



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

School of Computer Science and Statistics

Congestion-Aware Request-Matching in Ride-Sharing Systems using Multi-Agent Reinforcement Learning

Sophie Hegarty



April 30, 2020


A Dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI - Computer and Electronic Engineering

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at <http://www.tcd.ie/calendar>.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at <http://tcd-ie.libguides.com/plagiarism/ready-steady-write>.

Signed:  _____

Date: 30/04/2020

Abstract

With the growth in population throughout the world increasing, the need for transport, particularly around urban areas, the existing methods cannot keep up with the current demand. A better, more sustainable transport system must be found in order to keep up with the demand. Shared mobility-on-demand systems can improve the current systems of urban mobility. Autonomous vehicles partnered with the increase in demand for shared mobility apps such as UberPool, proves there is a clear want and need for this kind of technology. It seems only natural that optimizations are made in this area.

The main area of research needed for this kind of system is how to match requests with vehicles. This is a large area of research which focuses on using machine learning techniques to optimize matching. It mainly looks at research areas such as, predicting the demand of rides and using the shortest path as an optimizer. Whilst a lot of progress has been made in the area of shared mobility-on-demand systems, the methods that have been tested so far have been unrealistic. They focus on how models, e.g. Reinforcement Learning models, will work in an ideal simulation without taking into consideration real world attributes.

The main aspect that affects the time and profitability of a ride is traffic congestion. This is uncontrollable and will always be present when dealing with mobility systems. For this reason it must be taken in consideration before such models are implemented to the public. This research project will develop a model to take into account the congestion on the routes, to optimize ride time for clients and to optimize profits for the vehicle. This project will look specifically at congestion as the main real world attribute, that will have an effect on SaMOD. This thesis proposes a congestion-aware multi-agent reinforcement learning based solution to optimize request-matching sharing on SaMOD. It will simulate on a map of Dublin, as Dublin has yet to have a SAMoD study.

We evaluated the solution against a second scenario which was not congestion-aware. We looked at metrics that affected the rider, the vehicle and the environment. When the agents were congestion-aware, we saw a large increase in the number of shared miles however, this also increased the total distance that each vehicle travelled per hour. The passengers saw a larger wait time when the agents were congestion-aware, however they benefited from a lower travel time.

Acknowledgements

I would first like to thank my supervisor Prof. Ivana Dusparic for all her support, guidance and expertise throughout my masters thesis.

I'd like to thank Dr. Maxime Guériau for his guidance in SUMO traffic simulations.

Finally, I'd like to thank my family for their support and encouragement through my five years at TCD.

Contents

Abstract	ii
1 Introduction	1
1.1 Shared Autonomous Mobility on Demand	1
1.2 Thesis Aims and Objectives	2
1.3 Thesis Assumptions	3
1.4 Thesis Motivation	3
1.5 Thesis Contribution	4
1.6 Evaluation	5
1.7 Report Structure	5
2 Background Knowledge	7
2.1 Multi Agent Systems	7
2.1.1 Environments	8
2.1.2 Architecture	9
2.2 Machine Learning	10
2.3 Reinforcement Learning	11
2.3.1 Markov Decision Process	12
2.3.2 RL Characteristics	13
2.3.3 Reinforcement Learning Agents	13
2.3.4 Q-Learning	14
2.4 Deep Reinforcement Learning	16

3	State of the Art	17
3.1	Ride-Sharing	17
3.2	Types of Ride-Sharing	18
3.2.1	Meeting Points	19
3.2.2	Dynamic Requests	20
3.3	Shared Autonomous Vehicles in Ride-Sharing	20
3.4	Request Matching Optimization	21
3.4.1	Machine Learning	21
3.5	Reinforcement Learning	22
3.5.1	RL in Ride-hailing	22
3.5.2	RL in Ride-sharing	24
3.6	Deep Reinforcement Learning	25
4	Design	28
4.1	Request Matching Optimization Problem	28
4.2	State Representation	29
4.3	Action Space	31
4.4	Reward Function	31
4.5	System Design	32
4.5.1	System Architecture	32
4.6	Data	33
4.6.1	Request Data	33
4.6.2	Congestion dataset	35
4.7	Scenario Testing Design	37
4.8	Algorithm Design	38
5	Implementation	40
5.1	RL Libraries	40
5.1.1	Vehicle Class	40
5.1.2	Passenger Class	41

5.1.3	Stop Class	41
5.2	Multi-Agent Environment Set Up	42
5.3	QL Agent	43
5.4	Data	44
5.4.1	SCAT data	44
5.4.2	GPS File	45
5.4.3	DIS File	46
5.4.4	Data Pre-processing	46
5.4.5	Request File	47
6	Evaluation	49
6.1	Objectives	49
6.2	Simulation Environment	50
6.2.1	SUMO Files	51
6.2.2	Dublin Postal Code Shape-file	52
6.3	Scenarios	54
6.4	Metrics	55
6.4.1	Passenger Metrics	55
6.4.2	Vehicle Metrics	56
6.4.3	Environment Metrics	56
7	Results	58
7.1	Vehicle Evaluation	58
7.2	Passenger Evaluation	61
7.3	Environment Evaluation	62
7.4	Result Summary	63
8	Conclusion	64
8.1	Thesis Contribution	64
8.2	Future Work	65

List of Figures

2.1	RL Process, agent interacting with the environment through sensors and outputting an action with actuators	8
2.2	Multi-Agent Architectures: (1) Centralised (2) Decentralised (3) Hybrid .	11
2.3	The agent–environment interaction in a Markov decision process [1]. . .	12
3.1	Example of calculated meeting points for ride sharing, d=driver, r=rider, m=meeting point [2]	19
3.2	Example of an offline training, online decision making platform	23
3.3	Distributed Learning based Algorithm	23
3.4	Mean-Field MARL Framework	24
3.5	Hybrid Decentralised RL approach in SAMoD	26
3.6	Action Selection Q-learning	26
4.1	Creating the observation space	30
4.2	Creating the observation space	30
4.3	Sample NYC dataset	33
4.4	Histogram showing the passenger count in NYC requests	35
4.5	Histogram showing the passenger count in Dublin requests	36
4.6	Sample of a congestion dataset for 9 postal codes in Dublin	36
4.7	Sample of a congestion creation from traffic flow	37
4.8	Congestion levels in Dublin on 2012-03-01 01:00	37
4.9	Evaluation scenarios that show what states are observable by a single agent	38

5.1	Hierarchy of RLlib library	42
5.2	Multi-Agent RLlib control	42
5.3	SCAT sensor locations in Dublin	45
5.4	Sample of Request trips in 03/01/2012 – 4-5pm	47
5.5	Number of Requests Per Day over one year	48
5.6	Number of Requests Over One Day	48
6.1	TraCI API connecting with SUMO using TCP [3]	51
6.2	Overview of SAMoD system control	51
6.3	Network map of Dublin	53
7.1	Average Total Distance Travelled Per Vehicle Per Hour in KM's	59
7.2	Average Shared Distance Travelled Per Vehicle Per Hour in KM's	59
7.3	Average Empty Distance Travelled Per Vehicle Per Hour in KM's	60
7.4	Average Occupancy Per Vehicle Per Hour	60
7.5	Travel Time Per Request in Minutes	61
7.6	Wait Times per Request in Minutes	62
7.7	Percentage of Requests Served	63

List of Tables

4.1	Table showing column titles and descriptions of request data	33
4.2	Table showing column titles and descriptions of request data	35
4.3	Congestion Levels	36
5.1	Passenger Details	41
5.2	GPS file variables	45
5.3	DIS file variables	46
7.1	Vehicle Metric Results	58
7.2	Requests Metric Results	61
7.3	Requests Metric Results	62

1 Introduction

This thesis addresses the problem of the lack of real-world implementations of Shared Autonomous Mobility on Demand (SAMoD) systems. The lack of integration with real-world variables in these systems leads to inexperienced and untested mobility systems being introduced to the public. This ultimately leads to problems in integration in urban areas, and a lack of uptake in these types of systems. SAMoD systems have been the focus area of research for many years. However very little of this research focus on how these systems would be improved by allowing them to view other variables in their environment outside of their own system.

This thesis will present a decentralised multi-agent reinforcement learning ride-sharing system that has an enriched observation space using congestion information. It will be simulated in Dublin using real-open data to build and create congestion information that agents can have access to.

1.1 Shared Autonomous Mobility on Demand

Shared Autonomous Mobility on Demand (SAMoD) systems are future mobility services that aim to revolutionise urban mobility. Mobility on Demand (MoD) defines a system where passengers can hail a ride from any place at any time. This is a change from traditional mobility systems where stops and times are predefined, for example a bus. A traditional taxi service is an example of a MoD, riders define where and when they want picked up. As we have seen in the past number of years there

has been a change to shared taxi services where vehicles can pickup other requests at the same time. This reduces costs for the passengers while increasing profitability for the vehicle driver. As we look towards the future we will soon see taxi systems using autonomous vehicles. As mentioned, one of the most difficult problems with a SAMoD is matching requests to vehicles without the presence of a driver to accept them.

1.2 Thesis Aims and Objectives

This thesis aims to expand the current models that are available in the request-matching space, whilst also filling the gap of research into real-world attribute aware ride-sharing systems. This will take the basis of the reinforcement learning models already out there and make a more realistic optimization using features that exist in the real world. This project will look specifically at traffic and congestion as the main real world attribute that will have an effect on SAMoD. Decentralised reinforcement learning has been proven to create high performance SAMoD systems. Each agent has its own policy which it learns from interacting from the environment, which is an ideal system for a flexible taxi system that can be implemented into multiple scenarios. This thesis aims to implement and evaluate a decentralised multi-agent reinforcement learning based solution to optimize request-matching within a ride-sharing system by opening up the state space to congestion information.

We aim to create a improved system that reduces the travel time for clients and increases the number of miles that a vehicle has more than one request present. This will be simulated, trained and evaluated on a network map of Dublin. We aim to evaluate this against an identical solution that is not congestion aware so that we can easily see the effect of having the congestion information included.

1.3 Thesis Assumptions

This thesis has made a number of assumptions when designing the simulation system in Dublin. They are outlined below:

- Vehicles have a maximum capacity of 4 people - for ease of implementation each vehicle can hold a maximum of 4 people at one time. This is the standard capacity of a taxi in Dublin.
- Traffic flow is assumed to be correct from the SCAT data obtained from Dublin City Council 2012. We also assume that a four month period that we have data for can represent and be extrapolated for the other 8 months.
- The percentage of taxis on the road is assumed to be 0.2% of all traffic recorded. This percentage is used to create requests from traffic flow.
- Agents are assumed to be failure-free, and able to view all requests as they come into the system. It assumes they are able to connect with each request to gather the information required to complete a journey. It is also assumed they cannot share information with other agents in the environment.
- It is assumed that passengers are stationary, that their pick up or drop off location does not change

1.4 Thesis Motivation

In 2020 approximately 55% of the worlds population resides in urban areas. This is due to increase to 68% by the year 2050. This means the worlds urban population will almost double [4]. Europe's urbanisation is a lot higher, with 75% of its population in urban areas with this growing to 84% by the year 2050. Infrastructures need to adapt to deal with such growth in demand, in particular urban mobility [4].

Shared mobility-on-demand systems can improve the current systems of urban

mobility. The rising popularity of autonomous vehicles partnered with the increase in demand for such shared mobility apps such as UberPool, there is a clear want and need for this kind of technology. It seems only natural that optimizations are made in this area. The main area of research needed for this kind of system is optimization of matching ride requests with vehicles. Whilst a lot of progress has been made in the area, the methods that have been researched so far have been unrealistic. They focus on how these models, for example reinforcement learning models, will work in an ideal situation without taking into consideration real world attributes.

The main aspect that would affect time and profitability of a ride is traffic congestion. This is uncontrollable and will always be present when dealing with mobility systems. For this reason it must be taken into consideration before such models are implemented to the public. Currently, taxi systems rely on the drivers having the knowledge and experience of the area to know if a pickup or drop off in a certain area is worth while for their clients and for their profits. With apps like UberPool, suggestion of nearby clients are given to the driver, however, the final decision is down to the driver. This again uses the knowledge of the area acquired from driving in the city to make the decision. When we take the driver out of the car and look at autonomous vehicles, it is obvious that we must find a way to transfer the knowledge of a driver to the vehicle in order for it to choose clients appropriately. This research project will develop a model to take into account the congestion on the routes, to optimize ride time for clients and to optimize profits for the vehicle.

1.5 Thesis Contribution

This thesis identifies how request matching in SAMoD systems can be improved by introducing congestion information and what the motivation is for such a system. The main contribution of this thesis is an RL based solution that works in a multi-agent environment that optimizes request-matching between requests and vehicles. The solution proposes agents that are decentralised, meaning that they learn

independently without communication between other agents. Unlike other solutions, agents will have access to traffic congestion information that will influence their decision. Currently no other research has been done in this area. There is also no current study in Dublin of a ride-sharing system. This thesis will test the proposed solution in a map of Dublin using open data generated in Dublin.

1.6 Evaluation

In order for the model to be evaluated correctly we must look at the areas that will be impacted. We will look at the impact on the system, the impact on the riders and the impact on the vehicles. Each area will have its own metrics that are discussed in this report. We will use these areas to evaluate which areas will benefit most from the proposed solution and which will suffer.

By looking at these three areas of impact a clear idea of how well the system works can be seen. This also gives opportunity for the model to choose what it should focus on more to optimize, whether it should reduce travel times even though it could effect the number of empty miles travelled. It will be evaluated in SUMO simulation software with a map of Dublin over a simulated time-frame of 52 days. The proposed system will be compared to an identical system that does not have access to congestion information. This allows comparison on what exactly is affected by introducing congestion to the agents view.

1.7 Report Structure

The rest of the thesis is as follows:

- Chapter 2 introduces and describes information surrounding shared autonomous mobility on demand. It also introduces basic knowledge required about reinforcement learning and multi-agent systems that is needed to

understand the rest of the project.

- Chapter 3 reviews existing literature in the area of SAMoD, focusing on types of systems, models used to improve these systems before focusing on request-matching optimization.
- Chapter 4 describes the design aspects of the solution, including the problem space, the aspects of RL in the solution and the data needed to run simulations.
- Chapter 5 discusses the implementation and simulation of the proposed solution. It also looks at the libraries used and adapted to build the system.
- Chapter 6 presents the evaluation design that is used to evaluate the results that follow in chapter 7.
- Chapter 7 presents the results from the solution that are evaluated according to chapter 6
- Chapter 8 concludes this thesis with a summary, outlines the issues that remain open and areas for future work.

2 Background Knowledge

This section contains knowledge that is required to read and understand Section 3. It will present, Multi-Agent Systems and Reinforcement Learning.

2.1 Multi Agent Systems

Multi-Agent Systems (MAS) have grown in popularity over the years to solve complex problems involving more than one agent. Agents are any entity that makes a choice in a network. This choice is autonomous, meaning that it has its own goals and behaviours. These behaviours are defined by the environment that they are in and its attributes. MAS involves multiple agents that make decentralised decisions what actions they will take. Agents usually interact with each other, sharing complexities and resources to give optimums globally in the task that they are trying to complete. However, this is not always the case as agents can also independently make decisions viewing their own environments that other agents interact with. MAS can be seen not only in the world of computer science, but in bio-based applications. Ant colonies are great examples of this. Gordon studies how ants, who appear to move randomly can allocate and complete tasks without a central entity informing them [5]. Thus giving global optimum living conditions for the ants. This has inspired many algorithms for MAS such as the Ant-Colony Optimization algorithm. [6].

Russel and Norvig define an agent as anything that can perceive its environment (through its sensors) and act upon that environment (through its effectors), also known as a rational agents or intelligent agents. [7] These actions are learnt from interacting with the environment as seen in figure 2.1.

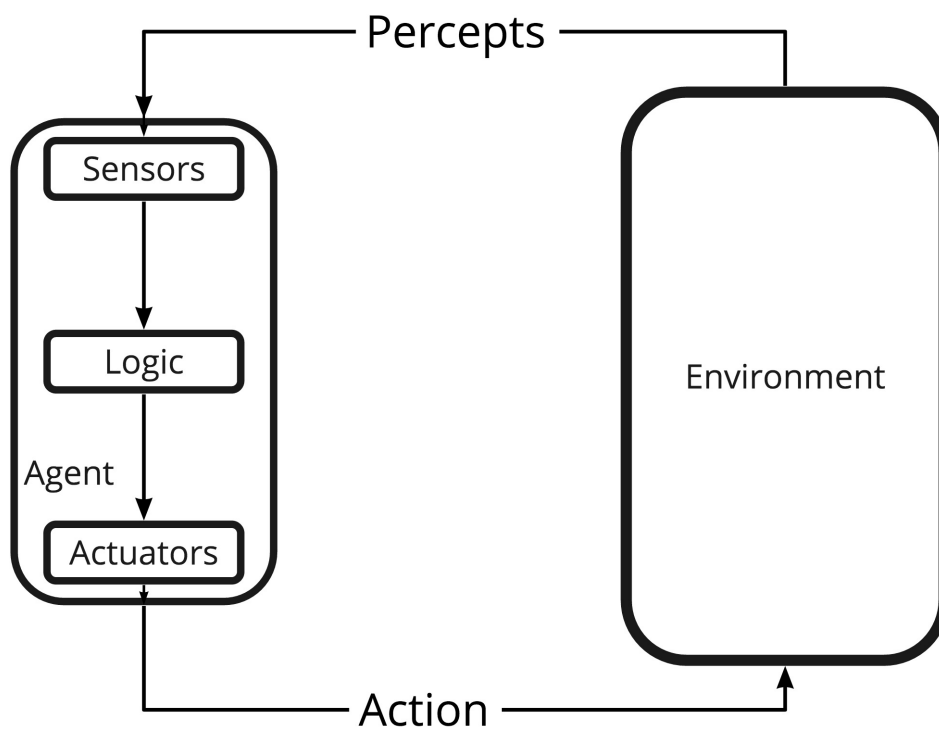


Figure 2.1: RL Process, agent interacting with the environment through sensors and outputting an action with actuators

[1]

2.1.1 Environments

Environments can also be classified into different categories depending on what interactions can occur [7].

- **Fully observable vs Partially observable**

Fully observable environments allows all states of the environment to be observed by the agents. In partially observable only predefined states are available, this can be due to a number of reasons.

■ **Deterministic vs Stochastic**

In deterministic environments, the next state choice is fully determined by the current state of the agent. Stochastic environments possess some randomness, and do not have a defined effect on the next state.

- **Episodic vs Sequential**

A sequential environment uses memory of past events/states to determine its next action. Episodic environments only use the current environment to make a decision.

- **Static vs. dynamic**

A static environment will react only to the agents actions, if the environment changes without input from the agent it is known as dynamic.

- **Discrete vs. continuous**

A discrete environment will have a predefined set of states and actions that an agent can move into. A continuous one will have an infinite number of states and actions.

- **Competitive vs. Collaborative**

A competitive environment will have individual agent goals as well as global goals. For example in Pacman, a ghost agent has a goal to find Pacman, whereas Pacman has a goal to eat as much food as he can without getting caught by a ghost. The global optimum for the ghosts is to find Pacman, however each ghosts competes with the other to increase their own goal. Collaborative environments have only global optimums

2.1.2 Architecture

Three main forms of architectures are defined for MAS, centralised, decentralised and hybrid. Each has its own applications and problems. Centralised systems have been the main architecture in research for years, however more and more research is being

put into decentralised systems which is evident in Section 3.5.2.

- **Centralised**

Centralised architectures have one central agent that controls the rest of the agents. It tells the other agents which actions to take based upon the controlling agents knowledge. The central agent acting as a master and others acting as slaves.

- **Decentralised**

Decentralised approaches do not have a single controller, instead each agent works autonomously. They interact with its environment and other agents autonomously, and make their own decisions. In some decentralised architectures agents can communicate and share information, and in others agents cannot communicate but only observe the effect other agents have on an environment.

- **Hybrid**

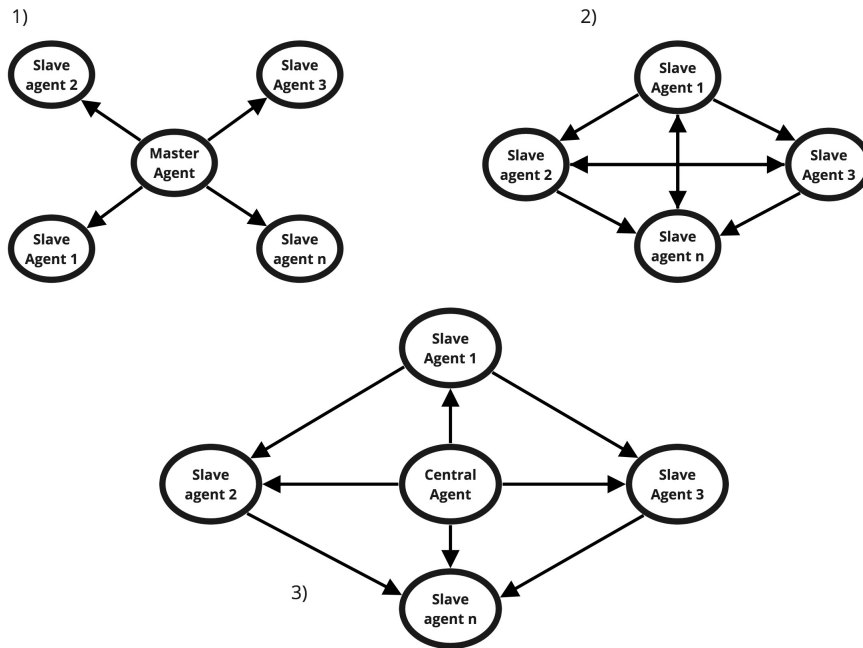
Hybrid architectures can have a central agent that does one task, but all other agents work autonomously for other tasks. They call on the controller to make the decision that they are unable to make. Examples can be found in Section 3.5.2.

This thesis will follow a decentralised approach.

2.2 Machine Learning

Machine learning is a subset of Artificial Intelligence. It is an area of research that involves improving and learning through experience. There are three main forms of learning techniques These are:

- **Supervised Learning** - Supervised learning trains and learns from a labelled historic dataset. It takes the labelled input data and labelled output data and learns the regression or categorization models. It then uses this knowledge to



2.2: Multi-Agent Architectures: (1) Centralised (2) Decentralised (3) Hybrid

make future predictions, based on the inputs it receives [8].

- **Unsupervised Learning** - This learning technique takes unlabelled data and groups together similar data to create clusters. It uses these patterns that it has gathered to categorise new inputs into the categories it has learnt[9].
- **Reinforcement Learning** - This learning technique involves an agent interacting with its environment while working towards a goal. This is discussed in detail in Section 2.3 [1].

2.3 Reinforcement Learning

Reinforcement Learning (RL) is one of the most active areas of research in AI. It is a machine learning approach where an agent evaluates a current state, takes an action and then receives a reward on this action [1]. Training takes a cut-and-try approach to maximize this reward, using positive feedback as a reward and negative feedback as a punishment. The agent learns through interacting with an environment.

2.3.1 Markov Decision Process

RL problems are modelled as Markov Decision Processes (MDP). MDP's are a mathematical representation of complex decision making, it is defined as a tuple of (S, A, P_a, R_a, γ) , they are defined with the following:

- a finite set of actions $(A) \{a_1, a_2, \dots, a_n\}$
- a state space $(S) \{s_t, s_{t+1}, \dots, s_n\}$
- The state transition probability function $(P_a(S, S'))$. This defines the probability that an action in state S at the current time t , will lead to a state S' in the next time step $t + 1$.
- The reward function - $(R_a(S, S'))$ which defines the expected reward that is given having moved from S to S' .
- The discount factor $(\gamma \in [0, 1])$, this holds the weights of futures rewards - discount factor is responsible for weighing future rewards against present rewards.

Typically in RL, an MDP where the transitions and reward functions are unknown is used to train on. The RL algorithm learns these, this provides the knowledge that agents bases their decisions on. The agent interacts with its environment to achieve the goal that they have been assigned. This goal is achieved by maximising the sum of reward throughout the process. The reward is a scalar feedback signal that indicates how well the agent is doing at time step t [1].

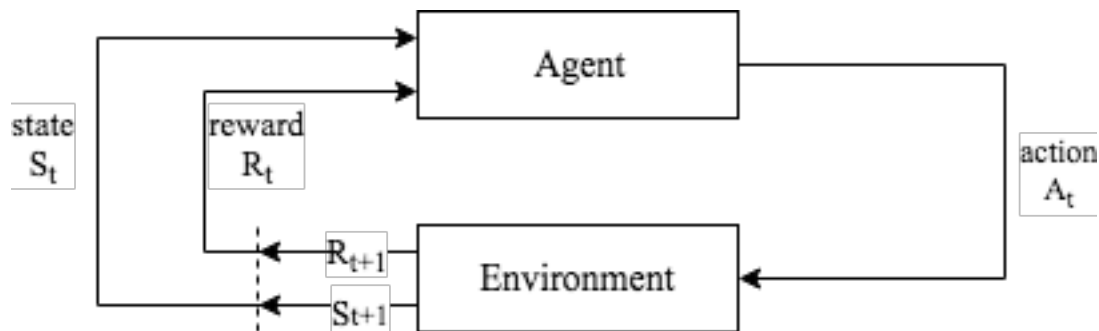


Figure 2.3: The agent–environment interaction in a Markov decision process [1].

2.3.2 RL Characteristics

A RL algorithm may contain one or more of these attributes [1]:

- A model - predicts what the environment will do next, it represents what the agents take on the environment is.
- A policy π , this represents the agents behaviour function ie. what the next action will be based on its state. A deterministic policy is represented by

$$a = \pi S \quad (1)$$

- A value function, this represents the reward that an agent can expect given it takes a certain state next

$$V_{\pi}(S) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots R_{t+n} | S_t = s] \quad (2)$$

2.3.3 Reinforcement Learning Agents

As described in the above sections, RL agents aim to optimize the sum of the reward.

We can categorize most RL agents into 5 categories:

- Value Based Agents - these agents develop their policy by estimating the value function of each possible state. This gives information on how good each action would be in each state, allowing the agent to make the best next action. eg. SARSA [10].
- Policy Based Agents - these agents choose actions based on an estimation of the current optimal policy. SARSA algorithm is policy based. Off policy algorithms use a behaviour policy to select its actions. Q-Learning which will be discussed further in Section 2.3.4 is off policy. [11].
- Actor-Critic Agents are both value and policy based as they use a value and a

policy in learning. An example would be: Advantage Actor-Critic (A2C) [12].

- **Model-Based Agents** - Agents try to understand the environment that its in before it decides its next action. Examples would be: Model-Based Value Expansion (MBVE) [13].
- **Model-Free Agents** - these algorithms directly use a policy or value function without building a model of the environment, examples of model free-algorithm are: Q-Learning and Policy Optimization eg PPO [14] and TRPO [15].

This thesis will focus on Q-learning, a model-free policy based agent.

2.3.4 Q-Learning

Q-learning is an off-policy RL algorithm as explored in Section 2.3.3 [16]. It is one of the most popular RL algorithms. It aims to find the optimal action to taken given the current state that the agent is in, without needing a prior model of the environment [17]. The Q in Q-learning stands for quality. Quality in how useful the action it will take is in gaining a reward. Q-values $Q(S, A)$ hold the agents experiences with interacting with the environment, the Q-table holds the long-term pays outs of taking certain actions A in certain states S the agent is currently in. These Q-values are updated every time step following the formula:

$$Q(S, A) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma(\max_A Q(S_{t+1}, A_{t+1})) - Q(S_t, A_t)) \quad (3)$$

where,

- S_t is the current state of the agent at time t
- A_t is the action the agent took at time t
- α is the learning rate for the algorithm, it determines how weighted new experiences in the new Q-value

- R_{t+1} is the reward given when moving from S to S_t
- γ is the discount factor, as explained in Section 2.3.1, determines how weighted previous experiences are

This is otherwise known as the Temporal Difference, Equation 3 is derived from the Bellman equation [18], shown below:

$$V(S_t) = \max_A (R(S_t, A_t) + \gamma V(S_{t+1})) \quad (4)$$

where,

- S is the state
- A is the Action
- $R(S, A)$ is the reward function that takes in A and S and outputs the reward associated
- $V(S)$ is the value at being at a state

This equation states that we take the maximum value that we can after considering all possible scenarios, which is what Q-learning aims to do.

An alpha value can range between 1 and 0, a value of 1 tells the agent to forget all previous experiences when deciding on the new one. A value of 0 shows that it should remember all previous experiences. A gamma value also ranges between 1 and 0, 1 gives all experiences equal importance whereas 0 gives current values a higher importance.

A Q-table is used to store these Q-values, it is a lookup table where the maximum expected future reward is stored for each (S, A) tuple. The agents goal is to get the highest long-term reward, if it chooses only the highest rewards in the next state it may not find the correct path to find the global optimum reward. In order to do this it must explore many different options to try and find the maximum value. A popular

[REDACTED] to implement this is through the epsilon-greedy method. This method will randomly choose an action to take, the number of times it does this is dependant on how high ϵ is. For example, if we set $\epsilon = 0.1$, then the algorithm will choose the highest reward for the next step that is available to it 90 % of the time. The other 10 % it will randomly choose. This minimizes the chances of the agent choosing what seems to be the best policy, when there is a better one available.

2.4 Deep Reinforcement Learning

Advances have been made in RL towards Deep Reinforcement Learning(DRL). This couples deep learning with RL in order to speed up the process of finding optimums, in other words to make algorithms more efficient. A popular example of this is DeepMinds AlphaGo, an algorithm that plays the game Go. [19] It beat the world champion in Go, and was cited as one of the reasons DRL is becoming more popular. Artificial Neural Networks (ANN) [20] are used as function approximators in RL, used when the action and state space is particularly large and is not known. This makes it almost impossible to use RL to solve the problem, as it would take too much time and data to come to a convergence. The ANN can approximate a value function, or a policy function. Instead of a table holding all of the possible state-action pairs, the neural net trains on samples of the state/action space and predicts if they lead to optimizing the goal in RL. It works similarly to classic deep learning, however it will rank the possible actions that the agent can perform from the state that its in. It maps a policy to an action. This thesis does not use DRL, however it is important to mention as some of the state of the art papers use this method.

3 State of the Art

This chapter reviews published work in the area of shared autonomous vehicles. It will discuss current techniques used to overcome technicalities and problems that are present with autonomous shared vehicles, in particular ride-sharing.

3.1 Ride-Sharing

Globally in 2015 , an estimated number of 8 million people used a form of car-sharing services [21], this number is expected to rise by 450 percent to 35 million people by the year 2021[22]. This increase in demand for shared travel has fueled an surge in research into the areas. The sharing economy, as it is referred to, is "the peer-to-peer-based activity of obtaining, giving, or sharing the access to goods and services, coordinated through community-based online services"[23]. The expansion in the demand for these services comes as the consumers begin to make choices influenced by climate change, sustainability and the economical benefits of using such services compared with ownership. This is particularly prevalent with the younger generations [23, 24].

Aside from consumer influences, urban transport has been pushing for the move towards ride-sharing and autonomous vehicles. The UN estimates that that over 6.3 billion people will live in urban centers by 2050 [25]. A number of cities have conducted studies on how these types of services would affect their cities and help alleviate problems stemming from increase in populations and urbanisation.

Currently cities are close to maximum capacity with public transport. Studies, from Asia [26, 27], Europe [28, 29] and North America[30, 31], show how shared autonomous vehicle services can reduce congestion, decrease CO2 levels from transport and provide shorter wait times for passengers. In recent years more focus has been put on the implementation of these types of services, in particular shared autonomous mobility-on-demand systems. This service allows passengers to request a ride from any location with a drop off location of their choosing. A driver-less vehicle will then service this ride, whilst also picking up or dropping off other passengers along the way. Therefore it is known in wider media as ride-sharing. This has become popular with taxi apps, each launching there own versions for example, UberPool [32] and LyftLine. Other versions have also been released including, Lyft Shared Saver, and Uber Express Pool, which will be evaluated in the next section.

Many operational issues still remain within this area of research. These include, re-balancing vehicles to minimise the time they are empty, the ride-matching optimization where drivers need to be dispatched to requests and how these services will work in real-life situations, among others. [33]. This research paper focuses on the ride-matching optimization problem and how real-world scenarios will affect these services.

3.2 Types of Ride-Sharing

As the scope for these types of problems are vast, attempts have been made to simplify the complexities. One complexity is where the passenger will be picked up. Currently, standard urban transport has stationary hubs where passengers must go to in order to get onto public transport, eg bus stops, train stations or shared bike stands. This reduces the number of stops a vehicle needs to make, therefore reducing travel time for everyone on the bus. Taxis however differ, they pick passengers up wherever they request. Ride-sharing uses the same concept, which causes large scope

for how passengers should be allocated to each vehicle to optimise cost and time for both passengers and the driver.

3.2.1 Meeting Points

To reduce the number of stops that a vehicle must make in order to fulfil multiple ride-sharing requests, suggestions have been made to introduce dynamic meeting points [34]. These meeting points would be calculated at a mean distance between two or more passengers within a given distance range, passengers would meet here so that the vehicle can pick them all up at once. This type of service has its benefits in reducing time and distances that a vehicle must travel if it were to pick each passenger up individually.[2] However, it also reduces the benefits to the passenger as they have to walk to find the car, which is what a traditional taxi's unique selling point is compared with public transport.

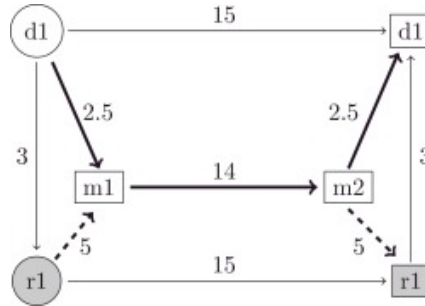


Figure 3.1: Example of calculated meeting points for ride sharing, d=driver, r=rider, m=meeting point [2]

Ride-Sharing companies have started to adopt this strategy as well, Uber in 2018 introduced 'Uber Express Pool' in 12 US cities. It costs about 30% to 50% less than UberPOOL [32], as users must walk to join other passengers at a common pickup and drop-off spot [35]. Another popular app, Lyft, has recently launched its own version 'Shared Saver', with the same dynamics as UberPool Express. This type of service, while innovative in reducing stops for vehicles, is not what the focus of this paper will be. It will instead focus on dynamic stop requests, this is the type of service that shared autonomous vehicles will most likely be first adopted into [36]. Proven by the

usage of this type of service in current markets. Therefore, implementation issues must be thoroughly researched before roll-out.

3.2.2 Dynamic Requests

Dynamic requests have been the method of choice for taxi services since adoption. The use of a taxi over public transport is mainly due to usefulness and reduction in travel time [36]. A dynamic request consists of a rider deciding the pickup and drop off location. This is how current non-autonomous ride-sharing apps work, UberPool and Lyft Line are the most common [24, 32]. Most of the research areas of request-matching optimization has focused on this style of services, for reasons mentioned in Section 3.2.1. This style of service gives riders the biggest range of flexibility, and adds to the quality of the service.

3.3 Shared Autonomous Vehicles in Ride-Sharing

As technology has improved over the years, more focus has been put onto autonomous vehicles. In particular, how they can transform urban transportation. As mentioned in Section 3.1, many large cities have completed studies on how shared autonomous mobility-on-demand will affect them [30, 31]. Ride-sharing companies have begun to roll-out tests in cities with driver-less vehicles to assess the usability of such a service [37]. While this tests the actual ability of a self-driving car to take a passenger from one point to another in a city. It is yet to be applied to a ride-sharing service. In order for this to happen, operational issues must first be solved. A key issue is the request-matching optimization, which is the focus of this paper.

Currently, most taxis are with a human driver. These drivers have experience in their cities, and have knowledge from doing the job for so long. We must ensure that this knowledge is transferred in some form to autonomous vehicles when the transition is made. After this transition, operating a fleet of mobility-on-demand vehicles becomes

more efficient as the variable of human error is eliminated, leading to cost savings and reduced emissions.

3.4 Request Matching Optimization

As discussed in the above sections request matching optimisation is a key operational issue that must be solved before a Shared Autonomous Mobility-on-Demand (SAMoD) system can be implemented into a public space and succeed. Basic approaches include using an algorithm to determine how many shared miles two riders could share, and match them based on maximising the profit for the driver [38]. More advanced research explores how machine learning and reinforcement learning can optimize this. The subsections below review current research and state of the art implementations.

3.4.1 Machine Learning

Machine learning techniques have been used to optimize the request matching problem. One paper suggests a best first algorithm that computes the top drivers that would suit a riders request [39]. This is based upon calculated probabilities of a rider's journey sharing with the current requests that a vehicle is undertaking. The driver with the highest probability is then assigned to the rider.

Other linear models have been created to solve matching optimization, Alonso-Mora and his team suggests using historical demand data to route vehicles in a way that they will take the optimal route to come across another request [40]. The algorithm predicts future demand from the past and current demand, so that the vehicle can make an informed decision about its route. The data comes from the New York City open source Taxi dataset [41], which a lot of research papers use. Another paper focuses on how a probabilistic approach can be used to maximise the number of passengers in each vehicle [42]. The difference being that this paper divides the area

into a grid and estimates how many requests are estimated to travel from region v to region u . Thus giving it a more tailored route to each specific vehicle.

While these approaches provide optimizations in profit and number of passengers in each car at one time, it does not provide a sustainable implementation. It would be difficult to scale this approach up when demand for these services increase. This would be due to the increase in the number of vehicles and the compute power that would be needed to support this at a central location.

3.5 Reinforcement Learning

Reinforcement learning has been up and coming in this space. It is an obvious choice for the application of SAMoD as vast amounts of data can be required and simulations are possible. This section will cover the application of reinforcement learning to ride-sharing as well as deep reinforcement learning.

3.5.1 RL in Ride-hailing

Many approaches have been taken to solve and optimize the problem of request-matching in SAMoD. Traditional reinforcement learning techniques have been applied in algorithms to solve the request-matching problem. Centralised and decentralised approaches have been done. AI Labs and Didi Chuxing suggest modelling this process as a 'large-scale sequential decision making process' [43]. It proposes to have two stages to the process, the first, ie. the offline learning stage, summarizes supply and demand based on historical data. This outputs a spatio-temporal quantization, that contains a value function of the expectancy of a driver being in a particular state. This is then used in the second step. This is the online planning stage which uses a markov decision process (MDP) to match requests with drivers using the value function. It uses a centralised decision maker that delegates to all agents. This can be seen in Figure 3.2. This does not take into

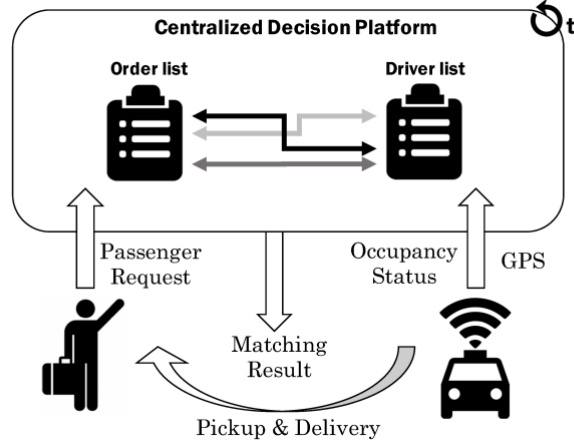


Figure 3.2: Example of an offline training, online decision making platform [43]

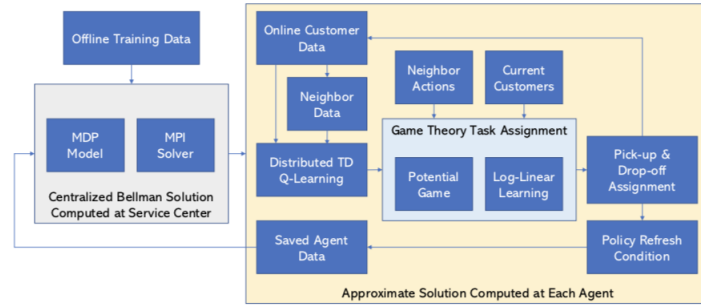


Figure 3.3: Distributed Learning based Algorithm [44]

consideration ride-sharing, however it shows how beneficial reinforcement learning can be in optimizing the goal of an entire system of vehicles.

A paper that proposes using a MDP with Bellman updates for solving request-matching with urban ride-sharing [44]. A central node computes a model from historical data, this is then sent to each agent which computes its own solution using Q-Learning and game theory task assignment, as seen in Figure 3.3. At each agent, local information is used to determine optimal and appropriate requests that it should fulfil. This hybrid solution, allows for the best of both worlds regarding centralised and decentralised approaches. However, with a centralised entity it will not scale well when more agents and requests are added.

3.5.2 RL in Ride-sharing

As proposed in this thesis, real-world attributes can enhance a reinforcement learning algorithm in regards to ride-sharing. It can be difficult to use data in this application as the scope can be huge, ie, travel time from one point to every other point in a city. This needs an almost infinite amount of compute and storage capabilities. It has been proposed to use mean-field multi-agent reinforcement learning to solve this matter [45]. Simplifying local interactions by using mean field approximation is proposed, this is done per grid space allowing the compute and storage to be decreased drastically. Average action of the interaction between neighbourhood (grid space) and taxis (agent) is computed. Interaction is the volume of requests per capita in each neighbourhood. A centralised critic is used during training to rate agents decisions on pickups/drop offs and to update their policies. After training, each agent works independently to choose and execute requests. This achieved a higher order response rate and a lower supply-demand gap.

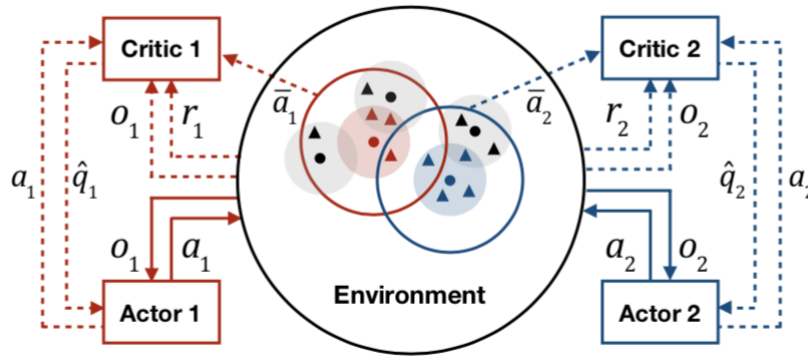


Figure 3.4: Mean-Field MARL Framework [46]

Completely decentralised approaches have also been proposed [47]. Here, decentralised reinforcement learning is used for dispatching and for re-balancing of vehicles to meet demand in ride-sharing. Q-learning is used by each agent to determine its optimal behaviour, re-balancing and request-matching, based on current and historical demand. NYC taxi data is used for historical and current

demand. Each agent can then work independently to re-balance and fulfil requests.

This algorithms lead to global decreases in riders wait times, and a higher level of service. The paper also highlights the trade-offs vehicles must make in order to provide a service. Trading off individual rider travel time for example to decrease overall wait times.

3.6 Deep Reinforcement Learning

Deep reinforcement learning is the next step in reinforcement learning. It has been used in a number of proposals in ride-sharing. A 2019 paper from Didi Labs suggests a multi-agent deep reinforcement learning approach to solve order-dispatching through order-vehicle distribution matching [46]. It uses a grid-based order system, similar to this thesis. They suggest using KL-divergence optimization at each time step to balance vehicles to areas of high demand. They aim to maximise total driver income and the order response rate, striking a balance between these two metrics is difficult as the driver can only go for high cost rides creating a higher salary but would decrease the overall satisfaction of the other requests as they wouldn't be filled. It is a decentralised approach with all agents making their own decisions, this is to ensure the system is capable of dealing with a high influx of requests as the system grows. The learning process is centralised, similar to the paper mentioned in subsection 3.5.1, which is then passed onto the independent agents once the system is live. Thus, a somewhat hybrid approach, shown in figure 3.5

It develops a Q-learning network with a state-action input, to work out the Q-value. As seen in figure 3.6, it inputs the state and action before concatenating them together in a hidden layer.

Another approach is called DeepPool, it is a distributed model-free algorithm that learns optimal dispatch policies from its environment [48]. It aims to incorporate

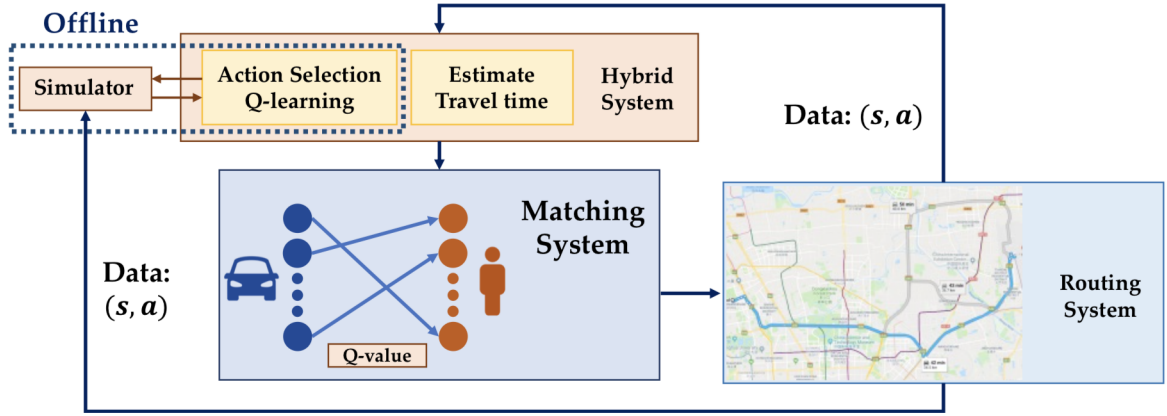


Figure 3.5: Hybrid Decentralised RL approach in SAMoD [46]

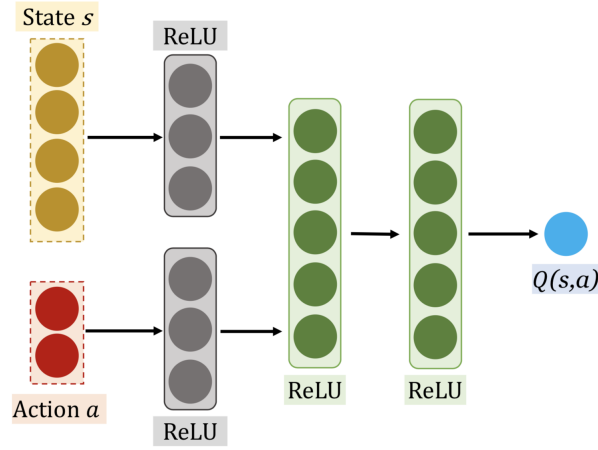


Figure 3.6: Action Selection Q-learning [46]

demand into its algorithm to manage dispatching decentrally. Each vehicle solves its own DQN problem, (Deep Q-Network), without coordinating with other vehicles which reduces the complexity of the system. Each vehicle assigns itself to a request, which also takes into consideration ride-sharing. The framework reduces the amount of vehicles needed to serve its requests with the same quality of service as with the extra vehicles.

Model-free approaches have become popular in recent years, a paper using this approach suggests a different way to deal with the complexities of ride-sharing. A multi-hop approach is developed, in which riders can use ride-sharing to get from

A-B but may change vehicle up to two times throughout this journey [49]. It uses DRL to find the routes and hops for each passenger. This approach does not seem realistic with the standard ride-sharing application. It defeats the purpose of usability for passengers, and would be more suited for a public transport journey generator.

This thesis suggests using a model-free decentralised approach to solve the request-matching problem. In evaluating the research space, it shows there is very little focus on real-world applications of ride-sharing systems, and how real-world attributes may affect them. This is the motivation that thesis will use to focus on how traffic data can be utilised to provide a quality service for users.

4 Design

In the previous chapter state of the art approaches to the topic request-matching in shared autonomous vehicles were explored. From this we can gather that whilst the research area is vast and many different approaches have been applied to the problem, there is also a lot of room for steps forward. This thesis hopes to address this by focusing on how real world attributes will affect request matching. Little work has been done in this area, although it is something that must be looked at before a fully autonomous shared mobility system is rolled out into a city or area. We will focus on a decentralised approach as this is the most advanced approach in the space with not a lot of research done on it, however it holds a lot of potential. This chapter will define the design aspects of this research and aspects taken into consideration will be defined.

4.1 Request Matching Optimization Problem

Request matching optimization can be translated into a RL problem where we define the agent, a vehicle, interacting with the environment which is a city's road network. The agent observes a state $s_t \in S$ at the beginning of a time step t , then it chooses and actuates an action which corresponds to one of the actions a taxi can take, for example pick up a passenger. After taking the action, the agent transitions to the next state and receives a reward. The next subsections define these components. An episode is complete for an agent when a ride has been completed. The goal for each

agent is to maximise the reward, ie. fulfil as many requests as possible. The next sections define and describe this in more detail.

4.2 State Representation

When defining the observation space we must look at two things, the agents state and the state of the environment.

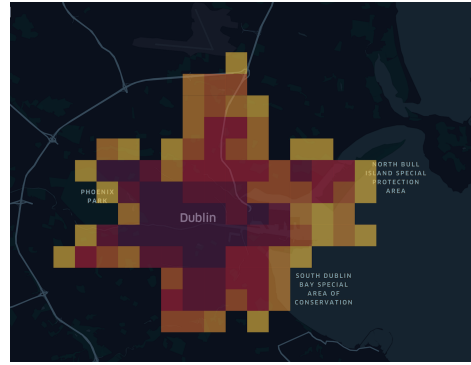
The state of the environment is defined as how heavy the traffic flow is in a certain area. The hypothesis being that if the traffic is higher it will take longer to drive through it. As seen in Figure 4.1b, we can see congestion in a grid space in Dublin. To reduce the observation space we define each postal code as a grid space. We can see an example of this in Figure 4.1a. This not only reduces complexity for the observation space, but it also makes it easier to apply to any city by simply using the postal code. To simplify the state space, and to reduce the complexity we define a city's district as a separate grid space. Each grid space will have a traffic flow level that will be defined as low, medium and high (LMH). This again is to reduce complexity, making the flow discrete changing from a continuous integer flow rate. The levels of low, medium and high will be relative to the flows that we see in the city at each time frame. We can see a sample of this in Figure 4.1c. This type of sectioning is common place in shared mobility studies, as explored in Section 3.

To calculate these matrices, for which level (LMH) it will fall into, the average traffic flow is calculated per district. Each postal code is then compared with all others at each time frame and is given its level relative to all the post codes.

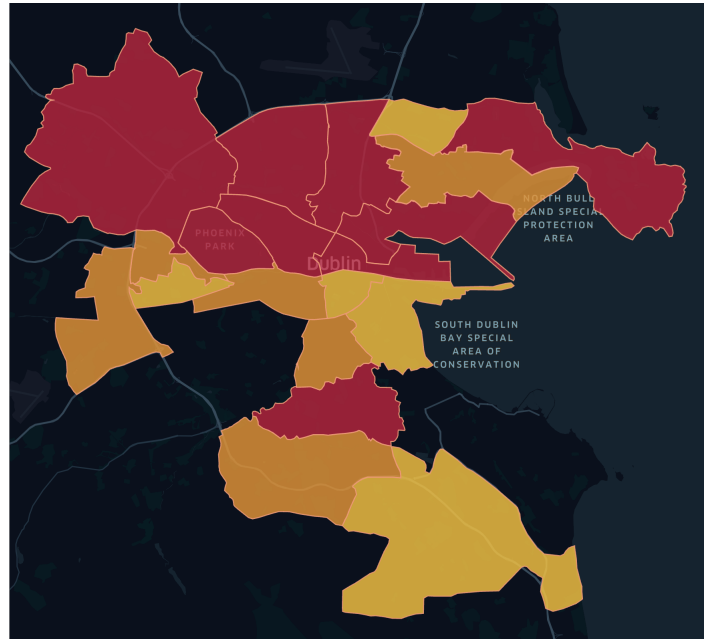
The states of the taxi define the occupancy of the taxi, states being 'no passengers', 'has passengers' and 'full'. Each being essential for this research as the mobility is shared, the agent needs to know if they have room for more passengers and can still accept rides. If it was not shared, it would only search for new requests once it picks up one request.



(a) Dublin postal codes



(b) Sample of traffic flow in an area



(c) Sample of Dublin postal districts with congestion

Figure 4.1: Creating the observation space

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(a) Sample postal code matrix

$$\begin{bmatrix} Low & High & Low \\ High & Medium & Medium \\ Low & Medium & Low \end{bmatrix}$$

(b) Sample congestion matrix

Figure 4.2: Creating the observation space

4.3 Action Space

In RL, once the agent has observed its state it chooses an appropriate action. The action space A is the defined set of actions that the agent can make. This simulation will have three actions that can be made. These include: accepting a request, declining a request, finishing a request and do nothing. $A = \{A, D, F\}$. Each of these actions triggers the agent to complete this action. The agent, in this case the vehicle, will trigger one of these actions depending on the state. Once a person has requested a trip the agent decides on what it should do with it. By accepting a request the vehicle must travel from its current destination to the location of the pickup.

A vehicle should accept a request in a number of situations, when it currently is not serving any trips or if it is currently serving a ride but it has room for more passengers. The agent will learn which requests it should pickup depending on how the traffic in the areas will affect the metrics of the ride. A vehicle should decline a request if the vehicle is at full capacity or if it decides it would impact its current request too much. Finishing a request involves the vehicle dropping a passenger off at the requested location.

4.4 Reward Function

The reward function in an RL algorithm proves to be the most difficult to fine tune. Given the MAS RL architecture that has been chosen, as described in Section 4.5.1, each agent will have its own reward. This means there will not be an overall reward for the entire environment, instead each agent will aim to maximise their own reward. Once an agent completes a trip by dropping off a set of passengers it will be rewarded a score of 100. The goal being to drop off as many passengers in the set time, by doing so each agent should learn when to pick up a passenger and when not to.

4.5 System Design

This section will define how the RL simulation will look like, and give more details on the architecture, and data needed.

4.5.1 System Architecture

In Section 3, it can be seen that many different approaches are used. The main three being centralised, decentralised and a hybrid architecture, shown in Figure 2.2. Each has its own benefits, centralised architectures have been the main system used for years in research and industry. For example, in a taxi company a central office takes calls and sends the request to an appropriate vehicle. More recently, a decentralised approach can be seen in industry. Companies, such as Uber, have pivoted from this to a decentralised approach. This project focuses on a decentralised approach. This means that each agent will work independently of one another. They will work in the same environment with the same observation space, as the congestion level in each postal code will be the same for each agent. The agents can't communicate with each other or share information.

Each agent will make its own decisions on which requests it will accept. This decision process will be learned and based on the q-learning algorithm implemented. Each agent has access to all the requests, based on a first come-first serve approach. Whichever agent picks up the request first will be given the request. This algorithm should train itself to pick these requests optimally.

As the system is decentralised each agent will have its own policy that it has learned through its experience in the environment. This multi-agent algorithm will use a library that allows multi-agent RL, giving each taxi its own q-table and opportunity to update this individually. Each agent will have its own goal and reward as mentioned in Section 4.4.

4.6 Data

The project requires two data-sets to run the algorithm. These are the request dataset which will contain a list of taxi requests within a range of time. We also require a dataset for the observation space that contains information about the congestion levels in each area. Each are described below.

4.6.1 Request Data

As mentioned above a request dataset is needed to allow the taxis to have requests to pick up. There are currently no open data sources for Dublin, as discussed in Section 3, most other papers in this space use an open source dataset from New York [41]. As this is tried and proven in these papers, whilst also containing all the information that is needed for this research, a dataset for Dublin will be created based on the NYC dataset.

	id	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag
0	id3004672	1	30/06/2016 23:59	1	-73.988129	40.732029	-73.990173	40.756680	N
1	id3505355	1	30/06/2016 23:59	1	-73.964203	40.679993	-73.959808	40.655403	N
2	id1217141	1	30/06/2016 23:59	1	-73.997437	40.737583	-73.986160	40.729523	N
3	id2150126	2	30/06/2016 23:59	1	-73.956070	40.771900	-73.986427	40.730469	N
4	id1598245	1	30/06/2016 23:59	1	-73.970215	40.761475	-73.961510	40.755890	N
...

Figure 4.3: Sample NYC dataset [41]

Variable Name	Description
requestID	integer with a unique id number
pickUpLat	latitude coordinate for pickup location
pickUpLong	longitude coordinate for pickup location
dropOffLat	latitude coordinate for drop off location
dropOffLong	longitude coordinate for drop off location
numPeople	integer, number of people per request

Table 4.1: Table showing column titles and descriptions of request data

Due to the fact that there is no open dataset, the dataset that is curated must have a realistic demand of requests. In order to do this traffic data is used to determine the

probability of a request popping up at a certain place and a certain time. As we have divided out the map of Dublin into postal code sections, we can estimate the number of taxi requests that start and end in each section at a given time frame, eg, every 15 minutes. This is done by using traffic flow data that is averaged over each section. We estimate how much of this traffic flow is taxis by multiplying by an assumption named taxi-rate. We then randomly generate a location in the area and a random time within that 15 minute time frame. We do the same to get a location coordinate. The algorithm to do this can be seen in Figure 4.6.1.

Algorithm 1 Request Data Creation PseudoCode

Calculate congestion value in each postal code at each timeframe
 Multiply by taxi-rate
 For congestion value at each TimeFrame and PostalCode lgenerate pickup coordinate
 generate drop off coordinate
 generate timeframe
 generate how many people

Algorithm 2 Pick up request data creation algorithm

Read in traffic flow data
while *file is not empty* **do**
 | iterate through each traffic sensor at each time
 | determine which postal code
 | determine which time frame
 | add flow value to current total for area and time-frame
end
 multiply each congestion value by taxi-rate
while *time frame != NONE* **do**
 | **while** *postal code != NONE* **do**
 | | **while** *congestion count != 0* **do**
 | | | generate random pickup location in postcode
 | | | generate random time within time-frame
 | | | generate drop off location
 | | **end**
 | **end**
end

To get an idea of the number of passengers that each ride would require, an assessment of the NYC open taxi dataset was done. We can see in Figure 4.4 that the majority of all requests have only one passenger. These values were converted into

probabilities to allow the Dublin dataset to have a realistic passenger count, as seen in the Table 4.2.

Passenger Number	Probability
1	0.8
2	0.01
3	0.05
4	0.05

Table 4.2: Table showing column titles and descriptions of request data

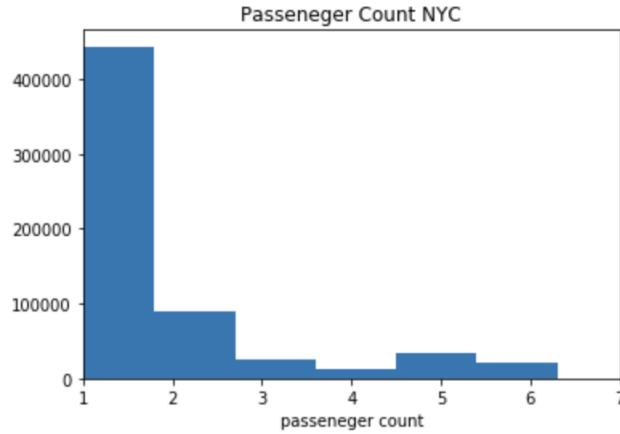


Figure 4.4: Histogram showing the passenger count in NYC requests [41]

4.6.2 Congestion dataset

As mentioned in Section 4.2, a dataset must be created for the observation space. This must include congestion information on each postal code at any given time frame. As seen in figure 4.1b, each postal area in Dublin has a congestion level. These levels are *Low*, *Medium*, *High*, represented by integers 1, 2, 3 respectively. In order to build such a dataset again traffic flow data is needed. A data set must be created that has each of our postal areas and for each time step a corresponding congestion value. In this project we have used a time step of 5 minutes. A sample of this can be seen in Figure 4.6.

To build this dataset an average traffic flow is calculated for each postal code per time step. Then at each time step, all 9 values (9 postal codes in this sample) are compared

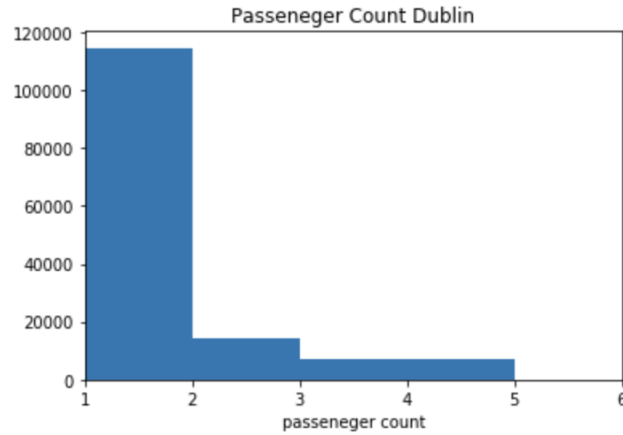


Figure 4.5: Histogram showing the passenger count in Dublin requests

	postal_code	01/03/2012 00:05	01/03/2012 00:10	01/03/2012 00:15	01/03/2012 00:20	01/03/2012 00:25
0	1	3	3	3	2	2
1	2	3	3	3	3	3
2	3	2	2	2	1	2
3	4	3	2	2	1	1
4	5	2	1	1	2	2
5	6	2	1	1	1	1
6	7	3	2	1	1	1
7	8	1	2	1	1	1
8	9	1	2	1	1	1

Figure 4.6: Sample of a congestion dataset for 9 postal codes in Dublin

and are then given a score from 1-3 relative to the other scores. In figure 4.7 it shows how the traffic flow is converted to the relative level for each postcode. In this sample the levels shown in Table 4.3:

traffic flow	relative level	High/medium/low
0 - 3	1	Low
4 - 6	2	Medium
7 - 9	3	High

Table 4.3: Congestion Levels

In Algorithm 3 4.6.2 pseudo code is available to calculate relative levels.

This should be repeated for each time-frame in the test set that is being used in order to build up a complete dataset for the q-learning algorithm.

	postal_code	traffic_flow (cars/5min)	relative_level
0	1	7	3
1	2	9	3
2	3	4	2
3	4	7	3
4	5	5	2
5	6	5	2
6	7	9	3
7	8	1	1
8	9	2	1

Figure 4.7: Sample of a congestion creation from traffic flow

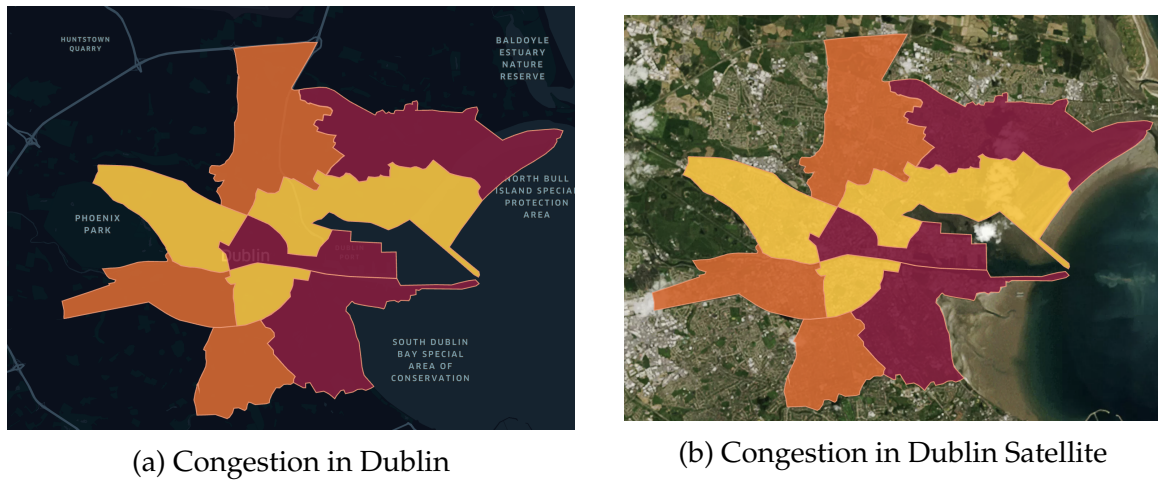


Figure 4.8: Congestion levels in Dublin on 2012-03-01 01:00

4.7 Scenario Testing Design

In the NYC data set information such as drop off time and call time are available. This gives readily available metrics that can be compared without the need for extra scenarios. For example, wait time and travel time can be calculated and used to compare against these metrics from the algorithm suggested in this project. Due to the fact that this information is not available for Dublin we must have some scenario that the algorithm that can be compared to. There are two possible ways this can be done. The first being comparing the travel time from the simulation to a calculated travel time from openstreetmap.org. This would be a time calculated without being in real-time, meaning that each would only be an estimation. For this reason the main

Algorithm 3 Congestion data creation algorithm for state space

```

while file is not empty do
  iterate through each traffic sensor at each time
  determine which postal code
  determine which time frame
  add flow value to current total for area and time-frame

for ( each time-frame ) {
  compute relative levels
  add relative level to each postcode
}

```



(a) Evaluation Scenario 1, with congestion

(b) Evaluation Scenario 2, without congestion

Figure 4.9: Evaluation scenarios that show what states are observable by a single agent

evaluation scenarios will be comparing the algorithm with a similar algorithm that has a smaller state space. The reduced state space only including the taxi states. This will allow a comparison on how the traffic information being introduced affects the systems performance. This will be described in more detail in section 5, an overview of this can be seen in Figure 4.9. The system will have a boolean value to trigger either scenario.

4.8 Algorithm Design

We must define how the algorithm is designed. The algorithm below shows how each agent will carry out it's task of accepting requests. In Section 5, a more in depth outline is available.

Algorithm 4 Agent Learning Process

```
env ← Environment()          /* env is an instance of the environment */
                                /* begin learning episode */
state ← env.getStates() action ← qlearning.pickAction()
if action == "accept" then
  | pickupPasseneger(RequestID)
else
end
if state == "full" then
  | listenForRequests()
  | pickupPasseneger(RequestID)
else
end
dropOffPassenger(RequestID)
updateQLearning(RequestID)
updateMetrics(RequestID)
```

We can see that the agent pulls the states from the environment, before then triggering the q-learning algorithm to give it its next request. If this request is pickup then it will go to that location. If the state of the taxi is not full while this is happening then the taxi will continue to listen out for new requests. This is followed by the vehicle dropping off the passenger and getting its reward.

5 Implementation

5.1 RL Libraries

To implement this system a number of RL and external sumo libraries were used. A library called SUMOoD [50], that was built to test on-demand bus systems, was used to base the system implementation for this project. This was necessary as in order to provide a shared system, each passenger and vehicle needed to be tracked separately. This was to accommodate the shared aspect as well as metrics to be easily tracked as TraCI does not provide this level of detail on a per passenger basis. The main libraries are described below.

5.1.1 Vehicle Class

This library holds information about the agent (the taxi). It holds the vehicle's state as seen in ???. It also holds and controls the plan of the taxi, this means if an agent decides to pick up a passenger a stop is added into its plan. If another passenger is picked up, this is queued into the vehicles plan. It holds all information relating to a single agent such as its total capacity and current capacity.

```
class VehicleStatus:
    """ status of vehicle,represents state of the taxi """
    Empty = 0
    HasPassengers = 1
    Full = 2
```

5.1.2 Passenger Class

The passenger library holds information relating to a single passenger. In order to pull accurate metrics from each passenger, each is given a status. These let agents know if they are available for pickup, or if another vehicle has already accepted their request. A passenger request will time out after 10 minutes, this is another status that allows the system to keep track of how many passengers time out.

```
class PersonStatus:
    """ state of a person """
    UNALLOCATED = 0 #request waiting to be accepted
    ALLOCATED = 1 #request is accepted by a taxi
    TRANSIT = 2 #passenger is in a taxi
    ARRIVED = 3 #passenger has arrived
    TIMEOUT = 4 #request has been timed out
```

This class has set and get functions that determine the variables seen in Table 5.1.

variable	description
personID	unique identifier for each passenger
requestTime	time the passenger enters the simulation
pickupTime	time the passenger/s enters the taxi
dropoffTime	time the passenger is dropped off
origin	pick up location
destination	drop off location
status	status of the passenger
numPeople	lnumber of people in the request

Table 5.1: Passenger Details

5.1.3 Stop Class

This class holds information on what type of stop the taxi is making, this allows the system to know and track what each taxi is doing. It also helps the taxi know what type of stop it is making.

```
class StopType:
    """ Represents the different type of stops """
    PICKUP = 1 #stop is a pickup
    DROPOFF = 2 #stop is a drop off
    WAIT = 3 #the taxi is waiting at a location for a request
```

5.2 Multi-Agent Environment Set Up

The implementation of this RL algorithm is based on a gym environment. This RL algorithm when implemented with a single agent was built as a custom OpenAI gym environment. This was then upgraded to an RLlib environment [51]. This allows an OpenAI environment to be used if a single agent is called, and a RLlib environment otherwise. The environment itself must subclass the MultiAgentEnv interface as seen in Figure 5.1, which can return observations and rewards from multiple agents per step, as seen in Figure 5.2. RLlib has DRL algorithms that can be implemented however for this project we will only use it to control the multi-agent observations and rewards.

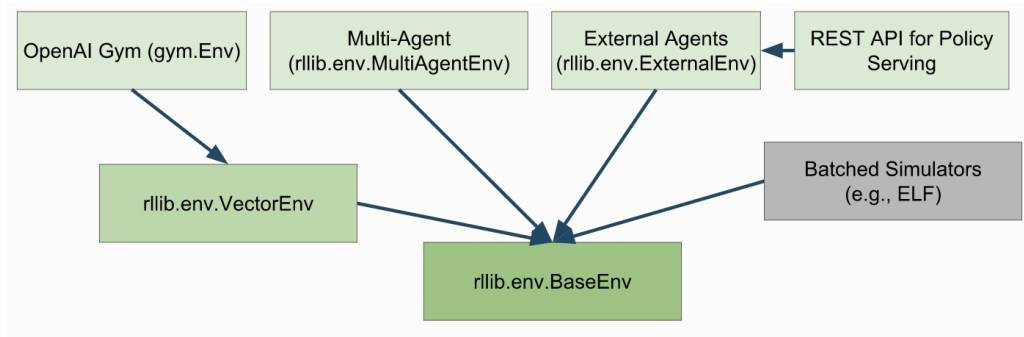


Figure 5.1: Hierarchy of RLlib library

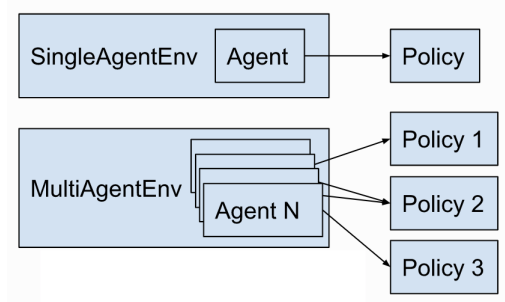


Figure 5.2: Multi-Agent RLlib control

5.3 QL Agent

The QL-Agent implementation is shown below. It uses an epsilon-greedy exploration algorithm that uses an epsilon value of 0.05 and a decay value of 0.99. It uses a standard q-learning set up to update, set and choose actions and rewards.

```
def __init__(self, state_space,
              action_space, alpha=0.5, gamma=0.95,
              exploration_strategy=EpsilonGreedy()):

    self.state_space = state_space
    self.action_space = action_space
    self.action = None
    self.q_table = {self.state:
                     [0 for _ in range(action_space.n)]}
    self.reward = 0

    def getAction(self):
        self.action = self.exploration.choose(
            self.q_table, self.state, self.action_space)
        return self.action

    def update(self, next_state, reward, done=False):
        if next_state not in self.q_table:
            self.q_table[next_state] =
                [0 for _ in range(self.action_space.n)]

        state = self.state
        s_next = next_state
        action = self.action

        self.q_table[state][action] = self.q_table[state][action]
            + self.alpha*(
                reward + self.gamma*max(self.q_table[s_next])
                - self.q_table[state][action])
        self.state = s_next
        self.reward += reward
```

5.4 Data

One of the most important variables in a RL project is data. RL needs a vast amount of data to simulate with in order for it to learn. As described in Section 4.6 two data-sets were built. Both data-sets that were built require a traffic flow dataset for Dublin. This information was not easily accessible. This proved to be the most difficult part of the project. However, a SCATS dataset from 2012 was acquired. This was used to build both data-sets, as described in Section 4.6.

5.4.1 SCAT data

'SCATS® (Sydney Coordinated Adaptive Traffic System)' is an adaptive urban traffic management system that synchronises traffic signals to optimise traffic flow across a whole city, region or corridor' [52]. In essence this means that traffic flow sensors are attached to traffic lights throughout the city, and record traffic flow 24 hours a day. The main use for this is to optimize traffic light signal timings, however we are only interested in the data side of this system. A SCATS dataset for Dublin from January 2012 through to April 2012 was available from Dublin City Council. For each date in the dataset two files were available, a .gps file and a .dis file described in Section 5.4.2 and 5.4.3.

In Dublin there are 740 traffic flow sensors as seen in Figure 5.3. These cover approximately $180km^2$ in county Dublin, however the network for the simulation covers only approximately $40km^2$. This covers most of inner city Dublin which can be seen in Figure 6.3 in our network map file. It covers the dates from January 2012 to April 2012, in order to get a full year these dates were extrapolated and duplicated for the remaining 9 months.



Figure 5.3: SCAT sensor locations in Dublin

5.4.2 GPS File

The first file is a .gps file, which contained the location information for each sensor.

The .gps file contains the variables seen in Table 5.2.

variable	description
streetSegId	unique identifier for a street segment
armNumber	ID for the arm on a street segment
lato	latitude in WGS84 20 meters into the arm
longo	longitude in WGS84 20 meters into the arm
latd	latitude in WGS84 of the centroid of the intersection
longd	longitude in WGS84 of the centroid of the intersection

Table 5.2: GPS file variables

The only information that is needed from this file is the streetSegID and the latd and longd. Each street segment can have multiple arms as it can have multiple traffic lights on one street section, for this reason we only need the location of the street segment and can average the flow over the multiple arms in each street section, as mentioned in Section 4.6.

5.4.3 DIS File

The second file is a .dis file, which contained the location information for each sensor. DIS extension is used for storing workbook files composed spreadsheet files created with Oracle Discoverer software. Both this and the .gps file were opened and converted into pandas dataframes for easy data cleaning and transformation. Pandas is a python library used for data manipulation and analysis [53].

The .dis file the variables seen in Table 5.3.

variable	description
streetSegId	unique identifier for a street segment
upperTime	timestamp of the 5 minute interval
armNumber	identifies the arm of the street segment
aggrateCount	aggregated volume count on that arm
flow	flow ratio

Table 5.3: DIS file variables

5.4.4 Data Pre-processing

The streetSegId and armNumber were used as 'join variables' between the two files. Before the data-sets could be built according to Section 4.6, a lot of pre-processing was needed.

Algorithm 5 Data Preprocessing

```

for ( Each date ) {
    Read in GPS and DIS Files
    Drop duplicate rows
    Drop longd, latd,aggreagateCount, armNumber', 'armAngle',
    Convert upperTime to pandas datetime
    Preform left join on streetSegID
}
if flow is NA then
    Average flow from available time-step before and after
else
    
```

After pre-processing a clean dataset is now available for transformation into the csv files described in Section 4.6.

5.4.5 Request File

The request mentioned previous, was built to have data to contain one day a week for one year. Every Tuesday from 2012 has been chosen, this is due to the stability of this day for traffic. Spikes in traffic, or unusual traffic is rare unlike for example on the weekend. On each day a time frame of 3 hours has been chosen, this time is from 3pm to 6pm. This time chosen as it has some sort of increase and decrease in traffic flow due to the rush hour.



Figure 5.4: Sample of Request trips in 03/01/2012 – 4-5pm

In the request file we have 53947 requests over 52 days, this averages to around 1,037 requests per 3 hour time slot. This gives around 345 requests per hour. This is a reasonable number of requests per hour in Dublin. New York has an average of 868 requests per hour with 13,500 taxis [54], an average of 345 per hour in Dublin is reasonable given that Dublin has 16,000 taxis [55] and a smaller population. In Figure 5.5 we can see a cyclical trend over 4 months, this is due to the flow data only being

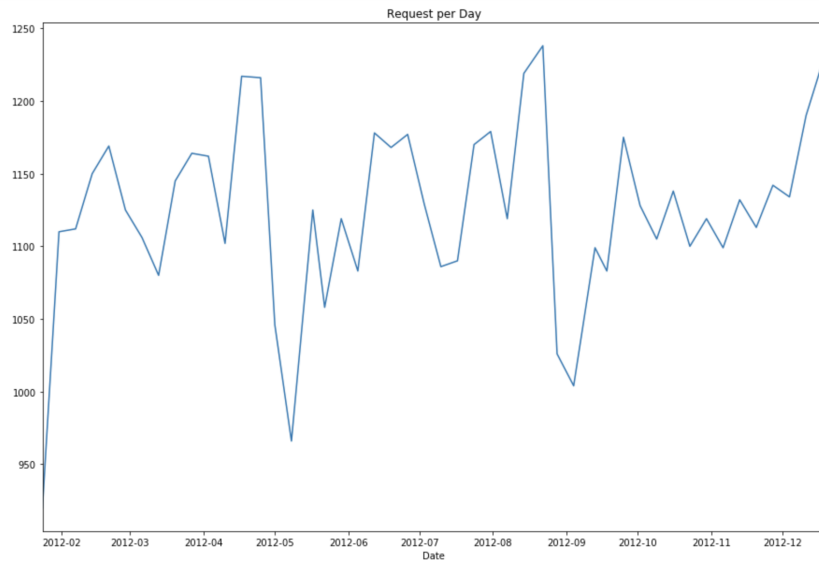


Figure 5.5: Number of Requests Per Day over one year

available for January through April meaning that the other months were built on the flow trends from these 4 months. Whilst this is not ideal, it is an area for improvement moving forward by finding a better data source.

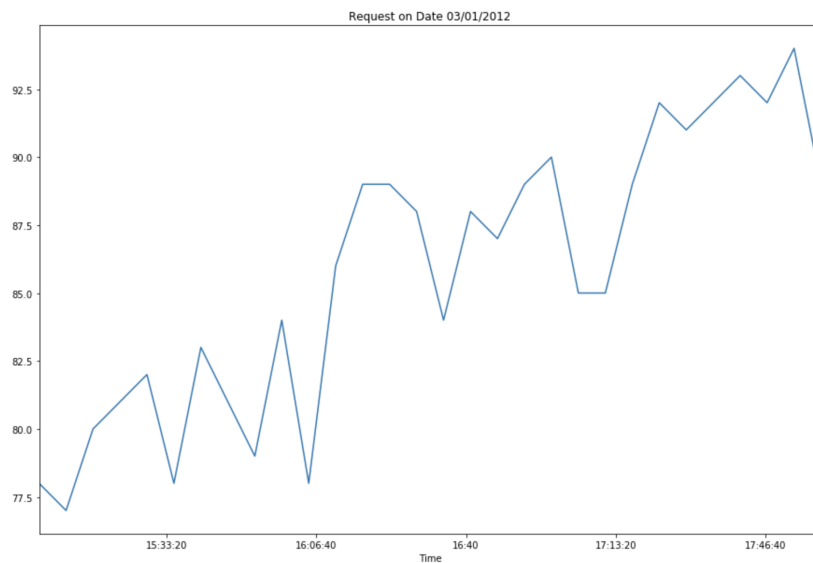


Figure 5.6: Number of Requests Over One Day

We can see in Figure 5.6 that the number of requests increases at the time moves towards 6pm, this is not unusual for rush hour and is as expected.

6 Evaluation

In this chapter we will present how we will evaluate the solution that we designed and implemented for a SAMoD system. We will look at the objectives of this evaluation, before looking at the simulation environment and the metrics we will evaluate on.

6.1 Objectives

The hypothesis states that, congestion data introduced to the observation space will improve the quality of a shared autonomous mobility on demand system. We can to evaluate how travel times are affected in this proposed solution and how vehicles empty miles are affected. This will allow us to have an overview of where trade-offs will need to be made in order to improve the quality of a ride-sharing system. This solution will be considered a success if the following conditions are met:

- Agents act autonomously by creating their own policies and accepting requests without any prompt.
- Agents will learn through experiences and through the simulation to choose requests optimally
- An improvement in one or more of the metrics can be seen
- A ride-sharing system can successfully be simulated in Dublin

6.2 Simulation Environment

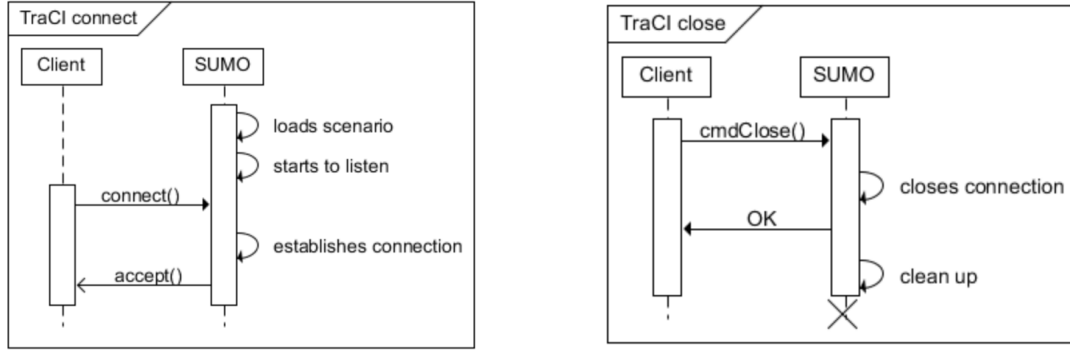
"Simulation of Urban Mobility" (Eclipse SUMO) is an open source, highly portable, microscopic and continuous road traffic simulation package designed to handle large road networks. The simulation environment used in the project is "Simulation of Urban Mobility" (SUMO)[3]. It is an open source, microscopic and continuous road traffic simulation package. It is designed to handle large road networks, which can be imported from open street map easily. This allows any road network to be experimented on.

It allows modelling of a variety of traffic systems and vehicles. This includes road vehicles, public transport and pedestrians. SUMO has a number of supporting tools, that handles tasks such as route finding, visualisation, network import and value retrieval. SUMO can be enhanced with custom models and provides various APIs to remotely control the simulation.

For this project a popular SUMO API will be used. The python TraCI API, which is short for "Traffic Control Interface" gives access to the elements needed for running a road traffic simulation. It allows retrieval of values of the simulated objects and allows manipulation of their behaviour "on-line". TraCI uses a TCP based client/server architecture to provide access to SUMO, as seen in Figure 6.1. It can handle multiple clients, which is necessary for multi-agent based systems. Each client must have its own id, and also be connected to the simulation before the simulation can be run.

The client sends commands after triggering a simulation step to SUMO to control the simulation. To control single vehicle's behavior or to ask for values from the environment. SUMO answers with a Status-response to each command and additional results that depend on the given command.

We can see in Figure 6.2 how the system is set up, and how each library and environment is connected. The RLLib/OpenAI environment holds the SUMO



(a) TraCI API opening connection to SUMO (b) TraCI API closing connection to SUMO

Figure 6.1: TraCI API connecting with SUMO using TCP [3]

simulation. TRaCI is used to connect between the agent and the environment. These TraCI values retrieved are sent to the agent so that it can update its states and choose an action, which is then fed through TraCI to SUMO.

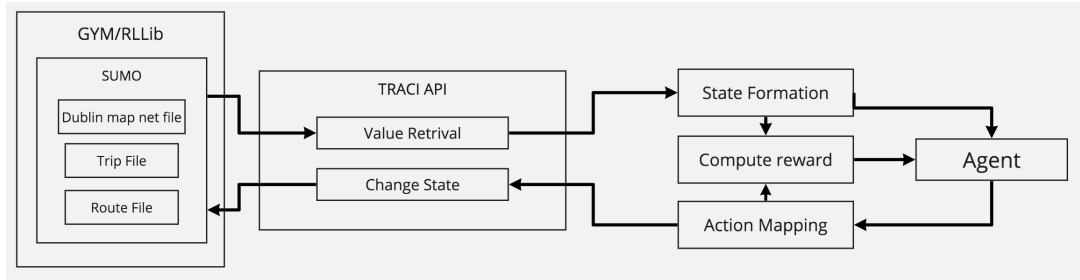


Figure 6.2: Overview of SAMoD system control

6.2.1 SUMO Files

Map

In order to run the simulation SUMO requires a number of files, one of these being the map of the network that you want to run it on. These can be generic roads or can be downloaded and converted from openstreetmap (OSM) [56]. As discussed throughout this report, Dublin is the focus of this evaluation. SUMO requires an XML scheme for the network. The map is downloaded from OSM and then converted into an XML schema using NETCONVERT. NETCONVERT is a command line application that takes versions of network maps and outputs a generated SUMO road

network. The Dublin map can be seen in Figure 6.3.

As seen in Figure 6.3c the map consists of the roads in Dublin, this figure was taken during a time-step in the simulation showing a number of cars on the road.

Route File

The route file is a necessary file needed for the simulation to run. It contains information on how traffic flows in the simulation. Each car is individually sent to the simulation from this file. Each car is uniquely identified with an integer to keep track of them, this way they can be controlled through TraCI. The file used contains a sample of 36,000 cars that can be added to the simulation at a particular time-step, destination and follow a certain route. In Listing 6.1 we can see a snippet of code that represents 1 car with ID 0. This car enters the simulation at a random destination at timestamp 0.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://sumo.dlr.de/xsd/routes_file.xsd">
  <vehicle id="0" type="car" depart="0.00" departLane="best"
    departPos="random" departSpeed="0.00">
    <route edges="32642508#2_..._32015967_4395931"/>
  </vehicle>
```

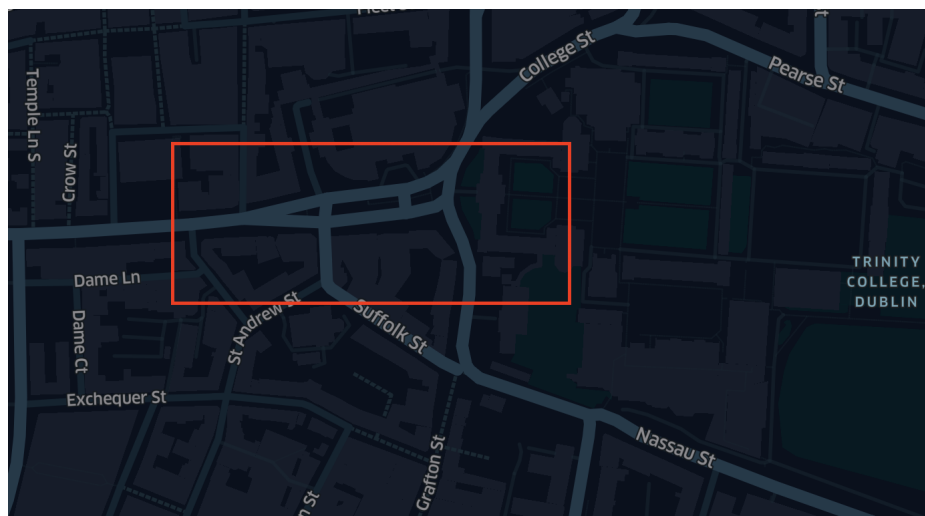
Listing 6.1: Sample route file (rou.xml)

6.2.2 Dublin Postal Code Shape-file

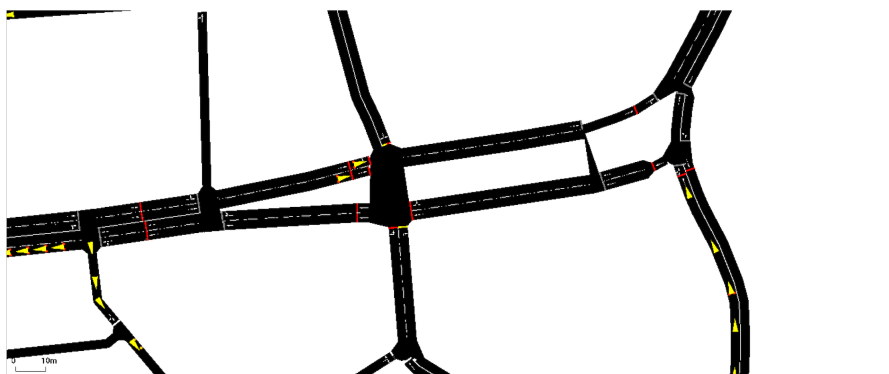
This file contains the areas of Dublin that the taxi can enter, these are the areas we can see highlighted in Figure 4.8a. These hold the coordinate boundaries for each grid space that the congestion data is found in. In Figure 6.2.2 we can see a sample of this



(a) Network Map of Dublin



(b) Satellite map showing area of network



(c) Close up of SUMO network outside TCD

Figure 6.3: Network map of Dublin

.geojson file that is read into SUMO as a shapefile. These were sourced from the following citation [57].

```
{
  "type": "FeatureCollection",
  "name": "DublinPostcodes_4326",
  "crs": { "type": "name", "properties": { "name": "urn:ogc
:def:crs:OGC:1.3:CRS84" } },
  "features": [
    { "type": "Feature", "properties": { "id": 1 }, "
      geometry": { "type": "MultiPolygon", "coordinates"
        : [ [ [
          [ -6.259245391258985, 53.361701960744242 ], ...,
          [ -6.268154304043104, 53.352918993675864 ],
          [ -6.259245391258985, 53.361701960744242 ] ] ] ] } }
```

6.3 Scenarios

Given that there are no baselines to compare this algorithm against, we must provide different scenarios to test whether or not the hypothesis has been proved. In order to prove that traffic information supplied to agents does increase the quality of a SAMoD system. We must compare scenarios where one has congestion levels in the observation space and one where it does not. This will give insight in how the congestion levels affects the metrics defined. It should also give insight into how adding congestion data into the agents observation space affects different areas of the system, either the passenger, the vehicle or the environment.

This gives an evaluation on:

- How ride-sharing systems can be affected by real-world attributes in particular congestion

- How ride-sharing systems can be improved by using information available in their environment
- How a system like this can work in Dublin

6.4 Metrics

In order to be able to compare the two scenarios, we need a number of metrics that we can compare against each other. The metrics have been broken down into three categories, each focusing on a particular area of the system. These categories are, the passenger, the vehicle and the environment.

6.4.1 Passenger Metrics

Passenger metrics should give an accurate representation of how the quality of the ride was for the passenger. This is vital in a mobility system because if the passenger does not have a good experience, the mobility system will not be used. The metrics we will compare are:

- **Waiting Time** - this will be the time calculated from the time that the passenger enters the simulation(the call time) and the time that they enter the vehicle. The maximum value that this can be is 10 minutes, after 10 minutes the request will time out and will disappear from the system. We anticipate that this will not be largely affected by the introduction of congestion information, the shared aspect of mobility would reduce this time.
- **Travel Time** - this is the time calculated from the time that a person gets into the vehicle to the time they are dropped off at their destination point. With this being a shared mobility system we can anticipate that travel time will be longer than the time that it would take to travel from A to B. This is due to the fact that a detour will be done to pickup another request. However, by introducing

congestion levels to the agents observation space we predict that travel time will be lower than when this data is not in the agents state space.

6.4.2 Vehicle Metrics

The vehicle metrics are important for the setup and running of an on demand mobility system. Through evaluating these metrics we will see how many vehicles would be necessary for such a system in Dublin. We will also see if the number of vehicles can be reduced if the congestion data is available for the vehicle to make decisions against. The vehicle metrics are:

- Total miles travelled - This is the total number of miles that the vehicle travels throughout the simulation.
- Empty miles - This is the number of miles that the vehicle is travelling while empty, for example when it is travelling to a request with no other passengers on board.
- Shared Miles - This is the number of miles that the vehicle has two or more requests at the same time.
- Time that taxi is empty - This is the time that the vehicle is empty, whether it be travelling to a request with no other passengers or while its waiting for a request to pop up.

6.4.3 Environment Metrics

The environment is the overall system, the main thing that we will look at and evaluate here is the number of requests that actually get served. A request will be timed out after 10 minutes, this is to replicate the average amount of time that a person waits before they can cancel an uber trip. The aim is to maximise the percentage of these requests that are served, this is vital to any mobility service.

The evaluation is ran with 75 agents. If these were non-shared taxis this would be an average of 4.6 fares per hour, which is reasonable. A sample of the taxi.add.xml file containing these taxis are shown.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://sumo.sf.net/xsd/routes_file.xsd">
  <vType id="taxi" accel="0.8" decel="4.5" sigma="0.5"
    length="5" maxSpeed="70" color="1,0,0"/>
  <vehicle id="taxi1" depart="60.00" type="taxi">
    <route edges="out"/>
  </vehicle>
```

Listing 6.2: Sample taxi file (rou.xml)

7 Results

This section will show the results from running the simulation scenarios mentioned in Chapter 5. In the results section, **Scenario 1 will represent the simulation with congestion** in the observation space, and **Scenario 2 represents the simulation that does not have congestion** information in its observation space. The results are taken from the last ten days of our simulation. We can assume that after 42 days of learning in the simulation, the agent has trained itself to choose optimally.

7.1 Vehicle Evaluation

Here we will discuss how the vehicle was effected between our two scenarios. In Table 7.1 we can see the vehicles results in each scenario. Values shown are an average per vehicle.

variable	Scenario 1	Scenario 2
Average Total KM per Hour	16.958	15.047
Average Shared KM per Hour	4.451	2.692
Average Empty KM per Hour	1.536	1.815
Percentage of Empty Miles	9.056	7.85
Percentage of Shared Miles	26.244	17.89

Table 7.1: Vehicle Metric Results

We can see that the scenario with congestion information travelled more miles per hour on average than the second scenario. This is expected as we can assume that the agents decided not to travel into other areas of higher congestion, and avoided these

areas by not accepting requests here. Scenario 1 also had a slightly higher percentage of empty miles, this could lead to the increased wait times for passengers. Scenario 1 had a higher percentage of shared miles compared to Scenario 2, with Scenario 1 having 26% shared miles and Scenario 2 with 18% shared miles. We can conclude that even though the vehicles travelled further, they had more passengers per mile which would lead to higher profitability.

This is also evident in Figures 7.1 - 7.4.

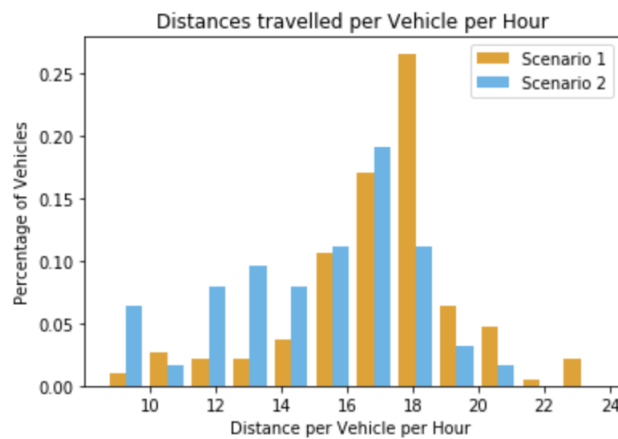


Figure 7.1: Average Total Distance Travelled Per Vehicle Per Hour in KM's

Figure 7.1 shows again that the scenario with congestion information available (Scenario 1) on average travelled a higher distance.

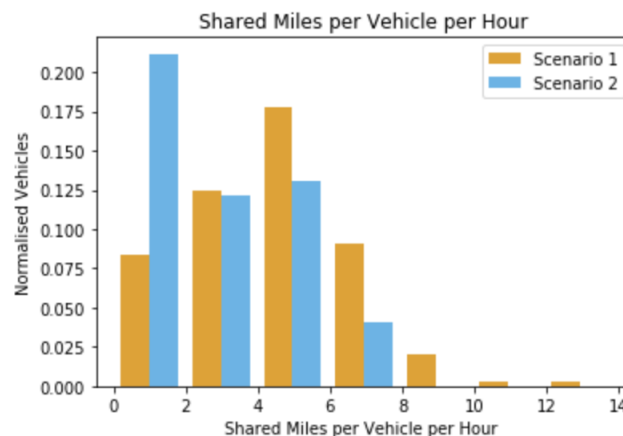


Figure 7.2: Average Shared Distance Travelled Per Vehicle Per Hour in KM's

Figure 7.2 shows that on average agents in Scenario 1 travelled a larger distance with more than one request present.

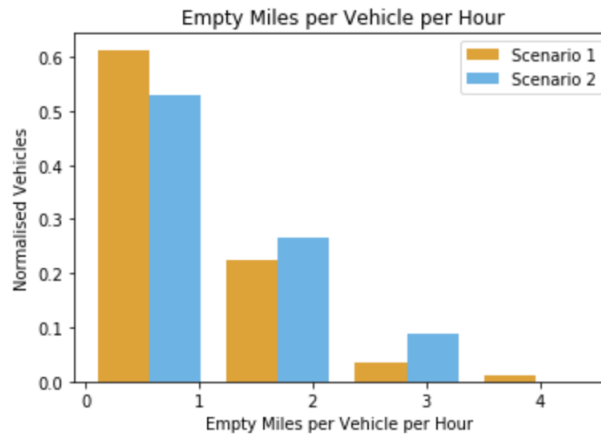


Figure 7.3: Average Empty Distance Travelled Per Vehicle Per Hour in KM's

Figure 7.3 shows the average distance per hour that the vehicles were empty. According to the National Transport Authority, which did an analysis of taxi drivers and work patterns, taxis are empty 58% of the time in Ireland [58]. In this project we average 9.056% empty for Scenario 1 and 7.85% empty for Scenario 2. Although this is a positive outcome, that vehicles are not empty for very long, this could lead to increased wait times and more requests being timed out as there are not enough vehicles to match the demand.

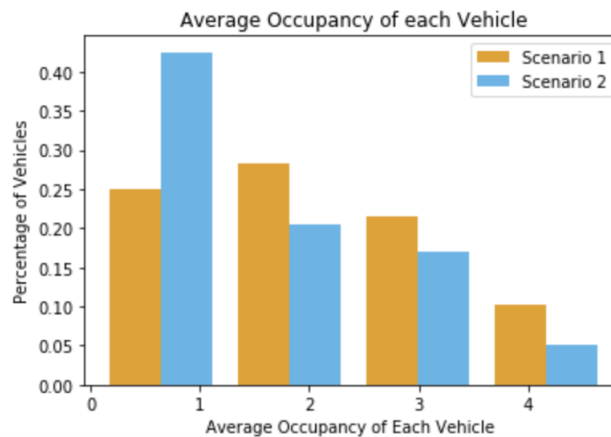


Figure 7.4: Average Occupancy Per Vehicle Per Hour

Figure 7.4 shows that Scenario 1 has a higher average occupancy rate than Scenario 2, this backs up the results that Scenario 1 had a higher percentage of shared miles. Scenario 2 had an average occupancy rate of 1.2 passenger, whereas scenario 1 had an average occupancy rate of 2.1.

7.2 Passenger Evaluation

Here we will discuss how the requests were effected between our two scenarios.

variable	Scenario 1	Scenario 2
Average Travel Time in Minutes	14.07	14.64
Average Wait Time in Minutes	3.18	1.86

Table 7.2: Requests Metric Results

We can see in Table 7.3 that as predicted, Scenario 1 has a lower average travel time compared to Scenario 2. It is 0.57 minutes lower, which seems small, however when we consider how the vehicles in this scenario had a higher average occupancy then this proves to be a good result. The wait time for Scenario 1 was higher than Scenario 2, with passengers waiting on average 1.32 minutes longer. Wait times do not include wait times for requests that were timed out. From this we can assume that a number of requests, that were in high congestion areas, were not picked up as quickly compared to in Scenario 2. This again shows that agents may have avoided areas of high congestion, as mentioned in Section 7.1. This is also evident in Figures 7.5 - 7.6.

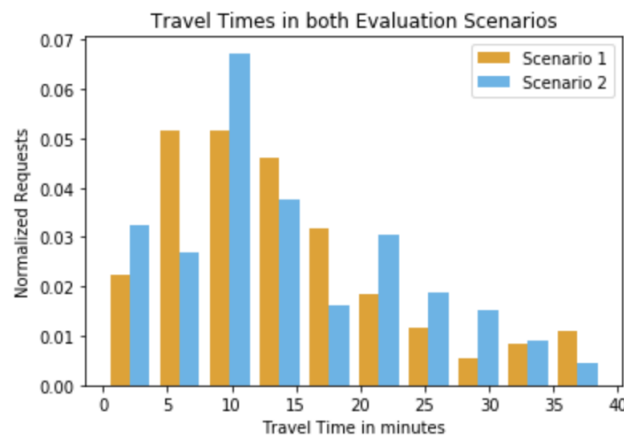
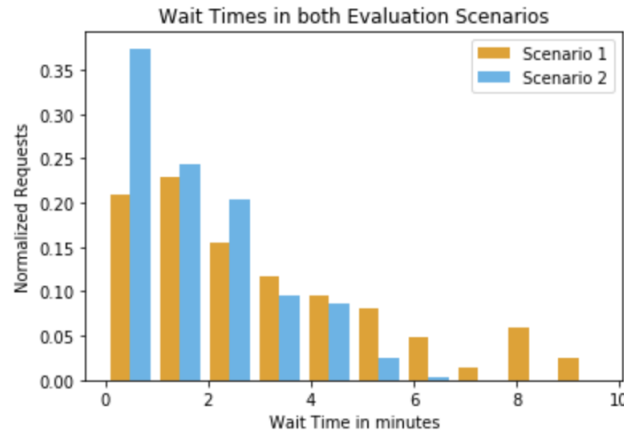


Figure 7.5: Travel Time Per Request in Minutes



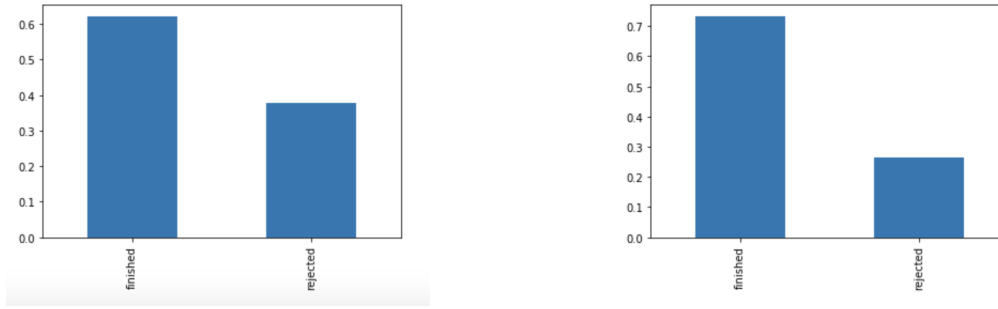
7.6: Wait Times per Request in Minutes

7.3 Environment Evaluation

We will now discuss how the number of requests served was affected in our two scenarios. In Figure 7.7 we can see how many requests were accepted and how many were timed out after ten minutes. We can see that in Scenario 1 more requests were timed out compared to Scenario 2. This supports the evidence in the requests results that wait times in scenario 1 were longer. This could be reduced by introducing more agents into the simulations in the future, in order to meet the demand for this size of city. As mentioned in Section 7.1, the rate of rejection could be reduced by using more agents to meet the demand.

variable	Scenario 1	Scenario 2
Requests served %	62.3	37.7
Requests Timeout %	73.5	26.5

Table 7.3: Requests Metric Results



(a) Scenario 1: Percentage of Requests Served (b) Scenario 2: Percentage of Requests Served

Figure 7.7: Percentage of Requests Served

7.4 Result Summary

The results from this evaluation state that an agent who has congestion integrated into their state space, has a lower travel time for passengers, approximately 0.57 minutes on average. Scenario 1 has a higher percentage of shared miles travelled, 26.244 compared to 17.89 in Scenario 2. As expected, we saw an increase in wait time for passengers of 1.32 minutes, however this could possibly be reduced by introducing more agents to meet the demand. We also saw that the percentage of empty miles increase in Scenario1 to 9.056% compared to 7.85% in Scenario 2. This stands with the increased wait times. These however are a lot lower than a standard taxi found in Dublin. This shows that trade-offs must be made in order to improve the quality of a ride-sharing system. In this case, passengers enjoyed a lower travel time with the proposed solution but with a longer wait time. The vehicles made a trade-off by having slightly more empty miles whilst having the benefit of having more requests in each vehicle at one time.

8 Conclusion

This chapter will summarise this thesis and review the outcomes that we saw. It will then discuss further work that could be done to enhance this and move forward.

8.1 Thesis Contribution

This thesis is the only work that involves integrating real-world attributes into the observation space of a SAMoD Multi-Agent Reinforcement Learning system, in particular introducing congestion information into the state space of the agents. It proposed and evaluated a congestion-aware request-matching ride-sharing system that uses decentralised multi-agent reinforcement learning. It is also the only research into a ride-sharing system in Dublin.

Chapter 1 defined what SAMoD is and how urban mobility is adapting to meet the needs of a growing population. It defined the problems associated with SAMoD and extracted the problem area that this thesis focuses on, which is request-matching optimization. We explored the assumptions that this thesis makes and used when designing the RL algorithm.

Chapter 2 defined and explained the necessary background information that a reader needs in order to understand the rest of the thesis, in particular the State of the Art section. This included reinforcement learning and multi-agent systems.

Chapter 3 explored the literature that is published in this space. It presented the most recent work in SAMoD systems. It looked at ride-sharing work and the types of

ride-sharing systems currently researched. It explored types of machine learning techniques used, in particular focusing on reinforcement learning and then looking at reinforcement learning in request-matching optimization. It also explored the small research area that looks at how SAMoD systems can use demand to improve services.

Chapter 4 shows how the SAMoD system is set up, including dataset creation, map definition and system architecture. It defined the RL attributes including, the action space, the state representation and the reward function. Whilst also introducing the scenario testing needed for the evaluation.

Chapter 5 shows how the system was implemented, what libraries were used and which libraries were adapted in order to implement the system. It also discusses the data that was sourced and used to build the data-sets described in Chapter 4.

Chapter 6 outlines the evaluation strategy that was used to evaluate the SAMoD system proposed in this thesis. It also describes the evaluation metrics used and why they are necessary to look at.

Chapter 7 showed the results of our evaluation, it showed that the decentralised multi-agent Q-learning RL algorithm with integrated congestion information in its state representation proposed in this thesis performed as we had hoped. It reduced travel time for passengers by on average 0.57 minutes and increased the percentage of miles that the vehicles have more than one request present. This shows that this area of research has performed well and gives insights on where the future work should lead towards.

8.2 Future Work

Here we will define next steps that this research can go in, learning and moving on from this thesis. When evaluating and implementing this project a number of areas were identified as to how it could be improved. These were:

- **Real-Time Data collection** - In this project data from 2012 was used to create data-sets to train the agents and run the simulation. In an ideal situation we would have used a more up-to-date dataset however this was not available. In future projects real-time data could be acquired for congestion levels in Dublin to be used in the simulation. A real dataset of taxi pickup and drop offs would be ideal if this could be acquired from a taxi company eg. Uber, as they have provided this for other cities eg, in New York.
- **Full Scale Dublin Simulation** - In order to accommodate computing power capabilities, the grid spaces were limited to 9 postal codes in inner city Dublin. In order to get a full idea of how a system would fair in Dublin all 24 postal codes need to be integrated into the system. This could be fulfilled by utilising cloud computing power.
- **Different Demand scenarios** - Similar to a full scale simulation in Dublin, a number of different congestion scenarios should be tested to see how the solution acts in different conditions. Throughout the day congestion fluctuates, we must test the system in a multitude of congestion conditions. This moves forward from the single time range that we have used in this project.
- **Optimized Number of Agent** - As mentioned in the results section, a number of requests were rejected meaning they were timed out after 10 minutes. In order to accommodate this more agents should have been used. A study could be done to find the optimal number of agents for a city the size of Dublin.
- **Reward Tuning** Due to time constraints a minimal amount of reward function tuning could be done, mainly due to simulations taking so long to run. In future more tuning should be done to find the best reward function to optimise travel time for passengers and to increase the number of requests served.
- **Deep Reinforcement Learning** - This model could be furthered by implementing a DQN to optimize this system. As mentioned in the report, the main reason this was not chosen was due to time and computing power

restraints. It would take a long time to train this high number of agents with a DRL algorithm therefore areas would need to be scaled back to implement this. This could also be trained on a cloud computing platform to speed up simulations.

- **Continuous observation state representation** In order to reduce complexity in this system congestion information is categorised into 3 relative levels. This system could be tested with a continuous state representation using the actual traffic flows for each area. This would most likely need a smaller test space, eg a smaller amount of grids to initially test it on.
- **Hybrid Multi-Agent Architecture** A hybrid architecture could be implemented to reduce training time for the system, this could involve a number of controller agents who have slave agents under them. The controller agents would send requests to these slave agents. Controller agents would be trained and slave agents would listen for actions.

Bibliography

- [1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.
- [2] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Miro Gradisar. The benefits of meeting points in ride-sharing systems. *SSRN Electronic Journal*, 01 2015. doi: 10.2139/ssrn.2567274.
- [3] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL <<https://elib.dlr.de/124092/>.
- [4] Department of Economic and Social Affairs. Population division, 2018.
- [5] Michael J. Greene and Deborah M. Gordon. Interaction rate informs harvester ant task decisions. *Behavioral Ecology*, 18(2):451–455, 01 2007. ISSN 1045-2249. doi: 10.1093/beheco/ar1105. URL <https://doi.org/10.1093/beheco/ar1105>.
- [6] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, Nov 2006. ISSN 1556-6048. doi: 10.1109/MCI.2006.329691.
- [7] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

- [8] CR Rao and Venkat N Gudivada. *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*. Elsevier, 2018.
- [9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, chapter 1, pages 485–585. Springer, 2009.
- [10] Theodore J Perkins and Doina Precup. A convergent form of approximate policy iteration. In *Advances in neural information processing systems*, pages 1627–1634, 2003.
- [11] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.
- [12] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic, 2012.
- [13] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.
- [14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.
- [16] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.
- [17] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.
- [18] Richard Bellman. The theory of dynamic programming. 1954.

- [19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354–359, 2017.
- [20] Jacek M Zurada. *Introduction to artificial neural systems*, volume 8. West St. Paul, 1992.
- [21] Frost and Sullivan. Future of carsharing market to 2025. URL <https://www.frost.com/future-of-carsharing-market-to-2025.html>.
- [22] Boston Consulting Group. Self-Driving Vehicles, Car Sharing, and the Urban Mobility Revolution. URL <https://www.bcg.com/industries/automotive/self-driving-vehicles-car-sharing.aspx>.
- [23] M Ukkonen A Hamari, J Sjöklint. The sharing economy: Why people participate in collaborative consumption. *Journal of THE ASSOCIATION FOR INFORMATION SCIENCE AND TECHNOLOGY*, 67(9):2047–2059, 2016.
- [24] W. Y. Lee Zach, Tommy K. H. Chan, M. S. Balaji, and Alain Y. Chong. Why people participate in the sharing economy: an empirical investigation of uber. *Internet Research*, 28(3):829–850, 2018. URL <http://elib.tcd.ie/login?url=https://search-proquest-com.elib.tcd.ie/docview/2047932572?accountid=14404>.
- [25] Barney Cohen. Urbanization, city growth, and the new united nations development agenda. *Cornerstone*, 3(2):4–7, 2015.
- [26] Biying Yu, Ye Ma, Mei-Mei Xue, Baojun Tang, Bin Wang, Jinyue Yan, and Yi-Ming Wei. Environmental benefits from ridesharing: A case of beijing. *Applied Energy*, 191:141–152, 04 2017. doi: 10.1016/j.apenergy.2017.01.052.
- [27] Hussein Dia and Farid Javanshour. Autonomous shared mobility-on-demand: Melbourne pilot simulation study. *Transportation Research Procedia*, 22:285–296, 12 2017. doi: 10.1016/j.trpro.2017.03.035.

- [28] David Bauer. Opportunities and barriers of ride-sharing in work commuting—a case study in sweden., 2017.
- [29] Christian Clausen. Niche management of autonomous vehicles for positive environmental outcomes in copenhagen. *IIIEE Masters Thesis*, 2017.
- [30] Daniel J Fagnant and Kara M Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas. *Transportation*, 45(1): 143–158, 2018.
- [31] Hussein Dia and Farid Javanshour. Dynamic ride-sharing: a simulation study in metro atlanta. *International Symposium on Transportation and Traffic Theory*, 45: 1450–1464, 2011.
- [32] JENNY LO and STEVE MORSEMAN. The perfect uberpool: A case study on trade-offs. *Ethnographic Praxis in Industry Conference Proceedings*, 2018(1):195–223, 2018. doi: 10.1111/1559-8918.2018.01204. URL <https://anthrosource.onlinelibrary.wiley.com/doi/abs/10.1111/1559-8918.2018.01204>.
- [33] Masabumi Furuhashi, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46, 2013.
- [34] Keiichiro Hayakawa and Eiji Hato. Dynamic traffic resources allocation under elastic demand of users with space-time prism constraints. *CoRR*, abs/1806.10719, 2018. URL <http://arxiv.org/abs/1806.10719>.
- [35] Susan Shaheen and Adam Cohen. Shared ride services in north america: definitions, impacts, and the future of pooling. *Transport reviews*, 39(4):427–442, 2019.
- [36] Yu Wang, Shanyong Wang, Jing Wang, Jiuchang Wei, and Chenglin Wang. An empirical study of consumers’ intention to use ride-sharing services: using an extended technology acceptance model. *Transportation*, pages 1–19, 2018.

- [37] Iis P Tussyadiah, Florian J Zach, and Jianxi Wang. Attitudes toward autonomous on demand mobility system: The case of self-driving taxi. pages 755–766, 2017.
- [38] Mohammad Asghari, Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, and Yaguang Li. Price-aware real-time ride-sharing at scale: An auction-based approach. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPACIAL '16*, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345897. doi: 10.1145/2996913.2996974. URL <https://doi.org/10.1145/2996913.2996974>.
- [39] Dinu Mol P Dr. R. Priya. A machine learning approach for maximizing ride shared route using iabfm framework. *International Organization of Scientific Research - Journal of Engineering*, 8:2–9, 2018.
- [40] J. Alonso-Mora, A. Wallar, and D. Rus. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3583–3590, Sep. 2017. doi: 10.1109/IROS.2017.8206203.
- [41] Taxi and Limousine Commission (TLC). Yellow Taxi Trip Data. URL <https://www.bcg.com/industries/automotive/self-driving-vehicles-car-sharing.aspx>.
- [42] Qiulin Lin, Wenjie Xu, Minghua Chen, and Xiaojun Lin. A probabilistic approach for demand-aware ride-sharing optimization. *CoRR*, abs/1905.00084, 2019. URL <http://arxiv.org/abs/1905.00084>.
- [43] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, KDD '18*, page 905–913, New York, NY, USA, 2018.

- Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219824. URL <https://doi.org/10.1145/3219819.3219824>.
- [44] Benjamin Riviere, Salar Rahili, and Soon-Jo Chung. Urban ridesharing with hybrid distributed reinforcement learning. 2019.
- [45] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference, WWW '19*, page 983–994, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313433. URL <https://doi.org/10.1145/3308558.3313433>.
- [46] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2645–2653, 2019.
- [47] M. Guériau and I. Dusparic. Samod: Shared autonomous mobility-on-demand using decentralized reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1558–1563, Nov 2018. doi: 10.1109/ITSC.2018.8569608.
- [48] Abubakr O. Al-Abbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20:4714–4727, 2019.
- [49] Ashutosh Singh, Abubakr Al-Abbasi, and Vaneet Aggarwal. A reinforcement learning based algorithm for multi-hop ride-sharing: Model-free approach. In *Neural Information Processing Systems (Neurips) Workshop*, 2019.

- [50] Shubham Jain, Stephan Winter, and Nicole Ronald. Feasibility analysis of simulating demand responsive transport using sumo. 01 2014.
- [51] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Kenneth Y. Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *ArXiv*, abs/1712.09381, 2017.
- [52] A. G. Sims and K. W. Dobinson. The sydney coordinated adaptive traffic (scat) system philosophy and benefits. *IEEE Transactions on Vehicular Technology*, 29(2): 130–137, 1980.
- [53] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL <https://doi.org/10.5281/zenodo.3509134>.
- [54] Business Insider. Congestion pricing could mark the beginning of the end of New York’s famous yellow taxis. URL <https://www.businessinsider.com/congestion-pricing-end-of-new-yorks-famous-yellow-taxis-2019-4?r=US&IR=T>.
- [55] The Irish Times. Dublin’s taxi problem: ‘There’s as many cabs here as New York’. URL <https://www.irishtimes.com/news/ireland/irish-news/dublin-s-taxi-problem-there-s-as-many-cabs-here-as-new-york-1.3958513>.
- [56] OpenStreetMap contributors. Planet dump retrieved from <https://planet.osm.org> . <https://www.openstreetmap.org>, 2017.
- [57] Jack H. Geocluster, 2012. URL https://github.com/AugmentedGeoBae/GeoCluster/blob/master/public/GeoJSON/DublinPostcodes_4326.geojson.
- [58] Tourism Department of Transport and Sport. Taxi statistics 2019. 2019.