Optimizing Autonomous Taxi Size With Reinforcement Learning

Darren Mulholland

Abstract

AI researchers have devoted considerable effort to determining optimal strategies for deploying, dispatching, and rebalancing fleets of autonomous taxis, a widely-anticipated early use case for autonomous vehicles with profound implications for established cultural norms and urban design.

An important related question which has received relatively little attention thus far is the optimal passenger capacity for individual vehicles within these autonomous taxi fleets. Current taxi size is in part the product of historical technological constraints which are unlikely to apply to the driverless, electrically-powered vehicles of the future, making radically new designs possible.

In this thesis I use q-learning, a form of reinforcement learning, to investigate this question of optimal autonomous vehicle size, training a simulated fleet of autonomous taxis using real-world taxi data from New York City. Q-learning is a foundational algorithm in reinforcement learning and provides an established theoretical framework to begin applying reinforcement learning techniques to this novel problem domain.

My q-learning model predicts that the optimal seating capacity for these simulated autonomous taxis is seven passengers. I discuss the plausibility and real-world implications of this prediction and consider alternative reinforcement learning strategies for optimizing autonomous taxi size. I develop and apply a Monte Carlo sample learning model to my simulated taxi fleet which both confirms the predictions of my q-learning model and at the same time raises interesting questions about its optimality.

Acknowledgements

Thank you to all the people who gave their time free of charge to create the open source software which made this project possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter I describe the motivation behind my project, list its aims and objectives, and outline the structure of the report.

1.1 Project Motivation

The widely-anticipated near-future arrival of driverless cars on our streets has fueled a wealth of academic research on related topics in the field of AI. Particular interest has been paid to the subject of autonomous taxis, a technology which has the potential to revolutionize urban transportation with profound long-term implications for established cultural norms and urban design [1].

One question which has so far been largely ignored by AI researchers is the optimal passenger-capacity for individual vehicles within these autonomous taxi fleets. Current car size is, in part, the product of historical technological constraints (such as the requirement to accommodate a

driver and an internal combustion engine and its associated drive train) which will not apply to the autonomous, electrically-powered vehicles of the future. Radically new car designs will certainly be possible in the near future; whether they will prove practical or desirable is an open question.

One factor which bears heavily on this question is the potential impact of ride-sharing on optimal vehicle size. Urban taxi services have traditionally operated on an exclusive basis, carrying one passenger or passenger group at a time, with most of their seating capacity going unused. (As we shall see, more than 70% of taxi trips in our New York dataset consist of passengers travelling alone.) The recent revolution in mobile communication technology combined with sophisticated automated dispatching systems has made ride-sharing (where multiple passengers with compatible itinaries share a single taxi simultaneously) a viable alternative [2]. This raises the prospect that in the near future fleets of autonomous taxis could operate more like small, free-floating, hailable buses than traditional taxis, servicing a continuous stream of ride-sharing passengers as they plot and replot efficient dropoff routes through their city's streets [3].

This potential for ride-sharing would seem to make larger vehicles sizes more attractive — if the additional capacity can be efficiently filled it could result not just in cheaper fares for commuters but simultaneously in higher profits for the taxi operators, a potential win-win outcome for all concerned [4]. But ride-sharing is not the only novel factor we need to take into account. The most significant operating cost for traditional taxis is the labour-cost of the ever-present driver. Driverless technology removes this factor from the equation entirely, significantly reducing the economies of scale represented by larger vehicle sizes, potentially making *smaller* taxis a more attractive prospect. If it turns out that empty or wasted capacity is the most significant factor in autonomous taxi prof-

itability then single-seat taxis, which by definition have no spare capacity to waste, may in fact be the most efficient design.

Clearly, optimal taxi size is a complicated question with no obvious or simple answer.

In this project I have investigated this question of optimal autonomous taxi size using reinforcement learning, a branch of AI which focusses on goal-seeking agents attempting to maximize their long-term rewards by interacting with their environment and learning from the consequences of their actions [5].

Q-learning, the reinforcement learning technique I apply to my simulated taxi fleet, was originally developed as a model for understanding animal intelligence [6] and has a rich history of being applied to the study of predator-prey interactions [7][8]. A fleet of autonomous, independent taxis can be viewed, on this model, as a population of predator-agents roaming their city streets in search of passengers to 'prey' on (or more prosaically, as gentle herbivors harvesting a geographically dispersed resource).

As with biological predators, the question of optimal vehicle size involves a fundamental trade-off between capability and efficiency, and there is no straightforward a priori way to determine an optimal solution in advance of empirical experimentation. Ideally we would, and no doubt at some point will, investigate this question by conducting real-world experiments with physical vehicles; in the meantime the best answers we can hope for will come from simulations and here the novel approach of applying reinforcement learning can help us at least provisionally answer an important question with practical implications for real-world fleet deployments.

1.2 Project Aims and Objectives

- To build a simulation engine capable of simulating a realistic fleet of autonomous taxis.
- To assemble a dataset of real-world trip data to use as input for the simulation.
- To design a reinforcement-learning training protocol to train the fleet of simulated taxis.
- To conduct the experiment, analyse its results, and detemine a predicted optimal size for the simulated taxis.

1.3 Report Structure

In Chapter 2 I outline the q-learning algorithm, review the state of the art, and summarize recent research related to the question of autonomous taxi size.

In Chapter 3 I describe the New York city taxi-demand dataset and training protocol I used to train my reinforcement-learning models.

In Chapter 4 I describe the taxi simulation engine I built to implement the training algorithm and discuss the assumptions and trade-offs inherent in its design.

In Chapter 5 I describe and evaluate the results of my investigation and consider alternative approaches to the question of determining optimal taxi size.

Chapter 2

Background and Related Work

In this chapter I describe the q-learning algorithm and review related work in the field.

2.1 Q-learning

Standard yellow taxis in New York city, as in many other cities around the world, are restricted by law to carrying 4 passengers at a time [9], but this limit is in part a product of historical technological constraints which may not apply to future autonomous vehicles. In this project I have attempted to investigate whether the sizes of individual vehicles in a hypothetical fleet of autonomous New York taxis can be optimized using **q-learning**, a type of **reinforcement learning**.

Reinforcement learning is itself one of the three main branches of machine learning, alongside supervised learning and unsupervised learning [10]. Reinforcement learning is a broad field of study but in general

concerns itself with goal-seeking agents attempting to maximize their cumulative reward by interacting with their environments and learning from the effects of their actions [11].

2.1.1 Background

Q-learning is a simple, model-free reinforcement learning algorithm first developed by Christopher Watkins in 1989 as a computational model for studying animal learning [6]. Just as animals explore their environment and learn from the consequences of their actions, a q-learning agent will explore its world of possible states and incrementally improve its evaluation of the 'quality' of taking particular actions in particular states as it learns from its experiences. (The 'Q' in the name q-learning comes from this idea of estimating the quality of actions.)

Q-learning can be described as **primitive** or **model-free** in the sense that the learning agent has no internal model of its world — it simply takes actions and observes their consequences [6]. Although simple, the algorithm can be proved to **converge**; that is, given enough time to explore its environment, a q-learning agent will eventually be able to determine its optimal policy for any state, the policy that maximizes its total expected long-term reward [12].

2.1.2 Algorithm

Let S be the set of possible states an agent can be in and A be the set of actions the agent can take in these states. The goal of the q-learning algorithm is to populate a look-up table, $Q(S, A)$, which can be thought of as a matrix in which each row represents a possible state and each

column represents a potential action the agent can take in that state.

Each entry in this table, $Q(s, a)$, is the agent's current estimate of the expected value of taking action a in state s, that is, the cumulative discounted reward for doing a in state s and then following the optimal policy. This q-value incorporates both the immediate reward the agent receives for choosing the action and the agent's estimation of the discounted future value of being in state s', the state it will arrive at after taking action a in state s.

Initially, we populate the Q-table with a set of arbitrary values, typically zeros. We then update the table iteratively as the agent learns from its experiences, where each **experience** is a tuple of the form:

$$< s, a, r, s' >$$

That is, the agent was in state s, chose action a, received reward r, and transitioned into state s'. On the basis of this experience we update the relevant q-value, $Q(s, a)$, according to the temporal difference equation:

$$Q(s, a) \longleftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(s', a') \right) \tag{2.1}$$

In simple terms, we replace the old q-value with a weighted average of the old value and the **return**, the new information resulting from the experience. This equation contains a number of new terms we haven't yet defined:

- $\boldsymbol{\alpha}$ is the **learning rate**, a value between 0 and 1 which determines

the agent's eagerness to learn from new experiences at the expense of its cumulative knowledge of past experiences. If $\boldsymbol{\alpha}$ is set to 0 the agent will learn nothing from new experiences and rely solely on its previous knowledge; if $\boldsymbol{\alpha}$ is set to 1 the agent will discount its previous knowledge entirely and rely solely on the return from its most recent experience.

- γ is the **discount rate**, typically a value between 0 and 1, which determines the importance the agent places on future rewards. If γ is set to 0 the agent will be 'myopic', considering only immediate rewards; a value closer to 1 encourages the agent to maximise its long-term payoffs. (If $\gamma \geq 1$ the q-values will diverge, which is generally not desired [13].)

- $\boldsymbol{max\, Q(s', a')}$ is the q-value of the agent's optimal action a' in its new state s'; this represents the agent's estimate of the future value of transitioning from state s to s' as a result of action a. (Much of the power of the q-learning algorithm comes from this deceptively simple idea — that the value of an action should reflect the values of the future actions which that action makes possible.)

Figure 2.1 shows a simple graphical illustration of the q-learning algorithm. Iteration will continue until the agent enters a terminal state or until some other sentinel condition is triggered, depending on the details of its application.

2.1.3 Policy

We have not yet discussed how an agent chooses which action to take in any given state. This is determined by the agent's **policy**, π; the strategy it uses to choose among its available actions.

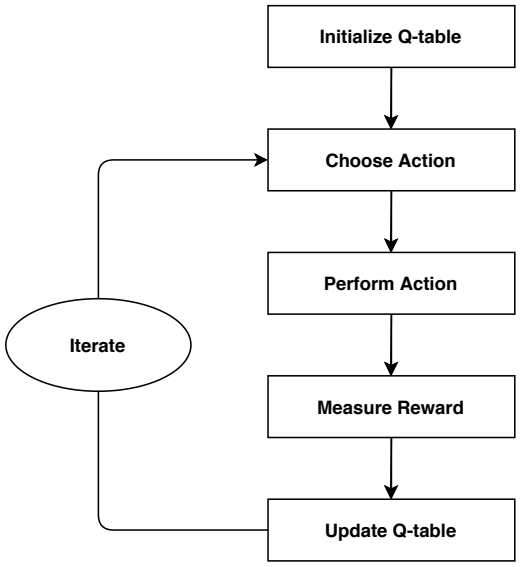


Figure 2.1: *Q-learning algorithm.*

An agent which always chooses its best action (or rather, its current estimate of its best action) is said to be following a **greedy** policy. A greedy policy maximizes the agent's expected payoff in the short-term but prevents it from exploring its state space and learning about the potentially greater rewards of untested actions.

A more useful policy, at least initially, is **epsilon-greedy**, where ε is the probability that an agent will choose to *explore* its environment and $1 - \varepsilon$ is the probability that the agent will instead choose to *exploit* its existing knowledge. An agent following an epsilon-greedy strategy will choose randomly from its available actions with probability ε and will choose its best action with probability $1 - \varepsilon$.

Notably, q-learning is an **off-policy** algorithm, meaning that an agent will learn its optimal policy regardless of the policy it happens to be fol-

lowing [14]. That is, even if an agent chooses randomly at every opportunity, its q-table will still be updated over time to reflect the best action it could have taken in each state. (We will take advantage of this feature of q-learning later by holding ε at 1 for a large number of iterations, forcing our taxis to explore their state spaces.)

2.2 Related Work

The widely anticipated arrival of autonomous vehicles on our streets in the near future has fueled a broad range of academic research on related topics. The research in the field of AI can be divided into two main categories: research on **functional AI**, concerned with the immediate practicalities of autonomous driving such as object detection and collision avoidance, and research on **operational AI**, concerned with higher-level questions such as optimal dispatching and rebalancing strategies for fleets of autonomous vehicles. Our concern here will be with the second category of research.

2.2.1 Definitions

Terminology in this fast-moving field is fluid, so we will begin by clarifying our definitions of a number of important terms.

AV

Autonomous Vehicle. A vehicle which operates independently without a human driver or controller [15][16].

SAV

Shared Autonomous Vehicle. An autonomous taxi [17][18][19][20][21].

MoD

Mobility-on-Demand. This term is used to describe transportation systems in which consumers make short-term use of hired vehicles [15][18][22][23]. It includes taxi services and short-term car, bike, and scooter rentals but not bus or train services which run on fixed routes.

AMoD

Autonomous Mobility-on-Demand. This term is commonly used in the literature to describe taxi services supplied by autonomous vehicles [15][18][23][24].

SAMoD

Shared Autonomous Mobility-on-Demand. An autonomous taxi service which incorporates ride-sharing, enabling multiple passengers with compatible itineraries to travel together simultaneously in a single taxi [18].

DRS

Dynamic Ride-Sharing. The practice of pooling multiple travelers with similar origins, destinations, and departure-times together in the same taxi [25].

2.2.2 Literature Review

Spieser et al. (2015) investigated potential rebalancing strategies for fleets of autonomous vehicles using rental data from existing free-floating car-sharing services operating in Europe and North America [23]. The authors argue that a fleet's rebalancing strategy in the face of uneven geographical demand has a critical impact on its performance metrics both from the consumer's and the fleet operator's perspectives.

Guériau and Dusparic (2018) used agent-based reinforcement learning to investigate decentralized solutions to this question of autonomous fleet rebalancing [18]. In their model independent ride-sharing autonomous vehicles use q-learning to optimise their individual repositioning policies, learning from their experience of historical demand patterns in different geographical regions. They trained their agents using taxi data from New York city and concluded that their decentralized approach could match the performance efficiency of centralized dispatching while accommodating the potential for heterogeneous vehicle fleets with multiple, non-coordinating operators. This study assumed a uniform four-seat capacity for its autonomous vehicles.

Alonso-Mora et al. (2017) also used taxi data from New York city to investigate vehicle routing strategies for autonomous taxis [22]. They developed a constrained optimization algorithm to generate optimal routes for taxis with varying passenger capacities up to a maximum of ten seats. They concluded that existing demand in New York city could be serviced by a fleet of just 2,000 ten-seat ride-sharing taxis, representing a reduction in the fleet size to 15% of its current value.

Dia and Jananshour (2016) conducted a small-scale simulation-based study to investigate the potential impact of shared autonomous vehicles on demand for vehicle ownership and on-street parking in an urban environment, using commuter data from Melbourne, Australia [15]. They concluded that the introduction of shared autonomous vehicles could result in significant reductions in both, though at the expense of a considerable increase in individual VKT (vehicle-kilometers travelled) per vehicle. Vehicles in this study were assumed to have a capacity of between two and four seats.

Bischoff and Maciejewski (2016) used a large-scale simulation of all road traffic in Berlin to investigate the potential for autonomous taxis to com-

pletely replace private car ownership in the city [16]. They focussed on optimizing their dispatching strategy to maximise taxi utilization and passenger throughput and concluded that a fleet of 100,000 autonomous taxis would be sufficient to service demand in the city, with each autonomous taxi replacing ten CDVs or 'conventionally driven vehicles'. This study did not consider ride-sharing as an option and individual vehicle size was not specified.

In a later study, Bischoff and Maciejewski (2017) used the same simulation engine with a real-world taxi data set from Berlin to investigate the potential efficiency of ride-sharing taxi services in the city [20]. They concluded that ride-sharing could reduce overall vehicle kilometers travelled by up to 20% with minimal impact on passenger journey times. Vehicles in this study were assumed to have uniform capacities of 4, 6, or 8 seats and passengers were assumed to travel alone or in pairs. The authors argue that high-capacity vehicles have higher fixed and variable operational costs, making capacities of 8 or greater inefficient as scheduling constraints mean that taxis are unlikely to be able to service more than three requests simultaneously.

Levin et al. (2017) developed a flow-based simulation model to investigate the impact of dynamic ride-sharing on traffic congestion in downtown Austin [19]. They compared scenarios involving private vehicles and SAVs and found that, in the absence of ride-sharing, empty repositioning trips by SAVs could increase traffic congestion and travel times. They concluded that ride-sharing is highly effective at reducing congestion by combining traveler trips, and that travel times can be optimized by keeping the fleet size as small as possible. Vehicles in this study had a uniform capacity of four seats.

2.2.3 Summary

There has been a considerable amount of academic research on potential dispatching, rebalancing, and ride-sharing strategies for fleets of autonomous taxis, but relatively little attention has been given to the question of optimal vehicle size within these fleets. In particular, this paper's application of agent-based reinforcement learning techniques represents a novel approach to an important question with significant practical implications for real-world deployments.

Chapter 3

Design

In this chapter I describe the New York City taxi dataset I used in this project as input for my taxi simulation engine. I also describe the reward metric and training protocol I used to train my reinforcement-learning taxi models.

3.1 New York Taxi Data

The New York City Taxi and Limousine Commission (TLC) is the agency responsible for licensing and regulating New York City's famous yellow (or 'Medallion') taxi cabs, along with a variety of other taxi and limousine services in the city. Since 2009 the TLC has publicly released an annual dataset containing records of every yellow taxi trip in the city, including the pick-up and drop-off locations, the pick-up and drop-off times, and the number of passengers carried [26]. These enormous datasets have been a treasure trove for AI researchers seeking real-world data to model demand for future autonomous taxi services [18] [22].

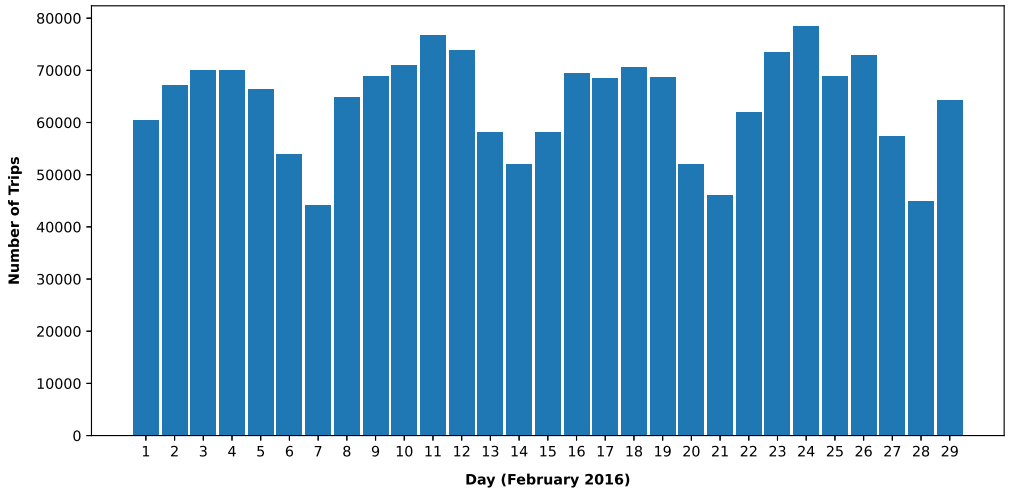


Figure 3.1: *Daily number of trips in the filtered dataset over the month of February 2016.*

For this project I used a subset of this trip data taken from the yellow taxi dataset from 2016 to model demand for a hypothetical fleet of autonomous taxis servicing commuters on the island of Manhattan. (I restricted my analysis to Manhattan as I believe it is likely that this kind of autonomous taxi system will be implemented first in geographically restricted areas where human-driven vehicles can feasibly be excluded. I chose the dataset from 2016 as this was the last year to include GPS coordinates for each trip's pick-up and drop-off locations; in subsequent years only the pick-up and drop-off *zones* are recorded.)

To assemble a representative sample I took the data for one full month (February), excluded any trips with a pick-up or drop-off location outside the island of Manhattan, and extracted four hours worth of requests for each day, taking the interval from 8 a.m. to 12 noon. This gave me approximately 60,000 requests per day, or about 1.85 million requests

overall.

Figure 3.1 shows a plot of the daily number of trips in this filtered dataset. The weekly demand cycle is particularly noticeable with demand being lightest at the weekends and peaking midweek on Wednesdays or Thursdays. (In calibrating the model I later developed to determine the number of taxis required to meet demand I took care to ensure that the fleet size met its minimum service requirements on its *worst* day as averaging performance over one or more weeks could mask unacceptable service levels on peak days.)

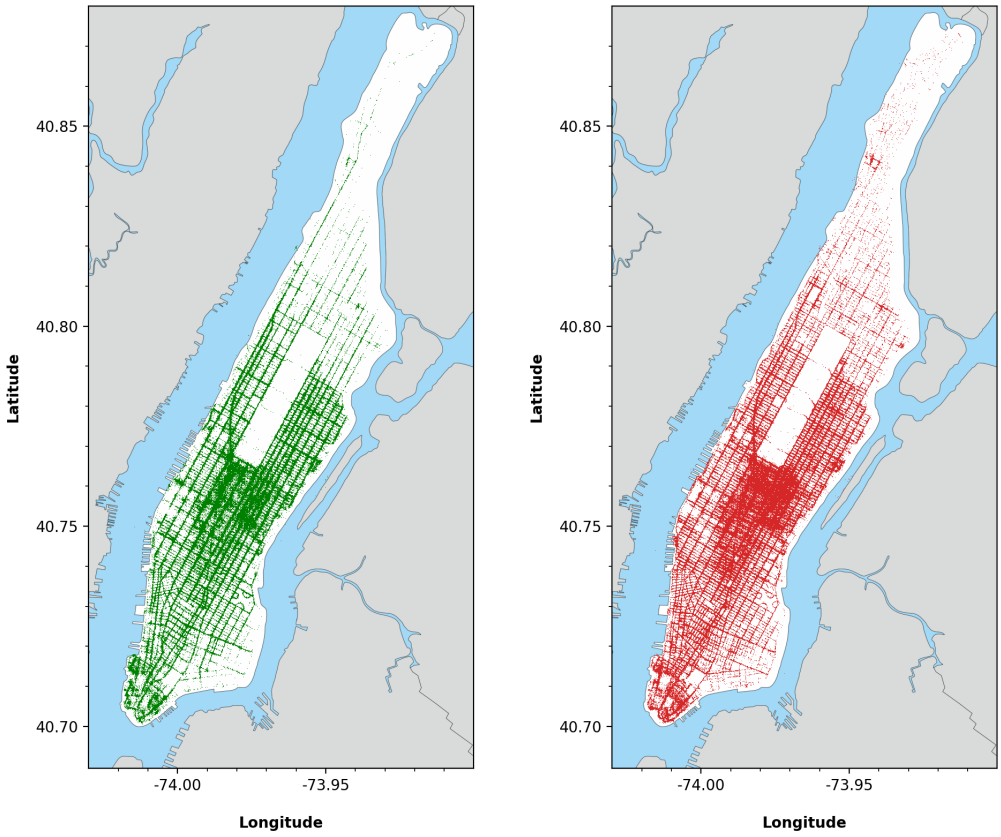
Figure 3.2 shows heatmaps of pick-up and drop-off locations in Manhattan constructed from the filtered dataset for the first week of February 2016. Both pick-ups and drop-offs show strong geographical clustering with particular hotspots at the south end of Central Park and in the financial district around Wall Street. (This clustering would be an important factor in designing a repositioning strategy for the model.)

3.2 Applying Q-learning

Reinforcement learning is not, perhaps, an obvious strategy for optimizing the size of autonomous taxis. Taxis do move around their environments a lot but they don't, in general, learn much from their experiences, and (even worse) have historically demonstrated an obstinate tendency to remain fixed in size.

If we want our taxis to learn then we first need to make them **teachable**, and to do that we need to make them **growable**.

We will assume, therefore, that for the purposes of this experiment every



(a) *Pick-up locations.*

(b) *Drop-off locations.*

Figure 3.2: *Heatmap showing pick-up locations (in green) and drop-off locations (in red) for the first week of February 2016, 8 a.m. to 12 noon.*

taxi in our fleet has the ability to grow and shrink its seating capacity at will, adding additional seats at some times and removing them at others. If we require a physical analogue we can perhaps imagine that these high-tech future vehicles are modular and can add or remove seating pods to their base configuration. More prosaically, we might imagine that the taxis themselves remain fixed but the taxi operator experiments with differently sized vehicles on different days.

We will revisit this assumption later and question if reinforcement learning is genuinely a good fit for our problem domain, but for now our taxis are teachable and our primary concern at any rate is with their final state rather than the path they take to get there.

3.2.1 Reward Metric

To apply q-learning or indeed any reinforcement learning algorithm to our taxis we need a suitable **reward metric**, a single number r which measures the taxi's performance and gives the algorithm a target to optimise.

In the real world the most obvious reward metric for a taxi is profit — taxi services are generally run as commercial enterprises and will naturally seek to maximise their profit over time. There are two significant problems with using profit as a reward metric however:

- Simply attempting to optimize operator profit ignores those factors which in economics are known as **externalities** — the set of social and environmental costs and benefits which are invisible to the financial calculus of profit and loss [27]. (Pollution from car exhausts is a classic example of an externality; the pollution imposes a cost on those who are affected by it which is not borne by the polluter.

Our autonomous taxis would presumably be electrically-powered but would undoubtedly have characteristic positive and negative externalities of their own.)

- Even if we were happy to use raw operator profit as our reward metric, determining the profit function to use would require an unfeasible number of assumptions about future technology, future operating costs, and the future economic environment in which our taxis will operate [28][29][30].

Fortunately there is a simple proxy we can use as a substitute for our ideal metric of externality-adjusted operator profit. This is **weighted passenger distance**.

The idea is straightforward. If a taxi's **passenger distance** for a trip is its distance-travelled times the number of passengers carried, then its **weighted passenger distance** is this figure weighted by the utilised proportion of its capacity, i.e.

$$distance\ travelled \times no.\ of\ passengers \times \frac{no.\ of\ passengers}{taxi\ capacity} \tag{3.1}$$

For example, if a three-seat taxi carries two passengers a distance of 15km, its weighted passenger distance for the trip would be:

$$15 \times 2 \times \frac{2}{3} = 20 \tag{3.2}$$

This simple metric is intuitively robust and captures two key ideas:

1. More passengers are better than fewer passengers.

2. Empty capacity is wasted capacity and imposes costs on the operator, on other commuters, and on the environment.

We should note immediately that point 1 is only valid for a ridesharing taxi model with per-seat pricing. This is not the model we currently use to price taxi fares — if I hire a taxi today I pay the same fare for the trip whether I travel alone or bring three friends along with me for the ride; we take it for granted that when we hire a taxi we're renting its entire capacity, regardless of how much of that capacity we actually use. Adopting weighted passenger distance as our reward metric assumes that our taxis will function more like small, hailable, free-floating buses than contemporary taxis.

Is this a realistic assumption? Ridesharing services have certainly increased in popularity in recent years and this trend seems likely to continue. Commuters will always be faced with trade-offs between privacy, personal space, financial cost, and time cost, and it seems plausible that ridesharing services where passengers pay individually by the seat will be a welcome additional option for many.

We will make one last adjustment to our metric to determine our final reward function; that is, to divide each taxi's weighted passenger distance by the total distance the taxi has travelled over the course of the day. This incorporates a penalty for empty travel into our metric, the third and last key idea we want it to capture.

Our reward metric r is then given by the formula:

$$r = \frac{\textit{weighted passenger distance}}{\textit{total distance travelled}} \tag{3.3}$$

3.2.2 Training Protocol

We now have a set of training data, a reinforcement learning algorithm, and a reward metric for our taxis. Our next steps will be:

1. To develop a taxi simulation engine capable of simulating taxi trips in Manhattan using real-world demand data drawn from the New York City dataset.

2. To calibrate this model to determine the number of standard 4-seat taxis required to adequately service demand.

3. To develop a reinforcement learning protocol, run it against our simulated taxi fleet, and analyse the results.

Chapter 4 describes the taxi simulation engine and its implementation of q-learning. Chapter 5 describes the training process and analyses its results.

Our provisional training protocol will proceed as follows:

- We will begin with our baseline fleet of 4-seat ridesharing taxis. Each taxi in the fleet will be modelled as an independent q-learning agent with its own individual q-table.

- A taxi's **state** will be its size and the **actions** available to it will be to subtract one from its size, to remain at the same size, or to add one to its size.

- The training process will be divided into days, each using one day's worth of requests from the filtered dataset, and running in a continuous loop as required.

- At the beginning of each day, each taxi will choose from its available actions. The taxi will choose to explore its state space with probability ε (choosing randomly) or will choose to exploit its existing knowledge with probability $1 - \varepsilon$ (choosing its best option).

- The simulation will then run for one day. At the end of the day each taxi will determine its reward metric and update its q-table.

The training process will last for 2,000 days in total, divided into three phases. For the first 1,000 days each taxi's ε parameter will be held fixed at a value of 1, forcing the taxis to explore their state space; over the next 500 days ε will slowly be reduced to zero, transitioning the taxis from explore to exploit mode; finally, the simulation will run for a further 500 days to allow the taxis to settle into a steady state.

This protocol is provisional — our first guess at a suitable training regimen for growable taxis. We will re-evaluate it as we go and adapt it as necessary in the light of its results.

Chapter 4

Implementation

In this chapter I describe the taxi simulation engine I built to train my simulated fleet of autonomous taxis and discuss the assumptions and tradeoffs inherent in its design.

4.1 Building a Simulation Engine in Python

Before I could begin training any taxis I had to build a simulation engine capable of simulating taxi trips using request data from the New York City dataset. I chose to build this simulation engine in Python, a language known for its flexibility, speed of development time, and extensive library support. (Python's disadvantages include its slow execution speed and poor native support for parallelization, issues we will return to later.)

4.1.1 Assumptions and Parameters

To complete the simulation engine in a reasonable amount of time I made one primary simplifying assumption — that simulated taxis would always travel in straight-line paths towards their current destination. This is obviously unrealistic and ignores the underlying geography of the road system but is a relatively good approximation for short trips in a small, convex region with a dense road network like Manhattan. (Restricting the analysis to trips within Manhattan had the advantage of excluding geographical choke-points like bridges and tunnels.)

To ensure the model didn't stray too far from reality I looked at all 1.85 million requests in the dataset and calculated the average speed at which these taxis would have travelled *had they taken straight line paths*. (The data for each request included the pick-up and drop-off times and GPS co-ordinates for the pick-up and drop-off locations, making this analysis possible.)

I used this average speed, which turned out to be approximately 12 km/h, as the speed for all taxis in the simulation. (Like all the numerical assumptions we will discuss in this chapter, taxi speed is not hard-coded in the model but implemented as a parameter which can be chosen per-simulation run.)

4.1.2 Tick Time

A second significant model parameter is the **tick time**, an interval measured in seconds of in-world simulation time which determines the frequency at which the engine recomputes the state of the world. (For example, a ten-second tick means that the engine recomputes the location

of every taxi and passenger and runs its dispatching and repositioning routines at ten second intervals.)

Selecting a tick time involves making a direct trade-off between the granularity of the simulation and its execution speed — a simulation using a one-second tick takes roughly ten times longer to run than a simulation using a ten-second tick.

I found through experimentation that I could increase the tick time to 60 seconds without significantly impacting the results of the simulation. This was surprisingly longer than I had expected and provided a welcome speed boost to the engine, making longer training times possible for the q-learning algorithm.

4.1.3 Dispatching

The dispatching routine is the most computationally expensive part of the simulation and proved to be the engine's primary bottleneck in terms of execution speed, particularly in simulations with ridesharing enabled.

The dispatching algorithm itself is conceptually simple. Incoming requests are added to a dispatch queue. The engine iterates over this queue once per tick and attempts to dispatch the closest available taxi to service each request, where 'available' means the closest idle taxi if ridesharing is disabled or the closest taxi with sufficient spare capacity and a compatible itinerary if ridesharing is enabled.

The problem is that to determine the *closest* taxi the engine needs to calculate the distance from the passenger's pick-up location to every available taxi, which in the worst case means every taxi in the simulation, and it needs to do this for 60,000 requests per day. (If ridesharing is enabled

the problem is even worse as the engine has to perform multiple distance calculations just to determine if a taxi is available.)

As the engine models pick-ups and drop-offs at a point-to-point level of granularity using arbitrary GPS coordinates (instead of using a fixed set of predetermined pick-up and drop-off locations), none of these distances can be precomputed but have to be calculated on the fly.

To solve this problem I introduced the idea of **dispatch zones**, as illustrated in Figure 4.1. Each small rectangle on this map is a 0.01° latitude by 0.01° longitude 'zone' (approximately 1,100m by 850m). The engine keeps track of which taxis are in which zones at all times and when dispatching a request checks only the passenger's own pick-up zone and the eight immediately-neighbouring zones for candidate taxis.

As a further optimization I introduced the idea of an **instant dispatch radius**, measured in minutes of driving time. Every available taxi within this radius of the passenger's pick-up location counts as being 'the closest', so the engine short-circuits to dispatching the first one it finds. (The instant dispatch radius is one of the engine's configurable parameters; I used a value of 1 minute for all my simulation runs.)

4.1.4 Repositioning

Determining an optimal repositioning strategy for idle taxis is a complex problem which can be studied in its own right [18], but as this question wasn't the focus of my project I decided to adopt a simple probabilistic model based on observed historical demand.

To construct this model I looked at all 1.85 million requests in the dataset and assigned a weight to each zone proportional to the total number of

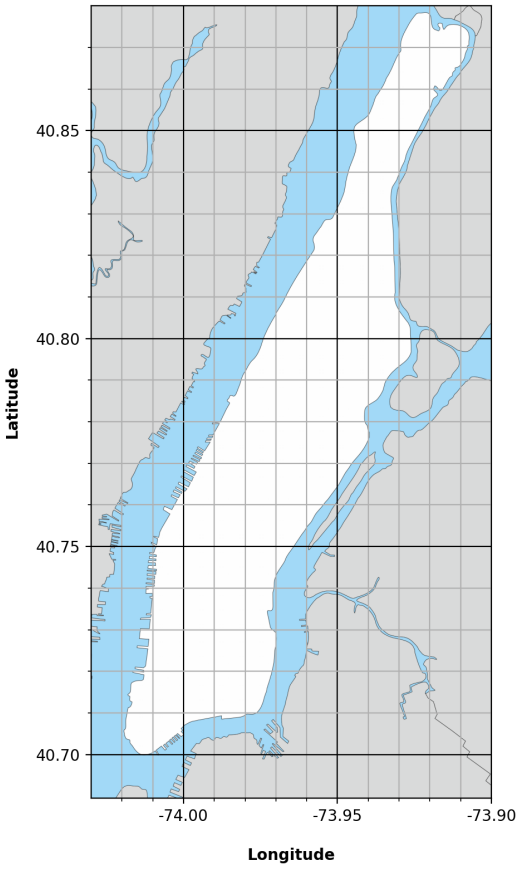


Figure 4.1: *Zone map of Manhattan. Each small block on this map is a* 0.01° *latitude by* 0.01° *longitude zone (approximately 1,100m by 850m).*

pickups originating in that zone.

The simulation engine has a configurable **mean time to reposition** parameter, measured in minutes, which determines the average time an idle taxi will wait before repositioning itself. (The default setting is 10 minutes.) The engine takes this parameter and, by modelling repositioning events as a Poisson point process, converts it into a per-tick probability that an idle taxi will decide to reposition. If a taxi does decide to reposition, it chooses its destination zone by selecting randomly from the weighted distribution.

4.1.5 Ridesharing

The simulation engine can be run with or without ridesharing enabled. When ridesharing is disabled, the dispatching routine considers only idle or repositioning taxis as available for picking up pending requests; when ridesharing is enabled, it also considers active taxis with passengers already onboard as available if they have sufficient spare capacity and a 'compatible' itinerary.

In deciding if a taxi's itinerary is compatible with a request there are two questions to ask: first, would the current occupants of the taxi be willing to accept the necessary detour to pick up the new passenger, and second, would the prospective passenger be willing to accept the offered position in the taxi's drop-off itinerary or would they prefer to wait in the hope of finding a more direct ride to their destination?

To answer these questions the simulation engine uses the idea of a **ridesharing multiplier**, a parameter with a default value of 1.1 corresponding to a 10% increase in journey time. That is, if the detour to pick up a new passenger will add less than 10% to their remaining journey time,

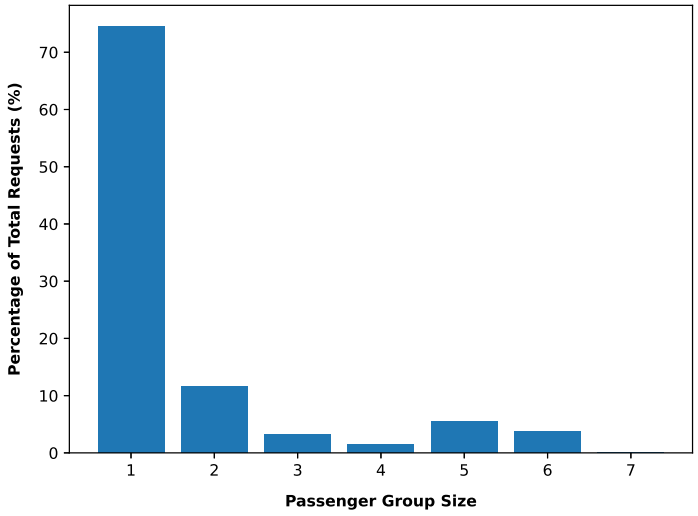


Figure 4.2: *Passenger group size as a percentage of total trips.*

we assume that the current occupants of the taxi will accept it. Similarly, if the offered rideshare adds less than 10% to the prospective passenger's direct journey time, we assume that the passenger will accept it.

Designing an optimal ridesharing algorithm for autonomous taxis is a complex problem whose subtleties are beyond the scope of this project but, as we shall see, this simple model turns out to be quite effective.

4.1.6 Groups and Splitting

Figure 4.2 show a plot of passenger group size as a percentage of total trips in the New York City dataset. Most trips consist of commuters travelling alone (74%) or in pairs (12%), but larger group sizes are still numerically significant — of the 1.85 million trips in the dataset, 100,000 were by groups of size 5 while 69,000 were by groups of size 6.

There is no obvious answer to the question of how we should handle these larger group sizes in our hypothetical future scenario of variably-sized taxis. Are these passenger groups atomic or would they be prepared to split up and travel separately if that meant they reached their destination sooner?

To handle this aspect of the model I introduced the idea of a **group split size**, a parameter with a default value of 4, and a **group split time**, a parameter with a default value of 5 minutes. With these settings, if a group of 4 or more passengers has waited longer than 5 minutes without finding a taxi, we assume that the group will split in half and each half will attempt to find a taxi separately. (The motivating assumption here is that a group will be prepared to split as long as no member has to travel alone.)

4.1.7 Execution Speed

Python is a slow language, typically two orders of magnitude slower than C, so getting the simulation to run at an acceptable rate on the available hardware (a Core i5 MacBook Pro) proved a challenge. My first implementation took two hours to process a single day's worth of requests and would have required 167 days to run the full 2,000 day training routine.

Thankfully I was able to improve its speed considerably. The initial bottleneck turned out to be an external geocoding library I was using to calculate distances between GPS coordinates — reimplementing this functionality myself resulted in a 1,500x speed boost for this section of the code.

(My implementation uses the haversine formula to determine the great-

circle distance between two points on the earth's surface, given their latitudes and longitudes [31]. The haversine formula models the earth as a perfect sphere which isn't strictly true so its accuracy is limited to 0.5% or 5 meters per kilometer, which I considered sufficient for the simulation. More accurate techniques which use an ellipsoidal model of the earth are possible but are considerably slower to calculate.)

This single change reduced the runtime of the simulation to roughly 3 minutes per day, an enormous improvement, although it would still have taken 100 hours to run the full training routine. I found that I could gain an additional 10x speed boost without making any changes to the code simply by running the simulation using PyPy rather than the default CPython interpreter. (PyPy is an alternative implementation of the Python language which uses a just-in-time compiler to boost the speed of long-running programs.) This brought the runtime down to approximately 20 seconds per day, or roughly 6 hours per thousand days of simulation time.

Along the way I tried some other speedup strategies which proved less successful. One of these was caching. The dispatch routine calculates certain point-to-point distances multiple times so it seemed a promising strategy to cache these results in RAM to avoid multiple calls to the distance function. In practice it turned out that the cache's dictionary lookups were more expensive than simply recalculating the distance each time and caching actually slowed down the simulation.

One other strategy which I considered but in the end didn't require was parallelization. The simulation's most computationally-expensive component, the dispatch loop, is highly parallelizable and in an ideal implementation its speed would scale directly with the number of CPU cores available. Python has poor native support for parallelizing CPU-bound tasks, however, and can only utilise multiple cores by running the par-

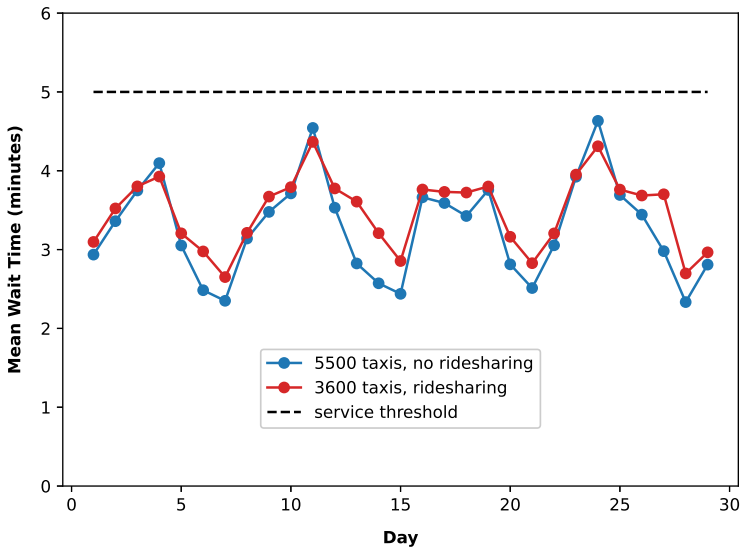


Figure 4.3: *Mean passenger wait time (calculated daily) over one month.*

allel threads in separate processes (i.e. using a multiprocessing rather than multithreading model). In the end I judged that the potential 2x speed boost available on my dual-core system couldn't justify the additional complexity that parallelizing the engine would have added to the codebase as the runtime was already fast enough for the project's purposes.

4.2 Calibrating the Model

To calibrate the model and determine the number of taxis required to service demand I decided to use two performance metrics — the number of **timeouts** and the **mean passenger wait time**.

- Passengers won't wait forever for a taxi to be dispatched; if they're left waiting too long they'll give up and find an alternative means of transportation. We say that a request *times out* if a passenger has been waiting longer than the engine's **timeout** parameter (by default, ten minutes) without a taxi being dispatched to service their request. To calibrate the model, I chose a service threshold of less than 0.5% for timeouts, that is, less than 5 requests out of every thousand should time out.

- The **passenger wait time** is the total time a passenger spends waiting for a taxi from the moment they submit their request until the moment their taxi arrives. To calibrate the model, I chose a service threshold of less than five minutes for the **mean passenger wait time**.

Applying these service thresholds to the taxi fleet's performance on its worst day, the model predicts that 5,500 taxis are required to service demand if ridesharing is disabled while only 3,600 taxis are required if ridesharing is enabled.

Figure 4.3 shows a plot of mean passenger wait time (calculated daily) over one full month for both the ridesharing (in red) and non-ridesharing (in blue) taxi fleets. Figure 4.4 shows a plot of the timeout percentage (calculated daily) over the same full month for both the ridesharing (in red) and non-ridesharing (in blue) taxi fleets.

Figure 4.5 shows the paths of two sample taxis, one each from the ridesharing and non-ridesharing fleets, over a four-hour period from 8 a.m. to 12 noon on a single day.

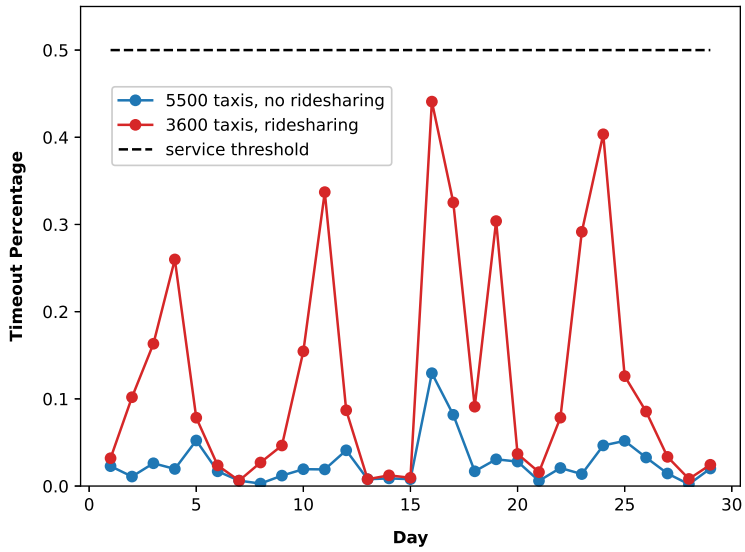
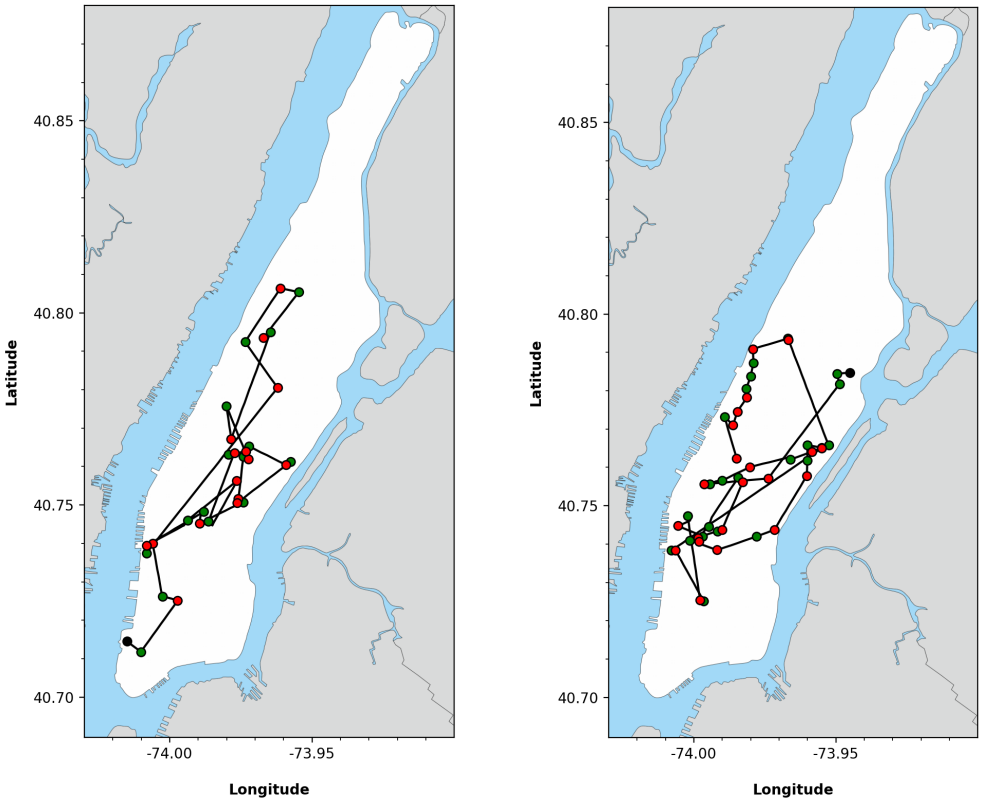


Figure 4.4: *Timeout percentage (calculated daily) over one month.*

4.3 Implementing Q-learning

With the simulation engine complete, implementing the q-learning algorithm was a simple task. Each taxi has its own individual q-table, a matrix in which each row represents a possible state the taxi can be in (a possible size) and the columns represent the actions available to the taxi in that state (typically to add one to its size, to subtract one from its size, or to remain unchanged). An individual entry in the matrix represents the taxi's estimate of the long-term value of choosing that particular action in that particular state (the 'quality' of the action). A sample q-table for a trained taxi is shown in Appendix B.

The q-learning algorithm has two parameters, α, the **learning rate**, and γ, the **discount rate**, which are implemented as engine parameters.



(a) *No ridesharing.*

(b) *Ridesharing.*

Figure 4.5: *Sample taxi paths over a four-hour period, 8 a.m. to 12 noon. A black dot marks the taxi's starting location, green dots indicate pick-ups, and red dots indicate drop-offs.*

- For α I chose a value of 0.25, low enough to ensure that taxis wouldn't place undue weight on uncharacteristic returns from particularly unlucky days.

- For γ I chose a value of 0.9 to ensure the taxis placed a high value on maximizing their long-term reward.

I also implemented a global **maximum size** parameter and chose an initial upper limit of 16 to ensure the taxis would actually explore their state space instead of embarking on a random walk to infinity. (This value was my first estimate of a number that would be safely 'too big' to be optimal. I intended to revise this estimate if the results showed that larger taxi sizes were favoured; in the event, this proved unnecessary.)

Chapter 5

Results & Evaluation

In this chapter I describe the results of applying the reinforcement-learning training protocol developed earlier to a fleet of simulated autonomous taxis. I discuss the optimality of the q-learning approach and develop and apply a simpler reinforcement-learning algorithm to the taxi fleet.

5.1 Preliminary Investigation

I began the investigation with a preliminary training run lasting for 2000 days of simulation time. This run was divided into three phases:

- For the first 1000 days, each taxi's ε parameter (its probability of choosing to 'explore' rather than 'exploit') was held constant at 1; this meant that each day each taxi would randomly choose from its available actions without attempting to maximize its payoff. This phase was intended to give the taxis time to explore their state space and populate their q-tables.

- Over the next 500 days, each taxi's ε parameter was slowly reduced to zero. This encouraged the taxis to focus-in on their most promising states and spend their time investigating the regions around them.

- The simulation then ran for a further 500 days with ε set to zero; each taxi now operated solely in 'exploit' mode and chose its best option at every opportunity. This final phase gave the taxis time to settle into a steady state.

5.1.1 Setup

For this training run I used a fleet of 3,600 taxis with ridesharing enabled. Each taxi's maximum size was set to 16.

5.1.2 Results

Figure 5.1 shows the distribution of taxi sizes over the course of the 2000-day training run. The result is a clear preference for size 7, with sizes 6 and 8 the closest runners-up. This preference for size 7 is even more clearly visible in Figure 5.2 which shows the distribution of sizes on the final day of training.

- Size 6 accounts for 817 out of 3600 taxis, or 23% of the total.
- Size 7 accounts for 1242 out of 3600 taxis, or 35% of the total.
- Size 8 accounts for 530 out of 3600 taxis, or 15% of the total.
- Together, sizes 6, 7, and 8 account for 73% of the final distribution.

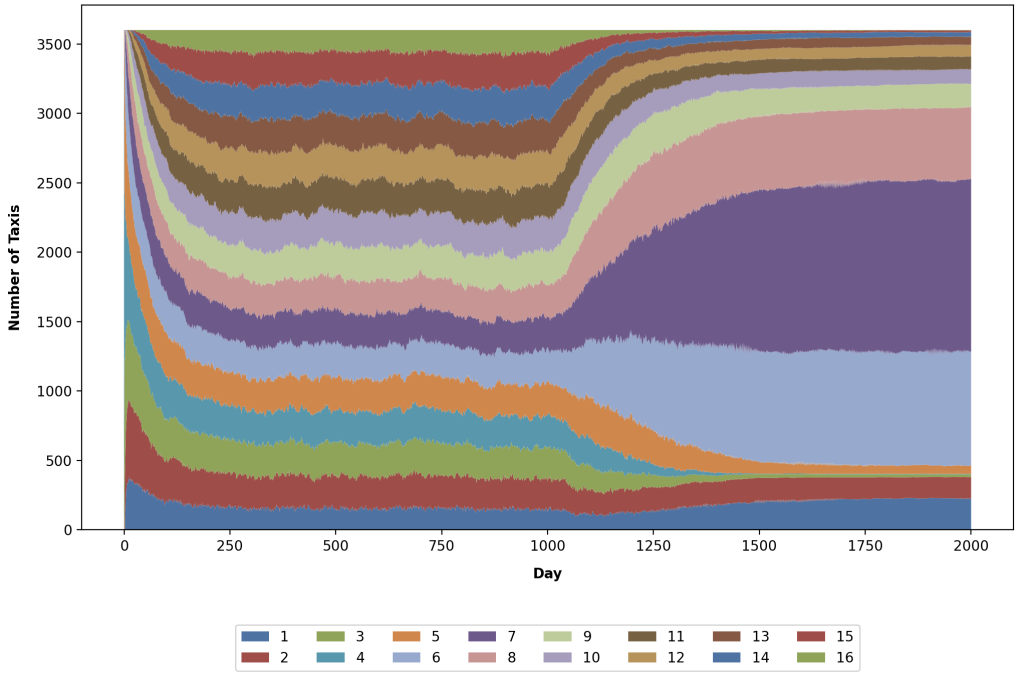


Figure 5.1: *Distribution of taxi sizes over 2000 days of training.*

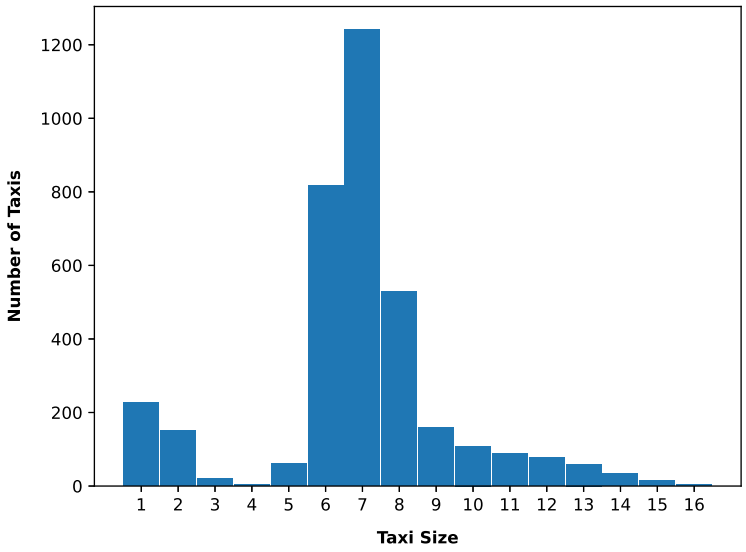


Figure 5.2: *Distribution of taxi sizes on the final day of training (2000 runs).*

5.1.3 Performance Metrics

So how does this 'optimal' size distribution affect our passenger service metrics? I re-ran the simulation for one month using the final distribution of taxi sizes from day 2000 of the training period. Figure 5.3 shows the daily percentage of timeouts for this distribution in red, contrasted in blue with the daily percentage of timeouts for the same simulation run using 3,600 4-seat taxis. Figure 5.4 shows mean passenger waiting times for the same two simulations and Figure 5.5 shows mean passenger journey times.

We can see that with respect to timeouts and mean waiting times the trained distribution performs significantly better than the uniform distribution; passenger journey times are also slightly reduced although the difference is negligible. This is an interesting result as the training al-

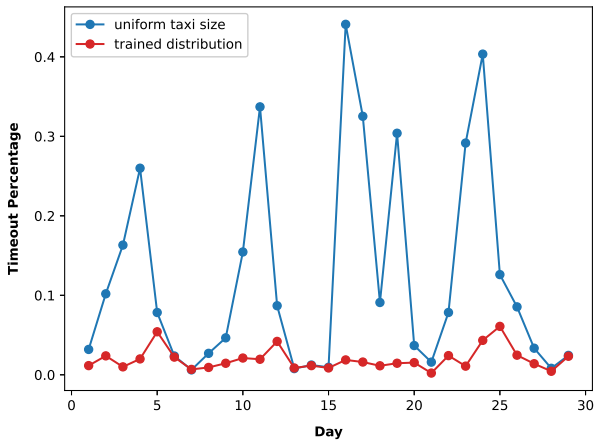


Figure 5.3: *Timeout percentage (calculated daily) over one month for the trained distribution (in red) and the default uniform distribution of size 4 (in blue). Both simulations used 3,600 taxis with ridesharing enabled.*

gorithm made no attempt to optimize these passenger metrics — it was primarily concerned with optimizing performance from the taxi's own perspective. The improved service for passengers is in this respect a happy coincidence.

This outcome is likely a straightforward product of an increase in overall system capacity resulting from the training. Our initial uniform distribution of 3,600 4-seat taxis had a total capacity of $3,600 \times 4 = 14,400$. Our trained distribution, in contrast, has a total capacity of 24,719, an increase of over 70%.

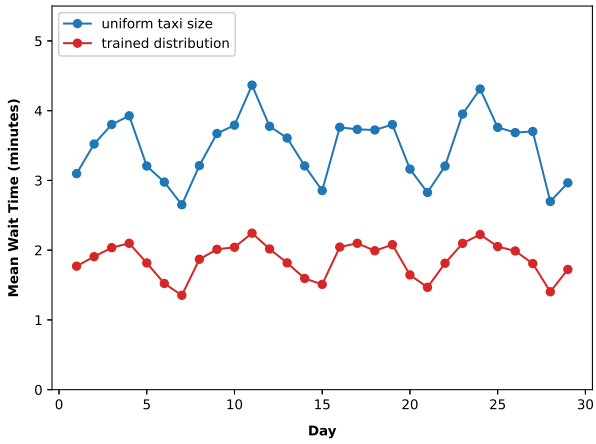


Figure 5.4: *Mean passenger waiting time (calculated daily) over one month for the trained distribution (in red) and the default uniform distribution of size 4 (in blue). Both simulations used 3,600 taxis with ridesharing enabled.*

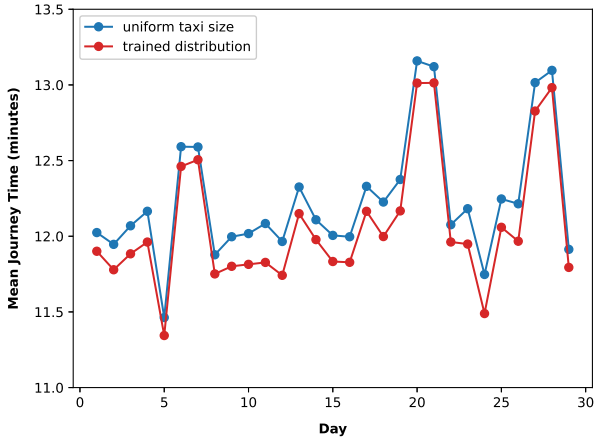


Figure 5.5: *Mean passenger journey time (calculated daily) over one month for the trained distribution (in red) and the default uniform distribution of size 4 (in blue). Both simulations used 3,600 taxis with ridesharing enabled.*

5.2 Confirmation

Our preliminary investigation has given us a clear result: in this model, taxis overwhelmingly prefer to be at or around size 7. But is this result reliable? In particular, the value of 7 looks suspiciously close to the midpoint of the allowed range of sizes. This raises the question: is this result simply an artifact of the training protocol, perhaps the product of some unanticipated bias towards the midpoint?

To investigate this hypothesis I decided to double the maximum possible taxi size to 32 and re-run the simulation over a substantially longer 6,000 day period. As before, this training run was divided into three phases:

- An initial 4,000-day exploration phase with each taxi's ε parameter held constant at 1.
- A 1,000-day transition phase with ε slowly decaying to zero.
- A final 1,000-day cooldown phase to allow the distribution to settle into a steady state.

Figure 5.6 shows the distribution of taxi sizes over the course of this training run. Once again 7 is the preferred size by a clear margin, with 6 and 8 as the closest runners-up. Figure 5.7 shows the final size distribution on day 6,000 — clear confirmation of our initial result. Indeed, the longer training run seems to have resulted in a significantly stronger preference for size 7.

- Size 6 accounts for 918 out of 3,600 taxis, or 26% of the total.
- Size 7 accounts for 1,445 out of 3,600 taxis, or 40% of the total.
- Size 8 accounts for 571 out of 3,600 taxis, or 16% of the total.

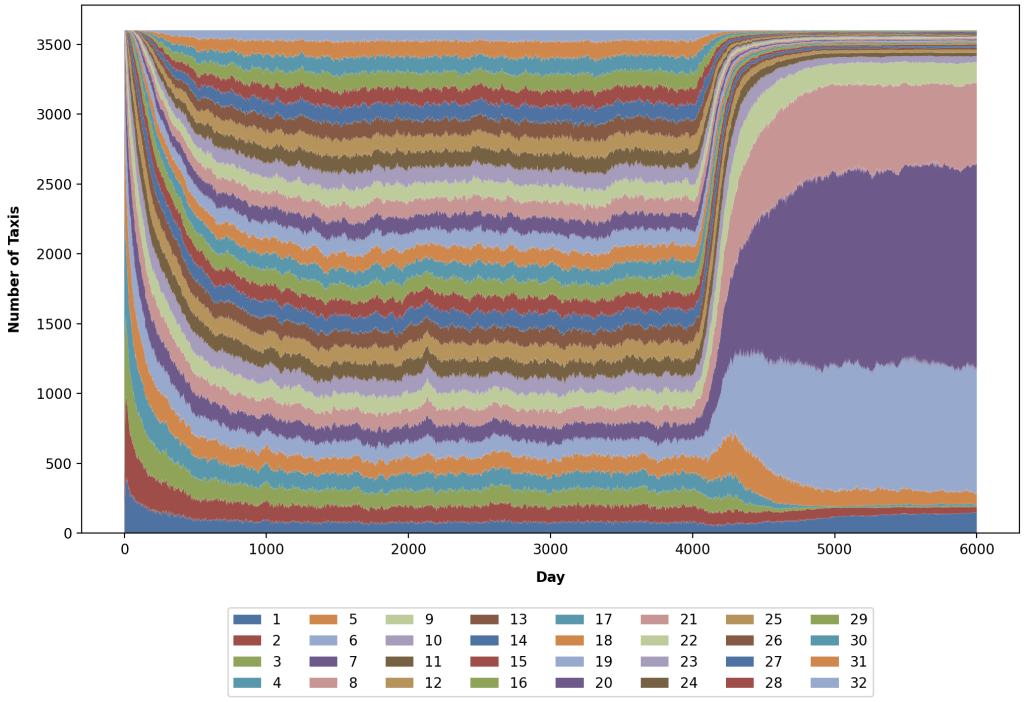


Figure 5.6: *Distribution of taxi sizes over 6000 days of training.*

- Together, sizes 6, 7, and 8 account for 82% of the final distribution.

5.3 Rethinking Q-learning

With the benefit of experience we can now re-examine our approach and ask if q-learning is genuinely a good fit for our problem domain. The answer, I think, is that it isn't a bad fit but it's unlikely to be optimally efficient.

The problem is that q-learning introduces an artificial path dependency

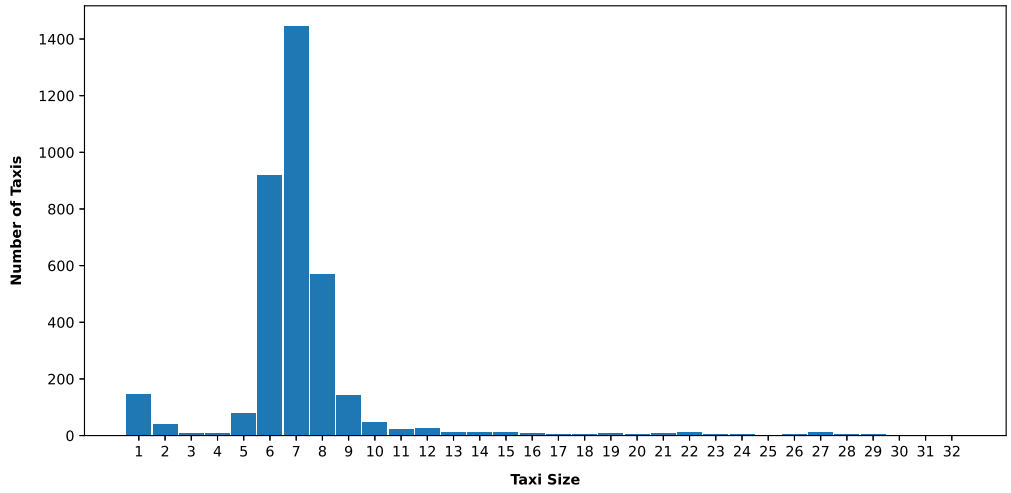


Figure 5.7: *Distribution of taxi sizes on the final day of training (6000 runs).*

into our model which has no parallel in our particular reality. Put simply, given any finite number of training episodes, a q-learning agent can get trapped at a local maximum with no path to a global maximum *that it already knows about*. In our case, a three-seat taxi could 'want' to be a seven-seat taxi but have no way of reaching that goal because four seats are worse than three and its estimate of its best action given its current state is to stay where it is. (Table B.2 in Appendix B shows a q-table from one of our trained taxis which has found itself trapped in just this sort of situation.)

This path dependence isn't a fundamental problem for applying q-learning to our domain; the algorithm is guaranteed to converge on a path to the global maximum given a sufficient number of training episodes. In practice, however, this number could be very large — inefficiently large, as the path dependency doesn't reflect any kind of fundamental real-world constraint on our taxis. A three-seat taxi doesn't really have to endure

the purgatory of being a four-seat taxi on its way to seven-seated nirvana. To the extent that it could change its size at all it could presumably add four seats in one go and be done with it.

An obvious solution to this problem is to give our taxis additional actions to choose from — to add or subtract two, three, or even four seats at a time. In practice this would increase the state space enormously and require even longer training times to explore. But alternative reinforcement learning techniques which don't assume path dependence may prove more efficient.

5.4 Monte Carlo Reinforcement Learning

One potentially promising alternative technique is **Monte Carlo sample learning**, machine learning terminology for trying different things at random and seeing what works best [11].

This algorithm is simple to implement. Its goal is to learn a value function $V(s)$, whose value is the expected return from being in state s calculated as the mean value of the agent's reward over all its experiences of s.

I applied this algorithm to our taxi simulation using a simple training protocol. Each taxi maintains an individual state table representing its current estimate of $V(s)$ for every possible state. At the start of each day each taxi is randomly assigned a size, s. The simulation then runs and at the end of the day each taxi determines its reward metric and updates its state table entry for $V(s)$ by calculating an incremental mean, i.e.

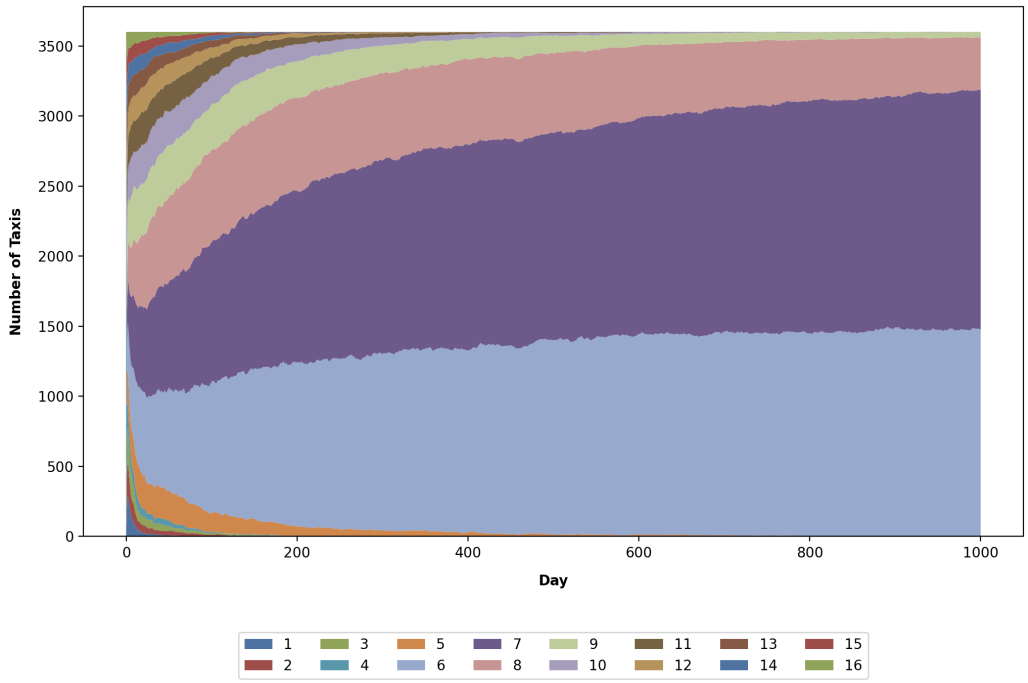


Figure 5.8: *Distribution of preferred taxi sizes over 1,000 days of training using Monte Carlo sample learning.*

$$V(s) \longleftarrow V(s) + \frac{r - V(s)}{n_s} \tag{5.1}$$

where r is the reward and n_s is the number of times the taxi has visited state s. At any given time a taxi's **preferred state** is the state with the highest value for $V(s)$.

I applied this training protocol to a fleet of 3,600 taxis with ridesharing enabled. Figure 5.8 shows the distribution of preferred states over 1,000 days of training. Figure 5.9 shows the distribution of preferred states on the final day. (A sample state table from one of these taxis is shown in

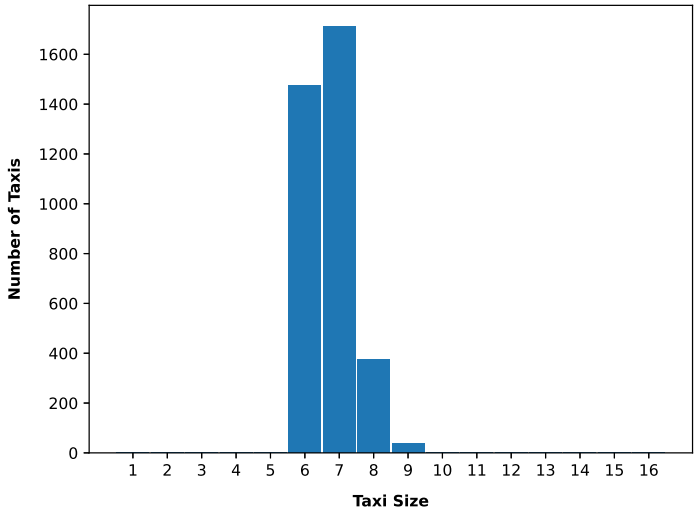


Figure 5.9: *Distribution of preferred taxi sizes after 1,000 days of Monte Carlo sample learning.*

Table C.1 in Appendix C.)

There are two main points to note. First, this model seems to confirm q-learning's preference for size 7, although the taxis are now largely indifferent between sizes 6 and 7. More interesting is the speed with which the taxis converged on this preference, reaching a good approximation of their final distribution within just 200 days. This is much faster convergence than q-learning could achieve.

So why does Monte Carlo sample learning favour size 6 more than q-learning? I suspect the discrepancy may be another side-effect of q-learning's path dependence. Part of the value of being in state 7 as far as q-learning is concerned is that it lets you transition into states 6 and 8, both of which are also pretty good. But this isn't a feature of the model we're interested in so for us sample learning may not only be more ef-

ficient than q-learning — it may also be giving us a truer picture of the intrinsic value of the states in-and-of themselves.

Chapter 6

Conclusion

This investigation of optimal autonomous taxi size has given us a clear prediction — AI taxi agents trained on real-world data using reinforcement learning overwhelmingly prefer a size of or about 7.

So how much weight should we place on this prediction? We can expect our results to be particularly sensitive to three important factors: the demand data used for training, the reward metric used to measure the taxis' performance, and the ridesharing algorithm used by the simulation engine to service trip requests.

- Our taxis have been trained on one set of data from one particular region of one particular city; our results may or may not transfer to other locations with their own unique demand patterns. This is a question which can only be answered empirically by further work.

- Our reward metric, weighted passenger distance, is a simple proxy for externality-adjusted operator profit. In particular, it assumes that autonomous taxis will function as small, hailable, free-floating

buses, charging passengers by the seat. Our prediction is unlikely to apply to taxi fleets which do not function in this manner.

- The simulation engine uses a conservative ridesharing algorithm which likely significantly underestimates the potential for ridesharing opportunities. If this is true, it suggests that in practice larger taxi sizes may in fact be optimal.

Granted these caveats, I believe we have a robust result with interesting real-world implications. Predicting the future may well be a job for a one-handed economist but when it comes to buying shares in autonomous taxi companies our model says that seven seats is the lucky number to watch out for.

Bibliography

[1] Lawrence D. Burns. Sustainable mobility: A vision of our transport future. *Nature*, 497:181–182, 2013.

[2] Niels Agatz et al. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Reserach*, 223(2):295–303, 2012.

[3] Shuo Ma et al. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[4] William J. Mitchell et al. *Reinventing the Automobile: Personal Urban Mobility for the 21st Century*. The MIT Press, 2015.

[5] L. P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[6] Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

[7] Megan M. Olsen and Rachel Fraczkowski. Co-evolution in predator prey through reinforcement learning. *Journal of Computational Science*, 9:118–124, 2015.

[8] Xueting Wang et al. A reinforcement learning-based predator-prey model. *Ecological Complexity*, 42, 2020.

[9] New York City Taxi and Limousine Commission. Passenger FAQ. `https://www1.nyc.gov/site/tlc/passengers/passenger-frequently-asked-questions.page`. Accessed: 2020-03-22.

[10] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning*. Morgan and Claypool, 2018.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[12] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[13] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[14] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, second edition, 2017.

[15] Hussein Dia and Farid Javanshour. Autonomous shared mobility-on-demand: Melbourne pilot simulation study. *19th EURO Working Group on Transportation Meeting*, EWGT2016, 2016.

[16] Joschka Bischoff and Michal Maciejewski. Simulation of city-wide replacement of private cars with autonomous taxis in Berlin. *Procedia Computer Science*, 83:237–244, 2016.

[17] Daniel J. Fagnant and Kara M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C*, 40:1–13, 2014.

[18] Maxime Guériau and Ivana Dusparic. SAMoD: Shared autonomous mobility-on-demand using decentralized reinforcement learning. *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[19] Michael W. Levin et al. A general framework for modeling shared autonomous vehicles with dynamic network-loading and dynamic ride-sharing application. *Computers, Environment and Urban Systems*, 64:373–383, 2017.

[20] Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in Berlin. *IEEE 20th International Conference on Intelligent Transportation Systems*, 2017.

[21] Wenwen Zhang et al. The performance and benefits of a shared autonomous vehicles based dynamic ridesharing system: An agent-based simulation approach. *95th Transportation Research Boarding Meeting*, 2015.

[22] Javier Alonso-Mora et al. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *PNAS*, 114(3):462–467, 2017.

[23] Kevin Spieser et al. Shared-vehicle mobility-on-demand systems: A fleet operator's guild to rebalancing empty vehicles. *Proc. TRB 95th Annu. Meeting Compendium Papers*, pages 1–15, 2015.

[24] Wen Shen et al. An online mechanism for ridesharing in autonomous mobility-on-demand systems. *Proceedings of the Twenty-*

Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), pages 475–481, 2016.

[25] Daniel J. Fagnant and Kara M. Kockelman. Dynamic ride-sharing and optimal fleet sizing for a system of shared autonomous vehicles. *Proceedings of the 94th Annual Meeting of the Transportation Research Board in Washington DC*, 2015.

[26] New York City Taxi and Limousine Commission. TLC trip record data. `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`. Accessed: 2020-03-22.

[27] Ashley Nunes and Kristen D. Hernandez. Autonomous vehicles and public health: High cost or high opportunity cost? *Preprint*, 2019.

[28] Patrick M. Bosch et al. Cost-based analysis of autonomous mobility services. *Transport Policy*, 64:76–91, 2018.

[29] Todd Litman. *Autonomous Vehicle Implementation Predictions: Implications for Transport Planning*. Victoria Transport Policy Institute, 2020.

[30] Lawrence D. Burns and Bonnie A. Scarborough. *Transforming Personal Mobility*. The Earth Institute, Colmbia University, 2012.

[31] Glen Van Brummelen. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2013.

Appendix A

Code

All the Python code used in this project has been submitted alongside this document.

Alternatively, the code can be viewed at or downloaded from the following public Git repository: **https://github.com/dmulholl/taxisim**.

Appendix B

Sample Q-Tables

Table B.1 below shows a sample q-table taken from a taxi after 2,000 days of training. The **state** is the taxi's size; the **actions** are the choices available to the taxi in that state, namely to subtract one from its size, to remain at the same size, or to add one to its size. The table entry or **q-value** is the percieved value to the taxi of taking that particular action in that particular state, based on its previous experience.

Table B.2 shows an example of q-learning failing. This 14-seat taxi 'wants' to be in state 7 but is trapped oscillating between states 13 and 14.

State	Actions		
	-1	0	$+1$
1	-1.0000	3.9614	4.2514
2	4.1890	4.5095	4.3567
3	4.5053	4.2871	3.8813
4	3.8403	4.3982	5.1306
5	4.4104	3.8018	6.1835
6	5.5831	6.2612	7.1076
7	6.2320	7.8267	6.8260
8	7.0060	6.7034	6.4217
9	7.0003	5.9032	4.2079
10	4.9094	4.0061	3.8962
11	4.1492	3.9793	2.8740
12	3.4580	3.8673	3.2053
13	3.7678	2.9496	2.5014
14	2.9747	2.8777	2.3896
15	2.7989	2.7324	2.5311
16	2.8553	2.5458	-1.0000

Table B.1: *Sample Q-Table from a 7-seat taxi after 2,000 days of training.*

State	Actions		
	-1	0	$+1$
1	-1.0000	1.5683	2.0120
2	1.7945	2.2223	2.1049
3	1.8395	2.5225	2.7699
4	2.7287	2.8513	3.3452
5	2.9234	3.1752	3.9989
6	3.4507	4.7165	5.1835
7	5.3119	5.4479	4.8346
8	5.2573	4.6518	4.3950
9	4.9286	4.8264	4.7628
10	4.5761	4.7334	4.4723
11	4.4399	4.2840	2.7962
12	3.2963	3.7476	3.5570
13	3.3172	4.1521	4.3933
14	4.3747	4.2370	4.2546
15	4.3254	4.2832	4.2681
16	4.3438	4.1678	-1.0000

Table B.2: *Example of a q-learning agent becoming trapped at a local maximum. This 14-seat taxi 'wants' to be in state 7 but is trapped oscillating between states 13 and 14.*

Appendix C

Sample Monte Carlo State Table

Table C.1 below shows a sample state table taken from a taxi after 1,000 days of Monte Carlo sample training. The **state** is the taxi's size; the table entries show the number of times the taxi has visited each state and the taxi's estimate of its expected return from being in that state.

A taxi's **preferred state** is its state with the highest expected return. For this particular taxi, its preferred state is 7.

State	Visits	Return
1	61	0.6444
2	61	0.6627
3	66	0.5981
4	69	0.5810
5	68	0.6360
6	67	0.7698
7	48	0.7821
8	47	0.7290
9	65	0.6565
10	69	0.6477
11	57	0.6390
12	58	0.5921
13	79	0.5805
14	58	0.4605
15	64	0.4755
16	63	0.4331

Table C.1: *Sample state table after 1,000 days of training.*