergency response incidents [14] and motor vehicle collisions [12]. Although these events and incidents have impact in real life applications, as shown in our previous investigations, we omitted these datasets in our agent development. Instead, these datasets were utilized in previous investigations to provide background knowledge about potential factors that influence the profitability of taxi trips.

4.2 Preprocessing

4.2.1 Game Board Representation

The Game Board Representation itself did not require any preprocessing as it was a complete product provided for this work. It is noted that although the file comprises of 3508 cells, only 3475 distinct cells in both pick-up and drop-off locations appear in the TLC taxi trips.

4.2.2 Taxi Trips Data

The preprocessing steps on the taxi trips data can be divided into cleaning, processing and feature engineering.

During the cleaning stage, we applied the same preprocessing steps undertaken in the test data to the train data. This is because the train data only provides the additional attributes to map the location to cell location, but nothing else. Specifically, we executed a cleaning script which removes instances that:

- have a trip time is less than a minute or longer than 10 hours,
- do not record the pickup or dropoff cells, or
- have payment type apart from Cash or Card (1 and 2 respectively).

Following the cleaning script, numerous features were engineered. These included:

- `earning`, sum of fare amount and tip of the trip,
- `dropoffX`, column of drop-off cell,

- `dropoffY`, row of drop-off cell,
- `pickupX`, column of pick-up cell,
- `pickupY`, row of pick-up cell,
- `duration`, duration of the trip,
- `mow`, minute of week of the trip,
- `doy`, day of year of the trip,
- `woy`, day of week of the trip,
- `moy`, month of year of the trip, and
- `year`, year of the trip.

In addition to the aforementioned cleansing script, trips that were inferred to have inaccurate records were also removed. This includes any instance which has a:

- trip speed greater than 65 mph,
- fare amount less than $2.50, or
- fare rate (i.e. fare amount divided by trip duration) less than $0.50 per minute.

Trips with speed over 65 mph are not logically valid because the maximum speed limit NYC is capped at 65 mph. The other two rules are inferred from the official NYC taxi fare rates [10]. We adhered to these additional rules such that our agents will not include any inference that is reliant on incorrect records.

As the training dataset is considerably large, attributes were converted to the lowest possible precision in order to conserve space. For example, `int64` data types are cast into `uint16` or `uint8` types (depending on the attribute range) to reduce memory consumption.

4.2.3 Other External Data

The external datasets followed the same preprocessing steps from previous investigations, and were fitted to the cells in the game board where possible.

5 Agents and Models

In this section, we provide the details of the proposed agents, including their respective pseudocode for reproducibility. In addition, the mathematical derivation for the Markov Model is provided to justify its feasibility.

5.1 Random Walk

RANDOM-WALK($cell$)
1 $neighbours \leftarrow cell.neighbours()$
2 **return** RANDOM-CHOICE($neighbours$)

Figure 1: The Random Walk Agent.

We were provided with a Random Walk Agent, which moves to an adjacent cell randomly. It is noted that this agent is always for hire. The Random Walk Agent is used as a point of comparison for the other Agents, and illustrates the difficulty of the problem. This is an appropriate baseline agent as it is trivial to implement, and is instantaneous in decision-making.

5.2 Pseudo Greedy

The Greedy Agent is an agent which utilizes a problem-solving heuristic in order to inform locally optimal choices, with the intent of approximating the globally optimal strategy. Solutions that make globally optimal choices tend to require an exploration of a large state space. In contrast, greedy approaches replace these expensive searches with estimations that are less time-consuming.

Generally, the implementation of a Greedy Agent will include:

- **Candidate set:** The set of candidate solutions for a given action.

```
DIJKSTRA(initial, goals)
 1  if initial.state() in goals
 2    then return 0
 3
 4  frontier ← PriorityQueue()
 5  frontier.add(initial)
 6  expanded ← {state : best_path}
 7  solution ← ∞
 8  while frontier and top(frontier) < solution
 9  do parent ← pop(frontier)
10     for child in successors (parent)
11     do state ← child.state()
12        if state not in expanded or child < expanded[state]
13          then expanded[state] ← child
14               frontier.add(child)
15               if child in goals and cost < solution
16                 then solution ← child
17
18  return solution

MAP-HEURISTIC(nodes, goal_states)
1  solution ← Dictionary()
2  for node in nodes
3  do solution[node] ← DIJKSTRA(node, goal_states)
4  return solution
```

Figure 2: Map Heuristic which computes the minimum path cost from every node to any goal state.

- **Selection function:** A function which determines the best candidate to be added to the solution pool.
- **Feasibility function:** An additional function which determines if a particular candidate will contribute any information to a solution.
- **Objective function:** A function which assigns a value or cost to an action or solution.
- **Solution Function:** The function which indicates whether or not the Agent has discovered the optimal solution.

In our case, the objective function is defined as a distance-based map heuristic, which we describe in the following section.

5.2.1 Map Heuristic

In order to build a map heuristic, we first require a set of cells to be defined as *goal states*. Since our goal is to maximize the earnings of a taxi driver, these goal states should correspond to the cells with high taxi demand. The choice of these specific cells as goal states are described in Section 7. The map heuristic for a given cell is then the minimum path distance from that cell to any of the goal states. A common shortest path algorithm is Dijkstra's algorithm, which is used to find the shortest path between a node to every other node. Here, we modify Dijkstra's algorithm to instead return the minimum path distance from a given cell to the goal states, as described in Figure 2.

5.2.2 Agent Logic

Since we have defined multiple goal states, a Greedy Agent should take the shortest path to reach one of these goal states. This is done by moving to the adjacent cell with the lowest heuristic cost. Once in a goal state, a Greedy agent will then remain within these goal states to optimize its utility. Hence, we designed the proposed Pseudo Greedy Agent to degenerate into a Random Walk Agent within the goal states. The workings of this agent is detailed in Figure 3. By nature, a Greedy Agent will always be for hire to obtain as many trips as possible.

5.3 Advanced Greedy Agent

One of the weaknesses of the Pseudo Greedy Agent is the lack of any substantial strategy when it is within the goal states. This is especially problematic because the taxi will remain in these goal states once they are reached. To alleviate this issue, we propose a refined agent to optimize cruising behaviour within the goal states.

```
PSEUDO-GREEDY(cell, nodes, goal_states)
 1  utility ← Array()
 2  map ← MAP-HEURISTIC(nodes, goal_states)
 3  for node in cell.neighbours()
 4  do min_cost ← (argmin map, min map)
 5     utility.add(min_cost)
 6
 7  node, cost ← min(utility)
 8  if cost is 0
 9    then return RANDOM-WALK(cell, cell.neighbours() ∩ goal_states)
10
11  return node
```

Figure 3: The Pseudo-Greedy Agent with Random Walk behaviour when the goal state is reached.

```
ADVANCED-GREEDY(cell, nodes, goal_states, BOOK)
 1  utility ← Array()
 2  map ← MAP-HEURISTIC(nodes, goal_states)
 3  for node in cell.neighbours()
 4  do min_cost ← arg min map, min map)
 5     utility.add(min_cost)
 6
 7  node, cost ← min(utility)
 8  if cost is 0
 9    then return EXPLORE(node.neighbours(), BOOK)
10
11  return node

EXPLORE(neighbours, BOOK)
1  for node in BOOK
2  do if node in neighbours
3        then return node
4
5  return RANDOM-CHOICE(neighbours)
```

Figure 4: The more Advanced Greedy Agent with an additional heuristics for actions inside a goal state.

We implement this via Book Learning, where a ranking of cells is to be specified and stored in a "book". We deduce the desirability of a cell by inferring from historical training data (the details are delayed until Section 7). An Advanced Greedy Agent (Figure 4) will then move the most highly ranked adjacent cell when it has reached a goal state. In the case of potential changes to the list of cells, a random choice will be returned if no neighbours of a cell exist in the "book". There should be no observable impact on the cost of running this algorithm, as its complexity is linear in the size of the board.

5.4 Markov Model

As greedy agents are designed to make locally optimal choices, we now turn our attention to build a model which makes *globally optimal* choices instead. In other words, the total earnings from a given shift is sought to be maximized. The game inherently possesses the Markov property as the trips obtained by a taxi driver at a given game state does not depend on the trip history of that taxi driver. Hence, the problem may be formulated as a Markov Decision Process (MDP).

In fact, this approach has been investigated by Li et al.[9], which has also been implemented in the context of New York City taxi trips. However, there are a few differences between this previous work and our problem setting. Firstly, the problem formulation found in [9] assumes that taxi drivers must always be for hire, whereas our agent has the flexibility to choose not to be for hire, which adds complexity into the decision making. Secondly, [9] considers a model with 2500 locations and 1 hour shifts (with minutely resolution), which corresponds to 1.5×10^5 states, a much smaller state space compared to that of our problem formulation.

For this section, the start time of a shift is fixed and denoted as t_{start}. Suppose that the agent has to make moves for up to the first t_{shift} minutes of the shift (specifically, $t_{\text{shift}} = 12 \times 60 = 720$). For this section, we impose $t \in \{t_{\text{start}}, t_{\text{start}}+1, \ldots, t_{\text{start}}+t_{\text{shift}}-1\}$ to model a specific shift. In addition, we also use $l_{\pi(s)}, z_{\pi(s)}$ to denote the location and availability as provided by $\pi(s)$.

For simplicity, we drop the restriction that the maximum time for transporting passengers is 10 hours. The game without this restriction is referred as the *simplified game* hereafter. This is done such that the game state can be completely captured by s. Let $U_\pi(s)$ be a random variable representing the earnings from state s until the end of the shift, given that the agent is following policy π. If the start time of a shift is fixed to be t_{start}, an ideal agent should choose a policy π^* and a starting location l_{start} according to

$$(\pi^*, l_{\text{start}}) = \arg\max_{(\pi,l)} \mathbb{E}\left[U_\pi(l, t_{\text{start}})\right].$$

In other words, we seek to maximize the expected earnings of an entire shift in the simplified game. The rest of this section describes an approximate approach to this optimization.

5.4.1 Dynamic Programming

Recalling that $U_\pi(s)$ represents the earnings from state s until the end of the shift, we may compute $\mathbb{E}\left[U_\pi(s)\right]$ by using the values of $\mathbb{E}\left[U_\pi(s')\right]$ where state s' may be drawn from any state that occurs later than state s. This suggests that dynamic programming is a viable approach to this optimization problem.

If the agent obtains a trip which arrives at state s', then[1] $U_\pi(s) = R(s,s') + U_\pi(s')$, where $R(s,s')$ is random variable denoting the earnings from a trip that begins at state s and ends at state s'.[2] Otherwise, if the agent does not obtain a trip during state s, then $U_\pi(s) = U_\pi(l_{\pi(s)}, t_s + 1)$ as the agent will move to a neighbouring location (or stay in its current location). Recalling that $z_{\pi(s)} = 0$ if and only if the agent decides to be unavailable during state s (under policy π), we have

$$P(U_{\pi,s} = u) = \begin{cases} \min\{z_{\pi(s)}, P(s'|s)\} & \text{if } u = R(s,s') + U_\pi(s') \\ 1 - \min\{z_{\pi(s)}, \sum_{s' \in S} P(s'|s)\} & \text{if } u = U_\pi(l_{\pi(s)}, t_s + 1) \end{cases},$$

where $P(s'|s)$ is the probability of obtaining a trip to state s' from state s. Note that if $t_{s'} \leq t_s$, then $P(s'|s) = 0$, as the time must strictly increase during a trip. Writing $V(s) = \mathbb{E}\left[U_\pi(s)\right]$ for some specific policy π, we then arrive at

$$V(s) = \begin{cases} V(l_{\pi(s)}, t_s + 1) & \text{if } z_{\pi(s)} = 0 \\ \sum_{s' \in S} P(s'|s)\left[\mathbb{E}R(s,s') + V(s')\right] + \left[1 - \sum_{s' \in S} P(s'|s)\right] V(l_{\pi(s)}, t_s + 1) & \text{if } z_{\pi(s)} = 1 \end{cases},$$

which may be simplified to

$$V(s) = V(l_{\pi(s)}, t_s + 1) + \sum_{s' \in S} \min\{z_{\pi(s)}, P(s'|s)\}\left[\mathbb{E}R(s,s') + V(s') - V(l_{\pi(s)}, t_s + 1)\right]. \tag{1}$$

Therefore, to maximize $V(s)$, the policy is chosen to be

$$l_{\pi(s)} = \arg\max_{l' \in N(l_s) \cup \{l_s\}} V(l', t_s + 1) \text{ and } z_{\pi(s)} = \begin{cases} 0 & \text{if } \sum_{s' \in S} P(s'|s)\left[\mathbb{E}R(s,s') + V(s') - V(l_{\pi(s)}, t_s + 1)\right] < 0 \\ 1 & \text{otherwise} \end{cases},$$

which can be achieved by classical dynamic programming. This is possible as the evaluation of $\pi(s)$ and $V(s)$ requires only the values of $V(s')$ where $t_{s'} > t_{s'}$, i.e. computation is performed in the order $t = t_{\text{start}} + t_{\text{shift}} - 1, t_{\text{start}} + t_{\text{shift}} - 2, \ldots, t_{\text{start}}$. Lastly, the starting location can be then be chosen by setting

$$l_{\text{start}} = \arg\max_{l \in L} V(l, t_{\text{start}}).$$

5.4.2 Efficient Optimization

The optimization proposed so far includes several challenges:

- The start time of the shift is to be given as an input to this algorithm, hence the algorithm must be efficient enough to support the demand of running this algorithm for multiple start times.
- The computation of $V(s)$ for a particular state s involves a summation of up to $\mathcal{O}(|S|)$ terms, hence the algorithm has worst-case time complexity $\mathcal{O}(|S|^2)$, which is far too costly.

[1] This only holds for the simplified game.

[2] If $t_{s'} \geq t_{\text{start}} + t_{\text{shift}}$, then $U_\pi(s') = 0$.

- $P(s'|s)$ and $\mathbb{E}R(s,s')$ are parameters to be estimated by utilizing historical taxi trip data, where the parameter space is of size $\mathcal{O}(|S|^2)$. Apart from concerns about space requirements, the model will be prone to overfitting as the number of trip records available is orders of magnitude smaller than $|S|^2$.

To address these issues, we decided to compromise by ignoring the time component of the dropoff state of each trip. Specifically, we make the following changes:

- $P(s'|s)$ is replaced by $P(l'|s)$, the probability of obtaining a trip to location l' given that the agent starts in state s.
- $\mathbb{E}R(s,s')$ is replaced by $\mu(l'|s)$, the expected trip earnings for a trip that starts in state s and ends at location l'.
- $t_{s'} - t_s$ is replaced by $\lambda(l'|s)$, the expected duration for a trip that starts in state s and ends at location l'.

The justification for this compromise is that given the pickup state s and the dropoff location l', there will only be a limited range of possible trip durations $t_{s'} - t_s$. Hence, the parameter space can be reduced if this duration is simply replaced by a constant value $\lambda(l'|s)$. Likewise, the pickup state s and the dropoff location l' should provide sufficient information to estimate the probability and expected earnings of a trip, without leveraging the trip duration $t_{s'} - t_s$. The result is that the optimization is far more approximate, but the parameter space and overall time complexity is now $\mathcal{O}(|S||L|)$. Though this may still seem large, in practice this approach is efficient enough as the parameters $P(l'|s)$ are rather sparse (due to how taxi trips are mostly concentrated around the goal states).

Equation 1 is now replaced by the approximation

$$V(s) \approx V(l_{\pi(s)}, t_s + 1) + \sum_{l' \in L} \min\{z_{\pi(s)}, P(l'|s)\} \left[\mu(l'|s) + V\left(l', t_s + \lambda(l'|s)\right) - V(l_{\pi(s)}, t_s + 1)\right]. \tag{2}$$

The details of the algorithm can be found in Figure 5. The policy and starting location are then stored to physical memory, to be extracted by the agent during the runtime of the game.

5.4.3 Parameter Estimates

The remaining component to this approach is to estimate the parameters P, μ and λ, which only have to be computed once. Although the parameter space has already been reduced, it is still larger than the number of historical trips available. This leads to a potential downfall of overfitting to the training data. Therefore, it is advantageous to bin the time component to further decrease the number of parameters to be estimated. In particular, every time $t \in \{1, 2, \ldots, 10080\}$ may be partitioned into bins of B minutes, by the function

$$b(t) = \left\lfloor \frac{t}{B} \right\rfloor.$$

The reduction in parameter space is then achieved by enforcing the restrictions

$$\begin{aligned}
\hat{P}(s'|l,t) = \hat{P}(s'|l,t+1) &= \cdots = \hat{P}(s'|l,t+B-1) \\
\hat{\mu}(s'|l,t) = \hat{\mu}(s'|l,t+1) &= \cdots = \hat{\mu}(s'|l,t+B-1) \\
\hat{\lambda}(s'|l,t) = \hat{\lambda}(s'|l,t+1) &= \cdots = \hat{\lambda}(s'|l,t+B-1)
\end{aligned}$$

for all $l \in L$ and $t \in \{1, B+1, \ldots, 10081 - B\}$.

Let D denote the dataset of historical trips available, and let

$$D^{(l,l',b)} = \{d \in D : \text{trip } d \text{ begins in location, ends in location } l', \text{ and starts during time bin } b\}.$$

Denote the trip earnings and trip duration of a trip $d \in D$ as $r(d)$ and $t(d)$ respectively. Since the time t_s component of a state s corresponds to a particular minute of any week, we obtain the estimate

$$\hat{P}(l'|l,t) = \min\left\{1, \frac{|D^{(l,l',b(t))}|}{\#\{\text{occurrences of time } t_s \text{within dataset timeframe}\}}\right\}.$$

The denominator should be equal to the number of weeks spanned by the dataset ± 1. Similarly, the expected earnings may be estimated by the aggregation

$$\hat{\mu}(l'|l,t) = \frac{1}{|D^{(l,l',b(t))}|} \sum_{d \in D^{(l,l',b(t))}} r(d).$$

```
MARKOV-MODEL(L, t_start, t_shift, P, μ, λ)
 1  V ← Array() of dimensions |L| × |T|
 2  π ← Array() of dimensions |L| × |T|
 3  t_end ← t_start + t_shift − 1
 4
 5  for t ← t_end to t_start
 6  do for l in L
 7     do dest ← l
 8        no_trip ← 0
 9        if t < t_end
10          then no_trip ← V[l][t + 1]
11               for l′ in l.neighbours()
12               do if V[l′][t + 1] > no_trip
13                     then dest ← l′
14                          no_trip ← V[l′][t + 1]
15
16        s ← (l, t)
17        margin ← 0
18        for l′ in L
19        do δ ← μ(l′|s) − no_trip
20           if t + λ(l′|s) ≤ t_end
21             then δ ← δ + V[l′][t + λ(l′|s)]
22           margin ← margin + P(l′|s) × δ
23
24        if margin < 0
25          then π[l][t] ← (dest, 0)
26               V[l][t] ← no_trip
27          else π[l][t] ← (dest, 1)
28               V[l][t] ← no_trip + margin
29
30  return π, arg max_l V[l][t_start]
```

Figure 5: Finding an approximately optimal policy and starting location based on a MDP framework.

As for the expected durations, we instead take an average of the log-transformed durations, and then invert the transform after aggregation

$$\hat{\lambda}(l'|l,t) = \exp\left\{ \frac{1}{|D^{(l,l',b(t))}|} \sum_{d \in D^{(l,l',b(t))}} \ln t(d) \right\}.$$

This is because the durations are known to be positively skewed, therefore a log-transformed aggregate is more robust against outliers of extreme trip durations. This calculation is equivalent to taking the *geometric median*, which is reported by Sauro and Lewis [16] to be the most adequate summary of durations, especially if the sample size $|D^{(l,l',b)}|$ is small.

Note that if $|D^{(l,l',b)}| = 0$, then the corresponding parameter estimates will be set to 0. As each record in D contributes exactly once for each of these parameters, this step may be performed in $\mathcal{O}(|D|)$ time.

6 Shift Strategy

6.1 Shift Strategy

Agents and models in Section 5 tackle the problem of determining a policy function, but do not decide on any starting time for the shifts throughout the week. Although it may be sensible to have a different shift selection strategy for each agent, we decided that all implemented agents will share a unified shift time strategy.

A natural way to determine the ideal shift times is to simulate an agent across different starting times. Consequently, the shifts that generate the most earnings will be chosen as the ideal shift times. For a given agent, we simulate it by sampling trips from the entire historical dataset. However, one has to be careful not to overestimate the number of trips. Given a state s, the agent

should get a trip with a probability of

$$\hat{P}(s) = \min\left\{1, \frac{|D^{(s)}|}{\#\{\text{occurrences of time } t_s \text{within dataset timeframe}\}}\right\},$$

where $D^{(s)} = \{d \in D : d \text{ begins in state } s\}$. The steps to simulate one shift carried out by an agent are described in Figure 6, which requires a policy function π and a starting location l_{start} as input. We then arrive at an estimate of the expected earnings made by the agent by taking the average earnings across repeated simulations.

```
SIMULATE-AGENT(π, l_start, t_start, t_shift, P, D)
 1  earnings ← 0
 2  l_curr ← l_start
 3  t_curr ← t_start
 4  t_end ← t_start + t_shift − 1
 5  t_passeger ← 0
 6
 7  while t_curr ≤ t_end and t_passenger < 600
 8  do s ← (l_curr, t_curr)
 9     l_next, z ← π(s)
10     if z is 1 and RANDOM-REAL([0, 1]) < P(l_curr, t_curr)
11       then trip ← RANDOM-CHOICE(D^(s))
12            earnings ← earnings + trip.earnings()
13            l_curr ← trip.dropoff_location()
14            t_curr ← t_curr + trip.duration()
15            t_passenger ← t_passenger + trip.duration()
16       else l_curr ← l_next
17            t_curr ← t_curr + 1
18
19  return earnings
```

Figure 6: Simulating the earnings of an agent using the entire historical dataset.

One of the disadvantages of obtaining expected earnings via simulations is the presence of noise due to randomness. As the minimum number of simulations required for reliable estimates is unknown, we seek to instead estimate the expected earnings deterministically. This can be achieved via dynamic programming, using a very similar algorithm to that of Section 5.4.1.

In order to formulate this algorithm, we utilize Equation (1) again, where π is the policy function of the given agent. In this case, overfitting is not as much of a concern as we require only one estimate of the expected earnings. Therefore, instead of reducing the parameter space, we simply process the entire dataset linearly to implement a dynamic programming algorithm in $\mathcal{O}(\max\{|D|, |S|\})$ time. The algorithm will then return the expected earnings made by the given agent over a shift. We name this algorithm FEED (Finding Expected Earnings Deterministically), and describe its details in Figure 7. It can be interpreted as a deterministic variant of obtaining expected earnings via agent simulation, and hence also requires a policy function π and a starting location l_{start} as input.

One disadvantage of this method is that it is an estimation based on the simplified game, which is required for the derivation of Equation (1). To alleviate this issue, one can replace the maximum shift duration t_{shift} of 12 hours with an *effective shift duration*, which is the expected shift duration of the given agent. The effective shift duration is most likely to be shorter than 12 hours as a shift ends when the taxi driver has transported passengers for 10 hours during that shift. This effective shift duration can be estimated by finding the average shift duration across simulations of the given agent.

7 Development

Having described the formulations of our agents and models, we now proceed to specify how we have developed instances of these agents. Prior to running the FEED algorithm, we select a fixed starting time for the shifts. We infer from our previous investigations that the optimal starting times should be around 12pm. This shift time is used to run our simulations for the initial comparisons.

```
FEED(π, l_start, L, t_start, t_shift, P, D)
 1  V ← Array() of dimensions |L| × |T|
 2  counts ← Array() of dimensions |L| × |T|
 3  V ← 0
 4  counts ← 0
 5  t_end ← t_start + t_shift − 1
 6
 7  for t ← t_end to t_start
 8  do for l in L
 9     do for trip in {trip ∈ D : trip.pickup_location() is l and trip.pickup_time() is t}
10        do t_endtrip ← t + trip.duration()
11           V[l][t] ← V[l][t] + trip.earnings()
12           if t_endtrip ≤ t_end
13             then V[l][t] ← V[l][t] + V[trip.dropoff_location()][t_endtrip]
14           counts[l][t] ← counts[l][t] + 1
15
16        l_next, z ← π(l, t)
17        no_trip ← V[l_next][t + 1] if t < t_end else 0
18        if z is 1 and counts[l][t] > 0
19          then V[l][t] ← no_trip + P(s) × (V[l][t]/counts[l][t] − no_trip)
20          else V[l][t] ← no_trip
21
22 return V[l_start][t_start]
```

Figure 7: Estimating the expected earnings of an agent without explicit simulations.

7.1 Ranking Locations

We first rank the locations by the total number of historical trips in a given location. This is because a location with a high taxi demand will most likely correspond to more earnings made by the taxi driver. This ranking is then output to a "book", as introduced in Section 5.3.

Apart from the Markov Model, all other agents proposed do not include the logic of choosing a starting location. We bridge this gap by setting the starting location to be the most highly ranked cell, regardless of the shift. The Markov Model is an exception, because its dynamic programming algorithm is capable of finding starting locations depending on the shift time.

The most highly ranked cell turns out to be 25:68, which is Pennsylvania Station. This is a highly interpretable result, as Pennsylvania Station is the busiest intercity railroad station in the Western Hemisphere[1][8].

7.2 Optimizing Greedy Behaviour

Here, we define the goal states for the Pseudo Greedy Agent to be an estimated range of cells. Since this is merely an initial approach, we selected some cells from Manhattan, LaGuardia Airport and JFK Airport by visual inspection to be the goal states (see Figure 8). These physical locations historically generate profitable trips, as inferred from our previous work.

The initial simulations (Figure 9) between the Pseudo Greedy Agent and Random Walk Agent revealed an average earning of ≈$2750, a significant increase to the Random baseline which had average earnings of ≈$1600. Additionally, the results of the Pseudo Greedy Agent verify the claims from previous analyses, which claimed a basic strategy of moving towards Manhattan or the Airports would be considerably more profitable than staying in the suburbs.

Because the Pseudo Greedy was able to outperform the Random Walk Agent as expected, we can look into additional optimizations and heuristic improvements. Additionally, the Pseudo Greedy Agent can now act as a baseline against the more Advanced Greedy Agent for future simulations.

The Advanced Greedy Agent inherently possesses an improved behaviour when in the goal states. Additionally, we also refine the goal states' boundaries (i.e. around Manhattan, LaGuardia Airport and JFK Airport). We implemented this by defining goal states to be the locations which had more than 600,000 historical taxi trips from July 2015 to June 2017 (see Figure 11). We found the threshold of 600,000 by trial and error, as changing the threshold significantly led to either too many or too few goal states chosen.

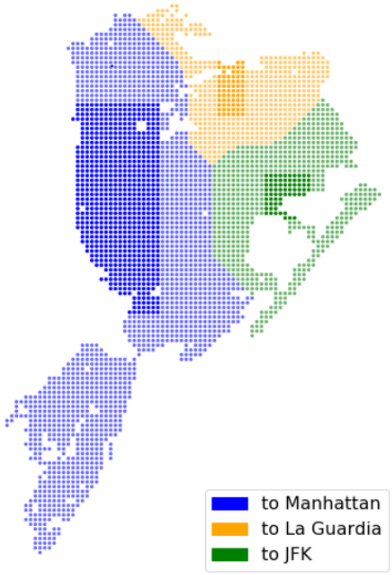

Figure 8: Goal states used by the Pseudo Greedy Agent.

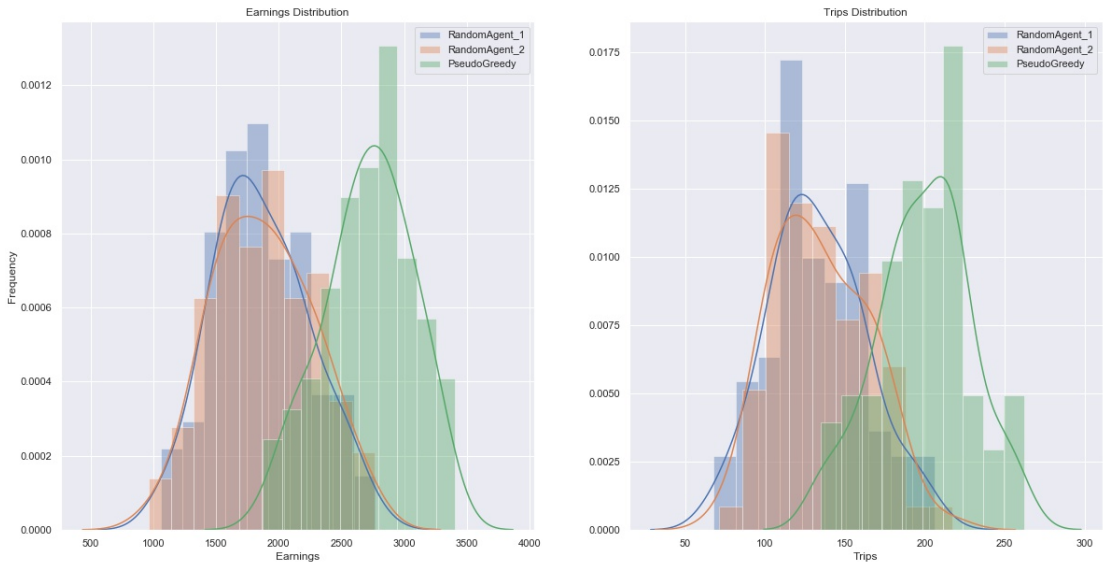

Figure 9: Pseudo Greedy vs Random Walk performance.

Specifically,

- The Manhattan area was further reduced to just Lower Manhattan.
- The approximate location of JFK Airport is now explicitly 2 cells which define the pickup zones in the Airport.
- The approximate location of LaGuardia Airport is now a 5 cell boundary which define the pickup zones and outskirts of the Airport.

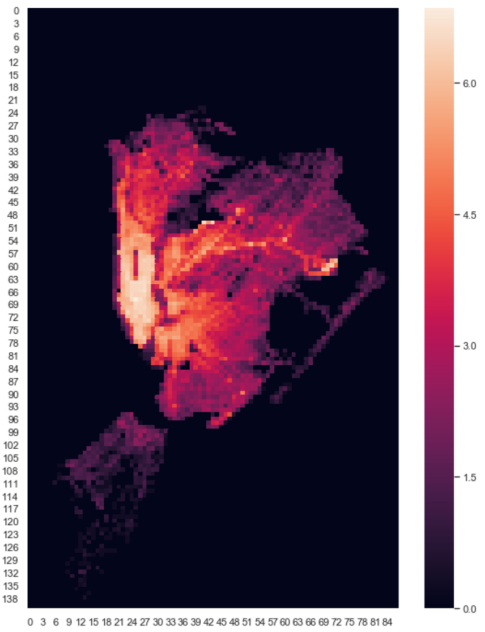

Figure 10: Total Trip Counts (in $\log_{10}$) for each cell.

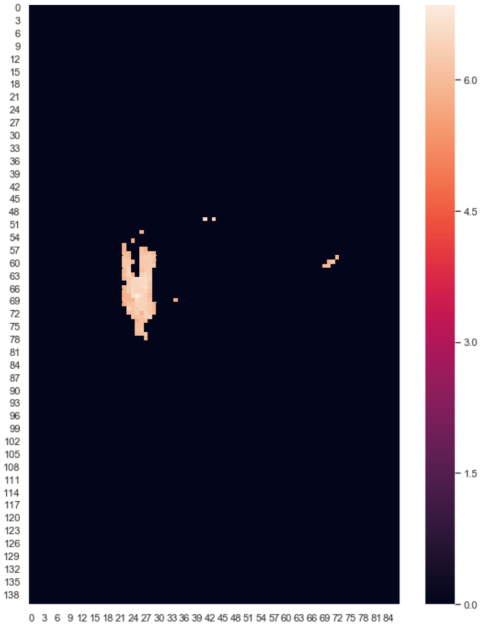

Figure 11: Total Trip Counts (in $\log_{10}$) for refined goal states.

7.3 Comparison of Greedy Agents

Between the two variants of Greedy Agents, the Advanced Greedy Agent successfully annihilates the Pseudo Greedy Agent with respect to the earnings between trips. Figure 12 shows the earnings accrued by each agent, where it can be observed that the best-case earnings of the Pseudo Greedy Agent is approximately the worst-case earnings of the Advanced Greedy Agent.

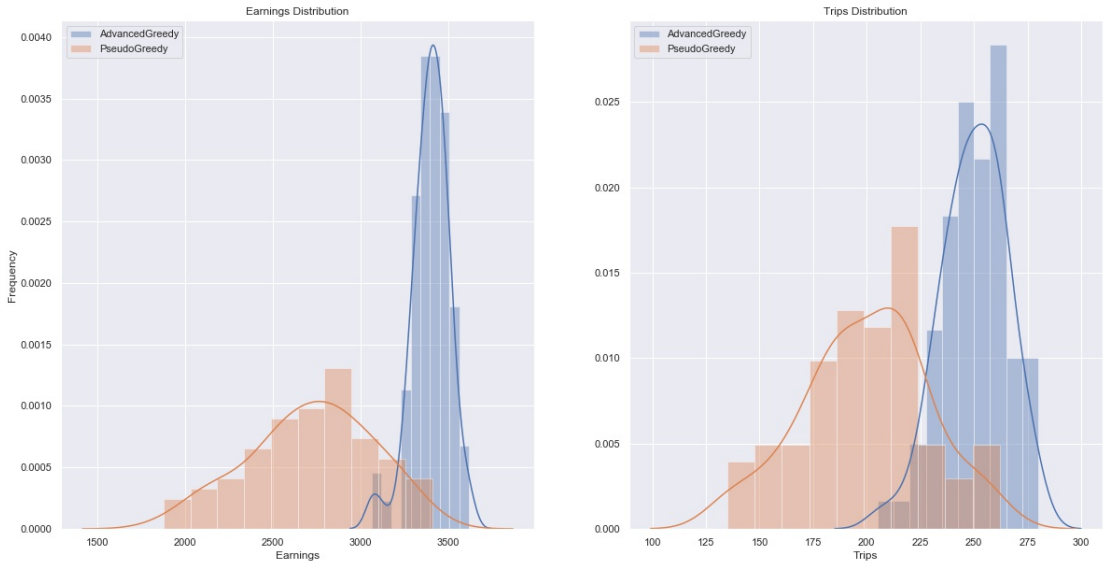

Figure 12: Pseudo Greedy vs Advanced Greedy performance.

7.4 Refining Starting Times

We have observed that the Advanced Greedy Agent exhibits the best performance out of all agents developed so far. Hence, we use the agent Advanced Greedy Agent as the input to the FEED algorithm to find the expected earnings for different starting times. A plot of expected earnings across different starting times (conditioned by day of the week) is shown in Figure 13.

Day of Week	Best Morning Shift Time	Revenue	Best Afternoon Shift Time	Revenue
Monday	05:30	$665.17	13:50	$661.20
Tuesday	05:20	$636.88	14:40	$638.47
Wednesday	05:30	$635.23	15:20	$630.52
Thursday	05:10	$634.91	16:10	$626.78
Friday	04:50	$644.49	17:00	$631.23
Saturday	05:40	$695.18	15:30	$631.56
Sunday	05:40	$736.10	N/A	N/A

Table 1: Maximal Expected Earnings of the Advanced Greedy Agent across the week

Based on this Section 7.4, we found out that starting before the morning rush hour will consistently lead to a good performance for almost every day, with the only exception being Tuesday. The precise starting times used by the agents are listed in Table 1. By selecting only the ideal morning shift times, the agents are able to satisfy the constraint of having at least 8 hours of rest between shifts.

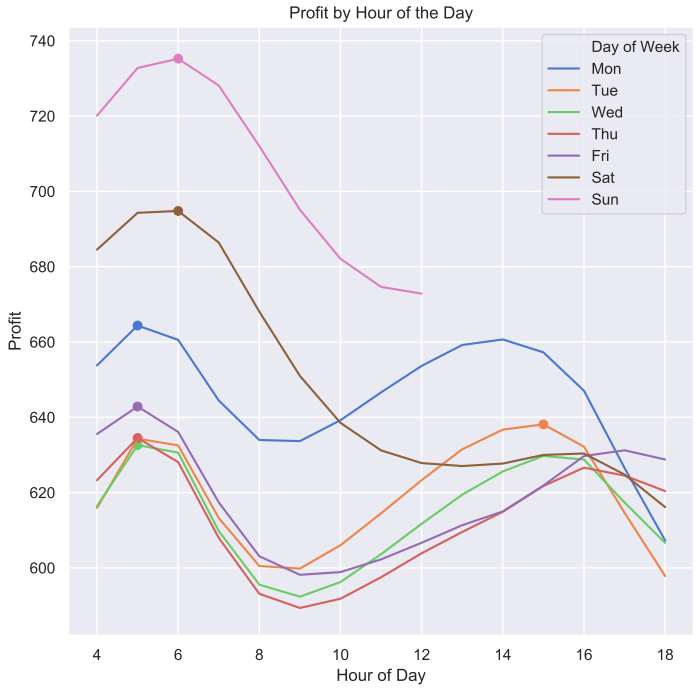

Figure 13: Expected shift earnings of Advanced Greedy Agent computed using FEED algorithm, given starting times from 4am to 6pm. The line for Sunday (pink) is discontinued after 12pm as a full shift of 12 hours will not be completed (week ends at Sunday 11:59pm).

8 Evaluation

8.1 Comparison Of All Agents

In this section, we will discuss and evaluate each agent and their performance against each other.

Upon inspecting Figure 14, while the trip count of the Advanced Greedy Agent matches the merit of the Markov Model, our final model performs significantly better compared to all previous agents with respect to earnings, where it is able to beat all other agents by a large margin. In addition to the expected performance of the agents, the Markov Model also performs consistently (small variance) well compared to most of other agents, rendering it both high performing and robust across different scenarios. However, it is worth noting that the distribution of earning of the Markov Model does have a higher variance compared to the Advanced Greedy Agent.

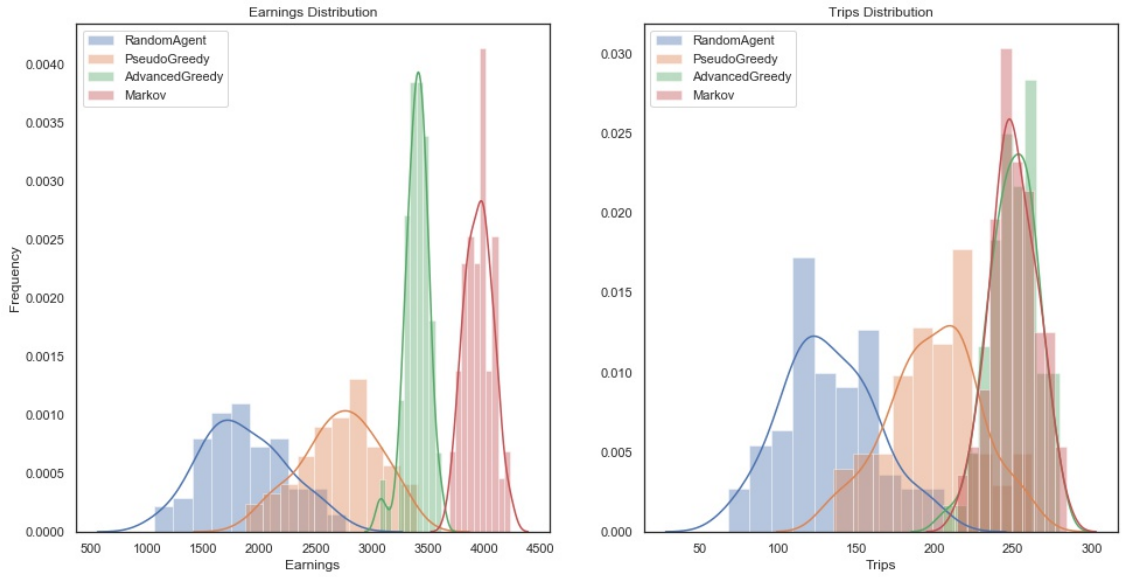

Figure 14: Agent to Agent Comparison.

8.2 Interpreting Decision of Agents

After evaluating the high level distribution of performance of the Agents, the following analysis was done to explore what decision the agent is making and explain the reasoning behind it.

To understand the decision making process the agents make during a game, we animated the agents' log files and visualized the exact decisions onto a map of a single instance (Figure 15). Specifically, the visualisation of the decisions are at Saturday 10pm for both Advanced Greedy Agent and Markov Model. Despite the initial impression that there is a significant difference between the two agents, we discovered that they actually perform relatively similar.

Notably, the most apparent differences between the two models are the sink spots represented by the red boxes. Here, if a taxi driver goes into the specified region, they will opt into waiting for a trip, rather than moving to an adjacent cell. From observations, the Advanced Greedy Agent tends to sink at the Hunterspoint Avenue (located in the outskirts of central Manhattan) whereas the Markov Model makes a more reasonable decision - choosing to navigate towards the train station before the city center, which has been shown toe be much more profitable due to its high demand. In addition, at Saturday 10pm, the demand of travelling from a recreational area such as the locations next to Train Station will are high, making it more preferable to the Station than Airports.

Another insightful, but not obvious trend is the variation of tendency moving to Airports for the two agents. These are highlighted with blue rectangle, where, the Markov Model will prefer moving to Airports even though it is much closer to Manhattan. Because the airports provide more profitable trips with less demand, the Markov Model identifies the Airports to be its best bet.

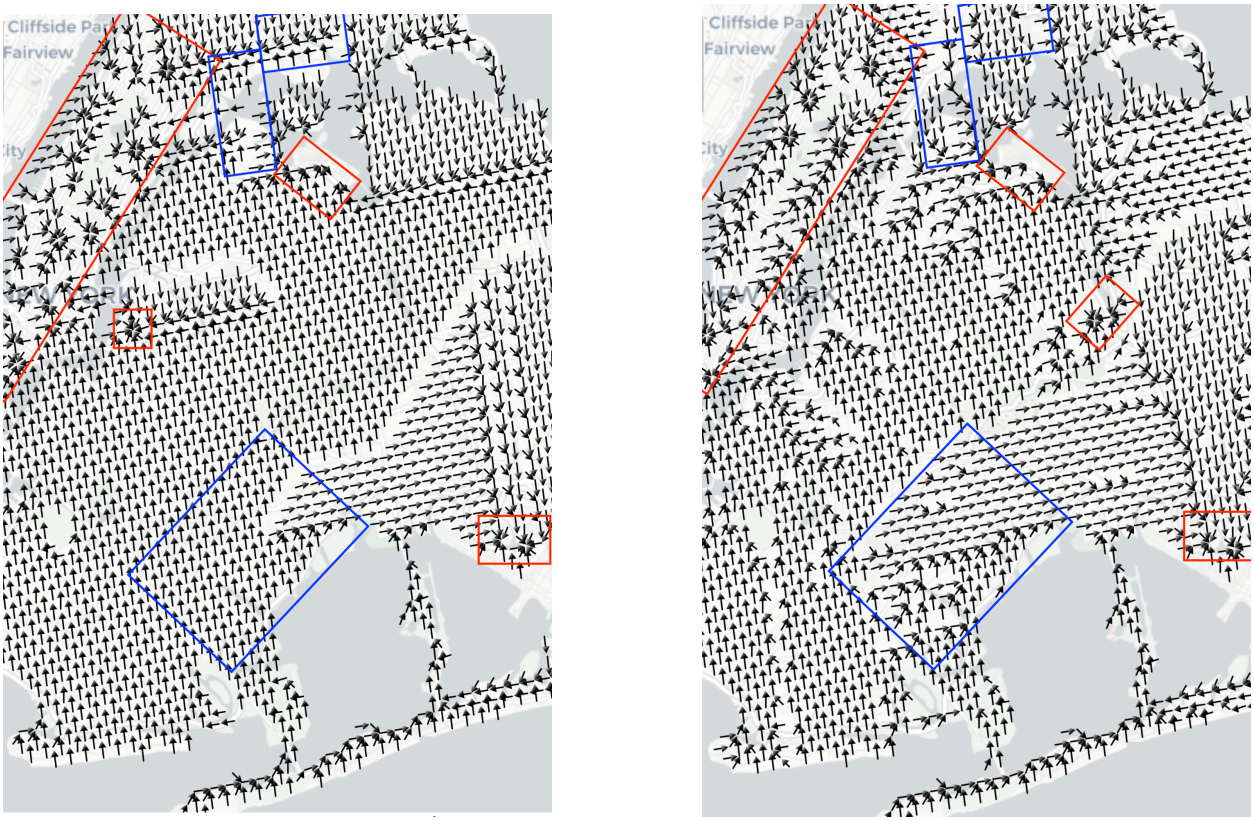

Figure 15: Visualised decision Book of the Advanced Greedy Agent (left) and the Markov Model (right) at 10pm on a Saturday. Each arrow indicates the direction the agent will move to given that it is not taking a passenger at the moment. The red rectangle denotes the sink region in the map, whilst the blue rectangle highlights the major differences between the two agents' decisions. The Manhattan Region is on top right side while the two Airports are indicated by the red boxes on the right hand side of the Advanced Greedy Agent map.

However, the trip counts heuristic used in the Advanced Greedy Agent will guide it to move downtown because there is more demand in Manhattan.

From the analysis above, it is clear that whilst Greedy Agent prefers higher trip counts, the combined evaluation of Markov Model over both trip counts and income makes it a better Agent competing in New York.

9 Discussion

As expected, the Pseudo Greedy Agent performed poorly when compared to its more informed counterpart, the Advanced Greedy Agent. This can be attributed due to its logic when the agent is located in the goal states, namely the Manhattan and Airports, where the Pseudo Greedy Agent would degenerate to a Random Walk Agent. In contrast, the Advanced Greedy Agent would continue to make locally optimal movements based on its "book".

An interesting observation is that the Advanced Greedy Agent is conceivably worse than the Markov Model despite a greater total trip count. This result can be attributed to the different internal decision making, since the Advanced Greedy Agent can be thought of as a Markov Model which only takes into account the next action instead of all the possible consequential actions. Moreover, the Advanced Greedy Agent is always for hire, and hence may pick up trips with low earnings per unit of time.

Both the Advanced Greedy Agent and Markov Model were simulated alongside 13 other agents from different teams. During these online simulations, our Advanced Greedy and Markov agents were able to consistently rank in the second place. The variation in performance for our models between local and online simulations were minimal, hence we claim that our model is robust and applicable to a wide variety of scenarios outside that of the provided train data.

9.1 Future Work

When doing basic statistical analysis of the training data, we noticed a significant change in the distribution of trip counts within the top cells, specifically with July 2016 (refer to Figure 16). This is because GPS coordinates were depreciated post July 2016 and were converted into zones to reduce accuracy of data. So, the provided dataset is suspected to have this problem and was later confirmed by the data provider (Dr Chris Culnane).

There are some additional improvements that can be made with the current implementations. Especially with the Markov Model, there is a possibility to train multiple Markov Models with respect to the time (e.g by binning the data with respect to seasons, months or year). This was not possible with the current implementation as the time required to train and re-estimate parameters was deemed unfeasible.

However, by combining the cells in NYC into larger zones, we can optimize each individual Markov Model by training on each zone. Furthermore, trend models can be built by considering the relative increases or reductions of taxi trips with respect to these different zones. If implemented correctly, these models can be combined with the existing algorithms to provide more effective online learning algorithm.

Furthermore, more traditional Artificial Intelligence based approaches such as Reinforcement Learning can be attempted - refer to Appendix for the specific details. It is expected that this method will yield a result better than the Pseudo Agent from initial training stages as well as provide a more general purpose and robust model.

10 Conclusion

In the end, the Advanced Greedy was considered to be the best compromise between performance and time complexity, although the Markov Model was used for final submission. This was because of the given constraints which allows for a reasonable space complexity, allowing the Markov Model matrices to be included. However, it is arguable from an overall standpoint that the instantaneous decisions from the Advanced Greedy Agent to be more applicable for real-time recommendations.

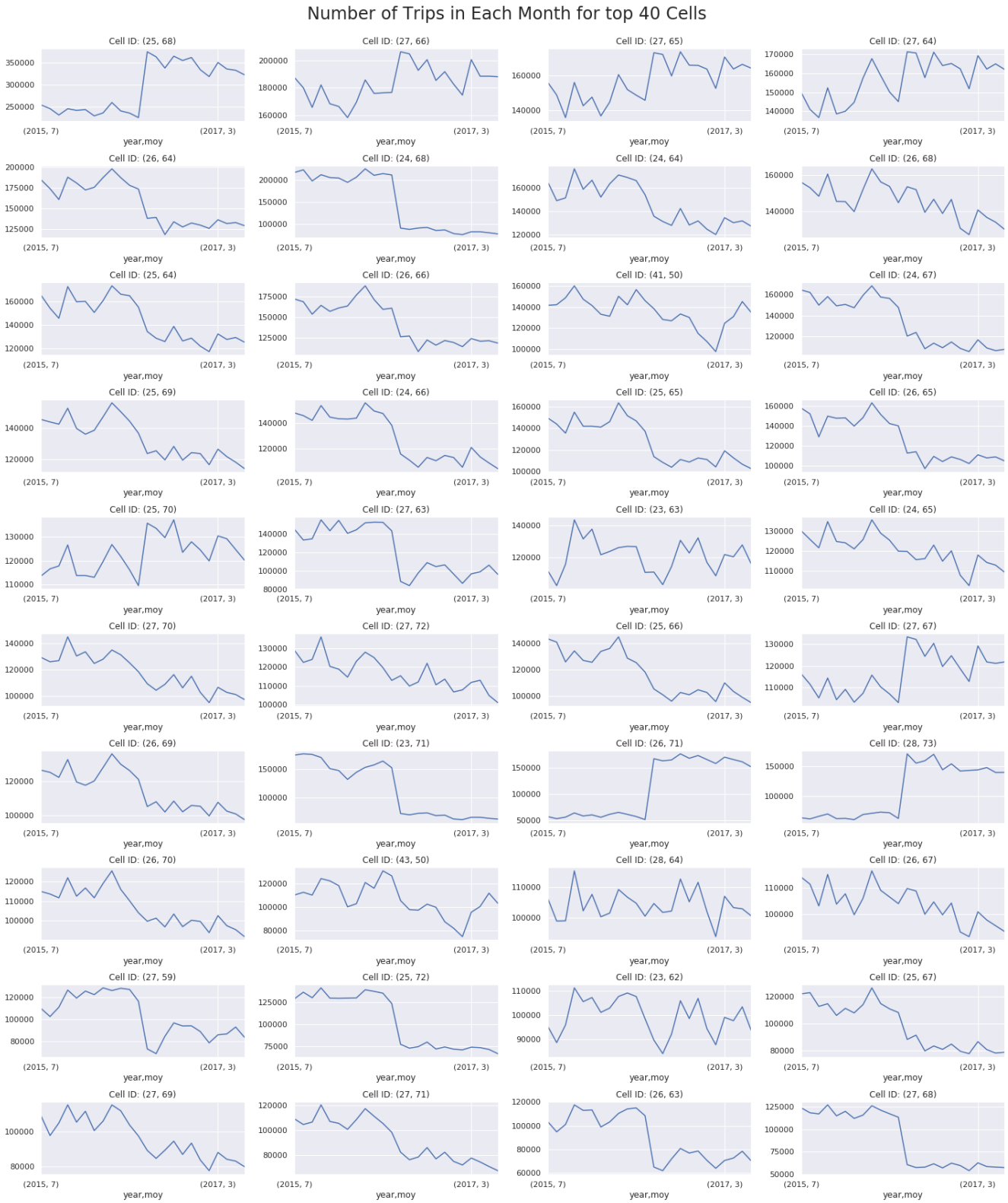

Figure 16: As we can see, in quite a lot of the cells, there is either a shape increase or decrease of trip counts at July 2016.

References

[1] The most awful transit center in america could get unimaginably worse - bloomberg. https://www.bloomberg.com/news/features/2018-01-10/the-most-awful-transit-center-in-america-could-get-unimaginably-worse. (Accessed on 24/10/2019).

[2] Daryl Herzmann Akrherz@iastate.edu˙Iem :: Nws coop raw observations download. https://mesonet.agron.iastate.edu/request/coop/obs-fe.phtml?network=NY_COOP.

[3] Apache Software Foundation. Welcome to the apache software foundation!

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Python Software Foundation. multiprocessing - process-based parallelism.

[6] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[7] Ernest W. Kent. *The Brains of Men and Machines*. McGraw-Hill, Inc., New York, NY, USA, 1981.

[8] Michael Kimmelman. When the old penn station was demolished, new york lost its faith - the new york times. https://www.nytimes.com/2019/04/24/nyregion/old-penn-station-pictures-new-york.html. (Accessed on 24/10/2019).

[9] P. Li, Sandjai Bhulai, and J.T. van Essen. Optimization of the revenue of the new york city taxi service using markov decision processes. In Sandjai Bhulai and Dimitris Kardaras, editors, *6th International Conference on Data Analytics, Barcelona (Spain), November 12-16*, pages 47–52. IARIA, 2017.

[10] NYC Taxi & Limousine Commission. Taxi fare - tlc. https://www1.nyc.gov/site/tlc/passengers/taxi-fare.page. Accessed on 23/10/2019.

[11] NYC Taxi & Limousine Commission. TLC Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2015. Accessed: 25/09/2019.

[12] Police Department (NYPD). Nypd motor vehicle collisions - crashes | nyc open data. https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions-Crashes/h9gi-nx95. Accessed on 23/10/2019.

[13] Department of Design and Construction (DDC). Active projects - infrastructure: Nyc open data. https://data.cityofnewyork.us/Housing-Development/Active-Projects-Infrastructure/rukc-mmqu, Jul 2018.

[14] Office of Emergency Management (OEM). Emergency response incidents | nyc open data. https://data.cityofnewyork.us/Public-Safety/Emergency-Response-Incidents/pasr-j7fb. Accessed on 23/10/2019.

[15] Huigui Rong, Xun Zhou, Chang Yang, Zubair Shafiq, and Alex Liu. The rich and the poor: A markov decision process approach to optimizing taxi driver revenue efficiency. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2329–2334, New York, NY, USA, 2016. ACM.

[16] Jeff Sauro and James R. Lewis. Average task times in usability tests: What to report? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2347–2350, New York, NY, USA, 2010. ACM.

[17] Wesm. wesm/feather, Sep 2019.

[18] X. Zhan, X. Qian, and S. V. Ukkusuri. A graph-based approach to measuring the efficiency of an urban taxi service system. *IEEE Transactions on Intelligent Transportation Systems*, 17(9):2479–2489, Sep. 2016.

[19] C. Zhu and B. Prabhakar. Reducing inefficiencies in taxi systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 6301–6306, Dec 2017.

Appendices

A Other Agents

A.1 Reinforcement Learning

Reinforcement learning is a common approach for the problem define here, it optimizes the model based on a environment(observation)-action-reward model, that is, the agent A with policy π_θ parameterized by θ will receive an observation given by environment at state s, make an action based on the observation, and receive a reward based on the action taken, then the environment progresses to the next state s'.

$$s \xrightarrow{a\sim\pi_\theta(.|s)} s' \xrightarrow{a\sim\pi_\theta(.|s')} s'' \xrightarrow{a\sim\pi_\theta(.|s'')} \dots$$

where

a is the action sampled from π at state s.

s, s', s'' are specific states resulted from taking the action a

Then the agent will improve the policy π based on the reward received, if it is positive, the agent will improve the policy such that it has less chance to make the same decision given the same observation, and vice versa.

In this specific work, we define the following simplified framework for the algorithm with a differentiable policy π_θ:

POLICY ITERATION()
1 $Initialize\ a\ random\ policy\ \pi_\theta$
2 **while** $not\ terminated$
3 **do**
4 $V^\pi(s) = r(s, \pi_\theta(s)) + \gamma \sum_{s'\in R} M(s'; s, \pi_\theta(s))V^\pi(s')$
5 $Compute\ \partial \frac{V^{\pi_\theta}(s)}{\partial\theta}$
6 $\theta' = \theta + \alpha\partial \frac{V^{\pi_\theta}(s)}{\partial\theta}$[3]
7 $\pi = \pi_{\theta'}$
8 **return** π_θ

where

γ is the discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.

M is thee transition matrix denoting the chance to transit to state s' given action a drawn from $\pi_\theta(s)$

α is the optimization hyperparameter controlling the speed of optimization

$V^\pi(s)$ stands for the value function for taking adopting policy π at state s This definition is not rigorous, but just to demonstrate the general guideline adopted when optimizing our agent. More specifically, three different agents are trialed and tested.

This will be an effective way to generalize the problem even further by assuming no explicit structure in our problem. The basic training environment is laid out using OpenAI Gym[4], and the architecture and training is done through Stable-baselines [6].

We implemented a soft environment for the agent. That is, the agent will not be take out of the game when invalid move is made, instead, the agent will be penalized heavily after making such moves. This is done because the training environment requires a homogeneous action space for all states the agent might have. And having invalid moves for the agent will disable the training process completely.

However, to fine tune and train such agent is time consuming, at current stage it is infeasible with our budget of time. So this idea was not carried further to make an competitive agent.