6-2020

Scalable multi-agent reinforcement learning for aggregation systems

Tanvi VERMA
Singapore Management University, tanviverma.2015@phdis.smu.edu.sg

SCALABLE MULTI-AGENT REINFORCEMENT LEARNING FOR AGGREGATION SYSTEMS

TANVI VERMA

SINGAPORE MANAGEMENT UNIVERSITY

2020

Scalable Multi-Agent Reinforcement Learning for Aggregation Systems

by

Tanvi Verma

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee

Pradeep Varakantham (Supervisor/Chair)
Associate Professor
Singapore Management University

Hoong Chuin Lau (Co-Supervisor)
Professor
Singapore Management University

Shih-Fen Cheng
Associate Professor
Singapore Management University

Sarit Kraus
Professor
Bar-Ilan University

Singapore Management University
2020

I hereby declare that this PhD dissertation is my original work
and it has been written by me in its entirety.
I have duly acknowledged all the sources of information
which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree
in any university previously.

Tanvi Verma
5 June 2020

Scalable Multi-Agent Reinforcement Learning for Aggregation Systems

Tanvi Verma

Abstract

Efficient sequential matching of supply and demand is a problem of interest in many online to offline services. For instance, Uber, Lyft, Grab for matching taxis to customers; Ubereats, Deliveroo, FoodPanda etc. for matching restaurants to customers. In these systems, a centralized entity (e.g., Uber) aggregates supply and assigns them to demand so as to optimize a central metric such as profit, number of requests, delay etc. However, individuals (e.g., drivers, delivery boys) in the system are self interested and they try to maximize their own long term profit. The central entity has the full view of the system and it can learn policies to maximize the overall payoff and suggest it to the individuals. However, due to the selfish nature of the individuals, they might not be interested in following the suggestion. Hence, in my thesis, I develop approaches that learn to guide these individuals such that their long term revenue is maximized.

There are three key characteristics of the aggregation systems which make them unique from other multi-agent systems. First, there are thousands or tens of thousands of individuals present in the system. Second, the outcome of an interaction is anonymous, i.e., the outcome is dependent only on the number and not on the identities of the agents. And third, there is a centralized entity present which has the full view of the system, but its objective does not align with the objectives of the individuals. These characteristics of the aggregation systems make the use of the existing Multi-Agent Reinforcement Learning (MARL) methods challenging as they are either meant for just a few agents or assume some prior belief about others. A natural question to ask is whether individuals can utilize these features and learn efficient policies to maximize their own long term payoffs. My thesis research focuses on answering this question and provide scalable reinforcement learning methods in aggregation systems.

Utilizing the presence of a centralized entity for decentralized learning in a non-cooperative setting is not new and existing MARL methods can be classified based on how much extra information related to the environment state and joint action is provided to the individual learners. However, presence of a self-interested centralized entity adds a new dimension to the learning problem. In the setting of an aggregation system, the centralized entity can learn from the overall experiences of the individuals and might want to reveal only those information which helps in achieving its own objective. Therefore, in my work I propose approaches by considering multiple combinations of levels of information sharing and levels of learning done by the centralized entity.

My first contribution assumes that the individuals do not receive any extra information and learn from their local observation. It is a fully decentralized learning method where independent agents learn from the offline trajectories by considering that others are following stationary policies. In my next work, the individuals utilize the anonymity feature of the domain and consider the number of other agents present in their local observation to improve their learning. By increasing the level of learning done by the centralized entity, in my next contribution I provide an equilibrium learning method where the centralized entity suggests a variance minimization policy which is learned based on the values of actions estimated by the individuals. By further increasing the level of information shared and the level of learning done by the centralized entity, I next provide a learning method where the centralized entity acts as an correlation agent. In this method the centralized entity learns social welfare maximization policy directly from the experiences of the individuals and suggests it to the individual agents. The individuals in turn learn a best response policy to the suggested social welfare maximization policy. In my last contribution I propose an incentive based learning approach where the central agent provides incentives to the individuals such that their learning converges to a policy which maximizes overall system performance. Experimental results on real-world data sets and multiple synthetic data sets demonstrate that these approaches outperform other state-of-the-art approaches both in the terms of individual payoffs and overall social welfare payoff of the system.

Contents

List of Figures

List of Tables

Acknowledgement

I am deeply grateful to my advisor Pradeep Varakantham for his guidance, support, encouragement and patience throughout my PhD journey. The passion and enthusiasm he has for his research has kept me motivated during tough times in the PhD pursuit. Pradeep's positive attitude towards research is contagious and I always used to be full of enthusiasm after our meetings. I could not have asked for a better advisor, thank you Pradeep, for helping me grow into a confident independent researcher.

I would also like to thank my co-advisor, Hoong Chuin Lau for his inspirational words and guidance. His personal attention and focus on individual growth has made my stint at Fujitsu-SMU Urban Computing and Engineering (UNiCEN) Corp. Lab memorable. Discussion and suggestion provided by Shih-Fen Cheng on multiple occasions has helped me immensely in understanding complex concepts. I wish to acknowledge the guidance provided by Sarit Kraus during early years of my PhD. Course works conducted by Archan Mishra and Hady W. Lauw has helped me greatly in growing as a researcher. I also thank UNiCEN Lab for their generous funding for my research.

I would like to thank my friends and colleagues: Meghna, Mehak, Rajiv, Pritee, Supriyo, Abhinav, Srishti, James, Sanket, Alolika and Han for their friendship and support. Meghna, a special thanks to you for always being there for me, for valuable discussions and for proofreading my papers. I have learnt a lot from you and I feel lucky to have shared my PhD journey with you.

Finally, I sincerely thank my family for their love and support. I am grateful to my parents, for believing in me and supporting me consistently in every decision of my life. Thank you, mummy and papa, for providing foundation for my castle in the air. I

am thankful to my brothers for taking pride in me, it motivates me to keep pushing the limit. I am extremely grateful to my in-laws for their love and patience towards me. A special thanks to my son, Advik, for patiently waiting for mumma to finish her studies and “retire” so that she can spend more time with him. I can not thank Rakesh, my childhood friend and my beloved husband enough. Thank you for assuming the role of “mom” when I was too busy and occupied to take care of our son. This thesis would not have been possible without your unconditional love, encouragement, patience and support.

To

our next generation

Advik, Ira, Vidhi, Kashvi & Niramay

Chapter 1

Introduction

Aggregation systems (e.g., Uber, Lyft, FoodPanda, Deliveroo) have been increasingly used to improve efficiency in numerous environments, including in transportation, logistics, food and grocery delivery. In these online to offline service systems, aggregation companies aggregate supply (e.g., drivers, delivery personnel) and matches demand to supply on a continuous basis. The individuals who are responsible for supply (e.g., taxi drivers,delivery bikes or delivery van drivers) earn more by being at the ”right” place at the ”right” time. However, the objective of the aggregation companies is to optimize a central metric such as maximizing profit, maximizing number of request served or minimizing customer delays. Due to optimizing a metric of importance to the aggregation companies, the interests of the individuals can be sacrificed. Therefore, in my thesis I focus on the problem of learning revenue maximizing policies for individuals in aggregation systems such that overall performance of the system is maximized.

I first describe the following three aggregation systems which provide motivation for my work.

Taxi Aggregation: Companies like Uber, Lyft, Didi, Grab etc. all provide taxi supply aggregation systems. The goal is to ensure wait times for customers is minimal or amount of revenue earned is maximized by matching taxi drivers to customers on a regular basis. However, these goals of the aggregation companies may not be correlated to the individual driver objective of maximizing their own revenue. In this thesis I develop

methods to guide individual drivers to ”right” locations at ”right” times based on their past experiences of customer demand and taxi supply (obtained directly/indirectly from the aggregation company) to improve their revenue.

Food or Grocery Delivery : Aggregation systems have also become very popular for food delivery (Deliveroo, Ubereats, Foodpanda, DoorDarsh etc.) and grocery delivery (AmazonFresh, Deliv, RedMart etc.) services. They offer access to multiple restaurants/grocery stores to the customers and use services of delivery boys/delivery vans to deliver the food/grocery. Similar to taxi aggregation systems, my work can be used to guide delivery boy/van to be present at right locations at right times so as to improve their individual revenue.

Supply Aggregation in Logistics: More and more on-line buyers now prefer same day delivery services and tradition logistic companies which maximize usage of trucks, drivers and other resources are not suitable for it. Companies like Amazon Flex, Postmates, Hitch etc. connect shippers with travelers/courier personnel to serve same day/on-demand delivery requests. The courier personnel in this system can employ the proposed methods to learn to be at ”right” place at ”right” time by learning from the past experiences.

Figure 1.1 provides the framework of an aggregation system. The central agent and individual agents are mutually dependent on each other as it is the individuals who execute the actions whereas the demand assignment is done by the central entity. The entire process can be summed up in following steps

- Individuals executes action based on their current location, ex. stay in the current location or move to a nearby location to maximize their long term payoff.
- Based on the incoming demand, the central entity assigns them to the individuals such that a central metric (maximum social welfare, minimum delay etc.) is optimized.
- Environment moves to the next global state and the central entity receives payoff for the joint-action.

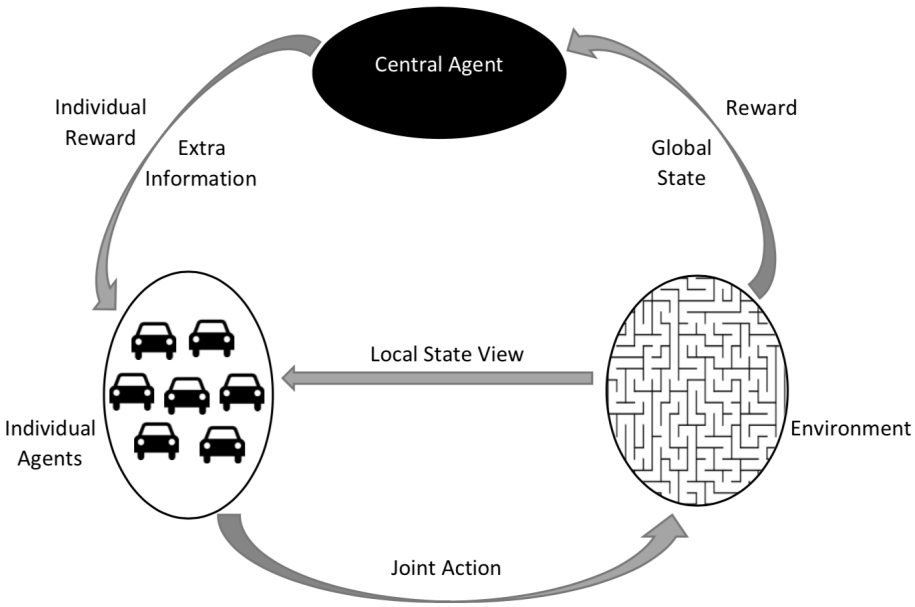


Figure 1.1: Framework of aggregation systems

- After deducting the commission fee, the central agent provides the individual payoffs to the individual agents. It also learns from the experience and optionally provides extra information to them.

- Based on their own experience, individuals learn policies to maximize their own revenues.

Maximizing the long term revenue by being at a location where probability of being matched with a demand is high requires the individuals to make a sequence of decisions. For example, wait in current zone for 5 minutes, if no demand is assigned, wait for 5 more minutes and if a demand is still not assigned then move to some other zone. Also, customer demand is uncertain in these domains. Hence, Multi-Agent Reinforcement Learning (MARL) is an ideal approach for the learning problem. However, there are following characteristics of the aggregation systems, which make them different from the other multi-agent systems.

- Typically a sizable number of individuals are part of the aggregation systems. For example, there are around 40000 taxis in a small city like Singapore.

- The interactions among the individuals are anonymous, i.e., the payoff of an agent depends on the number of other agents selecting the same action rather than the

identities of the other agents. For example, whether a demand is assigned to a taxi is dependent on number of other taxis present at the same location and not on the identities of the other taxis.

- Most importantly, the objectives of centralized agents (aggregation companies) and individual agents are different. For example, the centralized entity optimizes overall system performance whereas the individuals are interested in maximizing their own long term payoffs.

Hence, existing MARL approaches which are either suitable for only a few agents (Fictitious play (Brown, 1951), JAL (Claus & Boutilier, 1998), WOLF-IGA (Bowling & Veloso, 2002), Minimax Q-Learning (Littman, 1994), Nash Q-Learning (Hu & Wellman, 2003), Correlated Q-Learning (Greenwald, Hall, & Serrano, 2003), Friend-or-Foe Q-learning (FFQ) (Littman, 2001a), AWESOME (Conitzer & Sandholm, 2007), FSP (Heinrich, Lanctot, & Silver, 2015) etc.) or assume that the learning agent has some prior belief about the other agents (M* (Carmel & Markovitch, 1996), (Castellini, Oliehoek, Savani, & Whiteson, 2019), ToMnet (Rabinowitz et al., 2018), Bayes-ToMoP (T. Yang et al., 2019), LOLA (J. Foerster et al., 2018) etc.) are not suitable for the learning problem.

Many of the current algorithms assume presence of a centralized entity which provides extra information about environment state or joint action to the individuals either during training time (Mean Field Q-Learning (MFQ) (Y. Yang et al., 2018), Multi-Agent Actor Critic (MAAC) (Lowe et al., 2017), MADDPG (Lowe et al., 2017), COMA (J. N. Foerster, Farquhar, Afouras, Nardelli, & Whiteson, 2018)) or both at the training as well as at execution time (JAL, Minimax Q-Learning, FFQ, Nash Q-Learning, Correlated Q-Learning etc.). In fact, the existing algorithms can be classified based on the level of information shared by the centralized entity. As in aggregation systems the centralized agent has a full view of the system, it can learn policies which optimizes its own objective and suggest them to the individual agents. The presence of a *self-interested* centralized entity which acts as an intermediary between the environment and the individual learners adds a new dimension to the learning problem. Therefore, in my work I

provide methods of learning in aggregation system based on these two dimensions , i.e. the level of information sharing and the level of learning done by the centralized agent.

There are a few recent algorithms (Neural Fictitious Self Play (NFSP) (Heinrich & Silver, 2016), MFQ (Y. Yang et al., 2018), MAAC (Lowe et al., 2017)) which are relevant to the problem presented in this thesis, however they do not consider learning by the centralized agent and we show in our experiments that the proposed methods outperform these algorithms.

1.1 Motivating Example

In this section I discuss few scenarios which motivates me to provide different learning approaches for the aggregation domain. The level of information to be shared and the level of centralized learning to be done is completely dependent on the discretion of the centralized entity. For example, a food delivery aggregation company might not be interested in providing extra assistance to the individuals in the system. Hence, for such kind of domains I have provided independent learning methods where the individuals learn from their local experiences.

If the central entity is willing to participate in the learning, there are multiple ways in which it can do so. For example, it can share global state information to the individuals, or it can learn policies which improve performance of the overall system and suggest them to the individuals. Furthermore, as the individuals are self-interested, they might not be interested in following the suggested action. Hence, instead of suggesting a policy, the central entity can provide incentives to the individuals such that the overall system performance is maximized. This motivates me to provide learning approaches for the individuals in the presence of a self-interested central entity. The proposed approaches are suitable for different combination of level of information shared and level of learning done at the end of the centralized entity.

1.2 Contributions

In my work I provide five different learning methods in aggregation system based on multiple combinations of the two above-mentioned dimensions which are summarized as follows

- First, I propose ILT (Independent Learning from offline Trajectories), where the individuals learn completely from their local observation and no extra information is shared by the centralized entity. In this method the individuals learn from the real-world (Global Positioning System) GPS trajectories of the other taxis present in an offline data set. This approach is suitable when there are very few learning agents present in the environment and rest of the agents follow stationary policies. Experimental results using real-world data show that a learning agent is able to earn revenue which is comparable the revenue earned by the top 10 percentile of the real-world taxi drivers.

- Second, I present Density Entropy based Deep Q Network (DE-DQN) for the individual learners where other agents are also simultaneously learning. This approach utilizes the anonymity feature of the aggregation domain and consider the local count statistics of the other agents in their learning model. More specifically, DE-DQN predict and control the non-stationarity introduced due to the presence of other agents by learning policies that maximize the entropy on agent population distribution. DE-DQN performs better than other relevant algorithms both in the terms of individual payoffs as well as the social welfare values.

- Third, I provide value Variance Minimization Q-learning (VMQ) approach which learns $\epsilon-$ Nash equilibrium policies for the individual learners. This a *centralized learning decentralized execution* algorithm where the central agent learns from the learned values of the individual agents and provides the extra information only during the learning phase of the algorithm. Building insights from the non-atomic congestion games model, I provide theoretical properties of equilibrium in anonymous domains. Based on these properties, VMQ learns $\epsilon-$Nash

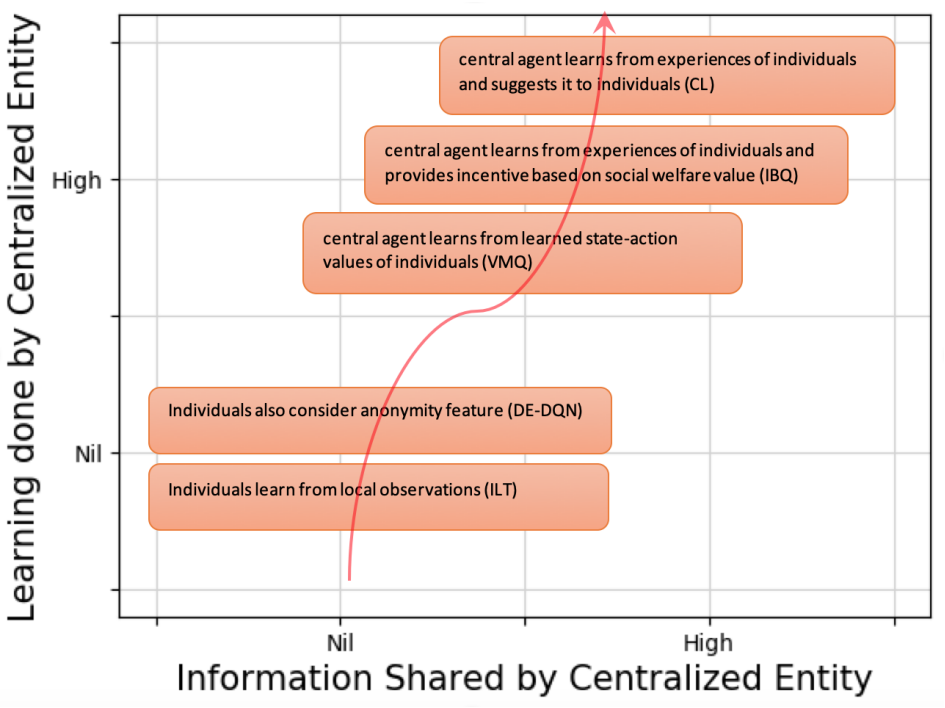


Figure 1.2: Thesis Evolution.

equilibrium policies. I experimentally show that VMQ learn better equilibrium policies than other state-of-the-art methods.

- Fourth, I propose Correlated Learning (CL) method where central entity learn directly from the experiences of the individuals. Based on the experiences, the centralized agent learns a policy which optimizes its objective of maximizing social welfare and suggests it to the individuals. Individual agents in turn learn best response policies to the suggested social welfare policy. Experimental results show that CL results into a ”win-win situation” where both central agents and individuals receive better payoff than the other learning approaches.

- Fifth, to maximize the overall performance of the system, I propose an Incentive Based Q-Learning (IBQ) method where the central agent learns a social welfare maximization policy. Based on the insights from potential based reward shaping, the central agent provides incentives to the individuals such that they converge to policies that maximize social welfare value. Experimental results show that providing incentive is advantageous and both central agent and individual agents receive better payoff than relevant MARL algorithms.

These approaches are scalable as they either consider only the individual action or they consider the aggregated joint action in their model. The experimental results depict that proposed algorithms outperform relevant state-of-the-art algorithms.

Figure 1.2 shows how my thesis has evolved with respect to the two dimensions. My initial work involved independent learning with local observations where the central agent is passive in the environment. Then by gradually increasing the levels, I worked on providing more robust methods which improve performance both in the terms of individual payoffs and overall social welfare payoff of the system.

Figure 1.3 categorizes the contribution of this thesis (in red) with respect to the level of extra information shared to the individual agents and the level of learning done by the centralized agent. It shows that my thesis covers all the combinations of the two dimensions where a few of them learn solely from the local observations whereas extra learning happens at the central agent's end for the rest of the proposed methods.

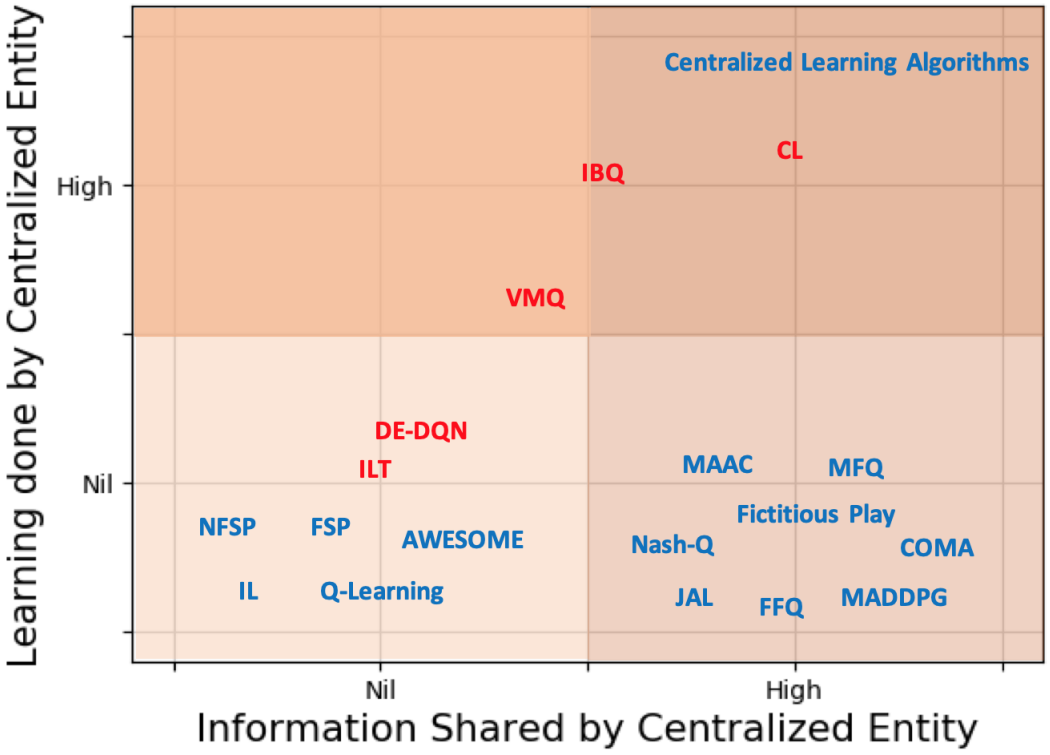


Figure 1.3: Thesis Contribution.

1.3 Overview

Following is the overview of the chapters presented in this thesis

- **Chapter 2** presents formal frameworks of related concepts such as RL, MARL and stochastic games. It provides the background necessary to understand the detailed research I present in later chapters.

- **Chapter 3** discusses the related research on multi-agent reinforcement learning and game theoretic approaches.

- **Chapter 4** introduces the underlying model for anonymous multi-agent reinforcement learning. It is a specialization of stochastic games model which consider interaction anonymity. In the chapter I also introduce experimental setups which have been used throughout the thesis to compare performances of the proposed methods.

- **Chapter 5** provides ILT, a method for an individual agent to learn from offline trajectories of other agents. We show that this method performs extremely well when number of learning agents are limited in the domain.

- **Chapter 6** presents DE-DQN, a method which exploits interaction anonymity of the domain. We show that by considering entropy of agent state distribution, an individual agent performs significantly better than the existing algorithms.

- **Chapter 7** provides VMQ, an equilibrium learning method where centralized agent learns based on the learning of individuals. I experimentally show that VMQ learns better ϵ-Nash equilibrium policies as compared to the other equilibrium learning algorithms.

- **Chapter 8** proposes CL, where the central agent learns social welfare maximizing policy from the experiences of the individuals. The individuals learn to play best response to the suggested social welfare policy.

- **Chapter 9** presents IBQ, an incentive based learning mechanism which maximizes the overall performance of the system.
- In **Chapter 10** I conclude my thesis with discussion about applicability, operationalization and future work.

Chapter 2

Background

In this chapter, we introduce the relevant background concepts.

2.1 Markov Decision Process

Markov Decision Processes (MDPs) are very popular mathematical framework for modeling decision making under uncertainty. Formally, an MDP is represented by the tuple $\langle S, A, T, R, \gamma \rangle$, where

- S is the set of states,
- A is the set of actions,
- $T : S \times A \times S \to [0, 1]$ is transition function,
- $R : S \times A \to \mathbb{R}$ is reward function, and
- $\gamma \in [0, 1)$ is the discount factor, which represents the difference in importance between immediate reward and future rewards.

Transition function defines a probability distribution over next states as function of agent's current state and its action. More specifically, $T(s, a, s')$ represents the probability of transitioning from state s to state s' on taking action a. The reward function defines the immediate reward received by the agent after selecting actions in given state,

i.e. $R(s, a)$ represents the reward obtained on taking action a in state s. The transition function T and reward function R together defines the *model* of the MDP.

Given an MDP $\langle S, A, T, R, \gamma \rangle$, the core problem is to find an optimal *policy* for an agent. A policy $\pi : S \rightarrow A$ is a function which determines an action $\pi(s) \in A$ for every state $s \in S$. An optimal policy for the MDP maximizes the long run expected reward received by following the policy. $V^{\pi}(s)$ is the policy π's state value function which determines the agent's expected reward starting from state s and then following policy π

$$V^{\pi}(s) = \mathbb{E}_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^{k} r^{t+k+1} | s^{t} = s \Big\}$$

where $\mathbb{E}_{\pi}\{\}$ is the expected value given that agent follows policy π.

The problem of providing guidance to taxi drivers on the "right" locations so as to maximize their long-term revenue can be modelled as an MDP as:

- Maximizing the long term revenue by finding customers during cruising requires making a sequence of decisions (ex: wait in current zone for 5 minutes, if no customer found, wait for 5 more minutes and if you cannot still find a customer move to zone B)
- Rewards are well defined, i.e., the sum of revenue earned from a customer (if customer on board) and cost from travelling between locations;
- Customer demand is uncertain and the transition function of an MDP can capture such uncertainty quite well.

Hence, in this thesis I model the problem as an MDP.

2.2 Single Agent Reinforcement Learning

Reinforcement Learning (RL) (Sutton & Barto, 1998) is a method of solving MDP when the model of MDP is not known. A single agent reinforcement learning framework is

based on the model given in Figure 2.1, where an agent interacts with the environment by selecting action and then the environment responds by sending a reward signal and presenting the new state to the agent. The agent learns a policy that maximizes the long term reward while only obtaining experiences (and not knowing the full reward, R and transition, T models) of transitions and reinforcements. An experience is defined as (s, a, r, s') where s is the current state, a is the action selected by the agent, r is the immediate reward of taking action a in state$s$ and s' is the next state. An episode is a sequence of experiences that ends when s' is a terminal state.

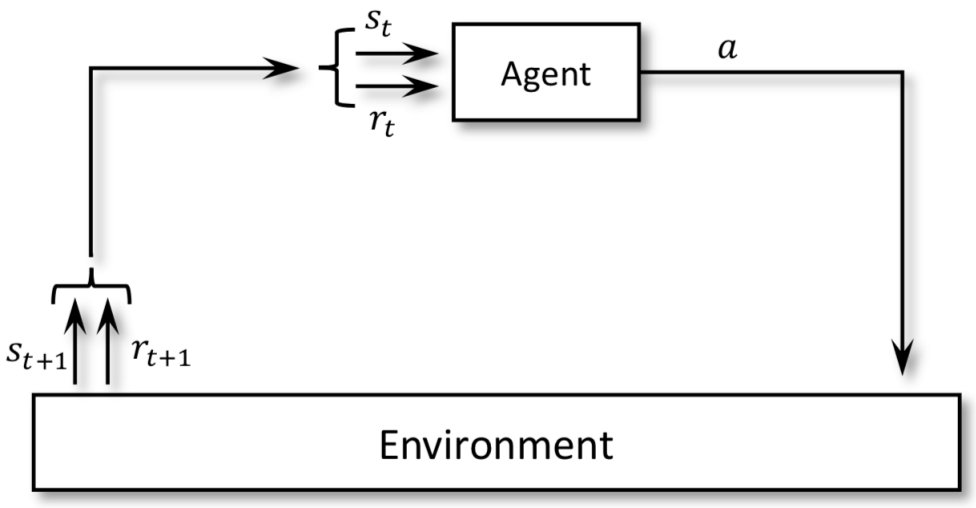


Figure 2.1: Single agent RL framework.

As RL algorithms learn from performing actions and interacting with environment, they usually estimate state-action value function (Q-value function), instead of estimating state value function $V^{\pi}(s)$. The Q-value $Q^{\pi} : S \times A \rightarrow \mathbb{R}$ function defines the expected discounted reward of choosing a particular action in a given state and then following the policy π.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^{k} r^{t+k+1} | s^{t} = s, a^{t} = a \Big\}$$

RL can be broadly divided into two classes, model-based learning and model-free learning. Model-based methods require a model of transition probabilities and the reward function to compute values of states. There are many existing works which deal

with learning transition and reward models (Schneider, 1997; Chakraborty & Stone, 2011; Hester & Stone, 2009). Temporal Difference (Tesauro, 1995) and Monte Carlo (Sutton & Barto, 1998) methods are well known model-free learning methods.

2.2.1 Q-Learning

Q-learning is one of the most popular model-free approaches for RL (Watkins & Dayan, 1992), where the Q function is represented as a table (and initialised to arbitrary fixed values) with an entry in the table for each state and action combination. The optimal Q-value function $Q^*(s, a)$ is the immediate reward of selecting action a in state s plus the expected discounted value attainable from the next state. Q^* satisfies the Bellman optimality equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a' \in A} Q^*(s',a') \tag{2.1}$$

Since the model of the MDP is not known, Q-learning modifies the Equation 2.1 into an iterative approximation approach based on sample of experiences (s, a, r, s') as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.2}$$

Where α is the learning rate which typically decreases over the course of many iterations. The optimal policy can be deduced from the learned Q-values as follows

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q(s,a) \tag{2.3}$$

Q-learning (Watkins & Dayan, 1992) is guaranteed to converge to the optimal solution for stationary domains. To ensure a good explore-exploit tradeoff, an ϵ-greedy (select a random action with probability ϵ and select greedy action with $1 - \epsilon$ probability) policy is typically employed.

2.2.2 Monte Carlo RL

Monte Carlo (MC) method is a way of solving the reinforcement learning problem based on averaging sample returns. It learns directly from episodes of experience. As the experiences of an IL in the domain of interest can be appropriately represented as episodes, MC methods are well suited to estimate Q-values. Return for an state-action pair for a given episode is the cumulative reward till the end of the episode. For example, for episode $s^0, a^0, r^0, s^1, a^1, r^1, ..., s^t, a^t, r^t, s^{term}$ (s^{term} is terminal state), return from (s^1, a^1) is given by :

$$Ret(s^1, a^1) = \sum_{k=1}^{t} r^k$$

Algorithm 1 shows how optimal policy can be computed using MC RL.

Algorithm 1 MC RL

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}$
  $Q(s, a) \leftarrow 0$
  $Count(s, a) \leftarrow 0$
  $Ret(s, a) \leftarrow$ empty list
for every episode in training episodes **do**
  for each (s, a) pair in the episode **do**
    $G \leftarrow$ return after first occurrence of (s, a)
    $Ret(s, a) \leftarrow Ret(s, a) + G$
    $Count(s, a) \leftarrow Count(s, a) + 1$
for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
  $Q(s, a) \leftarrow \dfrac{Ret(s, a)}{Count(s, a)}$
for all $s \in \mathcal{S}$ **do**
  $\pi(s) \leftarrow \underset{a}{argmax} Q(s, a)$

2.2.3 State Abstraction in RL

How the state space is identified is of utmost importance in our problem domains. For example, how we define the zone structure of a map (explained in detail in Chapter 5) will determine the size of state space. If zones are too large, though state space size will decrease, it will also decrease the granularity of the actions. State abstraction

[1] is a popular method for large state space RL to reduce the computational burden. State abstraction exploits the structure of an MDP to derive an abstract MDP having a compressed MDP with similar model. It is a powerful tool for scaling by simplifying complex models.

2.2.4 Deep Reinforcement Learning

In recent years, deep reinforcement learning has become extremely popular among the RL community due to its success in achieving human level performance in various games (Mnih et al., 2013; Silver et al., 2016). Deep RL uses deep neural network to approximate Q-values. Here, we explain two popular deep RL networks - deep Q-network and actor-critic network

Deep Q-Networks (DQN)

Instead of a tabular representation for Q function employed by Q-Learning, the DQN approach (Mnih et al., 2015) employs a deep network to represent the Q function. Unlike with the tabular representation that is exhaustive (stores Q-value for every state, action pair), a deep network predicts Q-values based on similarities in state and action features. This deep network for Q-values is parameterized by a set of parameters, θ and the goal is to learn values for θ so that a good Q-function is learnt.

Network parameters θ are learned using an iterative approach that employs gradient descent on the loss function. Specifically, the loss function at each iteration is defined as follows:

$$\mathcal{L}_\theta = \mathbb{E}_{(e\sim U(\mathcal{J}))}[(y^{DQN} - Q(s, a; \theta))^2] \tag{2.4}$$

where $y^{DQN} = r + \gamma \text{max}_{a'} Q(s', a'; \theta^-)$ is the target value computed by a target network parameterized by previous set of parameters, θ^-. Parameters θ^- are frozen for some time while updating the current parameters θ. To ensure independence across ex-

[1] Temporal abstraction (also known as action abstraction), augments the action space which allows the agents to act on a less granular time-scale reducing the number of of decisions to optimize for.

periences (a key property required by Deep Learning methods to ensure effective learning), this approach maintains a replay memory $\mathcal{J}$ and then samples experiences from that replay memory. Like with traditional Q-Learning, DQN also typically employs an ϵ-greedy policy.

Advantage Actor Critic (A2C)

DQN learns the Q-function and then computes policy from the learnt Q-function. A2C is a policy gradient method that directly learns the policy while ensuring that the expected value is maximized. To achieve this, A2C employs two deep networks, a policy network to learn policy, $\pi(a|s;\theta_p)$ parameterized by θ_p and a value network to learn value function of the computed policy, $V^{\pi}(s;\theta_v)$ parameterized by θ_v.

The policy network parameters are updated based on the policy loss, which in turn is based on the advantage ($A(s,a;\theta_v)$) associated with the value function. Formally, the policy loss is given by:

$$\mathcal{L}_{\theta_p} = \mathbb{E}_{(e\sim U(\mathcal{J})}[\nabla_{\theta_p} \log \pi(a|s;\theta_p) A(s,a;\theta_v)] \quad \text{where,}$$
$$A(s,a;\theta_v) = R - V(s;\theta_v) \tag{2.5}$$

Here R is the k-step (or till the end of the episode) discounted reward following the current policy and using a discount factor γ.

2.3 Multi-Agent Reinforcement Learning

The Multi-Agent Reinforcement Learning (MARL) framework is based on the same model given in Figure 2.1 but, now instead of single agent there are multiple agents interacting with the environment. As shown in Figure 2.2, there are multiple agents which interact with the environment simultaneously and state transitions are the result of joint action of all the agents. In this section, we first discuss the challenges of MARL and then provide the various classification of MARL algorithms.

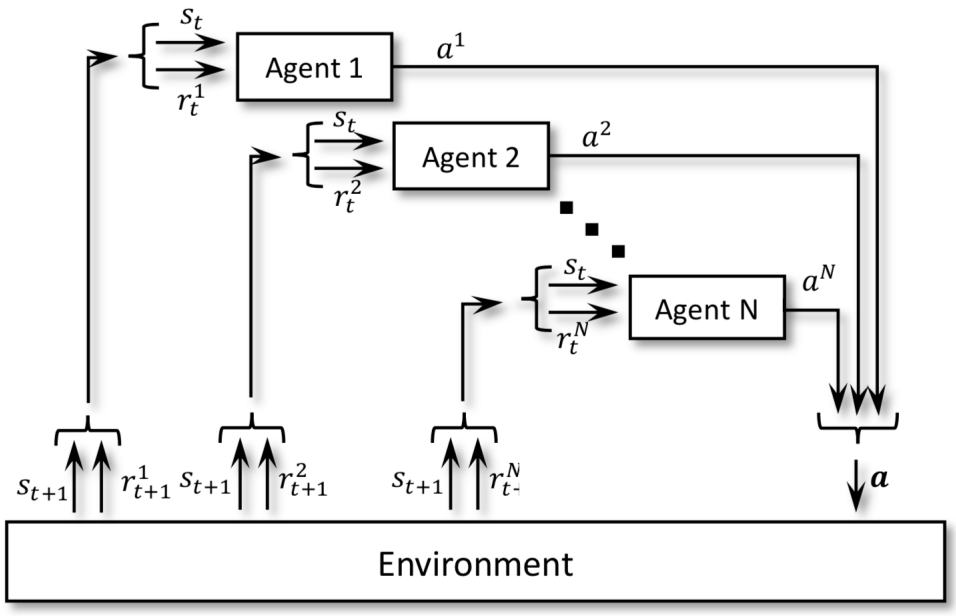


Figure 2.2: Multi-agent RL framework.

2.3.1 Challenges of MARL

Learning in multi-agent environment is inherently more complex than single-agent case, as agents interact with the environment at the same time. This makes the environment non-stationary and invalidates the guarantees established for single agent cases as most of the single agent algorithms assume a stationary environment. Following are the main challenges of MARL

- In single agent RL, agents adapt their strategy according to the reward signal they receive and the reward is purely due to their own action. Contrary to it, in MARL agents also need to adapt to other agents' learning and actions. The reward they receive is due to the joint action of the agents.

- Single agent RL learns values of each state-action pair. It can be extended to MARL by estimating values of state-joint-actions, but as number of agents increase, the size of state-action pair becomes intractable. Hence extending single agent RL to MARL is not scalable.

- MARL agents do not always have full view of the environment and to learn an optimal policy they need to predict/access actions of other agents. This makes

learning in multi-agent setting difficult.

- To learn the optimal/equilibrium policy in multi-agent setting, communication or coordination between agents are needed which is not always feasible in real-world settings.

2.3.2 MARL Algorithms

MARL algorithms can be classified based on multiple feature of both multi-agent systems and agents such as type of tasks to be performed, awareness of the agents, the way non-stationarity is handled etc. In this section we will focus of two types of classifications - independent learner (IL) vs. joint action learner (JAL) and cooperative learning vs. competitive learning.

IL vs. JAL

A straightforward way of extending single agent RL algorithms to MARL is to ignore the presence of other agents and learn independently. An independent learner estimates the local state-action value function $Q_i^{\pi_i}(s, a_i)$ and tries to find a local policy π_i that maximizes the expected long run reward for its own local action a_i.

$$Q^{\pi_i}(s, a_i) = \mathbb{E}_{\pi_i}\Big\{ \sum_{k=0}^{\infty} \gamma^k r^{t+k+1} | s^t = s, a^t = a_i \Big\}$$

The main advantage of IL approach is that size of the state-action table is independent of the number of agents, which make the approach scalable, but it comes at the cost of loss of theoretical guarantees.

On the other hand, JALs model the strategies of the other agents by learning state-joint-action values $Q_i^{\pi_i}(s, \boldsymbol{a})$ where $\boldsymbol{a}$ is the joint action. It implies that the agents can observe the actions of others. The JAL also maintains beliefs about the strategies of

other agents.

$$Pr_j^i(a_j) = \frac{C_j^i(a_j)}{\sum_{a'_j \in A_j} C_j^i(a'_j)}$$

Where $Pr_j^i(a_j)$ is agent i's model of agent j's strategy and $C_j^i(a_j)$ is the count of number of time agent i has observed agent j selecting action a_j. A JAL then computes expected value $EV(s, a_i)$ of its local action a_i as follows

$$EV(s, a_i) = \sum_{a_{-i} \in A_{-i}} Q_i(s, a_i, a_{-i}) \prod_{j \neq i} Pr_j^i(a_j)$$

Where a_{-i} is joint actions of other agents. The JAL then computes optimal policy based on the expected value of its local actions.

Cooperative vs. Competitive Learning

This classification of MARL is based on the goal of the learning. In cooperative MARL the agents have the same reward function and the learning goal is to maximize the common long term reward. In competitive MARL, on the other hand, each agent has a distinctive selfish goal. It should be noted that a cooperative MARL setting does not necessary mean that agents can view actions of other agents. Even if the agents can view the join action, to learn and execute the optimal policy there is a need of coordination/communication among the agents.

In a fully competitive MARL, a popular method is to apply minimax principle - maximize one's long term reward under the worst-case assumption that the opponents agents will always attempt to minimize it. The learning which is neither fully cooperative nor fully competitive falls under mixed category.

The aggregation system is an example of competitive domain.

2.4 Knowledge Based Learning

Knowledge based reinforcement learning deals with incorporating domain knowledge into the learning to guide exploration. Reward shaping is one of such approach which provides additional reward representative of prior knowledge in additional to the reward received from the environment. More formally, Q-values in Q-learning algorithm are updated as follows

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + F(s,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.6}$$

where $F(s,s')$ is a general form of any state based reward shaping function. Reward shaping has been shown to improve learning (Ng, Harada, & Russell, 1999b; Randløv & Alstrøm, 1998) in many experiments. However when used improperly, it can result into agent learning unexpected/undesired behavior. For example, (Randløv & Alstrøm, 1998) demonstrated that it might help to provide additional reward to stay balanced while learning to ride a bicycle. However, the learning agent discovered that it could benefit more by staying balanced and cycling in circles than reaching the destination.

To avoid such behavior, (Ng, Harada, & Russell, 1999a) proposed potential based reward shaping, where additional reward is computed as the difference in some potential function φ defined over states. The potential based reward is computed as follows

$$F(s,s') = \gamma\varphi(s') - \varphi(s) \tag{2.7}$$

γ is the same discount factor used in update rule provided in Equation 2.6. The potential of a state represents the goodness of the state, for example, potentials of states close to goal state are generally set to be more than the states that are far from the goal state. Potential based reward shaping algorithms do not alter the optimal policy in case of single agent learning. However, in case of multi-agent learning, it is not guaranteed that the algorithm will converge to the same policy if potential based reward shaping is used. In multi-agent system, it has been shown that potential based reward shaping does not

alter the underlying game structure. These guarantees hold if the potential of a state is static. If the potential is dynamic in nature, the above-mentioned guarantees only holds if the additional reward is of the form

$$F(s, t, s', t') = \gamma\varphi(s', t') - \varphi(s, t) \tag{2.8}$$

2.5 Stochastic Games

Stochastic games (Shapley, 1953) (also called Markov games) are the multi-agent extension of Markov decision processes with action space being the joint action space and reward being the vector of rewards, one for each agent. They were first introduced in the field of game theory, and have now become a formal framework for studying MARL. The game is played in sequence of stages. At the beginning of each stage, the environment is in some state. The players select action and receive payoff which is dependent on the current state and the joint action of players. The environment then probabilistically moves to the next state. The process is repeated at the new state and continues for a finite or infinite number of stages. The model of stochastic games is represented by the tuple

$$< \mathcal{N}, \mathcal{S}, (\mathcal{A}_i)_{i\in\mathcal{N}}, \mathcal{T}, (\mathcal{R}_i)_{i\in\mathcal{N}} >$$

- $\mathcal{N}(|\mathcal{N}| = n)$is the set of agents.
- $\mathcal{S}$ is a set of states.
- $\mathcal{A}_i$ is the set of available actions to agent i and $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the joint action space.
- $T : S \times A \times S \to [0, 1]$ is transition function,
- $R_i : S \times A \to \mathbb{R}$ is reward function of player i.

Researchers have used stochastic games as the underlying model for non-cooperative MARL. Like MDPs, the goal for the player i is to find a policy that maximizes its long-term reward. A stochastic policy for player i, π_i, is a mapping that defines probability of selecting an action in a state. Formally, $\pi_i \in \mathcal{S} \times \mathcal{A}_i \rightarrow [0, 1]$, where $\sum_{a \in \mathcal{A}_i} \pi_i(a|s) = 1, \forall s \in \mathcal{S}$. $\boldsymbol{\pi} =< \pi_i, ..., \pi_n >$ is the joint policy of all the agents. Π_i is used to denote set of all possible policy available to agent i and $\Pi =< \Pi_1, ..., \Pi_n >$ is the set of all the joint policies. $< \pi_i, \boldsymbol{\pi}_{-i} >$ refers to the join policy where player i follows π_i while the joint policy of other players is $\boldsymbol{\pi}_{-i}$. In the discounted reward framework, the value of the joint policy $\boldsymbol{\pi}$ for agent i in state $s \in \mathcal{S}$, with discount factor γ is

$$V_i^{\boldsymbol{\pi}}(s) = \mathbb{E}_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^k r_i^{t+k+1} | s^t = s \Big\}$$

where $\mathbb{E}_{\pi}\{\}$ is the expected value of player i given that agents follows joint policy $\boldsymbol{\pi}$.

Given a joint policy $\boldsymbol{\pi}_{-i}$ of the other players, the best-response function $BR_i(\boldsymbol{\pi}_{-i})$ for player i is the set of all policies that are optimal. Formally,

$$\pi_i^* \in BR_i(\boldsymbol{\pi}_{-i}) \text{ if and only if } V_i^{<\pi *_i, \boldsymbol{\pi}_{-i}>}(s) \geq V_i^{<\pi *_i, \boldsymbol{\pi}_{-i}>}(s), \forall s \in \mathcal{S} \text{ and } \forall \pi_i \in \Pi_i$$

As the players are self-interested, the algorithms that solve stochastic games focus on finding Nash equilibrium solution.

A Nash Equilibrium in a n-player stochastic game is a joint strategy $\boldsymbol{\pi} =< \pi_i \times \cdots \times \pi_n >$ such that $\pi_i \in BR_i(\boldsymbol{\pi}_{-i}), \forall i \in \mathcal{N}$.

2.6 Non-atomic Congestion Games

Non-atomic Congestion Games (NCG) has either been used to model selfish routing (Roughgarden & Tardos, 2002; Roughgarden, 2007; Fotakis, Kontogiannis, Koutsoupias, Mavronicolas, & Spirakis, 2009) or resource sharing (Chau & Sim, 2003; Krichene, Drighès, & Bayen, 2015; Bilancini & Boncinelli, 2016) problems. Though the underlying model is the same, there is a minor difference in the way the model is

represented. Here we present a brief overview of NCG from the perspective of resource sharing problem as that is of relevance to contributions in this thesis.

In NCG, a finite set of resources $\mathcal{L}$ are shared by a set of players $\mathcal{N}$. To capture the infinitesimal contribution of each agent, the set $\mathcal{N}$ is endowed with a measure space: $(\mathcal{N}, \mathcal{M}, m)$. $\mathcal{M}$ is a σ-algebra of measurable subsets, m is a finite Lebesgue measure and is interpreted as the mass of the agents. This measure is non-atomic, i.e., for an agent i, $m(\{i\}) = 0$. The set $\mathcal{N}$ is partitioned into Z populations, $\mathcal{N} = \mathcal{N}_1 \cup ... \cup \mathcal{N}_Z$.

Each population type z possesses a set of strategies $\mathcal{A}_z$, and each strategy corresponds to a subset of the resources. Each agent selects a strategy, which leads to a joint strategy distribution, $\boldsymbol{a}$:

$$\boldsymbol{a} = (f_z^a)_{a \in \mathcal{A}_z, 1 \leq z \leq Z} \textbf{ with } \sum_{a \in \mathcal{A}_z} f_z^a = m(\mathcal{N}_z) \forall z$$

Here f_z^a is the total mass of the agents from population z who choose strategy a. The total consumption of a resource $l \in \mathcal{L}$ in a strategy distribution $\boldsymbol{a}$ is given by:

$$\phi^l(\boldsymbol{a}) = \sum_{z=1}^{Z} \sum_{a \in \mathcal{A}_z : l \in a} f_z^a$$

The cost of using a resource $l \in \mathcal{L}$ for strategy $\boldsymbol{a}$ is:

$$c_l(\phi^l(\boldsymbol{a}))$$

where the function $c_l(.)$ represents cost of congestion and is assumed to be a non-decreasing continuous function. The cost experienced by an agent of type z which selects strategy $a \in \mathcal{A}_z$ is given by:

$$\mathcal{C}_z^a(\boldsymbol{a}) = \sum_{l \in a} c_l(\phi^l(\boldsymbol{a}))$$

A strategy $\boldsymbol{a}$ is Nash equilibrium if:

$$\forall z, \forall a, a' \in \mathcal{A}_z : \textbf{ if } f_z^a > 0, \textbf{ then } \mathcal{C}_z^{a'}(\boldsymbol{a}) \geq \mathcal{C}_z^a(\boldsymbol{a})$$

Intuitively, it implies that the cost for any other strategy, a' will be greater than or equal to the cost of strategy, a. In other words, it also implies that for a population $\mathcal{N}_z$, all the strategies with non-zero mass will have equal costs.

Chapter 3

Related Work

In this chapter we explore related work on learning in non-cooperative multi-agent systems. We will discuss how different algorithms focus on solving different challenges of multi-agent reinforcement learning.

As discussed in previous chapter, researchers have also utilized game theoretic framework such as stochastic games and congestion games to solve MARL problems. We will also explore MARL algorithms in game theoretic framework.

We first discuss dynamic state abstraction approaches which is needed to decide zone structure.

3.1 Dynamic State Abstraction in RL

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

same optimal action are aggregated.

All the approaches are closely relevant and our iterative dynamic abstraction approach presented in Chapter 5 builds on these approaches. There are two key differences:

1. In all the above-mentioned works, action space remains constant. However, in our problem setting, our action space is correlated to the state space, as actions correspond to ”moving to a certain location or state”. When state space is altered during abstraction, action space also changes.

2. Secondly, our abstraction approach employs multiple iterations to modify abstractions based on learned information.

3.2 Independent Learners

An Independent Learner (IL) learns values of its own local action by disregarding presence of other agents. Though it is difficult to guarantee convergence in independent learning, researchers have proposed many heuristic algorithms. Lauer and Riedmiller (2000) proposed Frequency Maximum Q-values (FMQ) which increases the Q-values of actions that frequently produced good rewards in the past steers the agent toward coordination. However Kapetanakis and Kudenko (2002) exhibited that FMQ can fail if reward is strongly stochastic. Distributed Q-learning (Lauer & Riedmiller, 2000) is based on ”optimistic independent agents”, i.e. agents optimistically assume that all other agents will act to maximize their reward. Optimistic learners are known to overestimate Q-values, to overcome this problem, Hysteretic Q-learning (Matignon, Laurent, & Le Fort-Piat, 2007) uses two learning rates, one to update the Q-value if the target Q-value is more than the current estimate and uses another rate otherwise. Most of these algorithms are more suitable for cooperative settings whereas our domain of interest is more competitive in nature.

Few approaches have also been proposed for independent learning in competitive setting. Bowling and Veloso (2002) introduced WoLF principle, standing for ”Win or

Learn Fast". The WoLF heuristic uses two learning rates to adjust the policy: δ_w is the agent is currently winning, δ_l otherwise. The agent is assumed to be winning if the expected value of current policy is greater that the current expected value of average policy. A number of algorithms such as WOLF-IGA, WoLF-PHC (Bowling & Veloso, 2002), GIGA-WOLF (Bowling, 2005), PD-WoLF (Banerjee & Peng, 2003) uses WoLF principle for independent learning in competitive settings.

While these approaches are relevant for domain of our interest, all these approaches assume that the IL has full view of the joint state. Our independent learner (Chapter 6) can view only local observation of the state. Based on the number of other agents present in the local view, the IL predicts and controls the non-stationarity introduced due to the presence of other agents by learning policies that maximize (or minimize) the entropy on agent state distribution.

Despite the lack of theoretical guarantees, independent learning has been successfully applied on some applications (Sen & Sekaran, 1998; Busoniu, De Schutter, & Babuska, 2006; Wang & De Silva, 2006; Guo & Meng, 2008). In fact, Wang and De Silva (2006) exhibited that IL does a better job than a JAL (Team Q-learning, (Littman, 2001b)) algorithm in a complicated and unknown environment with many obstacles for multi-robot box pushing task. Motivated by these results, we propose our IL approach which uses entropy of agent state distribution to improve its learning.

3.3 Non-Stationarity in MARL

Presence of multiple independent learners in a multi-agent setting introduces the problem of non-stationary environment. Each agent learns its own Q-value/policy based on the state and its own action disregarding the fact that the payoff received by it is the result of joint actions of all the agents. Researchers have proposed various methods to tackle the non-stationarity of the environment. J. Foerster et al.(2017) provided multi-agent variant of importance sampling and proposed use of fingerprint to track the quality of other agents' policy to tackle the non-stationarity. Lanctot et al. (Lanctot et

al., 2017) used a game-theoretic approach to learn the best responses to a distribution over set of policies of other agents.

A comprehensive survey on MARL dealing with non-stationarity has been provided by Hernandez-Leal, Kaisers, Baarslag, and de Cote (Hernandez-Leal et al., 2017) where based on how the algorithms cope up with the non-stationarity they are categorized into five groups: ignore, forget, respond to target opponents, learn opponent models and theory of mind. The first two categories do not represent other agents, while the last three categories explicitly represent other agents (in increasing order of richness in models used to represent other agents).

Learning with Opponent-Learning Awareness (LOLA) was introduced by J. N. Foerster et al. (2017) which explicitly accounts for the fact that the opponent is also learning. Approaches that explicitly learn policies of all other agents are typically suitable for domains with few agents, as the state and action spaces are typically combinatorial in the number of agents.

3.4 Entropy Regularized Learning

Mnih *et al.* (Mnih et al., 2016) and Haarnoja *et al.* (Haarnoja et al., 2017) use policy entropy regularizer to improve performance of policy gradient approaches to Reinforcement Learning. The density entropy approaches provided in Chapter **??** are also based on the use of entropy, however, there are multiple significant differences. Policy entropy is entropy on policy that is relevant for single agent learning. We employ density entropy, which is entropy on joint state configuration in multi-agent settings. Unlike policy entropy which is a regularization term, considering density entropy requires fundamental changes to the network and loss functions. Policy entropy is used to encourage exploration, while density entropy helps in improving predictability of joint state configuration and reduces non-stationarity (by moving all learning agents to high entropy joint states) due to other agents' strategies. Finally, since policy entropy is for single agent exploration, it is complementary to density entropy.

3.5 Centralized Training Decentralized Execution

Centralized training with decentralized execution (CTDE) is another thread of work in multi-agent domains which has become very popular recently. In CTDE, it is assumed that extra information is available to the individual learners during training time. Generally, a joint action value function is learned during the training phase by a central entity. During execution, each agent acts independently without direct communication. It should be noted that to act independently during execution phase, the individual agents do not model the extra information provided to them during training phase in their learning. Sunehag et al. (2018) proposed Value Decomposition Network (VDN), a method that computes the joint action value function by summing up all the action value functions of each individual agent. Individual agents are then trained as a whole by updating the joint action value functions iteratively. QMIX (Rashid et al., 2018) utilizes a neural network to represent the joint action value function as a function of the individual action value functions and the global state information. The authors of (Lowe et al., 2017) extend the actor-critic methods to propose Multi-Agent Actor Critic (MAAC). It allows a centralized critic Q-function to be trained with the actions of other agents, while the actor needs only local observation to optimize its policy. J. N. Foerster et al. (2018) propose counterfactual multi-agent policy gradient (COMA), which employs a centralized critic function to estimate the action value function of the joint, and decentralized actor functions to make each agent execute independently. There are CTDE algorithms which learn a joint centralized policy assuming agents act independently (Nguyen, Kumar, & Lau, 2017). Recently, CTDE algorithm is used innovatively to reduce traffic congestion in maritime domain (A. J. Singh, Nguyen, Kumar, & Lau, 2019; A. J. Singh, Kumar, & Lau, 2020).

These approaches are either suitable for cooperative domain, or learn a joint centralized policy or are appropriate only for a few number of agents and do not scale very well for problem of our interest.

3.6 Potential Based Reward Shaping in MARL

The potential of a state represents the desirability of being in the state, for example, potentials of states close to goal state are generally set to be more than the states that are far from the goal state. In case of single agent learning, (Ng et al., 1999a) proved that potential based reward shaping does not alter the optimal policy for both infinite and finite state MDPs. (Wiewiora, 2003) further proved that agents learning with potential based reward shaping with Q-tables initialized to zero will learn identically to agents with Q-values initialized to same potential values. With good heuristic used for potential function, single agent learning has been shown to converge quickly (Ng et al., 1999a; Wiewiora, 2003; Wiewiora, Cottrell, & Elkan, 2003; Asmuth, Littman, & Zinkov, 2008).

Potential based reward shaping has also been used to improve performance of multi-agent reinforcement learning methods (Babes, Munoz de Cote, & Littman, 2008; S. Devlin, Kudenko, & Grześ, 2011; S. Devlin & Kudenko, 2011). In multi-agent system, it has been shown that potential based reward shaping can change the converged joint policy, however it does not change the Nash equilibrium of the underlying game (S. Devlin & Kudenko, 2011). These guarantees hold if the potential of a state is static. (S. M. Devlin & Kudenko, 2012) proved that potential based reward shaping with dynamic potential function is not equivalent to Q-table initialization. They also demonstrated that the optimal policy is not altered (single agent case) or Nash equilibrium remains consistent (multi-agent case) if the potential of a state is evaluated at the time the state is entered and used in both the potential calculation on entering and exiting the state.

In this thesis, we provide a policy based dynamic reward shaping algorithm where the central agent computes incentives based on potential of the states.

3.7 Equilibrium Learner

There has been a line of research over the past decade in regards to the development of equilibrium learning algorithms, as well as determining their conditions for conver-

gence. Researchers have focused on computing equilibrium policies in MARL problems, which is represented as learning in stochastic games (Shapley, 1953). Minimax-Q (Littman, 1994) is one of the early equilibrium-based MARL algorithm that uses minimax rule to learn equilibrium policy in two-player zero-sum games. Nash-Q learning (Hu, Wellman, et al., 1998) is another popular algorithm that extends the classic single agent Q-learning (Watkins & Dayan, 1992) to general sum stochastic games. At each state, Nash-Q learning computes the Nash equilibria for the corresponding single stage game and uses this equilibrium strategy to update the Q-values. (Littman, 2001a) proposed Friend-or-Foe Q-learning (FFQ) which has less strict convergence condition compared to Nash-Q. Another algorithm similar to Nash-Q learning is correlated-Q learning (Greenwald et al., 2003) which uses value of correlated equilibria to update the Q-values instead of Nash equilibria. In fictitious self play (FSP) (Heinrich et al., 2015) agents learn best response through self play. FSP is a learning framework that implements fictitious play (Brown, 1951) in a sample-based fashion. Unfortunately, all these algorithms are generally suited for a few agents and do not scale if number of agents is very large, which is the case in problems of interest in this thesis.

3.8 Best-Response Learner

In the best response based MARL, individual agents try to learn policies which are best response to the joint policy of the other agents. NSCP (non-stationary converging policies) learning was proposed by (Weinberg & Rosenschein, 2004) which models other agents in the environment and learns a best response policy. (Lanctot et al., 2017) uses deep reinforcement learning to compute the best response to a distribution over policies, but it assumes prior knowledge of a set of opponent policies. Learning with Opponent Learning Awareness (LOLA) was introduced by (J. Foerster et al., 2018) which minutely modifies the objective of the player to take into account their opponents' goals. Though these algorithms which model opponents are relevant to our work, they do not scale to a large number of agents present in aggregation systems.

Fictitious self play (FSP) (Heinrich et al., 2015) is an excellent example of learning of best response through self play. FSP is a machine learning framework that implements fictitious play (Brown, 1951) in a sample-based fashion. (Heinrich & Silver, 2016) proposed neural fictitious self play (NFSP) which combines FSP with neural network function approximator. Learning best response through self play in MARL are known to be scalable. However, they are often sub-optimal because environment becomes non-stationary from a single agent's perspective. While learning best responses, some recent work (Lowe et al., 2017; Nguyen et al., 2017; Y. Yang et al., 2018) consider presence of a central agent which provides extra information to the individual agents. Our work uses the similar framework but we also consider the fact that the objective of the central agent might not be aligned with the objective of individual agents.

Chapter 4

Model and Experimental Setup

In this chapter I discuss the underlying model and few experimental setups which I have used in my thesis.

4.1 Model

As discussed in Chapter 1, the interaction among the individual agents is anonymous and the payoff of an agent is dependent on the number of other agents selecting the same action. Building of this characteristic, in this section I provide underlying model of Anonymous Multi-Agent Reinforcement Learning (AyMARL) by modeling the decision making problem of individuals and the central agent as Markov Decision Processes (MDPs). As everyone (including the central agent) is self-interested, they are endowed with their own MDPs. Though these MDPs are correlated, each individuals act independently and use reinforcement learning methods to solve their MDP. Whereas the central agent does not act directly and learns from the experiences of the individuals.

4.1.1 MDP of Central Agent

The model of central agent for AyMARL is represented using the tuple:

$$\left\langle \mathcal{N}, \mathcal{Z}, \mathcal{S}, \{\mathcal{A}_z\}_{z\in\mathcal{Z}}, \mathcal{T}, \{\mathcal{R}_i\}_{i\in\mathcal{N}} \right\rangle$$

- $\mathcal{N}$ is the set of individual agents.
- $\mathcal{Z}$ is the set of local states [1] (ex. zone of a taxi). At any given time instant, $\mathcal{N}$ is partitioned into $|\mathcal{Z}|$ disjoint sets based on the local state of all the individual agents, i.e. $\mathcal{N} = \mathcal{N}_1^s \cup ... \cup \mathcal{N}_{|\mathcal{Z}|}^s$, where $\mathcal{N}_z^s$ is the set of individual agents present in the local state z when the global state is s.
- $\mathcal{S}$ is the set of global states (joint states), which is factored over individual agents' local states. More specifically, by utilizing the anonymity feature of the domain, I consider global state to be the number of agents present in all the local states. In case of taxi domain, location (zone) of an agent can be considered as its local state while vector of number of agents (or fraction of agents) present in each zone is the global state, i.e. $s =< |\mathcal{N}_1^s|, ..., |\mathcal{N}_{\mathcal{Z}}^s| >$.
- $\mathcal{A}_z$ is the action set for the agents present in local state z. In case of the taxi problem, an action $a_i (\in \mathcal{A}_z)$ for agent $i \in \mathcal{N}_z^s$ represents the zone to move to if there is no customer on board. We use $\boldsymbol{a}$ to represent the joint action of agents. The joint action is the aggregated action of individuals agents, i.e. $\boldsymbol{a} = ((f_z^a)_{a \in \mathcal{A}_z})_{z \in \mathcal{Z}}$, where f_z^a is fraction of agents in $\mathcal{N}_z^s$ selecting action a in state s.
- $\mathcal{T}$ is the transitional probability of environment states given joint actions.
- $\mathcal{R}_i$ is the reward function of agent i.

I would like the readers to refer to the framework of aggregation system provided in Figure 1.1. Environment provides an immediate payoff for the actions executed by the individuals and the central agent provides the corresponding payoff to the individuals after deducting its commission fee. I use $\omega_{iz}(s, a_i, \boldsymbol{a}_{-i})$ to denote the immediate payoff provided by the environment when agent $i \in \mathcal{N}_z^s$ takes action a_i in global state s and the joint action executed by other agents is $\boldsymbol{a}_{-i}$. $r(s, \boldsymbol{a})$ is the immediate reward corresponding to the social welfare value. Here, social welfare reward is defined as

[1] We use the terms *local state* and *zone* interchangeably in this thesis.

the cumulative reward of all the individuals plus reward of the central agent. $r(s, \boldsymbol{a})$ is given as follows

$$r(s, \boldsymbol{a}) = \sum_{z \in \mathcal{Z}} \sum_{i \in \mathcal{N}_z^s} \omega_{iz}(s, a_i, \boldsymbol{a}_{-i}) \tag{4.1}$$

Let η is the commission fee of the central agent, i.e., the payoff received by the central agent due to the action of agent i is $\eta\omega_{iz}(s, a_i, \boldsymbol{a}_{-i})$ and the corresponding immediate reward (denoted by $r_{iz}(s, a_i, \boldsymbol{a}_{-i})$) for the agent i is

$$r_{iz}(s, a_i, \boldsymbol{a}_{-i}) = (1 - \eta)\omega_{iz}(s, a_i, \boldsymbol{a}_{-i}) \tag{4.2}$$

Similarly, immediate payoff (denoted by $r_c(s, \boldsymbol{a})$) for central agent in state s for joint action $\boldsymbol{a}$ is given by

$$r_c(s, \boldsymbol{a}) = \eta \sum_{z \in \mathcal{Z}} \sum_{i \in \mathcal{N}_z^s} \omega_{iz}(s, a_i, \boldsymbol{a}_{-i}) \tag{4.3}$$

4.1.2 MDPs of Individual Agents

As I have provided learning methods considering different level of information being shared by the centralized entity, the state space and action space of individuals change based on the level of extra information they receive from the central agent. For example, for the independent learning method ILT, where learning is done from the offline GPS trajectories (Chapter 5), there is no extra information provided and their state space comprises of spatio-temporal features of their locations. Similarly, DE-DQN method (Chapter 6) which exploits interaction anonymity, the state space of individuals agents comprises of their location and number of other agents present in that location. Whereas for VMQ (Chapter 7), CL (Chapter 8) and IBQ (Chapter 9), the assumption is that the central agent shares full view of the environment state. I have explained the state space and action space of individual agents in detail in each chapter.

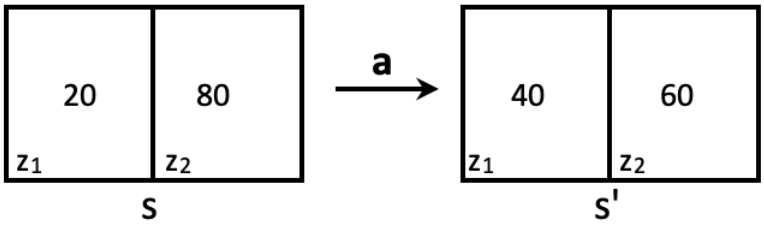


Figure 4.1: State transition of a grid-world example domain.

4.1.3 Learning Experiences

The learning experiences of individuals will have different level of granularity based on the level of information being shared to them. Whereas, the central agent learns from the aggregated experiences of the individuals. Let us assume a scenario where individuals are learning from their local observation and no extra information is being shared to them. Figure 4.1 represents state transition of a simple grid-world with 100 individual agents and 2 zones z_1 and z_2. The number in the grid represents number of agents present in that zone, i.e. $s =< 20, 80 >$ and $s' =< 40, 60 >$. Each zone has two feasible actions, a_1 = stay in the current zone and a_2 = move to the next zone. Suppose 10 agents in z_1 chose a_1 and remaining chose to move to z_2. Similarly, 48 agents in z_2 selected to stay whereas remaining agents selected action a_2. Hence, the joint action is given by $\boldsymbol{a} =<< 0.5, 0.5 >, < 0.6, 0.4 >>$. Suppose an agent i in z_1 selected action a_1 and transitioned to local state z_2 and received immediate payoff r_i. Also, let the reward for the central agent for joint action $\boldsymbol{a}$ is r_c. The learning experience of agent i is given by $< z_1, a_1, r_i, z_2 >$ whereas the central agent uses the aggregated experience $< s, \boldsymbol{a}, r_c, s' >$ for its learning.

4.2 Experimental Setup

In this section I explain the different experimental setups which have been used throughout this thesis to compare performances of the proposed algorithms. Following are the two simulators which I have used extensively for the experiments.

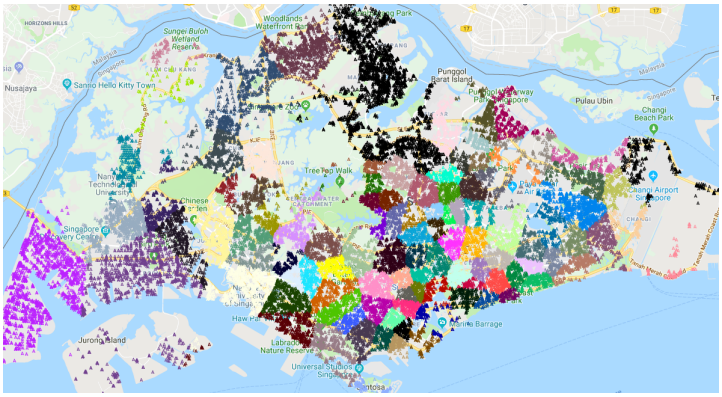


Figure 4.2: Road network of Singapore divided into zones

4.2.1 Taxi Simulator

I built a taxi simulator to simulate a taxi domain with the help of a real-world data set. The data set is from a large taxi-fleet company in Singapore and it contains GPS trajectory logs and information related to the corresponding trips. I used an algorithm (described in Section 5.4.3) to determine that the map of Singapore should be divided into 111 zones. Figure 4.2 shows the map of Singapore where road network is divided into zones. Then I used the data to compute the rate of demand arrival between any two zones and also the time taken to travel between the zones. Based on these values, I used a grid world to simulate the taxi domain. The taxi agents looking for demand decide either to stay in their current zone or move to the nearby zones. The demand in each zone is generated based on the value computed from the data and a demand is probabilistically assigned to an agent based on the number of agents present in the zone.

As discussed, each agent has their own MDP, hence they maintain their own learning framework (Q-table or deep Q network). However, simulating thousands of learning agents at the same time requires extensive computer resources hence, I could not perform a simulation with thousands of agents. Therefore, I computed the proportional demand for 100 agents and simulated for 100 agents.

4.2.2 Synthetic Online to Offline Service Simulator

For this example domain I generated synthetic data to simulate various combinations of demand and supply scenarios in online to offline service settings described in Chap-

ter 1. I used grid-world to depict map of a city and each grid is treated as a zone. Poisson process is a popular choice to model arrival of random and mutually independent events. Hence I used Poisson process to model the arrival of demand in our simulator with different zones assigned with different arrival rates. Demands are generated with a *time-to-live* value and the demand expires if it is not served within *time-to-live* time periods. Furthermore, to emulate the real-world jobs, the revenues are generated based on the distance of the trip (distance between the origin and destination grids). Aggregation companies typically deduct 20-25 % of the trip fare (Grab, 2019; Uber, 2019; Lyft, 2019) as commission, hence in our simulation, the central agent receives 20% of the share whereas individual agent which served the trip gets remaining 80% of the immediate payoff received due to the trip assignment. There are multiple agents in the domain and they learn to select next zones (they can reach within one time step) to move to such that their long term payoff is maximized. At every time step, the simulator assigns a trip to the agents based on the agent population density at the zone and the customer demand. In my experiments, I make a realistic assumption that it can take multiple time steps to complete a trip and the time taken to serve a trip is proportional to the distance between the origin and destination grids.

The revenue of an agent can be affected by features of the domain such as

- Demand-to-Agent Ratio (DAR): The average number of customers per time step per agent.
- Trip pattern: The average length of trips can be uniform for all the zones or there can be a few zones which get longer trips (for ex. airports which are usually outside the city) whereas few zones get relatively shorter trips (city center).
- Demand arrival rate: The arrival rate of demand can be either static w.r.t. the time or it can vary with time (dynamic arrival rate).

I performed exhaustive experiments on the synthetic data set where I simulated different combinations of these features.

4.2.3 Implementation Details

Apart from Chapter 5 where a tabular reinforcement learning method was feasible for independent learning, I have used deep neural network to estimate the values and policies. I used Adam optimizer (Kingma & Ba, 2014) and for For deep Q network the learning rate was set to 1e-4, whereas for policy gradient algorithms I used 1e-5 as learning rate. Two hidden layers were used with 256 nodes per layer. To prevent the network from overfitting, I also used dropout layer with 50% dropout between hidden layers. For DQN algorithms, I performed $\epsilon-$greedy exploration and ϵ was decayed exponentially. Training is stopped once ϵ decays to 0.05.

Chapter 5

Independent Learning from Offline Trajectories of Agents

In this chapter we provide Independent Learning from Trajectories (ILT) method where individuals learn from offline trajectories of other agents. More specifically, we provide methods to learn from offline data of Global Positioning System (GPS) trajectories collected from real-world taxi domain. As argued in Chapter 1, taxi aggregation domain is an anonymous domain and can be modeled as multi-agent system. A taxi in the domain can be treated as a self interested autonomous agent learning to act to maximize its log term revenue. Independent learners treat other agents as part of the environment and learn values for their local actions. In this chapter we show that independent learning using local observation yields excellent results and the individual agent is able to generate revenue at par with the top 10 percentile of the taxi drivers.

With the rapid development of information technology, sensing and networking technologies are widely used in transportation systems. Each taxi's status and its GPS location can be collected in real time. Relying on these advances, aggregation systems such as Uber, Grab, Lyft etc. have been able to activate more cars that act like taxis thereby significantly improved customer experience by reducing wait times and increasing availability. In the process, the companies generate enormous amount of GPS data which contains information related to trajectories of taxis and trips it served.

In this chapter, we focus on learning from an individual driver's perspective by using offline movement trajectories and trips of other drivers.

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

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

5.1 Challenges

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

- Typically in spatio-temporal problems, an abstraction (grouping) of locations and time is employed as the state space. For the problem of interest, such an abstraction can work for representing the state space; however, the "goodness" of the selected abstraction is not easy to ascertain. The effectiveness of abstraction is dependent on value of learned policy and learned policy is dependent on abstrac-

tion.

- Another challenge is with respect to understanding actions of drivers from the data, specifically when they are cruising. There can be multiple data points that are combined to retrieve one action and in many cases there can be multiple interpretations to drivers' action during "cruising", such as wandering around in circles, going to a taxi stand, going to a specific street, etc.

- RL relies on learning not just from single decisions but from a sequence of decisions. Therefore, the data needs to be annotated appropriately (with states, actions and reinforcements) and then condensed into learning episodes.

- The final challenge is evaluation. While performing human experiments with actual drivers would be the ideal case, due to the capital intensive nature of such experiments, it is not feasible to consider many drivers. Therefore, we provide a detailed simulator that allows performance comparison of actual drivers (using human intuition) and our agent that uses ILT.

To address the above mentioned challenges we first provide an annotation procedure that annotates the trajectory data with the decision taken by the taxi driver. This procedure also compactly represents annotated data as an activity graph for each cruising trajectory. Then, we employ Monte Carlo RL method to learn from the activity graphs by computing a static abstraction obtained by using clustering. To account for the cyclic dependence between state abstraction and learned policy, we provide an iterative abstraction approach that continually improves abstraction based on the learned information. Finally, we provide an evaluation method and use real data set to evaluate our policy.

5.2 Taxi Dataset

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

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

the following information:

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

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

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

5.3 From Taxi Dataset to Driver Activity Graphs

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

A ”cruising trajectory” starts when a taxi goes to ”free” state and ends when it goes to a ”non-free” state (passenger on board, busy, break, off-line, on call etc.). We mine cruising trajectories of drivers from the dataset and annotate the trajectories with the decisions made during the course of trajectory. To explain the details, let us consider the example trajectory shown in Figure 5.1, where a taxi driver’s cruising trajectory started at A and ended at E.

Initially, we start out by assuming that the driver made the decision to go to E at A

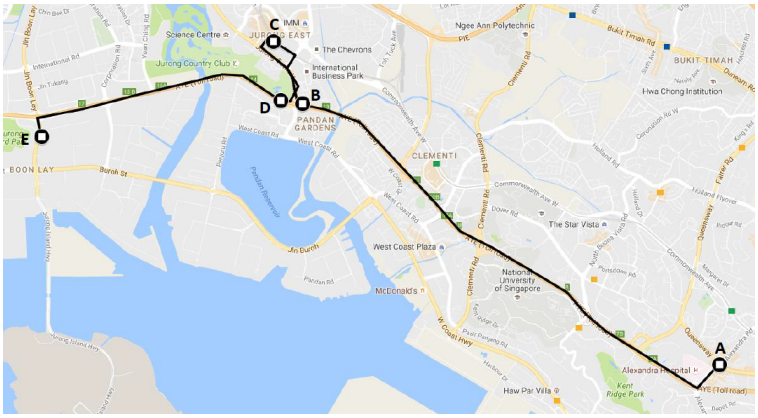


Figure 5.1: A cruising trajectory which starts at A and terminates at E. B, C and D are intermediate decision coordinates.

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

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

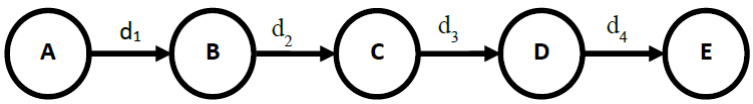


Figure 5.2: Activity graph for the cruising trajectory displayed in Figure 5.1. d_1, d_2, d_3 and d_4 are intermediate distances travelled. Nodes contain information about decision time epoch and GPS coordinate of the node. Terminating node E additionally stores trip information started there, if any.

Once the data is annotated, we then convert each cruising trajectory into an activity graph to get a summary view of driver activities. The activity graph can be viewed as

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

a directed graph with decision coordinates as nodes. Distance travelled between the coordinates is treated as weight of the edge between them. The terminating node of the activity graph also contains information about revenue earned. If the trajectory ended with getting a trip, the revenue earned is equivalent to the fare of the trip minus the cost of travel for the trip. The revenue earned is treated as zero if the trajectory ended without finding a passenger (taxi state changing to break, busy etc.).

5.4 Independent Learning from Trajectories (ILT)

We now provide reinforcement learning model of ILT. As the individual agent learns from the offline GPS trajectories and the objective is to learn from the local observations, we use spatio-temporal information from the data as the state space. Specifically, state is given as follows:

$$\langle \text{day-of-week, zone, time-interval} \rangle \tag{5.1}$$

We divide the entire map of Singapore into several zones[3] and zone division procedures are described in later parts of this section. Based on traffic intensity, time is divided into 6 time intervals: 0-6 hours, 6-9 hours, 9-12 hours, 12-17 hours, 17-20 hours and 20-24 hours. If there are n zones, there are n actions available to a cruising taxi, i.e., stay in the current zone or move to remaining $n-1$ zones.

5.4.1 Episodes

For reinforcement learning to be applied, we convert activity graphs into episodes. Each node in the activity graph represents a state and the subsequent node represents the action taken. We use the zone structure of the map and spatio-temporal information present in each node to convert activity graphs into a series of state-action pairs. The last node of the activity graph is always considered as terminal state of the episode. The cost of travel between nodes is determined by applying a fixed cost per km to the weight

[3]We use *zone* and *local state* interchangeably in this thesis.

of the edge. If the cruising trajectory ends with finding a passenger, a positive reward (equivalent to the fare of the trip minus cost to travel the trip) is awarded.

Equation 5.2 represents an episode for activity graph shown in Figure 5.2. S_x is the state and Z_x is the zone of node X, S_{term} is the terminal state.

$$(S_a, Z_b) \xrightarrow{c_1} (S_b, Z_c) \xrightarrow{c_2} (S_c, Z_d) \xrightarrow{c_3} (S_d, Z_e) \xrightarrow{c_4, R} S_{term} \tag{5.2}$$

$$R = \text{fare of trip } - \text{ cost to travel the trip}$$

$$c_i = d_i * \text{travelling cost per km}$$

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

5.4.2 Monte Carlo Estimation of Q Values

Monte Carlo (MC) method is a way of solving the reinforcement learning problem based on averaging sample returns. We use first-visit MC method to estimate the value of a state-action (s,a) pair (Sutton & Barto, 1998). Algorithm 2 provides the detailed algorithm to compute Q-values and the best action in each state.

Return of (s,a) pair ($Ret(s, a)$) in an episode is the cumulative reward accumulated till the end of the episode. For example in episode mentioned in equation 5.2, $Ret(S_c, Z_d)$ is $(R - c_4 - c_3)$. $Q(s, a)$, the value of (s, a) pair, is estimated as the average of the returns following the first time that the state s was visited and action a was taken in each episode. Value of s is defined as $\max_a Q(s, a)$. Line 9 computes the return for each (s, a) pair over all episodes and line 13 computes the average Q value for every (s, a) pair.

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

of actions. $\mathcal{S}_{learned}$ is the set of states for which we could learn optimal policy.

Algorithm 2 MC state-action value estimation

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}$
  $Q(s,a) \leftarrow 0$
  $Count(s,a) \leftarrow 0$
  $Ret(s,a) \leftarrow$ empty list
  $\mathcal{S}_{learned} \leftarrow$ empty set
for every episode in training episodes **do**
  for each (s,a) pair in the episode **do**
    $G \leftarrow$ return after first occurrence of (s,a)
    $Ret(s,a) \leftarrow Ret(s,a) + G$
    $Count(s,a) \leftarrow Count(s,a) + 1$
for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
  if $Count(s,a) \geq$ *min-count* **then**
    $Q(s,a) \leftarrow \dfrac{Ret(s,a)}{Count(s,a)}$
    if s not in $\mathcal{S}_{learned}$ **then**
      add s to $\mathcal{S}_{learned}$
for all $s \in \mathcal{S}_{learned}$ **do**
  $\pi(s) \leftarrow \underset{a}{argmax} Q(s,a)$

5.4.3 Zone Structure

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

Static Zones

In this method, we learn zone structure by merging zones to neighbour zones based on the training data available in each zone. A zone z maps to a state s and vice versa if

z is the zone of s as mentioned in equation 5.1. An episode is said to be relevant to z if there is any state s in the state-action pairs of the episode, which maps to z . We start with a large number of uniformly distributed zones and check how many relevant episodes are present in each zone, if the number is less than *min-count*, we merge the zone to its nearest zone. We repeat merging zones till each zone has sufficient data to learn from. In our experiment, we started with 500 zones and the final zone structure had 111 zones.

Dynamic Zones

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

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

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

Suppose s and a are state and action that map to zone z. If z is split, it affects

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

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

Proof. Suppose as per algorithm 2, $Ret(s', a) = x$ and $Count(s', a) = n$. After split all the (s', a) pair will either map to (s', a_1) or (s', a_2). Let, $Ret(s', a_1) = x_1$, $Ret(s', a_2) = x_2$, $Count(s', a_1) = n_1$ and $Count(s', a_2) = n_2$. We can see that

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

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

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

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

$$\textbf{Therefore: } Q(s', a_1) = Q(s', a) = Q(s', a_2) \tag{5.3}$$

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

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

$Q(s', a_1) > Q(s', a_2)$	$Q(s', a_1) > Q(s', a_2)$
$\Rightarrow \frac{x_1}{n_1} > \frac{x_2}{n_2}$	$\Rightarrow \frac{x_1}{n_1} > \frac{x_2}{n_2}$
$\Rightarrow \frac{x_1}{n_1} > \frac{x - x_1}{n - n_1}$	$\Rightarrow \frac{x - x_2}{n - n_2} > \frac{x_2}{n_2}$
$\Rightarrow \frac{x_1}{n_1}(n - n_1) + x_1 > x$	$\Rightarrow x > \frac{x_2}{n_2}(n - n_2) + x_2$
$\Rightarrow \frac{x_1}{n_1}(n - n_1 + n_1) > x$	$\Rightarrow x > \frac{x_2}{n_2}(n - n_2 + n_2)$
$\Rightarrow \frac{x_1}{n_1} > \frac{x}{n}$	$\Rightarrow \frac{x}{n} > \frac{x_2}{n_2}$
$\Rightarrow Q(s', a_1) > Q(s', a)$	$\Rightarrow Q(s', a) > Q(s', a_2)$

Therefore: $Q(s', a_1) > Q(s', a) > Q(s', a_2)$ (5.4)

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

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

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

Algorithm 3 Dynamic zoning

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

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

Algorithm 3 provides the pseudo code for learning zone structure dynamically. We start with dividing the map of Singapore into four large uniform zones and then split the zones repeatedly until further split is not possible. If tuples are from N months of data, T_n represents group of tuples which are from n^{th} month.

As after split there are new zone centres, it is possible that tuples which are mapped to a nearby zone now maps to new zones. We construct a list of affected tuples and realign them after we have split a zone. Algorithm 4 provides steps to decide if it is favourable to split a zone. Table 5.1 displays the learned number of zones for each time-interval on weekdays and weekends.

Algorithm 4 WorthSplitting(z)

Divide z into z_1 and z_2 using k-means clustering
if size of children zones $\leq$ *min-size* **then**
    return false
Construct tuple lists and Q-values for children zones
if $\max_a Q(s,a) \geq \max_a Q(s_1,a) + \max_a Q(s_2,a)$ **then**
    return false
if $\text{argmax}_a Q(s,a) == \text{argmax}_a Q(s_1,a)$
and $\text{argmax}_a Q(s,a) == \text{argmax}_a Q(s_2,a)$ **then**
    return false
return true

Table 5.1: Number of Dynamic Zones

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

5.5 Experiments

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

5.5.1 Evaluation Method

In this section "driver" means real world taxi drivers for whom we have historical data and "agent" means our learning agent which follows the learned policy. To evaluate the

Table 5.2: Notations

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

quality of policies learned by our approaches, we compare average revenue earned by our learning agent with the top percentile revenue of drivers. We also compare against revenue earned by greedy heuristics typically employed by drivers during cruising. For our experiments, we simulate the agent movements on real data. Since, we only advise one driver, we can assume that rest of the data about other drivers' movements does not change. Table 5.2 describes the terms and notation used in this section.

Simulation of Agent Movements

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

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

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

Driver revenue

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

Heuristic strategy

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

Agent revenue

We compute the agent's revenue for each time-interval for a given date. The actual time (t) of the day is also maintained. t is initialized with a start time of the interval. The evaluation ends when the value of t reaches the end of the time-interval. We observe the top earning drivers of the given date and time-interval and find their GPS location at the start of the interval. The corresponding state of the GPS location is used to initialize the agent's starting state. Following are the steps employed to compute the agent's revenue:

1. We use p^{st}_{assign} to assign a trip originated from state s. One random trip is assigned from the list of available trips.

2. If a trip was assigned to the agent, its next state becomes end zone of the trip and time t is updated. The fare of the trip (in the data) is considered as the revenue of

the agent.

3. If trip was not assigned, the agent waits for *decision-interval* (δ) minutes in the current zone before again taking decision based on learned policy.

4. After taking action, the agent moves to the suggested zone and t is updated based on precomputed travel time between zones for the given hour of the day.

5. After the agent reaches the next state s', it tries to assign a trip to the agent based on the $p^{s't}_{assign}$.

6. If the action was to stay in the same zone, a trip assignment is attempted after δ minutes. The assignment strategy remains same as explained in step 1.

7. Steps 1-6 are repeated until t is equal to the end time of the time-interval.

8. If a policy is not available for state s, the agent takes action based on heuristic strategy.

9. The agent's revenue is equivalent to the fare of all the assigned trips during the time interval minus cost of travelling associated with all the trips minus cost of cruising during the time interval.

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

5.6 Results

To evaluate the learning, we selected one month (not used for learning) and compared average agent revenue against average of top percentile revenues earned by drivers during that month. There were 20 weekdays and 10 weekends in the evaluation month. We find starting state of top 500 drivers in each time interval and use those states as initial

Algorithm 5 Agent's Revenue

Initialize the agent's initial state s
  t_{start} = start time of the time-interval
  t_{end} = end time of the time-interval
  $t \leftarrow t_{start}, rev \leftarrow 0$
  $fare \leftarrow 0, cost \leftarrow 0, cruising\_time \leftarrow 0$
while $t \leq t_{end}$ **do**
  assign trip with p^{st}_{assign} probability
  if trip was assigned **then**
    $s \leftarrow$ end state of trip
    $fare \leftarrow fare+$ trip fare
    $cost \leftarrow cost+$ cost of travelling the trip distance
    $t \leftarrow t+$ trip duration
  else
    $t \leftarrow t + \delta$
    if $s \in \mathcal{S}_{learned}$ **then**
      use learned strategy
    else
      use heuristic strategy
    go to advised zone
    $t \leftarrow t+$ time to travel to advised zone
    $cruising_time \leftarrow cruising_time+$time to travel to the advised zone
$rev \leftarrow fare - cost - cruising\_time * cruising\_cost$
return rev

state of the agent for evaluation. For each initial state, the revenue is averaged over 1000 executions. Hence for a given time interval and day, the agent revenue is averaged over 500,000 executions (500 different initial states * 1000 executions).

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

Tables 5.3 and 5.4 present the evaluation results for weekdays and weekends respectively. We see that early morning hours when drivers mostly roam to find passengers, the agent performs much better than the other drivers. During morning time interval on weekdays, the agent's revenue is 14.7% higher for static zones and 30% higher for dynamic zones. Apart from peak-hour in the evening, the agent always fares better than

Table 5.3: Performance of ILT on weekdays (revenues in SGD)

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

Table 5.4: Performance of ILT on weekends (revenues in SGD)

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

top 10 percentile revenue of drivers. We believe the slightly lower performance during 17-20 hours time-interval is due to traffic congestion. The agent always follows the shortest distance route while cruising, and there might be longer routes which take less time. As time taken to travel is taken from the real data, the agent might be wasting time while following the shortest route during peak hours. To get a better sense of the revenue earned by the agent with other drivers, we select a weekday and a weekend from the test data set. For the selected days we generate a list of top 500 drivers during each interval in terms of their revenues. We then find the corresponding starting state of drivers from their GPS logs. We use these starting states as the initial state of the agent during our evaluation. Figure 5.3 compares revenues of samples of an individual agent (a sample corresponds to one instantiation of trip allocation to the agent) with the top 500 drivers over different time intervals. The X-axis represents driver IDs of the best 500 drivers or 500 samples for the agent. We also evaluate utilization of the agent, which is computed as the percentage of time the agent is occupied during a given

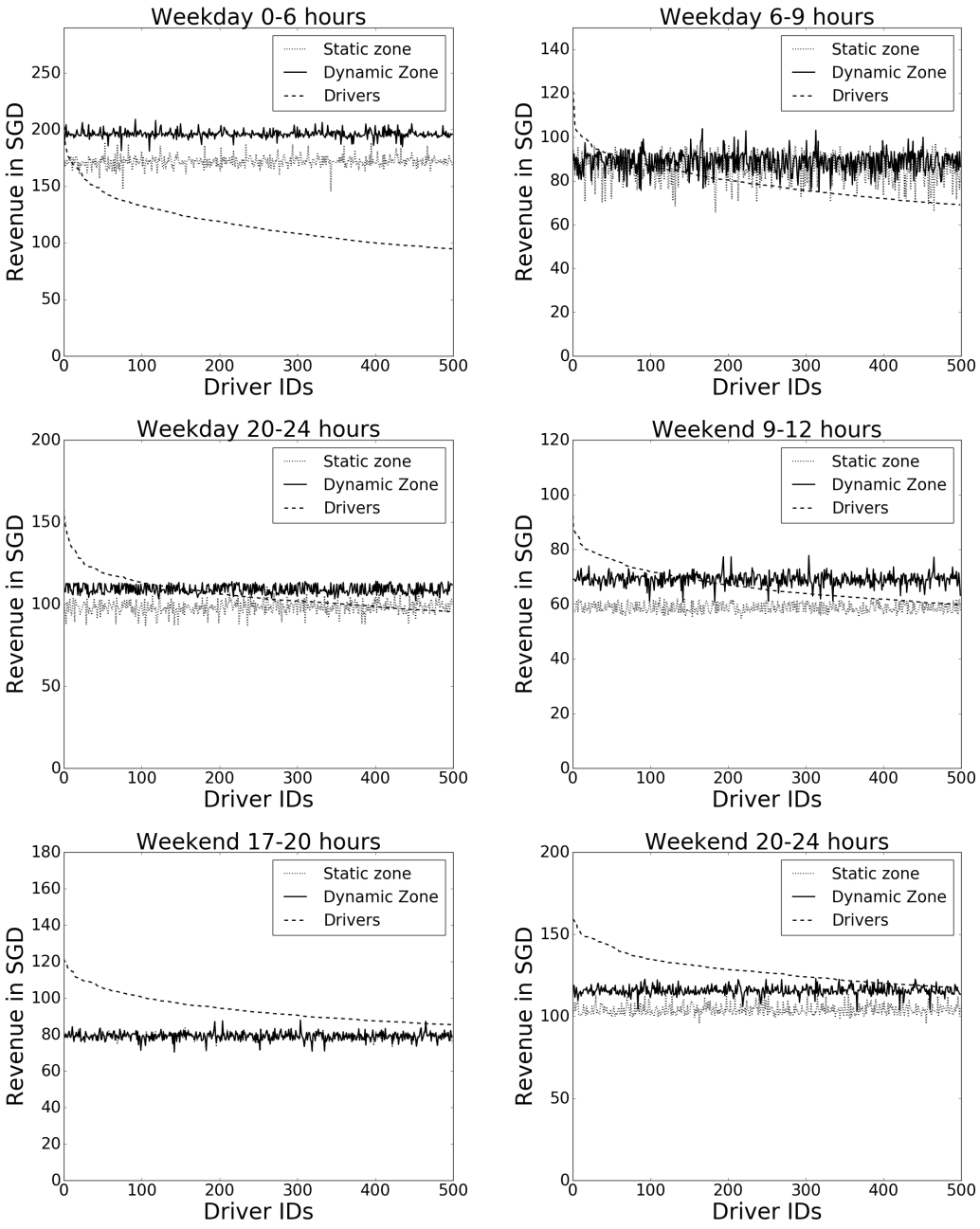


Figure 5.3: Revenue comparison for various time-intervals on weekdays and weekends.

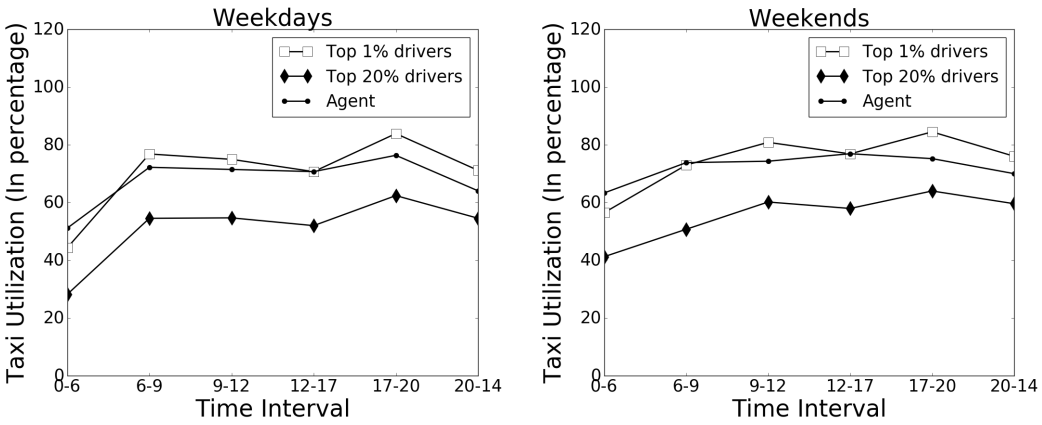


Figure 5.4: Taxi utilization comparison for weekdays and weekends.

time-interval. Figure 5.4 compares utilization on weekdays and weekend. We observe that though maximizing taxi utilization is not the learning objective, the agent's taxi utilization is always high (always better than 20 percentile and slightly worse than 1 percentile value of driver utilizations).

Table 5.5: Performance of ILT with multiple individual learners present in the system (weekdays)

Number of ILs in the environment	**0-6 hours**	**6-9 hours**	**9-12 hours**	**12-17 hours**	**17-20 hours**	**20-24 hours**
1	164.77	84.98	57.71	93.32	74.76	100.99
10	164.08	84.78	57.62	92.93	68.85	100.51
50	160.41	83.65	57.55	89.93	68.6	100.19
100	149.28	82.43	56.96	86.99	68.07	99.84
500	96.65	68.91	49.99	67.83	64.21	92.72
1000	69.89	55.69	41.28	54.57	58.62	81.74

Table 5.6: Performance of ILT with multiple individual learners present in the system (weekends)

Number of ILs in the environment	**0-6 hours**	**6-9 hours**	**9-12 hours**	**12-17 hours**	**17-20 hours**	**20-24 hours**
1	189.35	69.67	59.91	108.13	79.24	104.81
10	189.09	69.15	59.68	107.55	78.91	104.57
50	181.26	68.38	59.43	106.84	78.63	103.31
100	175.78	66.4	59.29	106.7	78.52	102.07
500	124.31	54.48	57.01	102.5	74.83	92.45
1000	103.62	39.31	49.56	93.41	66.58	82.87

Tables 5.5 and 5.6 compare the performance of ILT when there are multiple inde-

pendent learners present in the environment. As the number of free taxis in the data set are not constant and vary with time, we can not ascertain exact percentage of individuals present in the system. In the data set the number varied from 5500 to 8000 free taxis at any given time. Hence if there are 1000 independent learner in the simulation, we can safely say that the percentage of individuals in the environment is 11-15%. As can be seen in the tables, as the number of independent agents increase the performance of ILT start deteriorating.

5.7 Summary

In this chapter, we show that an independent agent, with no knowledge of the environment or taxi demand scenario, is capable of obtaining revenue which is comparable to (and in some cases higher than) revenue earned by top 10 percentile of drivers. Experimentally, we found that ILT is effective and except in one time interval (evening peak hour), the average revenue earned by the learned policy is better than the top 10 percentile revenue among all drivers. For some time intervals the agent performance is better than top 1 percentile revenue among all drivers. Though we employ the revenue maximization objective, we show that taxi utilization also increases significantly.

We also observed that with increase in number of individuals in the system, the performance starts declining. Hence, in the next chapter we provide learning methods when the individuals utilize anonymity feature of aggregation systems by considering that other individuals are also learning.

Chapter 6

Exploiting Interaction Anonymity to Improve Independent Learning

In Chapter 5 we presented learning method for individuals to learn a decision making policy. However, we assumed that other agents employ stationary and fixed strategies, rather than adaptive strategies. This led to a decline in performance with increase in number individuals present in the environment. Continuing our work on providing learning methods to the individuals in aggregation systems, in this chapter we propose methods where agents also account for the presence of other learning agents in the environment. More specifically, they utilize the anonymity feature of aggregation systems and consider number of other learning agents present in local state in their learning model. Though the individuals consider more information as compared to ILT, we still assume that the count statistics can be observed in their local state and no extra information is shared by the centralized entity. Therefore for the proposed method in this chapter we do not consider the presence of a centralized entity.

In this chapter, we improve the leading independent RL methods, specifically DQN (Deep Q-Networks) and A2C (Advantage Actor Critic) for solving problems of interest. Specifically:

- We provide mechanisms to exploit interaction anonymity in independent learning by extending the well known DQN (Deep Q-Network) and A2C (Advan-

tage Actor-Critic) methods. Specifically, we aim to predict and control the non-stationarity introduced due to the presence of other agents by learning policies that maximize the entropy on agent population distribution.

- To demonstrate the utility of our approaches, we performed extensive experiments on a synthetic data set (that is representative of many problems of interest) and a real taxi data set. We observe that our individual learners based on DQN and A2C are able to learn policies that are fair (minor variances in values of learning individuals) and most importantly, the individual and joint (social welfare) values of the learning agents are significantly higher than the existing benchmarks.

6.1 State Transition in AyMARL

In this section, compute state transition function based on individual agent's transitions in AYMARL. Figure 6.1 shows the global state transition dynamics for an anonymous multi-agent domain. We argue that in anonymous domain with large number of agents, individual agent's transitions are not dependent on other agents' states and actions directly but through agent population distribution, $\mathbf{d}$. That is to say, given $\mathbf{d}$, individual agent transitions are independent of each other in AyMARL. Let $d_z^s = |\mathcal{N}_z^s|$ is the number of agents in zone z in joint state s and $\mathbf{d}^s = (d_z^s)_{z \in \mathcal{Z}}$.

The state of an individual agent $i \in \mathcal{N}_z^s$ is given by (z, d_z^s) and we use $a_i \in \mathcal{A}_z$ to represent the action of the agent. We use $\mathcal{T}_i(z, d_z^s, a_i, z')$ to represent the probability of agent i to move to zone z' after taking action a_i in zone z given the environment state is s. The transition function of the environment state is given by

$$\mathcal{T}(s'|s, \mathbf{a}) = \prod_{z \in \mathcal{Z}} \prod_{i \in \mathcal{N}_z^s} \mathcal{T}_i(z, d_z^s, a_i, z') \tag{6.1}$$

Equation 6.1 can be rewritten in terms of $p(\mathbf{d}^{s'_{-i}}|\mathbf{d}^s, \mathbf{a}_{-i})$, which is the probability of other agents (except agent i) having an agent population distribution $\mathbf{d}^{s'_{-i}}$ given current

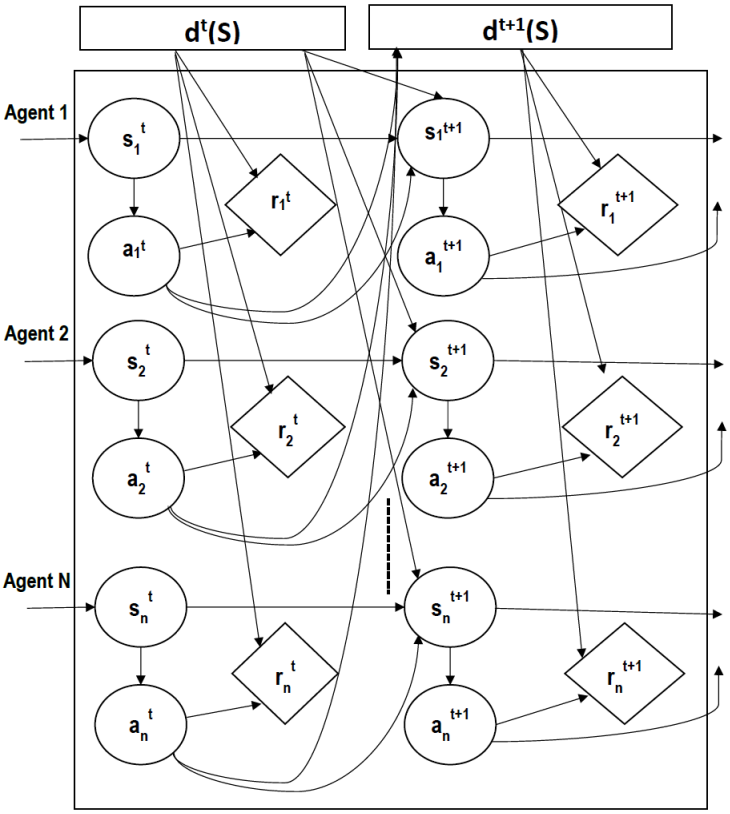


Figure 6.1: Agent population distribution $\mathbf{d}^t(s)$ and action a_i^t affects reward r_i^t of agent i. Distribution $\mathbf{d}^{t+1}(s)$ is determined by the joint action at time step t.

distribution $\mathbf{d}^s$ and joint action of the other agents $\mathbf{a}_{-i}$

$$\mathcal{T}(s'|s,\mathbf{a}) = \mathcal{T}_i(z, d_z^s, a_i, z') \cdot p(\mathbf{d}^{s'_{-i}}|\mathbf{d}^s, \mathbf{a}_{-i}) \tag{6.2}$$

Like with transitions, as shown in DBN, agent rewards are independent given agent population distribution, $\mathbf{d}$. In AyMARL, it is given by $r_{iz}(d_z^s, a_i)$.

6.2 Entropy based Independent Learning in Anonymous Domains

As discussed earlier, in this chapter we focus on independent learning approaches for AyMARL problems. In this section, we provide mathematical intuition and a general framework for independent learning based on the use of principle of maximum entropy in settings of interest.

Q-function expression for a given agent i in stochastic games (Hu & Wellman, 2003) is given by:

$$Q_i(s, \mathbf{a}) = r_i(s, a_i) + \gamma \cdot \sum_{s'} p(s'|s, \mathbf{a}) \cdot \max_{\mathbf{a}'} Q_i(s', \mathbf{a}') \tag{6.3}$$

The assumption in this chapter is that the individual agents do not get access to the global state s, or joint action $\mathbf{a}$. Q-function expression for an individual agent i that can observe number of other agents in the zone of agent i, i.e., d_z^s in stochastic games setting will then be:

$$Q_{iz}(d_z^s, a_i) = r_{iz}(d_z^s, a_i) + \gamma \sum_{z', d_{z'}^{s'}} \Big[p\big(z', d_{z'}^{s'} | z, d_z^s, a_i\big) \cdot \underset{a_i' \in \mathcal{A}_{z'}}{max} Q_{iz'}(d_{z'}^{s'}, a_i') \Big] \tag{6.4}$$

The above expression is obtained by considering (z, d_z^s) as the state of agent i in Equation 6.3.

The probability term in Equation 6.4 is a joint prediction of next zone and number of agents in next zone for agent i. Assuming a Naive Bayes approximation for $p\big(z', d_{z'}^{s'} | z, d_z^s, a_i\big)$, we have:

$$p\big(z', d_{z'}^{s'} | z, d_z^s, a_i\big) \approx p\big(z' | z, d_z^s, a_i\big) \cdot p\big(d_{z'}^{s'} | z, d_z^s, a_i\big) \tag{6.5}$$

While the term $p\big(z' | z, d_z^s, a_i\big)$ is stationary, the term $p\big(d_{z'}^{s'} | z, d_z^s, a_i\big)$ is non-stationary. $d_{z'}^{s'}$ is dependent not only on action of agent i but also on actions of other agents (as shown in Figure 6.1) and hence $p\big(d_{z'}^{s'} | z, d_z^s, a_i\big)$ is non-stationary.

Directly adapting the Q-learning expression of Equation 2.2 to the settings of interest, we have:

$$Q_{iz}(d_z^s, a_i) \leftarrow Q_{iz}(d_z^s, a_i) + \alpha[r + \gamma \cdot \max_{a_i' \in \mathcal{A}_z'} Q_{iz'}(d_{z'}^{s'}, a_i') - Q_{iz}(d_z^s, a_i)] \tag{6.6}$$

Since a part of the transition dynamics are non-stationary, this Q value update results in significantly inferior performance (as shown in experimental results) for existing

approaches (Q-learning, DQN, A2C). This is primarily because prediction of $d^{s'}_{z'}$ can become biased due to not representing actions of other agents. We account for such non-stationarity by ensuring that prediction of $d^{s'}_{z'}$ does not get biased and all options for number of agents in a state (that are viable given past data) are feasible.

The *principle of maximum entropy* (Jaynes, 1957) is employed in problems where we have some piece(s) of information about a probability distribution but not enough to characterize it fully. In fact, this is the case with the underlying probability distribution of $p\big(d^{s'}_{z'}|z, d^{s}_{z}, a_i\big)$ for all z' or in other words normalized $\mathbf{d}^{s'}$. For purposes of easy explainability, we abuse the notation and henceforth refer to the normalized agent population distribution as $\mathbf{d'}$ (i.e., $\sum_{z'} d_{z'} = 1$) corresponding to actual agent population distribution, $\mathbf{d}^{s'}$.

The *principle of maximum entropy* states that the best model of a probability distribution is one that assigns an unbiased non-zero probability to every event that is not ruled out by the given data. Intuitively, anything that cannot be explained is assigned as much uncertainty as possible. Concretely, when predicting a probability distribution, principle of maximum entropy requires that entropy associated with the distribution be maximized subject to the normalization (sum of all probabilities is 1) and known expected value constraints observed in data (for example average density $\bar{d}'_{z'}$ observed in $K_{z'}$ experiences). In case of $\mathbf{d'}$, this corresponds to:

$$\begin{aligned} &\max - \sum_{z'} d'_{z'} log\big(d'_{z'}\big) && :: [MaximizeEntropy] \\ &\textbf{s.t.} \sum_{z'} d'_{z'} = 1 && :: [Normalization] \\ &\frac{1}{K_{z'}} \sum_{k \in K_{z'}} d'^{k}_{z'} = \bar{d}'_{z'} && :: [ExpectedValue] \end{aligned} \tag{6.7}$$

The key challenge is in performing this constrained maximization of entropy for normalized agent density distribution, $\mathbf{d'}$ along with a reinforcement learning method. We achieve this by making two major changes to existing RL methods:

[*MaximizeEntropy*]: Including a term for entropy of $\mathbf{d'}$ referred to as $\mathcal{H}_{\mathbf{d'}}$ as

part of the reward term. This will ensure entropy is maximized along with the expected value.

[*Normalization*]: Normalized prediction is achieved through softmax computation on $p\big(d^{s'}_{z'}|z, d^{s}_{z}, a_i\big)$ prediction.

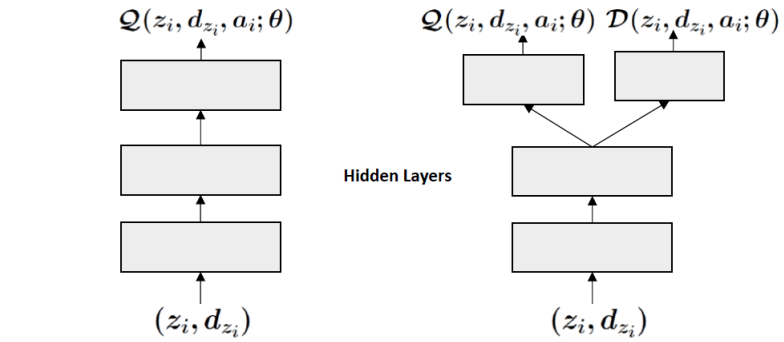
[*Expected Value*]: We predict normalized $\mathbf{d}'$ given observed experiences of $d^{s'}_{z'}$. The objective of the prediction is to minimize mean square loss between predicted and observed value of $d'_{z'}$. Due to minimizing mean square loss, we approximately satisfy the expected value constraint.

In domains of interest, there are two other key advantages to constrained entropy maximization of $\mathbf{d}'$:

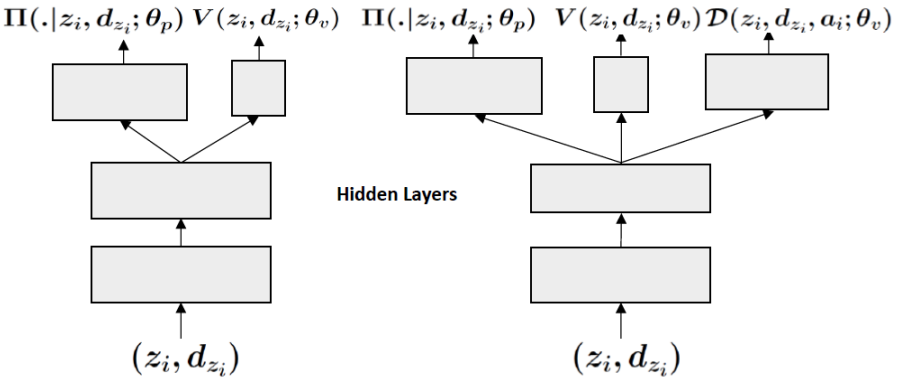
- *Controlling the non-stationarity*: The domains of interest have reward/expected reward values that decrease with increasing number of agents (e.g., chances of taxis getting assigned to customers and hence earning revenue are higher when there are fewer taxis in the same zone) . Due to this property, it is beneficial for agents to spread out across zones with demand rather than assemble only in zones with high demand. In other words, agents taking actions that will maximize the entropy of $\mathbf{d}'$ introduces predictability in agent actions and therefore reduces non-stationarity.

- *Reduced variance in learned policies*: There is homogeneity and consistency in learning experiences of agents because the rewards (revenue model) and transitions (allocation of demand to individuals) are determined consistently by a centralized entity (e.g., Uber, Deliveroo). The only non-stationarity experienced by an agent is due to other agent policies and/or potentially demand, so the impact of better predicting $\mathbf{d}'$ and controlling non-stationarity through maximizing entropy minimizes variance experienced by different learning agents.

Given their ability to handle non-stationarity better than tabular Q-learning, we implement this idea in the context of DQN (Mnih et al., 2015) and A2C (Mnih et al., 2016)



(a) DQN and DE-DQN



(b) A2C and DE-A2C

Figure 6.2: DE-DQN and DE-A2C networks

methods. It should be noted that an experience e in AyMARL is more extensive than in normal RL and is given by $\left(z, d_z, a_i, r_i, z', d_{z'}\right)$.

6.2.1 Density Entropy based Deep Q-Networks, DE-DQN

We now operationalize the general idea of applying principle of maximum entropy in the context of Deep Q-Networks by modifying the architecture of the neural network and also the loss functions. We refer to entropy for the predicted future agent population distribution as density entropy. We use $\boldsymbol{d}'$ to denote the predicted density distribution while $\mathbf{d}$ is the true density distribution.

As indicated in Equation 6.7, there are three key considerations while applying principle of maximum entropy: (a) maximizing entropy alongside expected reward; (b) ensuring expected value of the computed distribution follows the expected observed samples of $\mathbf{d}'$; and (c) finally ensure prediction of $\mathbf{d}'$ is normalized. We achieve these considerations using three key steps:

- *Including entropy in the Q-loss*: This step enables us to maximize entropy alongside expected reward. The Q-loss, $\mathcal{L}_\theta^Q$ is updated to include entropy of the predicted agent density for the next step, i.e., $\boldsymbol{d}'$. Extending from the description of DQN in Section 2.2.4, $\mathcal{L}_\theta^Q$ is given by:

$$\mathcal{L}_\theta^Q = \mathbb{E}_{(e\sim U(\mathcal{J}))}\Big[\Big(y^{DE-DQN} - Q_{iz}\big(d_z, a_i; \theta\big)\Big)^2\Big]$$

To maximize entropy of $\boldsymbol{d}'$ alongside the expected reward, the target value is:

$$y^{DE-DQN} = r + \beta\mathcal{H}_{\boldsymbol{d}'} + \gamma \max_{a_i'} Q_{iz'}(d_{z'}, a_i'; \theta^-)$$

Here $\mathcal{H}$ is the density entropy and β is a hyperparameter which controls the strength of the entropy.

- *Softmax on output layer*: We modify the architecture of the neural network as shown in Figure 6.2a. This involves introducing a new set of outputs corresponding to prediction of $\mathbf{d}^{s'}$. By computing softmax on the output, we ensure $\boldsymbol{d}'$ is normalized.

- *Introduce a new density loss, $\mathcal{L}_\theta^{\mathcal{D}}$*: This step enables us to minimize loss in prediction of agent density for the next step. Specifically, by minimizing loss we ensure that density entropy maximization occurs subject to observed samples of $\mathbf{d}'$. The independent learner agent gets to observe only the local density, $d_{z'}^{s'}$ and not the full density distribution $\mathbf{d}'$. Hence, we compute mean squared error (MSE) loss, $\mathcal{L}_\theta^{\mathcal{D}}$ for the local observation.

$$\mathcal{L}_\theta^{\mathcal{D}} = \mathbb{E}_{(e\sim U(\mathcal{J}))}[(d_{z'} - \mathcal{D}(z'|z, d_z, a_i; \theta))^2] \tag{6.8}$$

$\mathcal{D}(z, d_z, a_i; \theta)$ is the density prediction vector and $\mathcal{D}(z'|z, d_z, a_i; \theta)$ is the predicted density in zone z' if action a_i is taken in state (z, d_z). DE-DQN optimizes a single combined loss with respect to the joint parameter θ, $\mathcal{L}_\theta = \mathcal{L}_\theta^Q + \lambda\mathcal{L}_\theta^{\mathcal{D}}$. Here λ is the weightage term used for the density prediction loss.

6.2.2 Density Entropy based A2C, DE-A2C

Figure 6.2b depicts how A2C network can be modified to DE-A2C network. Similar to DE-DQN, DE-A2C also considers density entropy in value function and a density loss. In addition, DE-A2C has to consider policy network loss and this is the main difference between DE-A2C and DE-DQN.

DE-A2C maintains a policy network $\pi(.|z, d_z; \theta_p)$ parameterized by θ_p and a value network parameterized by θ_v. Value network maintains a value function output $v_{iz}(d_z; \theta_v)$ and a density prediction output $\mathcal{D}(z, d_z, a_i; \theta_v)$. R is k-step return from the experience. While computing the value function loss $\mathcal{L}^v_{\theta_v}$, density entropy is included as follows

$$\mathcal{L}^v_{\theta_v} = \mathbb{E}_{(e\sim U(\mathcal{J})}[(R + \beta\mathcal{H}_{\boldsymbol{d}'} - v_{iz}(d_z; \theta_v))^2] \tag{6.9}$$

Density prediction loss $\mathcal{L}^{\mathcal{D}}_{\theta_v}$ can be computed as given in equations 6.8. The value network loss $\mathcal{L}_{\theta_v}$ is the combination of value loss and density loss, $\mathcal{L}_{\theta_v} = \mathcal{L}^v_{\theta_v} + \lambda\mathcal{L}^{\mathcal{D}}_{\theta_v}$. Similarly, density entropy is included in the policy network loss as follows.

$$\mathcal{L}_{\theta_p} = \mathbb{E}_{(e\sim U(\mathcal{J})}[\nabla_{\theta_p}\log\pi(a_i|z, d_z; \theta_p) \cdot (R + \beta\mathcal{H}_{\boldsymbol{d}'} - v_{iz}(d_{z_i}; \theta_v))]$$

6.3 Experiments

In this section, we demonstrate that our approaches that employ density entropy alongside Q-function or value function are able to outperform leading independent RL approaches (DQN, A2C, Q-Learning). DQN, A2C and Q-Learning serve as lower bound baselines on performance. We also compare our results with Mean Field Q (MFQ) learning algorithm (Y. Yang et al., 2018) which is a centralized learning decentralized execution algorithm. MFQ computes target values by using previous iteration's mean action (mean of actions taken by neighboring agents), but in our experimental domain the previous mean actions are different for different zones, hence we use the modified MFQ algorithm where agents present in same zone are considered neighbors and previous iteration's mean actions of every zone is available to the all the learning agents.

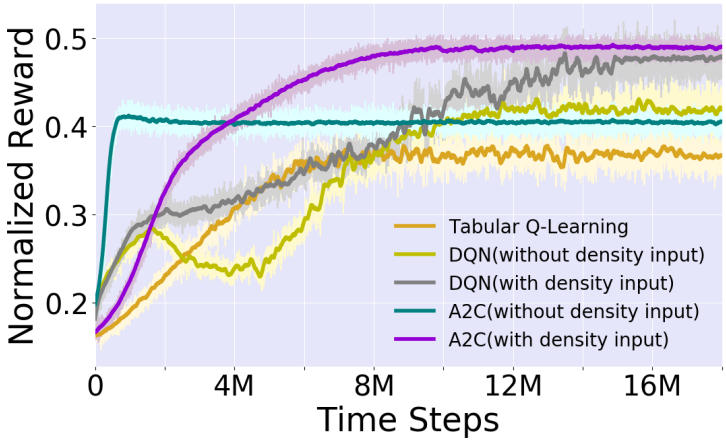


Figure 6.3: Comparison of tabular Q-learning, standard DQN and standard A2C .

As MFQ agents have more information than our DE algorithms, it serves as the upper bound baseline on performance.

We first provide results on a taxi simulator that is validated on a real world taxi dataset. Second, we provide results on a synthetic online to offline matching simulator under various conditions.

Computing Density Entropy from Local Observation

The independent learner agent gets to observe only the local density, $d_{z_i'}$ and not the full density distribution $\mathbf{d}'$. This makes computing a cross-entropy loss (or any other relevant loss) for density prediction difficult. Hence, as shown in Equation 6.8, we compute MSE loss and to get normalized density prediction $\boldsymbol{d}'$, we apply softmax to the density output $\mathcal{D}(z, d_z, a_i; \theta)$. Density entropy can be computed as $\mathcal{H}_{\boldsymbol{d}'} = -\boldsymbol{d}' \cdot \log(\boldsymbol{d}')$

6.3.1 Results

We now benchmark the performance of our learning approaches (DE-DQN and DE-A2C) with respect to DQN, A2C and MFQ. One evaluation period consists of 1000 (1e3) time steps[1] and revenue of all the agents is reset after every evaluation period. All the graphs plotted in the upcoming sub-sections provide running average of revenue

[1] A time step represents a decision and evaluation point in the simulator.

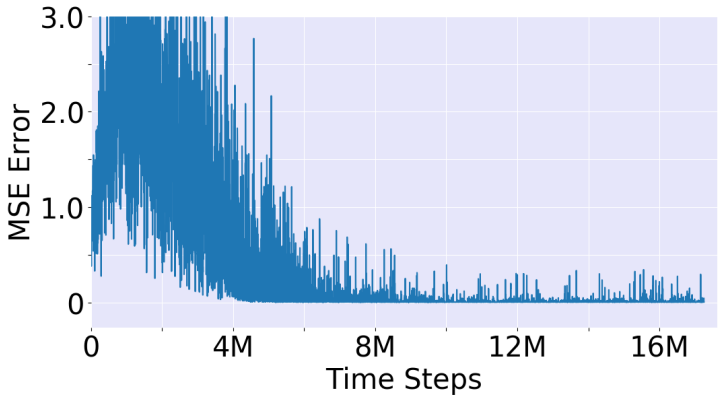


Figure 6.4: Error is density prediction.

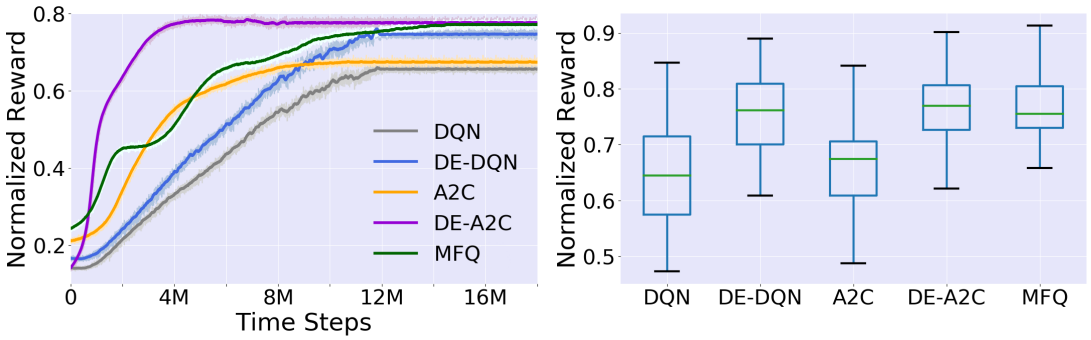
over 100 evaluation periods (1e5 time steps).

We evaluated the performance of all learning methods on two key metrics:

1. Mean payoff of all the individuals. This indicates social welfare of all the individual agents. Higher values imply better performance.

2. Variation in payoff of individual agents after the learning has converged. This is to understand if agents can learn well irrespective of their initial set up and experiences. This in some ways represents fairness of the learning approach. We use box plots to show the variation in individual revenues, so smaller boxes are better.

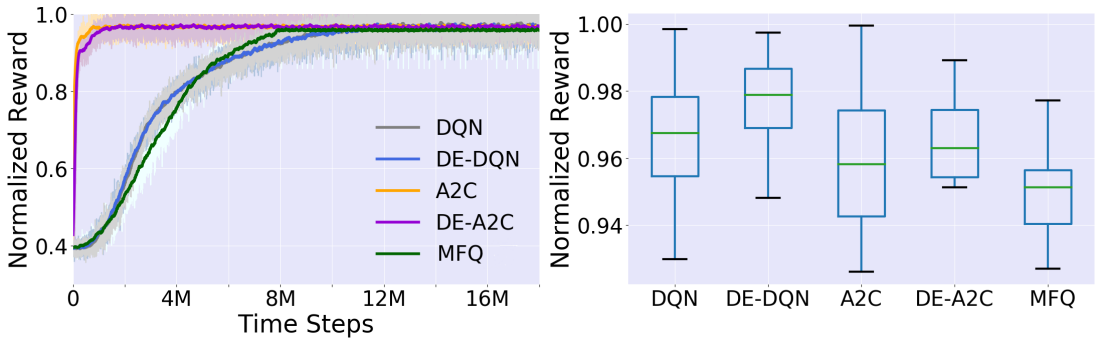
To provide a lower benchmark, we first compared our approaches with tabular Q-Learning. We also performed experiments with DQN and A2C algorithms when density input is not provided to the neural network. Figure 6.3 shows the comparison between tabular Q-learning; DQN with and without agent density input and A2C with and without agent density input where we plot average payoff of all the individuals (social welfare). We used non-uniform trip pattern with DAR $= 0.6$ for this experiment. We can see that DQN with agent density input and A2C with agent density input perform significantly better than the other three algorithms. For other demand/supply experimental setups also we obtained similar results. Hence, for our remaining experiments we used DQN and A2C with agent density input as our lower baseline algorithm. In Figure 6.4 we empirically show that the MSE loss between predicted density $\boldsymbol{d}$ and true density



(a) Social welfare of individuals

(b) Variance in individual payoffs

Figure 6.5: Real world dataset



(a) Social welfare of individuals

(b) Variance in individual payoffs

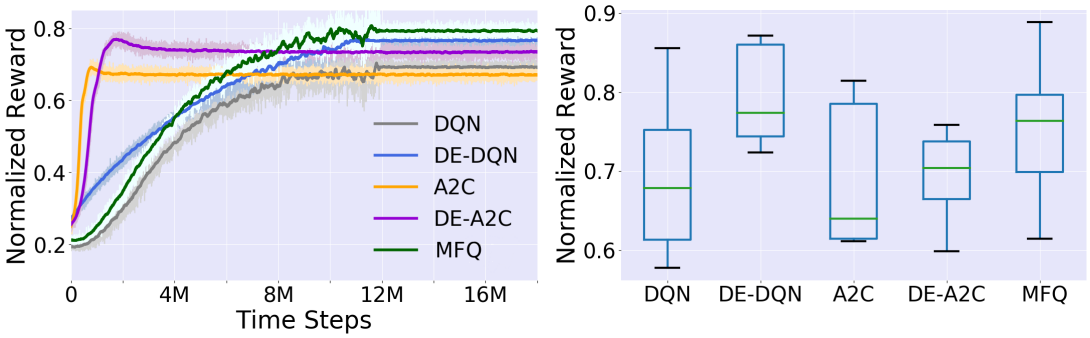
Figure 6.6: DAR $= 0.25$ with uniform trip pattern.

distribution **d** converges to a low value (≈ 0.2).

First plots in Figures 6.5 - 6.10 provide comparison of social welfare value of all the individuals for the respective experimental setups[2]. Second plots uses boxplots to show the variation in the individual average revenues after the learning has converged.

Experiment on real-world dataset (Figure 6.5) show that DE-DQN and DE-A2C outperformed DQN and A2C approaches by ≈10%. For low DAR (Figure 6.6) there is no overall improvement in the social welfare (Figure 6.6a) which is expected as with too many agents and too less demand, random action of some or the other agent will serve the demand. However, the variance is lower than the baseline algorithms. In Figures 6.7 -refent-dar75 we can see that with increase in DAR, the performance gap starts increasing ($\approx 7\%$ for DAR $= 0.4$, ≈10% for DAR $= 0.6$ and ≈15% for DAR $= 0.75$) DE-DQN and DE-A2C were able to perform as well as MFQ even with local obser-

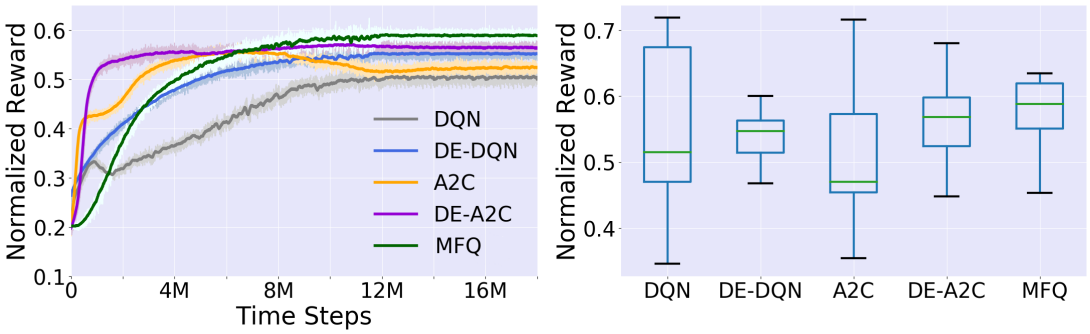
[2]The demand arrival rate is static and trip pattern is uniform for the experimental setups until stated otherwise.



(a) Social welfare of individuals

(b) Variance in individual payoffs

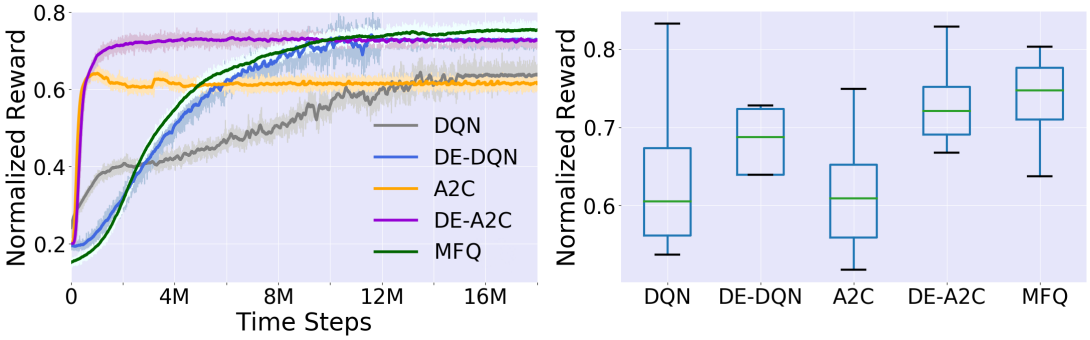
Figure 6.7: DAR $= 0.4$ with dynamic demand arrival rate.



(a) Social welfare of individuals

(b) Variance in individual payoffs

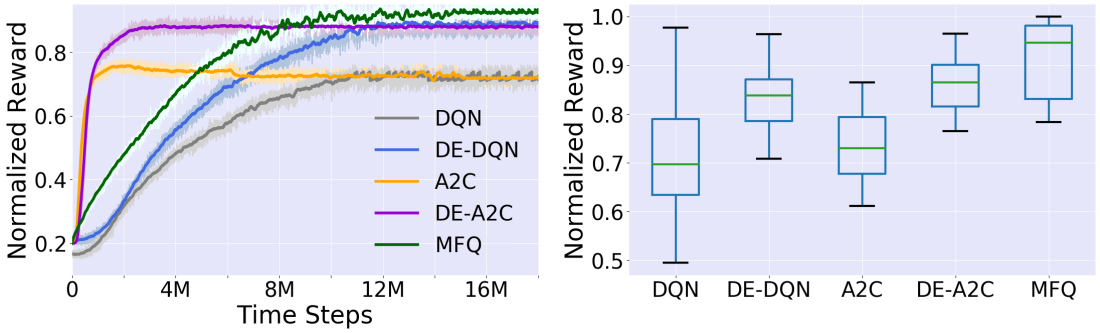
Figure 6.8: DAR $= 0.5$ with non-uniform trips pattern.



(a) Social welfare of individuals

(b) Variance in individual payoffs

Figure 6.9: DAR $= 0.6$ with uniform trip pattern.



(a) Social welfare of individuals

(b) Variance in individual payoffs

Figure 6.10: DAR $= 0.75$ with non-uniform trip pattern.

vations on the real world dataset validated simulator. Even on the synthetic simulator, the gap was quite small (about 5% for DAR $= 0.75$). The gap is lesser for lower values of DAR. The variation in revenues was significantly[3] lower for DE-DQN and DE-A2C compared to other approaches, thus emphasizing no unfair advantage for any agents.

6.4 Summary

In this chapter we provided methods where individuals exploits the anonymity feature of the aggregation systems. Due to the key advantages of predicting agent density distribution (required for accurate computation of Q-value) and controlling non-stationarity (due to agents changing policies), our key idea of maximizing agent density entropy is extremely effective for the individuals in anonymous multi-agent settings. We were able to demonstrate the utility of our approaches on a taxi simulator validated on real world data set and an online to offline service simulator simulated using synthetic data set. The results exhibited that the learning methods provide better results than existing algorithms with respect to the overall value for all agents and fairness in values obtained by individual agents.

The proposed methods presented in the thesis till now learn only from the local observation and do not utilize the presence of the central agent in the system. In upcoming chapters, we show that how presence of a central agent can be utilized to improve the

[3]Differences between DE algorithm and their counterpart base algorithm are statistically significant at 5% level.

both individual and system wide performances.

Chapter 7

Learning Equilibrium Policies

In Chapter 5 and Chapter 6, we provided methods for individual learning from local observation in the aggregation system. In this chapter, we assume increased level of information sharing from the centralized entity. As discussed in Chapter 1, the aggregation companies have full view of the system and they can learn policies which improve overall performance. Hence, in this chapter we propose a method where central agent learns from the learning of individual agents and share information about the global state. More specifically, the central agent learns policy based on the action-values estimated by the individual agents.

In aggregation systems, some suppliers can receive lower profits (e.g., due to servicing low demand and high cost areas) in maximizing overall profit for the centralized entity. This results in suppliers moving out and creating instability in the system. One way of addressing this instability is to learn equilibrium solutions for all the players (centralized entity and individual suppliers). We note the fact that even though there are thousands of individual players, their contribution to overall social welfare is infinitesimal and the effect of a single agent on the environment dynamics is negligible. We also observe that the learning in aggregation system bears a resemblance to learning problems in multi-stage non-atomic congestion games. Using these insights, we focus on learning equilibrium policies in this chapter.

Specifically, the key contributions of this chapter are as follows:

- We use the equilibrium solution from games with infinite players to derive a learning method for games with large (yet finite) number of agents.
- We propose Stochastic Non-atomic Congestion Games (SNCG) model which is suitable for infinite number of infinitesimal agents. It also considers anonymity in the interaction of the agents.
- We then provide key theoretical properties of equilibrium in SNCG problems.
- Inspired from the equilibrium property of SNCG, we provide an algorithm for finitely many agents which reduces the variance in agent values to move joint solutions towards equilibrium solutions.
- We provide detailed experimental results on multiple benchmark domains from literature and compare against leading MARL approaches.

In this chapter, we build on key results from non-atomic congestion game (Fotakis et al., 2009; Roughgarden & Tardos, 2002; Roughgarden, 2007; Chau & Sim, 2003; Krichene et al., 2015; Bilancini & Boncinelli, 2016) by accounting for transitional uncertainty. While, there has been some research (Angelidakis, Fotakis, & Lianeas, 2013) on considering uncertainty in congestion games, the uncertainty considered there is in cost functions and not in state transitions. There has been other work (Varakantham, Cheng, Gordon, & Ahmed, 2012) that has considered congestion in the context of stochastic games. However, the focus there is on planning (and not learning) without a centralized entity and there is also an approximation on value function considered in that work.

7.1 Stochastic Non-atomic Congestion Games

In this section we propose Stochastic Non-atomic Congestion Game (SNCG) model which represents both anonymity in interactions and infinitesimal agents in aggregation

systems. Formally, SNCG is represented using the tuple:

$$\langle \mathcal{N}, \mathcal{S}, \mathcal{Z}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$$

$\mathcal{N}$: Similar to NCG, $\mathcal{N}$ is the set of agents endowed with a measure space, $(\mathcal{N}, \mathcal{M}, m)$, where $\mathcal{M}$ is a σ-algebra of measurable subsets and m is a finite Lebesgue measure. For an agent i, $\{i\}$ is a null-set and $m(\{i\})$ is zero. $\mathcal{N}$ is an element of the interval [0, 1].

$\mathcal{Z}$: is the set of local states of individual agents (e.g., location of a taxi).

$\mathcal{S}$: is the set of global states (e.g., distribution of taxis in the city). The set of agents present in local state z in global state s is given by $\mathcal{N}_z^s$ and the mass of agents present in the local state z is given by $m(\mathcal{N}_z^s)$. The distribution of mass of agents is considered as the global state, i.e.,

$$s =< m(\mathcal{N}_1^s), m(\mathcal{N}_2^s), ..., m(\mathcal{N}_{|\mathcal{Z}|}^s) >, \text{ with}$$

$$\sum_{z=1}^{|\mathcal{Z}|} m(\mathcal{N}_z^s) = 1 \forall s \in \mathcal{S}$$

The total mass of agents in any global state s is 1.

$\mathcal{A}$: is the set of actions where $\mathcal{A}_z$ represents the set of actions (e.g., locations to move to) available to individual agents in the local state z.

$$\mathcal{A} = \{\mathcal{A}_z\}_{z \in \mathcal{Z}}$$

Let $act(i)$ provides the action selected by agent i. We define $f_z^a(s)$ as the total mass of agents in $\mathcal{N}_z^s$ selecting action a in state s, i.e. $\sum_{a \in \mathcal{A}_z} f_z^a(s) = m(\mathcal{N}_z^s)$. If the agents are playing deterministic policies, $f_z^a(s)$ is given by

$$f_z^a(s) = \int_{i \in \mathcal{N}_z^s} \mathbb{1}_{(act(i)=u)} dm(i) \tag{7.1}$$

The joint action $\boldsymbol{a}$ is given by

$$\boldsymbol{a} = ((f_z^a)_{a \in \mathcal{A}_z})_{z \in \mathcal{Z}} \tag{7.2}$$

$\mathcal{R}$: is the reward function[1]. The total mass of agents selecting action a for a joint action $\boldsymbol{a}$ in state s is given by

$$\phi^u(\boldsymbol{a}) = \sum_{z=1; a \in \mathcal{A}_z}^{|\mathcal{Z}|} f_z^a(s)$$

Similar to the cost functions in NCG, the reward function is assumed to be a non-decreasing continuous function. The immediate reward is dependent on the mass of the agents selecting the same action. Also, all the agents in local state z which select action a receive equal reward[2] which is given by

$$\mathcal{R}_z(s, \phi^a(\boldsymbol{a}))$$

$\mathcal{T}$: is the transitional probability of global states given joint actions. We define $p_z(z'|s, a_i, \boldsymbol{a}_{-i})$ as the probability of moving to local state z' when agent $i \in \mathcal{N}_z^s$ take action a_i and the induced joint action is $\boldsymbol{a}$. $\boldsymbol{a}_{-i}$ is the joint action induced by all the agents except i and s'_{-i_z} is the global state without agent i where i_z indicates that agent i is in local state z. The global transition from the perspective of an individual agent i is provided as

$$\mathcal{T}(s'|s, \boldsymbol{a}) = \sum_{z'} \mathcal{T}(s'_{-i_{z'}}|s, \boldsymbol{a}) \cdot p_z(z'|s, a_i, \boldsymbol{a}_{-i}) \tag{7.3}$$

The policy of agent i is denoted by π_i. We observe that given a global state s (we assume increased level of information sharing from the central agent where the central

[1]Researchers generally use the term "cost" in the context of NCG. To be consistent with the MARL literature we use the term "reward". However, reward and cost can be used interchangeably by observing that reward is negative of cost.

[2]In aggregation systems, expected reward is equal for all the agents in a local state who perform the same action in a local state, i.e. who select to move to the same zone.

agent provides information about the global state), an agent will play different policies based on its local state z as the available actions for local states are different. Hence, π_i can be represented as

$$\pi_i = (\pi_{iz}(s))_{s\in\mathcal{S},z\in\mathcal{Z}} \text{ such that } \sum_{a\in\mathcal{A}_z} \pi_{iz}(a|s) = 1$$

We define Π_z as the set of policies available to an agent in local state z, hence, $\pi_{iz}(s) \in \Pi_z \forall i \in \mathcal{N}_z^s, \forall s \in \mathcal{S}$. $\boldsymbol{\pi} = (\pi_i)_{(i\in\mathcal{N})}$ is the joint policy of all the agents.

Let γ be the discount factor and $\rho_{\boldsymbol{\pi}}$ denotes the state-action marginals of trajectory distribution induced by the joint policy $\boldsymbol{\pi}$. $\rho_{i\boldsymbol{\pi}}$ is the local state-action trajectory distribution of agent i induced by the joint policy $(\pi_i, \boldsymbol{\pi}_{-i})$. The value of agent i for being in local state z given the global state is s and other agents are following policy $\boldsymbol{\pi}_{-i}$ is given by

$$\begin{aligned} v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}_{-i}) &= \mathbb{E}_{((s,\boldsymbol{a})\sim\rho_{\boldsymbol{\pi}},(z',a)\sim\rho_{i\boldsymbol{\pi}})}\Big[\sum_{t=0}^{\infty}\gamma^t \mathcal{R}_{z'}(s, \phi^a(\boldsymbol{a}))\Big] \\ &= \mathcal{R}_z(s, \phi^{\pi_{iz}(s)}(\boldsymbol{a})) + \gamma \int_{s'}\sum_{z'} \mathcal{T}(s'_{-i_{z'}}|s, \boldsymbol{a}) \cdot p_z(z'|s, a_i, \boldsymbol{a}_{-i}) v_{iz'}(s', \pi_{iz'}, \boldsymbol{\pi}_{-i}) ds' \end{aligned} \tag{7.4}$$

The goal in an SNCG is to compute an equilibrium joint strategy, where no agent has an incentive with respect to their individual value to unilaterally deviate from their solution.

Here, we provide key properties of value function and equilibrium solution in SNCG that will later be used for developing a learning method for SNCGs.

Proposition 7.1.1. *Values of other agents do not change if agent i alone changes its policy. For any agent j in any local state z:*

$$v_{jz}(s, \pi_{jz}, \boldsymbol{\pi}_{-j}) = v_{jz}(s, \pi_{jz}, \boldsymbol{\pi}'_{-j})$$

where $\boldsymbol{\pi}_{-j} = \big(\pi_i, (\pi_k)_{k\in\mathcal{N}\setminus\{i,j\}}\big)$ *and* $\boldsymbol{\pi}'_{-j} = \big(\pi'_i, (\pi_k)_{k\in\mathcal{N}\setminus\{i,j\}}\big)$

Proof. Adapting Equation 7.4 for agent j in local state z, we have:

$$v_{jz}(s, \pi_{jz}, \boldsymbol{\pi}_{-j}) = \mathcal{R}_z(s, \phi^{\pi_{jz}(s)}(\boldsymbol{a})) + \gamma \int_{s'} \sum_{z'} \mathcal{T}(s'_{-i_{z'}}|s, \boldsymbol{a}) \cdot p_z(z'|s, \pi_{jz}(s), \boldsymbol{a}_{-j}) v_{jz'}(s', \pi_{jz'}, \boldsymbol{\pi}_{-j}) ds' \quad (7.5)$$

When policy of agent i is changed, the main factor that is impacted in the RHS of the above expression is **a** and due to that, the reward and transition terms can be impacted. **a** is solely dependent on $f_z^a(s)$ values and $f_z^a(s)$ values are dependent on the mass of agents taking action a in local state z and global state s (Equation 7.1):

$$f_z^a(s) = \int_{k \in \mathcal{N}_z^s} \mathbb{1}_{(act(k)=a)} dm(k)$$

If policy change makes agent i move out of local state z then the new mass of agents selecting action a in z is:

$$\tilde{f}_z^a(s) = \int_{k \in \mathcal{N}_z^s \setminus \{i\}} \mathbb{1}_{(act(k)=a)} dm(k)$$

Since f is primarily mass of agents (which is a Lebesgue measure), using the *countable additivity* property of Lebesgue measure (Bogachev, 2007; Hartman & Mikusinski, 2014), we have:

$$= \int_{k \in \mathcal{N}_z^s} \mathbb{1}_{(act(k)=a)} dm(k) - \int_{k \in \{i\}} \mathbb{1}_{(act(k)=a)} dm(k) \quad (7.6)$$

Since integral at a point in continuous space is 0 and mass measure is non-atomic, so we have $\{i\}$ is a null set and $m(\{i\}) = 0$

$$= \int_{k \in \mathcal{Z}_z^s} \mathbb{1}_{(act(k)=a)} dm(k) \quad (7.7)$$

Since $f_z^a(s) = \tilde{f}_z^a(s)$, action, $\boldsymbol{a}$ remains same. Hence neither reward nor transition values change. Thus, RHS of Equation 7.5 remains same when $\boldsymbol{\pi}_{-j}$ is changed to $\boldsymbol{\pi}'_{-j}$ in the LHS. □

7.1.1 Nash Equilibrium in SNCG

A joint policy $\boldsymbol{\pi}$ is a Nash equilibrium if for all $z \in \mathcal{Z}$ and for all $i \in \mathcal{N}_z^s$, there is no incentive for anyone to deviate unilaterally, i.e.

$$\begin{aligned} v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}_{-i}) \geq v_{iz}(s, \pi'_{iz}, \boldsymbol{\pi}_{-i}) \\ \forall s \in \mathcal{S}, \forall i \in \mathcal{N}_z^s, \forall z \in \mathcal{Z}, \forall \pi_{iz}(s), \pi'_{iz}(s) \in \Pi_z \end{aligned} \tag{7.8}$$

Proposition 7.1.2. *Values of agents present in a local state are equal at equilibrium, i.e.,*

$$\begin{aligned} v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}_{-i}) = v_{jz}(s, \pi_{jz}, \boldsymbol{\pi}_{-j}), \\ \forall s \in \mathcal{S}, \forall i, j \in \mathcal{N}_z^s, \forall z \in \mathcal{Z}, \forall \pi_{iz}(s), \pi_{jz}(s) \in \Pi_z \end{aligned} \tag{7.9}$$

Proof. In the proof of Proposition 7.1.1, we showed that adding or subtracting one agent from a local state does not change other agent's values, as contribution of one agent is infinitesimal. Thus,

$$\begin{aligned} v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}_{-i}) = v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}) \textbf{ and also} \\ v_{iz}(s, \pi'_{iz}, \boldsymbol{\pi}_{-i}) = v_{iz}(s, \pi'_{iz}, \boldsymbol{\pi}) \end{aligned} \tag{7.10}$$

This implies that the value is dependent only the policy of the individual agent given its state and joint policy. Hence if agent i in local state z gets a highest value of $v_{iz}(s, \pi_{iz}, \boldsymbol{\pi})$ over all policies, then any other agent j in the same local state z should get the same value. Otherwise, agent j can swap to the same policy (all agents have access to the same set of policies in each local state) being used by i. Thus,

$$v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}) = v_{jz}(s, \pi_{jz}, \boldsymbol{\pi})$$

and from the arguments in proof of Proposition 7.1.1, we have

$$v_{iz}(s, \pi_{iz}, \boldsymbol{\pi}_{-i}) = v_{jz}(s, \pi_{jz}, \boldsymbol{\pi}_{-j})$$ □

When there are multiple types of agents, we can provide a similar proof that values of same type of agents would be equal in a local state at equilibrium.

While SNCG model is interesting, it is typically hard to get the complete model before

hand. Hence, we pursue a multi-agent learning approach to compute high-quality and fair joint policies in SNCG problems.

7.2 Value Variance Minimization Q-learning, VMQ

Since aggregation systems have a large number of agents (not infinite, but large) with each agent making minimal contribution, we extrapolate the property of equilibrium in SNCG to propose a variance minimization algorithm. As argued in Proposition 7.1.2, the values of all the agents[3] present in any local state are equal at equilibrium. However please note that the converse is not true, i.e., even if the values of agents in local states are equal, the policy is not guaranteed to be an equilibrium policy. For a joint policy to be an equilibrium policy, agents should also be playing their best responses in addition to having values of agents in same local states being equal.

This is an ideal insight for computing equilibrium solutions in aggregation systems, as the centralized entity can focus on ensuring values of agents in same local states are (close to) equal by minimizing variance in values, while the individual suppliers can focus on computing best responses.

VMQ is a *centralized training decentralized execution* algorithm which assumes that during training a centralized entity has the access to the current values of the agents. The role of the central entity is to ensure that the exploration of individual agents moves towards a joint policy where the variance in values of agents in a local state is minimum. The role of the individual agents is to learn their best responses to the historical behavior of the other agents based on guidance from central entity.

Algorithm 6 provides detailed steps of the learning:

- Central agent suggests joint action $\boldsymbol{a}^c$ based on the joint policy it has estimated to all the individual agents. Line 11 of the algorithm shows this step. For the central agent, we consider a policy gradient framework to learn the joint policy. $\sigma(s, \boldsymbol{a})$ is the long term mean variance in the values of agents in all the local states if they perform joint action $\boldsymbol{a}$.

  We define two parameterized functions: joint policy function $\mu(s; \theta_\mu)$ and variance func-

[3]Values of all the agents of same type in a local state are equal if there are multiple types of agent population present in the system.

Algorithm 6 VMQ

Initialize replay buffer $\mathcal{J}$, action-variance network $\sigma(s, \boldsymbol{a}; \theta_\sigma)$, policy network $\mu(s; \theta_\mu)$ and corresponding target networks with parameters θ_σ^- and θ_μ^- respectively for the central agent
Initialize replay buffer $\mathcal{J}_i$, action-value network $Q_{iz}(s, a; \theta_i)$ and corresponding target network with parameter θ_i^- for all the individual agents i
while not converged **do**
  for $z \in \mathcal{Z}$ **do**
    for $i \in \mathcal{N}_z^s$ **do**
      compute value of i, $v_{iz} = max_a Q_{iz}(s, a; \theta_i)$
    Compute ν_z, variance in v_{iz} values for $i \in \mathcal{N}_z^s$
  Compute mean variance $\nu = \frac{1}{|\mathcal{Z}|} \sum_{z \in \mathcal{Z}} \nu_z$
  Compute suggested joint action by the central entity $\boldsymbol{a}^c \leftarrow \mu(s, \theta^\mu)$.
  for all $z \in \mathcal{Z}$ and for all agent $i \in \mathcal{N}_z^s$ **do**
    with probability ϵ_1,
      $a_i \leftarrow$ sample from $\boldsymbol{a}_z^c$
    with remaining probability $1 - \epsilon_1$
      $a_i \leftarrow \epsilon_2$-greedy$(Q_{iz})$
    Perform action a_i and observe immediate reward r_i and next local state z'
  Compute true joint action $\boldsymbol{a}$ and observe next state s'
  Store transition $(s, \nu, \boldsymbol{a}, s')$ in $\mathcal{J}$ and respective transitions $(s, z_i, a_i, r_i, s', z_i')$ in $\mathcal{J}_i$ for all agents i
  Periodically update the network parameters by minimizing the loss functions provided in Equations 7.12-7.14
  Periodically update the target network parameters

tion $\sigma(s, \boldsymbol{a}; \theta_\sigma)$. Since the goal is to minimize variance, we will need to update joint policy parameters in the negative direction of the gradient of $\sigma(s, \boldsymbol{a})$[4]. Hence, policy parameters θ_μ can be updated in the proportion to the gradient $-\nabla_{\theta_\mu}\sigma(s, \mu(s; \theta_\mu); \theta_\sigma)$. Using chain rule, the gradient of the policy will thus be

$$-\nabla_{\theta_\mu}\sigma(s,\mu(s;\theta_\mu);\theta_\sigma) = \\ -\nabla_{\theta_\mu}\mu(s;\theta_\mu)\nabla_{\boldsymbol{a}}\sigma(s,\boldsymbol{a};\theta_\sigma)|_{\boldsymbol{a}=\mu(s;\theta_\mu)} \tag{7.11}$$

- Individual agents either follow the suggested action with ϵ_1 probability or play their best response policy with $1 - \epsilon_1$ probability. While playing the best response policy, the individual agents explore with ϵ_2 probability (i.e. ϵ_2 fraction of $(1 - \epsilon_1)$ probability) and with the remaining probability ($(1 - \epsilon_2)$ fraction of $(1 - \epsilon_1)$) they play their best response action. Line 13 shows this step. The individual agents i maintain a network $Q_i(s, z, a; \theta_i)$ to approximate the best response to historical behavior of the other agents in local state z when global state is s.

- Environment moves to the next state. All the individual agents observe their individual reward and update their best response values. Central agent observes the true-joint action $\boldsymbol{a}$ performed by the individual agents. Based on the true joint-action and variance (ν) in the values of agents, the central agent updates its own learning.

As common with deep RL methods (Mnih et al., 2015; J. Foerster et al., 2017), replay buffer is used to store experiences ($\mathcal{J}$ for the central agent and $\mathcal{J}_i$ for individual agent i) and target networks (parameterized with θ') are used to increase the stability of learning. We define $\mathcal{L}_{\theta_\sigma}$, $\mathcal{L}_{\theta_\mu}$ and $\mathcal{L}_{\theta_i}$ as the loss functions of σ, μ and Q_i networks respectively. The loss values are computed based on mini batch of experiences as follows

$$\mathcal{L}_{\theta_\sigma} = \mathbb{E}_{(s,\nu,\boldsymbol{a},s')\sim\mathcal{J}}\Big[\Big(\nu + \gamma \cdot \sigma(s', \mu(s'; \theta^-_\mu); \theta^-_\sigma) - \sigma(s, \boldsymbol{a}; \theta_\sigma)\Big)^2\Big] \tag{7.12}$$

$$\mathcal{L}_{\theta_\mu} = \mathbb{E}_{(s)\sim\mathcal{J}}\Big[-\nabla_{\theta_\mu}\mu(s;\theta_\mu)\nabla_{\boldsymbol{a}}\sigma(s,\boldsymbol{a};\theta_\sigma)_{(\boldsymbol{a}=\mu(s;\theta_\mu))}\Big] \tag{7.13}$$

$$\mathcal{L}_{\theta_i} = \mathbb{E}_{(s,z,a,r,s',z')\sim\mathcal{J}_i}\Big[\big(r + \gamma \cdot \max_{a'} Q_{iz'}(s', a'; \theta^-_i) - Q_{iz}(s, a; \theta_i)\big)^2\Big] \tag{7.14}$$

$\mathcal{L}_{\theta_\sigma}$ and $\mathcal{L}_{\theta_i}$ are computed based on TD error (Sutton, 1988) whereas $\mathcal{L}_{\theta_\mu}$ is computed based on

[4]Joint action $\boldsymbol{a}$ is continuous.

the gradient provided in Equation 7.11.

7.3 Experiments

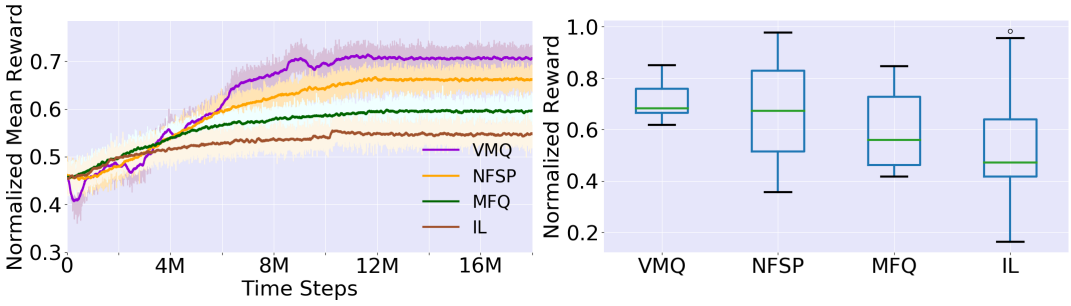
We perform experiments on four different domains: taxi simulator based on real-world data set (described in Section 4.2.1), a synthetic online to offline service simulator (described in Section 4.2.2), a single stage packet routing (Krichene, Drighes, & Bayen, 2014) and a muti-stage traffic routing (Wiering, 2000). In all these domains there is a central agent that assists (or provides guidance to) individual agents in achieving equilibrium policies. For example, a central traffic controller can provide suggestions to the individual travelers where as discussed earlier, the aggregation companies act as a central entity for the taxi domain.

As argued in Proposition 7.1.2, for SNCG the values of all the agents in a local state would be the same or variance in their values should be zero. Hence, we use variance in the values of all the agents as comparison measure (we use boxplots to show the variance). We compare with three baseline algorithms: Independent Learner (IL), neural fictitious self play (NFSP) (Heinrich & Silver, 2016) and mean-field Q-learning (MFQ) (Y. Yang et al., 2018). IL is a traditional Q-Learning algorithm that does not consider the actions performed by the other agents. Similar to VMQ, MFQ is also a *centralized training decentralized execution* algorithm and it uses joint action information at the time of training. However, NFSP is a self play learning algorithm and learns from individual agent's local observation. Hence, for fair comparison, we provide joint action information to NFSP as well. We compared with original NFSP as well, but it performed worse than the NFSP with joint action information. Hence we do not include those results here.

7.3.1 Taxi Simulator

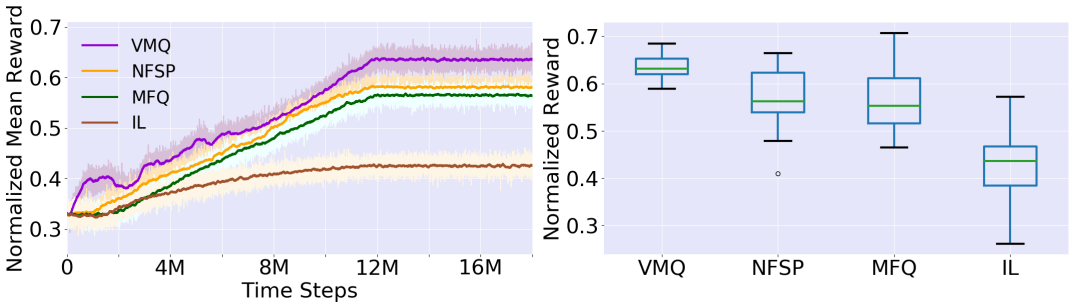
Apart from comparing variances in the payoffs of individuals, we also provide mean payoff of agents (with respect to the time) as the learning progresses. The mean payoff is an indicator of social welfare of individual agents. We show that VMQ learn policy which yield higher social welfare values for the individuals. The social welfare plots are for the running average of mean payoff of all the agents for every 1000 time steps.

Figure 7.1 show results for simulation based on the real-world data set. Plot in Figure 7.1a show that agents earn $\approx$5-10% more value than NFSP and MFQ. Boxplots in Figure 7.1b exhibit



(a) Social welfare of individuals (b) Variance in the values of agents

Figure 7.1: Taxi simulator using real-world data set



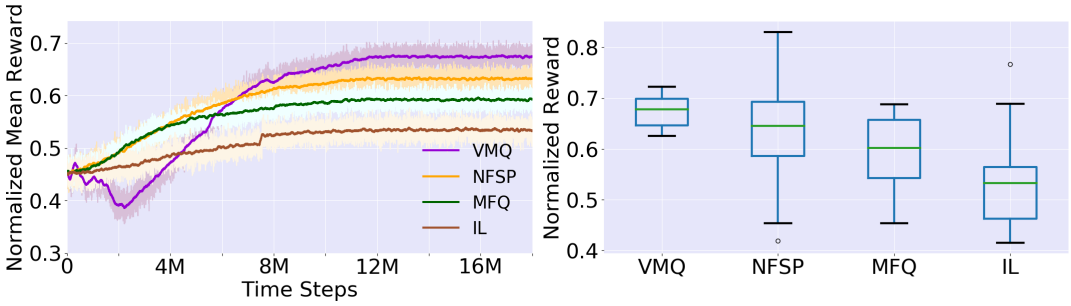
(a) Social welfare of individuals (b) Variance in the values of agents

Figure 7.2: DAR=0.4 with dynamic demand arrival rate

that the variance in the values of individual agents is minimum for VMQ. As agents are playing their best response policy and variance in values is low as compared to other algorithms, we can say that VQM learn policy which is closer to the equilibrium policy.

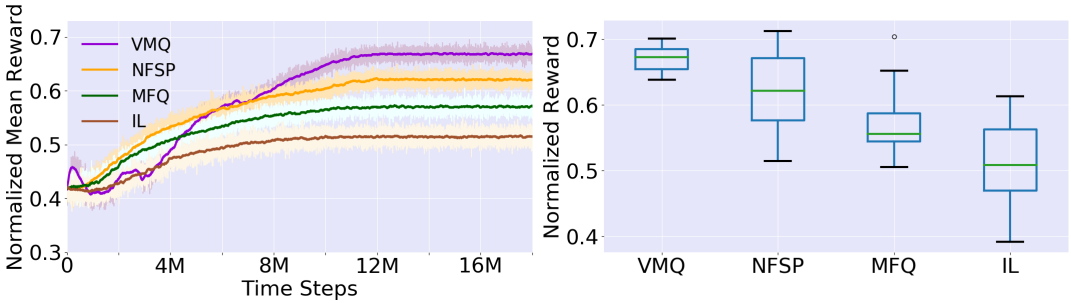
7.3.2 Synthetic Online to Offline Service Simulator

Figures 7.2 - 7.4 show results for synthetic data set where we include results for various combination of features. Figures 7.2a and 7.2b plot social welfare and variance in values of agents for



(a) Social welfare of individuals (b) Variance in the values of agents

Figure 7.3: DAR=0.5 with non-uniform trips pattern



(a) Social welfare of individuals

(b) Variance in the values of agents

Figure 7.4: DAR=0.6 with uniform trip pattern.

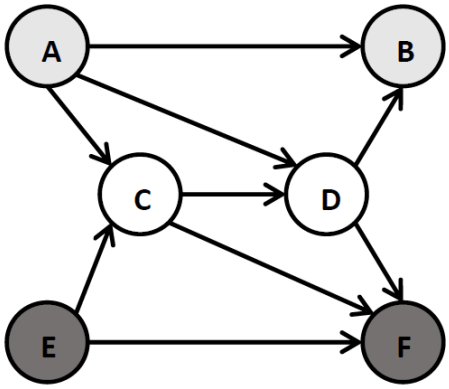


Figure 7.5: Routing network

a setup with dynamic arrival rate, non-uniform trip pattern with DAR=0.4. The social welfare value for VMQ is ≈8-10% higher that NFSP and MFQ. Figures 7.3a and 7.3b show results for a setup with dynamic arrival rate, uniform trip pattern and DAR=0.5. VMQ outperforms NFSP and MFQ by ≈5-10% in terms of social welfare of all the individual agents.

Comparison for an experimental setup with static arrival rate, non-uniform trip pattern and DAR=0.6 is shown in Figures 7.4a and 7.4b. Similar to other setups, social welfare for VMQ is ≈5-10% more than NFSP and MFQ respectively. For all the setups the variance in values of individual agents is minimum for VMQ. Hence VMQ provides better approximate equilibrium policies.

7.3.3 Packet Routing

We first performed experiments with a single stage packet routing game (Krichene et al., 2014). Two population of agents $\mathcal{N}_1$ and $\mathcal{N}_2$ of mass 0.5 each share the network given in Figure 7.5. The first population sends packets from node A to node B, and the second population sends

Table 7.1: Comparison of policies and ϵ values for packet routing example

Method	policy	ϵ value
Equilibrium Policy	((0, 0.187, 0.813),(0.223, 0.053, 0.724))	0
VMQ	((0, 0.180, 0.820), (0.220, 0.040, 0.740))	0.07
NFSP	((0.004, 0.116, 0.88), (0.01, 0.164, 0.826))	0.792
MFQ	((0, 0.162, 0.838), (0.220, 0.040, 0.740))	0.15
IL	((0.055, 0.176, 0.769), (0.217, 0.088, 0.695))	0.971

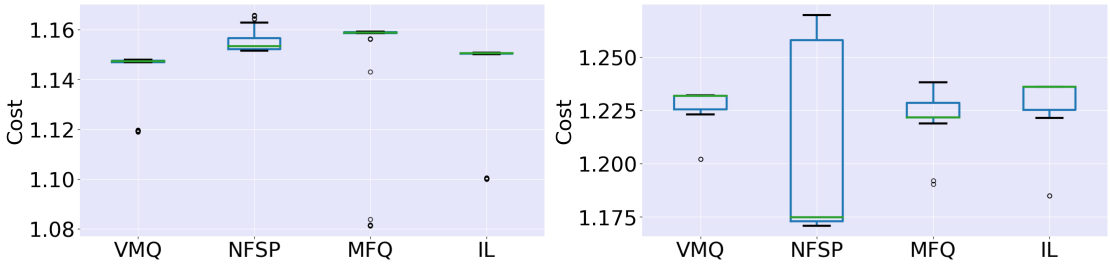
from node E to node F. Paths $AB, ACDB, ADB$ are available to agents in $\mathcal{N}_1$ whereas paths $EF, ECDF, ECF$ are available to agents in $\mathcal{N}_2$. The cost incurred on a path is sum of costs on all the edges in the path. The costs functions for the edges when mass of population on the edge is ϕ are given by:

$$c_{AB}(\phi) = \phi + 2, c_{AC}(\phi) = \frac{\phi}{2}, c_{AD}(\phi) = \phi,$$
$$c_{DB}(\phi) = \frac{\phi}{3}, c_{CD}(\phi) = 3\phi, c_{EC}(\phi) = \frac{1}{2},$$
$$c_{CF}(\phi) = \phi, c_{DF}(\phi) = \frac{\phi}{4}, c_{EF}(\phi) = \phi + 1$$

If the cost functions are known, equilibrium policy can be computed by minimizing Rosenthal potential function (Rosenthal, 1973). We use equilibrium policy and costs on paths computed by minimizing potential function to compare quality of the equilibrium policy learned. We performed experiments with 100 agents of each type. We also compute ϵ values of the learned policy, which is the maximum reduction in the cost of an agent when it changes its policy unilaterally. Table 7.1 compares the policies and ϵ values where the first row contain values computed using potential minimization method. The policy is represent as

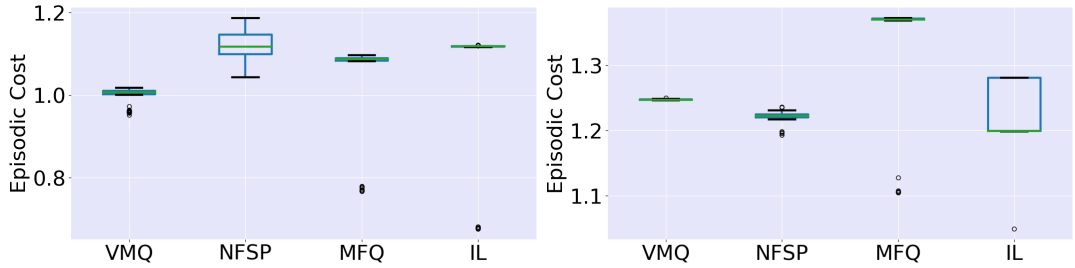
$$((\pi_1^{AB}, \pi_1^{ACDB}, \pi_1^{ADB}), (\pi_2^{EF}, f_2^{ECDF}, \pi_2^{ECF}))$$

where π_i^p is the fraction of mass of population of type i selecting path p. We see that the VMQ policy is closest to the equilibrium policy and ϵ value is also lowest as compared to NFSP, MFQ and IL. The equilibrium cost on paths as computed by the potential minimization method are: $AB = 2, ACDB = ADB = 1.14, EF = ECDF = ECF = 1.22$, i.e. at equilibrium agents in population $\mathcal{N}_1$ incur a cost of 1.14 whereas cost for agents in population $\mathcal{N}_2$ is 1.22. Figures



(a) Population $\mathcal{N}_1$ (b) Population $\mathcal{N}_2$

Figure 7.6: Variance in costs of agents for packet routing example.



(a) Population $\mathcal{N}_1$ (b) Population $\mathcal{N}_2$

Figure 7.7: Variance in values of agents for multi-stage traffic routing

7.6a and 7.6b provide variance in costs of agents for population $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively. We can see that not only variance in the costs of agents is minimum for VMQ but the values are also very close to the equilibrium values computed using potential function minimization method.

7.3.4 Multi-Stage Traffic Routing

We use the same network provided in Figure 7.5 to depict a traffic network where two population of agents $\mathcal{N}_1$ and $\mathcal{N}_2$ navigate from node A to node B and from node E to node F respectively. Unlike to the packet routing example, agents decide about their next edge at every node. Available edges to population type at every node remains the same as explained in the previous example. As the decision is made at every node, the domain is an example of SNCG where agents make a sequence of decision to minimize their long term cost. Hence, the values of agents from a population at a given node would be equal at equilibrium.

In this example, agents perform episodic learning and the episode ends when the agent reach their respective destination nodes. The distribution of mass of population over all the nodes is considered as state. We perform experiments with 100 agents of each type. Figures

7.7a and 7.7b show the variance in values of both the population. Similar to the packet routing domain, the variance is minimum for VMQ. Furthermore, we notice that for both single-stage and multi-stage cases, the values of agents from $\mathcal{N}_2$ is affected only by their own aggregated policy and fraction of agents from $\mathcal{N}_1$ selecting path AC. However, for agents from $\mathcal{N}_1$, the values would be different from single-stage case. For example, agents selecting path $ACDB$ and ADB would reach the destination node at different time steps and hence cost of agents on edge DB would be different from the single-stage case. Hence we can safely assume that the equilibrium value of agents from $\mathcal{N}_2$ would be 1.22 as computed for the single-stage case which is the value for VMQ as shown in Figure 7.7b.

7.4 Summary

In this chapter we proposed a Stochastic Non-atomic Congestion Games (SNCG) model to represent anonymity in interactions and infinitesimal contribution of individual agents for aggregation systems. We show that the values of all the agents present in a local state are equal at equilibrium in SNCG. Based on this property we proposed VMQ which is a *centralized learning decentralized execution* algorithm where the central agent learns policy from the state-action values of individual agents and suggests it to the individuals. Individual agents use the suggested policy to explore and learn equilibrium policies. Experimental results on multiple domains depict that VMQ learn better equilibrium policies than the other state-of-the-art algorithms.

In the next chapter we explore methods where the centralized agent learns directly from the experiences of the individual agents.

Chapter 8

Correlated Learning

In Chapter 7 we provided VMQ, which is a *centralized learning decentralized execution* algorithm where central agent ensures that the exploration of individual agents moves towards a joint policy where the variance in values of individuals is minimum (as we have shown that at equilibrium, the values of individuals in a local state are equal). However equilibrium policies are known to be sub-optimal. Hence in this chapter we focus on learning policies which increase long term payoff of individuals as well as ensures that overall performance of the system is improved.

In this chapter we increase the level of learning done by the centralized agent. Now the central agent learns from the direct experiences of the individuals and learn a social welfare maximizing policy. MARL algorithms which consider presence of a central agent assume that the central agent shares true joint action with the learning agents. However in the proposed method here, instead of sharing the true joint action, the central agent suggests the social welfare maximizing policy it has learned to the individuals. The individuals in turn learn to play a best response policy to the suggested social welfare policy.

Existing work (Littman, 1994; Hu et al., 1998; Hu & Wellman, 2003) has focused on computing equilibrium policies by representing MARL problems as learning in Stochastic Games (SG). Due to the existence of multiple equilibria and the challenge of coordinating agents to focus on the same equilibria, other alternatives have been considered (Shoham, Powers, & Grenager, 2003; Weinberg & Rosenschein, 2004). (Bowling & Veloso, 2001) proposed the following criteria for multi-agent learning:

1. *rationality*: learning should terminate with a best response to the play of other agents and

2. *convergence*: learning should converge to a stationary policy.

Though the above mentioned criteria (Shoham et al. (2003) named the criteria as *AI Agenda*) look similar to criteria of Nash equilibrium, however the difference lies between their approach towards *bounded rationality*. The idea of *bounded rationality* was introduced by Simon (1972) which states that in decision making, rationality of individuals is limited by the information they have, the cognitive limitation of their mind and the finite amount of time they have to make a decision. Traditional game theory assumes perfect reasoning and infinite mutual modeling of agents. However, *AI Agenda* focuses on achieving above-mentioned criteria and do not care about reaching an equilibrium solution. In this chapter, we focus on these two criterion for MARL problems with a large number of homogeneous agents.

The key contribution of this chapter is in developing a generic learning approach that exploits the presence of an aggregation system (e.g., Uber, Lyft, FoodPanda). We propose Correlated Learning (CL), where individual agents learn to play best response against a central agent, which learns joint actions that maximize social welfare. Similar to the work by (Bowling & Veloso, 2001), we demonstrate that CL satisfies the *rationality* and *convergence* criteria when the agent population is large.

We empirically show on multiple MARL problems that CL results into a "win-win situation" where both central agent and individual agents receive better payoff than the other MARL algorithms suitable for individual learning in the presence of a large number of agents. For aggregation systems, where there are many similar agents, we consider Anonymous MARL model that can capture homogeneity in agent models and anonymity in agent interactions to ensure scalable and efficient learning.

8.1 Correlated Learning for Homogeneous Agents

To understand the core facets of the approach, we first develop correlated learning method for homogenous agents in this section. We extend it to AyMARL problems in the next section. As discussed earlier, the objective of the central agent is to maxi-

mize the social welfare whereas individual agents try to maximize their own payoff.

There are set of $\mathcal{N}$ individual agents and one central agent present in the environment. Central agent does not interact directly with the environment and learns only from the experiences of the individual agents. Just to reiterate, $a_i \in \mathcal{A}$ denote action of agent i, where $\mathcal{A}$ is the action space of the individual agents. As agents are homogeneous, instead of modeling their joint action as a vector of actions of individual agents, we model it as a vector of number of agents selecting each available action. We use $\boldsymbol{a} = (n_j)_{j \in \mathcal{A}}, \boldsymbol{a} \in \boldsymbol{\mathcal{A}}$ to denote the joint action where n_j is the number of agents selecting action $j \in \mathcal{A}, \sum_{j \in \mathcal{A}} n_j = |\mathcal{N}|$ and $\boldsymbol{\mathcal{A}}$ is the joint action space. Superscript c is used to denote that action $\boldsymbol{a}^c$ has been derived from the central agent's policy. $\boldsymbol{a}_{-i}$ is the joint action of all the agents except agent i, i.e., $\boldsymbol{a} = (\boldsymbol{a}_{-i}, a_i)$. Note that as $\boldsymbol{a}$ is count based and does not consider agents' identities, there are $|\mathcal{A}|$ possible combinations of $\boldsymbol{a}_{-i}$ and a_i which will produce a single joint action $\boldsymbol{a}$. State space is denoted by $\mathcal{S}$.

Algorithm 7 Correlated Learning

1: Initialize central agent's Q-values arbitrarily, $Q_c(s, \boldsymbol{a}) \forall s \in \mathcal{S}, \forall \boldsymbol{a} \in \boldsymbol{\mathcal{A}}$
2: For all the individual agent i, initialize best response Q-values arbitrarily $Q_i(s, \boldsymbol{a}_{-i}, a_i) \forall s \in \mathcal{S}, \forall (\boldsymbol{a}_{-i}, a_i) \in \boldsymbol{\mathcal{A}}$
3: **while** not converged **do**
4: compute joint action for the central entity
$\boldsymbol{a}^c \leftarrow \epsilon_1$-greedy$\big(Q_c(s, \boldsymbol{a})\big)$.
5: **for** All agent i **do**
6: with probability ϵ_2,
$a_i \leftarrow$ follow $\boldsymbol{a}^c$
with remaining probability $1 - \epsilon_2$
$a_i \leftarrow \epsilon_3$-greedy$\big(Q_i(s, \boldsymbol{a}^c_{-i}, a)\big)$
7: Perform action a_i and observe next sate s' and reward r_i. Update agent's learning.
$Q_i(s, \boldsymbol{a}^c_{-i}, a_i) \leftarrow (1 - \alpha) Q_i(s, \boldsymbol{a}^c_{-i}, a_i) + \alpha \Big[r_i + \gamma \max_{a', \boldsymbol{a'}^c_{-i}} Q_i(s', \boldsymbol{a'}^c_{-i}, a') \Big]$
8: Compute true joint action $\boldsymbol{a}$, central agent's reward $r_c = \sum_i r_i$. Update central agent's learning.
$Q_c(s, \boldsymbol{a}) \leftarrow (1 - \alpha) Q_c(s, \boldsymbol{a}) + \alpha \Big[r_c + \gamma \max_{\boldsymbol{a'}} Q_c(s', \boldsymbol{a'}) \Big]$

Algorithm 7 provides the key four steps involved in CL.

- Central agent suggests current social welfare policy to all the individual agents. Line 4 of the algorithm shows this step.

- Individual agents either follow the suggested action with ϵ_2 probability or play their best response policy with $1-\epsilon_2$ probability. While playing the best response policy, the individual agents explore with ϵ_3 probability (i.e. ϵ_3 fraction of $(1-\epsilon_2)$ probability) and with the remaining probability ($(1-\epsilon_3)$ fraction of $(1-\epsilon_2)$) they play their best response action. Line 6 shows this step.

- Environment moves to the next state. All the individual agents observe their individual reward and update their best response values assuming that the other agent followed the suggested action. Line 7 shows this step.

- Central agent observes the true-joint action performed by the individual agents. Based on the cumulative reward of all the individual agents and the true joint-action, the central agent updates its own learning. Line 8 shows this step.

To follow $\boldsymbol{a}^c$ is step 6, individual agents take action j with probability $n_j/|\mathcal{N}|$. $\boldsymbol{a'}^c$ in step 7 is the social welfare policy of central agent for state s'. The individual agents play their best response based on their belief that others are following the suggested policy whereas the central agent acts as a correlation entity, hence we call it correlated learning. There is a difference between the way individual agents and central agent update their Q-values in steps 7 and 8. While central agent updates the value based on true joint action performed, the individual agents update their values based on the recommendation of joint action by the central entity.

Central agent's learning is dependent on the experiences of the individual agents. Hence, if individual agents explore sufficiently (infinite exploration is a criteria for the learning to converge), the central agent's joint-action exploration would also be sufficient. We now argue that social welfare policy and individual agents' best response policies converge to a stationary policy.

Lemma 1. *Central agent's learning converges to a stationary policy.*

Proof. Given the ϵ-greedy approach for each of the individual agents, all the joint state and joint action combinations will be explored in the limit with infinite exploration (GLIE) (S. Singh, Jaakkola, Littman, & Szepesvári, 2000). Since learning of central

agent is only dependent on the combined experiences of the individual agents and since individual agent exploration is exhaustive, central agent's learning is equivalent to reinforcement learning. Therefore, it will converge to a stationary policy. □

8.1.1 Discussion

As indicated earlier, our focus is on finding rational policies that are stationary. While we cannot yet guarantee, we are able to intuitively argue for these properties.

First, as individual agents update their Q-values over individual action and joint recommended action for all other agents, it will be ideal if our approaches learn the true best response values for ensuring rational behavior. This is feasible if we generate sufficient experiences of the suggested joint action. In Algorithm 7, each individual agent either follows the suggested action, or selects a random action. Selecting random action while exploration can be considered as idiosyncratic i.i.d. (independent and identically distributed) noise. We argue that during high exploration phase they do generate experiences of the suggested joint action. The intuition comes from the common assumption in the game theory that the idiosyncratic noise is averaged away if the agent population is large (Sandholm, 2010; Carmona & Delarue, 2014; Nutz, 2018). Hence, during exploration they generate experiences of suggested joint action and thus learn the values for true joint action.

Once the individual agents have learned values of true joint actions, the action suggested by the central agent works as a synchronization mechanism for them. Suppose for suggested joint action $\boldsymbol{a}^c$ for state s, agents have figured out their respective best response action and the resulting aggregated joint action is $\mathfrak{a}$. Even if in reality $\boldsymbol{a}^c$ and $\mathfrak{a}$ are not same, whenever joint action $\boldsymbol{a}^c$ is suggested, agents will play their respective best response and the true joint action will always be $\mathfrak{a}$. As shown in Lemma 1, the social welfare policy converges to a stationary policy, hence the best response policies will also converge to a stationary policy.

Hence under normal assumptions of Q-learning, CL intuitively satisfies the two criteria of the *rationality* and *convergence* if the agent population is large.

8.2 CL for Aggregation Systems

Algorithm 8 CL for aggregation systems

for central agent **do**
  Initialize replay memory $\mathcal{J}_c$
  Initialize DDPG state-action value network $Q_c(s, \boldsymbol{a}; \theta_c)$ and corresponding target network with parameter θ_c^-
  Initialize DDPG policy network $\mu(s; \theta_\mu)$ and corresponding target network with parameter θ_μ^-
for each individual agent i **do**
  Initialize replay memory $\mathcal{J}_i$
  Initialize best response network $Q_{iz}(s, \boldsymbol{a}_z, a; \theta_i)$ and corresponding target network with parameter θ_i^-
while not converged **do**
  compute joint action for the central entity
    $\boldsymbol{a}^c \leftarrow \mu(s, \theta_\mu)$
  for All agent $z \in \mathcal{Z}$ **do**
    for All agent $i \in \mathcal{N}_z^s$ **do**
      with probability ϵ_1,
        $a_i \leftarrow$ sample from $\boldsymbol{a}_z^c$
      with remaining probability $1 - \epsilon_1$
        $a_i \leftarrow \epsilon_2$-greedy$\big(Q_{iz}(s, \boldsymbol{a}_z^c, a)\big)$
      Perform action a_i and observe next sate (s', z') and reward r_i
      Store transition $(s, z, \boldsymbol{a}_z^c, a_i, r_i, s', z')$ in $\mathcal{J}_i$
  Compute central agent's reward $r_c = \eta \sum_i r_i$
  Store transition $(s, \boldsymbol{a}, r_c, s')$ in $\mathcal{J}_c$
  Periodically update the networks with following losses

$$\mathcal{L}_{\theta_c} = \mathbb{E}_{(e \sim \mathcal{J}_c)}\Big[\Big(r_c + \gamma Q_c(s', \mu(s'; \theta_\mu^-); \theta_c^-) - Q(s, \boldsymbol{a}; \theta_c)\Big)^2\Big]$$
$$\mathcal{L}_{\theta_\mu} = \mathbb{E}_{(e \sim \mathcal{J}_c)}\Big[\nabla_{\boldsymbol{a}} Q_c(s, \boldsymbol{a}; \theta_c)_{(\boldsymbol{a}=\mu(s;\theta_\mu))} \nabla_{\theta_\mu} \mu(s; \theta_\mu)\Big]$$
$$\mathcal{L}_{\theta_i} = \mathbb{E}_{(e \sim \mathcal{J}_i)}\Big[\Big(r_i + \gamma max_{a'} Q_{iz'}(s', (\mu(s'; \theta_\mu^-)_z), a'; \theta_i^-) - Q_{iz}(s, \boldsymbol{a}_z^c, a; \theta_i)\Big)^2\Big]$$

  Periodically update target network parameters

While the basic version of CL described in the previous section is easy to understand, it does not scale very well due to the combinatorial state and action spaces when considering large numbers of agents. We now extend CL for large scale AyMARL problems in aggregation systems by converting discrete combinatorial state and action spaces into continuous values.

The underlying model for CL in anonymous settings remains the same as described

in Section 4.1.1. Both joint state and joint action are vector of continuous values (mass/fraction of agent population), hence, a tabular learning of Q-values is difficult. Deep neural network, which is popular for function approximation, can be used to estimate $Q(\boldsymbol{s}, \boldsymbol{a})$ values for the central agent[1]. Deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) is an excellent algorithm to learn deterministic policies for domains with continuous actions. The assumption is that the policy is deterministic, which is a reasonable assumption for the problems of interest. Central agent maintains an actor network $\mu_c(\boldsymbol{s}; \theta_\mu)$ and a critic network $Q_c(\boldsymbol{s}, \boldsymbol{a}; \theta_c)$. Central agent learns a policy to maximize the social welfare (as given in Equation 4.1) from the experiences of the individual agents and its learning experience is given by $< s, \boldsymbol{a}, r(s, \boldsymbol{a}), s' >$. The target value mentioned in Equation 2.4 for the central agent is computed as follows.

$$y = r_c(s, \boldsymbol{a}) + \gamma Q_c(\boldsymbol{s'}, \mu_c(\boldsymbol{s'}; \theta_\mu); \theta_c)$$

The experience of an individual agent is given by $< s, z_i, \boldsymbol{a}_c^z, a_i, r_i, s', z_i' >$ which can be interpreted as after taking action a_i in state (s, z_i) agent i received r_i as immediate payoff and moved to state (s', z_i') given the joint action suggested by the central agent was $\boldsymbol{a}_c^z$. The agents maintain their best response value network $Q_{iz}(s, \boldsymbol{a}_{z_i}^c, a_i; \theta_i)$ where $a_i \in \mathcal{A}_z$ is the available actions in zone z. Furthermore, the individual agents compute their target value as follow.

$$y_i = r_{iz}(s, \boldsymbol{a}_z^c, a_i) + \gamma \max_{a_i'} Q_{iz'}(s', \boldsymbol{a}_{z_i'}^c, a_i'; \theta_i)$$

Here $\boldsymbol{a}_c^{z'}$ is the social welfare policy of the central agent for zone z' for the next joint state s'. CL steps for aggregation system has been given in the Algorithm 8.

[1]Use of non-linear function approximator such as neural network does not preserve any mathematical convergence guarantees. However in practice it has been seen to converge (Mnih et al., 2013, 2015).

8.3 Experiments

We analyze and evaluate CL on four different example domains. The first two domains are taxi simulator and a synthetic online to offline service aggregation simulator as described in Sections 4.2.1 and 4.2.2 respectively. To show the effectiveness of CL, we also perform experiments on two stateless games described as follows

8.3.1 Traffic Game

Traffic game (Chen et al., 2018) is a stateless coordination game motivated by traffic control task where players need to select a route such that they do not cause congestion. $\mathcal{N}$ players present in the domain need to coordinate in a way such that congestion on route k, $l_k = \sum_{i \in \mathcal{N}} 1_{(x_i=k)}$ is neither too many or too few, where x_i is the route selection of agent i. The *reward* function for maintaining desired congestion on a route k is Gaussian function $l_k \cdot e^{-(l_k-\mu_k)^2/\sigma_k^2}$ where μ_k is the desired mean value of congestion and σ_k is the penalty for deviation from the mean value. All the players on the same route receives same reward. The objective of a central traffic controller is to have congestion on each route to be as close to the respective mean value as possible. Hence its goal is to maximize the social welfare whereas the other players are self interested player maximizing their own reward.

8.3.2 When does the meeting start?

"When does the meeting start?" (Guéant, Lasry, & Lions, 2011) is a stateless coordination game where a number of participants need to attend a meeting. The meeting is scheduled to start at T but it will start only after a minimum number of participants are present for the meeting. Hence very often it starts several minutes after the scheduled time. As a result, each participant i has their own preference T_i when they would like to arrive for the meeting. Also, when participant i decides to arrive at t_i, in reality he arrives at $t_i \pm \sigma_i$ due to uncertainties (traffic etc.). More precisely, t_i is the action taken by the participant and σ_i is the uncertainty he is subjected to.

To decide about their arrival time, each participant optimize their cost. Suppose the meeting started at t and participant i arrived at t_i, then he incurs following cost

$$\beta_1(|t - t_i|) + \beta_2(T_i - t_i) + \beta_3 max(0, t_i - T)$$

Here first part is lateness/waiting cost, second part is deviation from the preference cost and the last part is reputation cost. β parameters are weigtages of these cost components. Participants learn when to arrive given T and T_i such that their individual cost is minimum. We can assume presence of a central entity (say manager of the team whose member are supposed to attend the meeting) whose objective is to minimize the social welfare cost.

8.4 Results

We compare CL with three baseline algorithms: (1) density entropy based deep Q-learning (DE-DQN) presented in Chapter 6(Verma, Varakantham, & Lau, 2019), (2) neural fictitious self play (NFSP) (Heinrich & Silver, 2016) and (3) mean-field Q-learning (MFQ) (Y. Yang et al., 2018). As discussed in Chapter 6, DE-DQN is an independent learning algorithm which learns from local observation. It predicts agent population density distribution and uses entropy of population density distribution to improve individual learning. NFSP is a fictitious self play learning algorithm where agents learn from their local observation. As CL has advantage of having a central agent providing more information, for fair comparison we provided joint action information to NFSP. As we have mentioned in previous chapter, we compared with original NFSP as well, but it performed worse than the NFSP with joint action information. Hence we do not include those results here. Also, we considered the action space of the central agent to be continuous for all our experimental domains.

DE-DQN algorithm is suitable for the setup with partial observation and uses entropy of population density distribution to improve learning. As we consider full view of joint action in case of the two stateless coordination game example, we do not per-

form experiments for DE-DQN for these examples.

A centralized cooperative learning will ensure an optimal social welfare revenue given the reward function and transition function are same for all the individual agents. Hence, as an upper bound we also compare with social welfare (SW) policy where the all the agents execute the social welfare policy cooperatively.

We evaluated the performance of all learning methods on two key metrics:

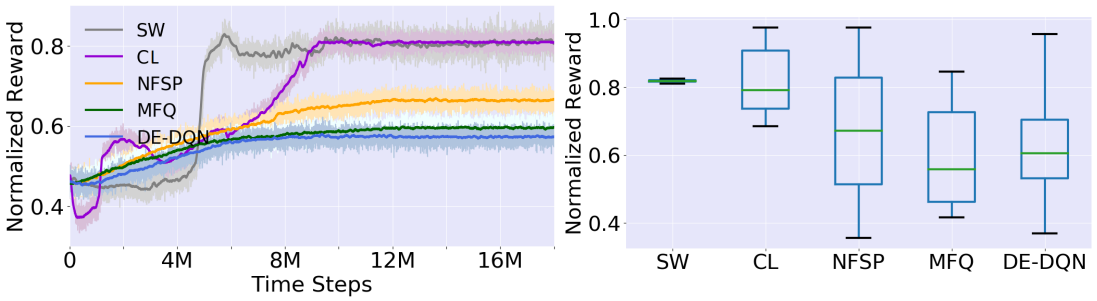
- Social welfare payoff of all the individuals computed by aggregating payoffs of all the individual agents. Higher values imply better performance.
- Variation in payoff of individual agents after the learning has converged. This is to understand if agents can learn well irrespective of their initial set up and experiences. This in some ways represents fairness of the learning approach. We use box plots to show the variation in individual revenues, so smaller boxes are better.

Payoff of all the agents is reset after every evaluation period time steps[2]. For taxi simulator and online to offline service simulator one evaluation period consisted of 1000 (1e3), whereas for traffic game and "when does the meeting start" experiments, it was set to 100 time steps. The graphs where social welfare has been compared provide running average of revenue over 100 evaluation periods for taxi simulator and online to offline service simulator, whereas for the remaining two experimental domains it provides running average of 20 evaluation periods.

8.4.1 Taxi Simulator

Figure 8.1 presents the performance comparison for taxi simulator where the zone structure and demand distribution were simulated using real-world data. In Figure 8.1a we can see that CL's social welfare value is similar to that of centralized cooperative learning. However Figure 8.1b shows that variance in the payoff of individual agents is

[2]A time step represents a decision and evaluation point in the simulator.



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 8.1: Taxi simulator

minimum for SW. This is expected as all the agents follow the same mixed policy computed by the central agent. Furthermore variance for CL is lower than the other three algorithms.

As seen in Figure 8.1a, the social welfare value of CL is $\approx 13 - 20\%$ higher than the other three algorithms. It means that there are some "lost demand" (demands that were not served) for NFSP, MFQ and DE-DQN which are being served by SW and CL. Figure 8.2 displays error between central agent's social welfare policy and the

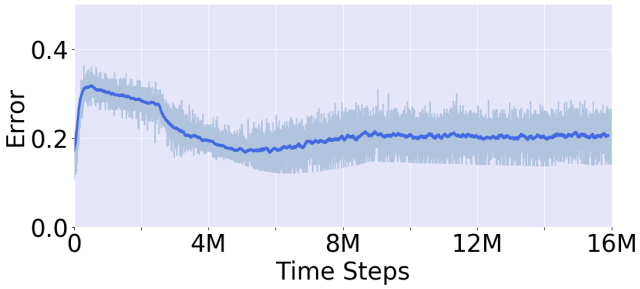
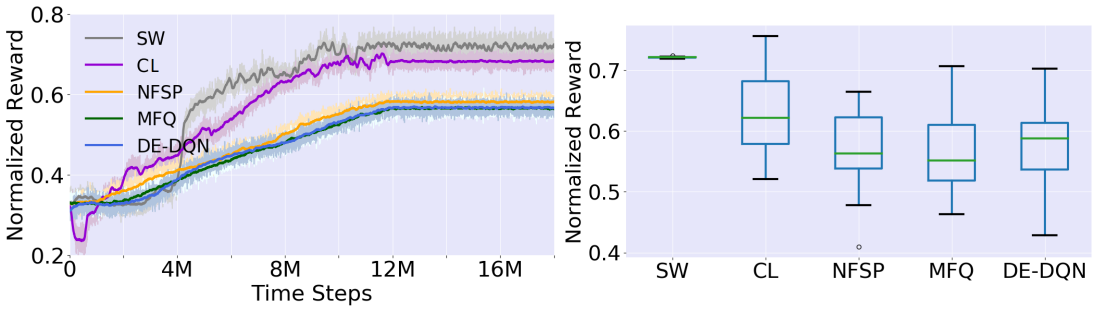


Figure 8.2: Difference in central agent's social welfare policy and aggregated policy of individual agents

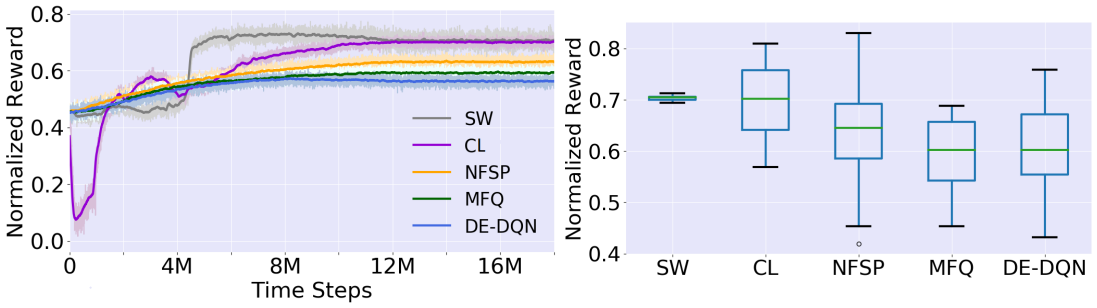
aggregated joint policy of individual agents. To compute aggregated joint policy of individual agents in zone z, we compute best response of agents present in the zone and then compute $\boldsymbol{a}^z_{bs} = (a^k_{bsz})_{k \in \mathcal{A}_z}$, where a^k_{bsz} is the fraction of agents present in zone z whose best response is to move to zone k. Then we compute root mean squared error (RMSE) between vectors $\boldsymbol{a}^z_c$ and $\boldsymbol{a}^z_{bs}$ to compute error in policies for zone z. Finally, we compute average error over all the zones to get the error between social welfare policy and the joint best response policy.

We can see that the social welfare policy and the joint best response policy converges



(a) Social welfare of individuals (b) Variance in individual revenue

Figure 8.3: DAR=0.4 with dynamic demand arrival rate



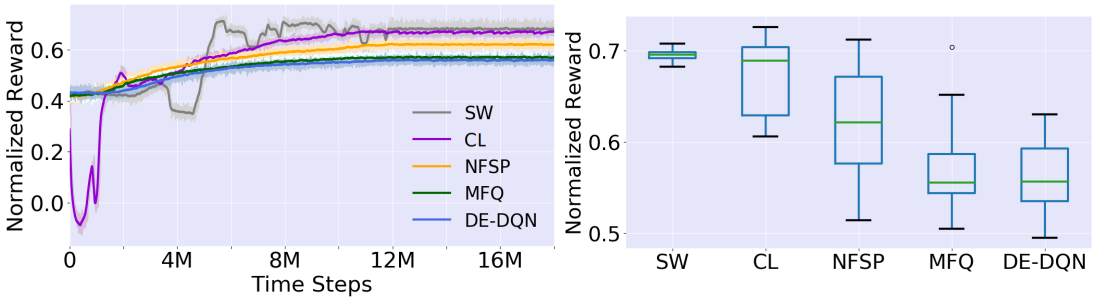
(a) Social welfare of individuals (b) Variance in individual revenue

Figure 8.4: DAR=0.5 with non-uniform trips pattern

to different policies as the error between them does not go down to zero. However our argument in Section 8.1.1 that the policy converges to a stationary policy is validated empirically here.

8.4.2 Synthetic Online to offline service simulator

Figures 8.3 - 8.5 presents plots for social welfare and boxplots for variance in individual revenue for various experimental setups. The first result is for a setup with dynamic arrival rate, non-uniform trip pattern with DAR=0.4. Social welfare value of CL is ≈10-12% more than NFSP, MFQ and DE-DQN (Figure 8.3a). In Figure 8.3b we can see that the best agent of CL generates revenue more than its counterpart for the algorithms. For setup with dynamic arrival rate and DAR=0.5 in Figure 8.4a, social welfare value of CL is ≈8-12 % more than NFSP, MFQ and DE-DQN. Also as seen in boxplots of Figure 8.4b, there are around 50 % of the agents who earn more than the social welfare value of SW. We observed similar results for the setup with non-uniform trip pattern and



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 8.5: DAR=0.6 with uniform trip pattern.

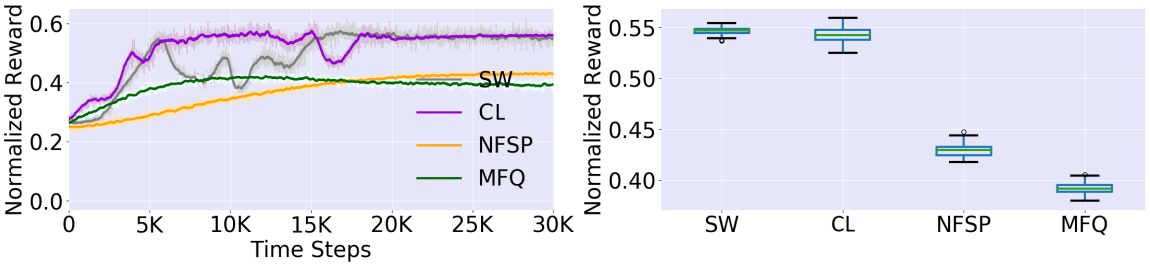
DAR=0.6 (Figures 8.5a and 8.5b), CL generated ≈6-10 % more social welfare revenue than NFSP, MFQ and DE-DQN.

Here are the key conclusions:

- SW performs best due to its cooperative learning and uniformity of the agents.
- Performances of NFSP, MFQ and DE-DQN with respect to SW vary with varying complexity of the experimental setup (with and without dynamic arrival rate and non-uniform trips). However CL's social welfare was consistently at par with SW's social welfare. This provides the motivation to the central entity to compute social welfare policy and share it with the non-cooperative individual learners.
- Apart from having lesser variance in the individual revenues, the best and worst performing agents of CL always perform better than the best and worst performing agents of NFSP, MFQ and DE-DQN respectively. This provides motivation for the individual agents to use CL instead of other individual learning algorithms.
- A considerable number (20-50%) of individual agents earn more than the social welfare value of SW. This justifies for the self-interested agents to play best response policy (CL) instead of learning cooperatively (SW).

8.4.3 Traffic Game

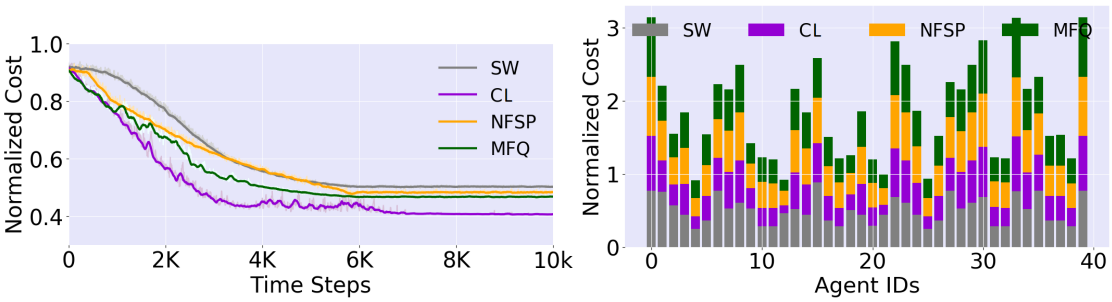
Figure 8.6 shows results for traffic game with 100 agents with 10 routes. We generated different values and μ and σ for each route to make the learning more complex (as



(a) Social welfare of individuals

(b) Variance in individual reward

Figure 8.6: Traffic game



(a) Social welfare of individuals

(b) Individual costs of 40 agents

Figure 8.7: Social welfare and individual costs for "when does the meeting start?" experiment

opposed to having same μ and σ for every route). As seen in previous example domains, the social welfare value of CL is similar to that of SW and is ≈15-18% more than NFSP and MFQ. As it is stateless game, other algorithms were also able to achieve low variance in the individual values.

8.4.4 When does the meeting start?

We performed experiments with 40 participants with options of arriving at 15 different time steps. We used normal distribution to introduce uncertainty σ_i and β values were set to 1. Each participant had their own preference of arrival time which makes the domain asymmetric and hence, a central policy might not provide a social optimal. Figure 8.7a confirms this and we see that performance of SW is worst. Also, cost for CL is ≈6-8 % lesser than MFQ and NFSP. Figure 8.7b illustrates costs of 40 participants where the results has been shown as stacked bar chart. Each bar represents cost of single participant for these forur algorithms. We can observe that the cost of individual agents

are minimum for CL.

8.5 Summary

In this chapter we presented correlated learning (CL) for aggregation systems. The centralized agent learn social welfare maximizing policy by observing learning experiences of the individual agents. Instead of sharing information about the true joint action, it suggest the social welfare maximizing policy to everyone. The central agent acts as a correlation agent where non-cooperative individual agents learn to play best response against the suggested social welfare policy. We first provide a generic CL and then extend it to use in anonymous domains. Our experiments on multiple example domains show that both central agent and individual agents get benefited by using CL as compared to other individual learning algorithms.

To generate the relevant experiences, CL assumes that the individuals will follow the suggested policy during the high exploration phase of the individual learning. However this is an optimistic assumption and the self interested agents might not follow the suggestion. Hence, in the next chapter, we propose an incentive based learning where the central agent provides incentive based on social welfare value of the current joint policy.

Chapter 9

Incentive Based Q-Learning

We saw in the last chapter that CL performs extremely well, where the fairness is high (in terms of individual payoffs) and the social welfare value is at par with a cooperative centralized algorithm. However, it is optimistic in its assumption that the individuals will follow the suggestion during exploration phase. Hence, in this chapter we provide an incentive based learning where the central agent still learns social welfare maximization policy from the learning experiences of the individuals. However, instead of suggesting the policy/corresponding joint action, it provides incentives (charges penalty, which can be deducted from the immediate payoffs of the individual agents) to the agents based on the value of the current joint policy.

Potential based reward shaping is a way to include prior domain knowledge in the learning. Taking inspiration from potential based reward shaping methods, in this chapter we propose Incentive Based Q-Learning (IBQ) where the incentive is computed based on the social welfare values. In experiments we show that due to providing incentives the agents learn a joint policy which provide similar results to CL which works on atring assumption that individual agents would sometime follow the suggested policy during high exploration phase. This results into scenario where the social welfare is high as well as the variance in individuals' payoffs is minimum.

9.1 Dynamic Potential Function

In this chapter, we propose use of joint policy based potential function for reward shaping. Suppose, $\boldsymbol{\pi}^c$ is the joint policy which maximizes the social welfare and $\boldsymbol{\pi}$ is the current joint policy of the individual agents. We define potential of a state as the difference in the social welfare value for these two policies as follows

$$\varphi(s) = v(s, \boldsymbol{\pi}) - v(s, \boldsymbol{\pi}^c) \tag{9.1}$$

Intuitively, it represents how far the current policy is from the optimal policy. As both $\boldsymbol{\pi}^c$ and $\boldsymbol{\pi}$ continuously change during learning, the above computed potential function is dynamic.

9.2 Potential Difference based Incentive

During a global state transition, the local state transition of individuals depend on their current local state. Hence computing incentive based on potential values of global state might not be helpful. Hence, we further increase the granularity and compute potential value of a global state - local state pair as

$$\varphi(s, z) = v(s, \boldsymbol{\pi}_z, \boldsymbol{\pi}_{-z}) - v(s, \boldsymbol{\pi}^c_z, \boldsymbol{\pi}_{-z}) \tag{9.2}$$

$\boldsymbol{\pi}_z$ is the joint policy of agents in local state z and $\boldsymbol{\pi}_{-z}$ is the joint policy of agents from all the remaining local states. To ensure that the underlying game structure remains unchanged, we need to compute the potential of a state at the time it is entered and use the same value when the state is exited. We use superscript t to denote policy at time t. Hence potential at time t is computed as follows

$$\varphi^t(s, z) = v(s, \boldsymbol{\pi}^t_z, \boldsymbol{\pi}^t_{-z}) - v(s, \boldsymbol{\pi}^{tc}_z, \boldsymbol{\pi}^t_{-z}) \tag{9.3}$$

For an individual agent transitioning from (s, z) to (s', z'), the additional reward is provided as

$$F(s, z, t, s', z', t') = \frac{1}{|\mathcal{N}|}\Big[\gamma\varphi^{t'}(s', z') - \varphi^{t}(s, z)\Big] \tag{9.4}$$

For the ease of notation we use $\delta_{zz'}$ to denote $F(s, z, t, s', z', t')$. The corresponding value update for agent i is given by

$$\begin{aligned} Q_{iz}(s, a_i, \boldsymbol{a}_{-i}) \leftarrow & Q_{iz}(s, a_i, \boldsymbol{a}_{-i}) + \alpha\Big[(1-\eta)\omega_{iz}(s, a_i, \boldsymbol{a}_{-i}) + \\ & \delta_{zz'} + \max_{a'_i} Q_{iz'}(s', a'_i, \boldsymbol{a}'_{-i}) - Q_{iz}(s, a_i, \boldsymbol{a}_{-i})\Big] \end{aligned} \tag{9.5}$$

9.3 Incentive Based Q-Learning (IBQ)

We now provide IBQ, where the central agent provides incentives/charges penalties based on the potential difference value given by Equation 9.4. As discussed in Chapter 4, both joint state and joint action are vector of continuous values (fraction of agent population), hence we assume that the social welfare policy is deterministic. The assumption helps us in utilize policy gradient theorem of DDPG (Deep Deterministic Policy Gradient) (Lillicrap et al., 2015), which is popular to learn deterministic policies for domains with continuous actions. Hence, to estimate social welfare policy, the central agent maintains an actor network $\mu(\boldsymbol{s}; \theta_\mu)$ and a critic network $Q_c(\boldsymbol{s}, \boldsymbol{a}; \theta_q)$. To compute the potential value, central agent needs to know the current joint policy of the agents. Hence, it also maintains an average policy network $\pi(s; \theta_\pi)$ which estimates the current joint policy of the agents based on the joint action execute by them.

The experience of an individual agent is given by $< s, z, a_i, r_i, s', z' >$ which can be interpreted as after taking action a_i in state (s, z) agent i received r_i as immediate payoff and moved to state (s', z'). The agents maintain their value network $Q_{iz}(s, a_i; \theta_i)$ which estimates value of taking action $a_i \in \mathcal{A}_z$ in local state z and global state s.

As common with deep RL methods (Mnih et al., 2015; J. Foerster et al., 2017), replay buffer is used to store experiences ($\mathcal{J}_c$ for the central agent and $\mathcal{J}_i$ for individual

agent i) and target networks (parameterized with θ^-) are used to increase the stability of learning. We define $\mathcal{L}_{\theta_\mu}$, $\mathcal{L}_{\theta_\pi}$, $\mathcal{L}_{\theta_q}$ and $\mathcal{L}_{\theta_i}$ as the loss functions of μ, π, Q_c and Q_i networks respectively. The loss values are computed based on mini batch of experiences as follows

$$\mathcal{L}_{\theta_\mu} = \mathbb{E}_{(s)\sim\mathcal{J}}\Big[\nabla_{\boldsymbol{a}} Q(s, \boldsymbol{a}; \theta_v)|_{\boldsymbol{a}=\mu(s,\theta_\mu)} \nabla_{\theta_\mu} \mu(s; \theta_\mu)\Big] \tag{9.6}$$

$$\mathcal{L}_{\theta_\pi} = \mathbb{E}_{(s,\boldsymbol{a})\sim\mathcal{J}}\Big[-\boldsymbol{a} log(\pi(s; \theta_\pi))\Big] \tag{9.7}$$

$$\mathcal{L}_{\theta_q} = \mathbb{E}_{(s,\boldsymbol{a},r,s')\sim\mathcal{J}}\Big[\Big(r + \gamma \cdot Q_c\big(s', \mu(s'; \theta_\mu^-); \theta_q^-\big) - Q_c(s, \boldsymbol{a}; \theta_q)\Big)^2\Big] \tag{9.8}$$

$$\mathcal{L}_{\theta_i} = \mathbb{E}_{(s,z,a_i,r_i,s',z')\sim\mathcal{J}_i}\Big[\big(r_i + \gamma \cdot \max_{a'} Q_{iz'}(s', a'; \theta_i^-) - Q_{iz}(s, a_i; \theta_i)\big)^2\Big] \tag{9.9}$$

Algorithm 9 Incentive Based Q-Learning

for central agent **do**
  Initialize replay memory $\mathcal{J}_c$.
  Initialize social welfare value network $Q_c(s, \boldsymbol{a}; \theta_q)$ and a target network with parameter θ_q^-
  Initialize social welfare policy network $\mu(s; \theta_\mu)$ and a target network with parameter θ_μ^-
  Initialize an average policy network $\pi(s; \theta_\pi)$.
for all the individual agents i **do**
  Initialize replay memory $\mathcal{J}_i$.
  Initialize the action-value Q network, $Q_{iz}(s, a; \theta_i)$ and a target network with parameter θ_i^-
while not converged **do**
  for all $z \in \mathcal{Z}$ **do**
    for all agent $\in \mathcal{N}_z^s$ **do**
      $a_i \leftarrow \epsilon$-greedy$\Big(Q_{iz}\Big)$
      Perform action a_i and observe next local state z'
  Compute true joint action $\boldsymbol{a}$ and observe total reward $r = \sum_{z\in\mathcal{Z}} \sum_{i\in\mathcal{N}_z^s} \omega_{iz}(s, a_i, \boldsymbol{a}_{-i})$ and next state s'.
  Compute incentive values $\delta_{zz'}$ as per Equation 9.4.
  for all agent i **do**
    update $r_i = (1 - \eta)\omega_{iz}(s, a_i, \boldsymbol{a}_{-i}) + \delta_{zz'}$ as per their local transition
  Store transition $(s, r, \boldsymbol{a}, s')$ in $\mathcal{J}_c$
  for all agent $i \in \mathcal{N}$ **do**
    Store transition (s, z, a_i, r_i, s', z') in $\mathcal{J}_i$
  Periodically update the network parameters with loss functions provided in Equations 9.6-9.9
  Periodically update the target network parameters.

Algorithm 9 provides detailed steps of IBQ. Individual agents act as per their own

learning (Line 12). Central agent observes r values (Line 14) and computes incentive values for local state transition (Line 15). It then provides corresponding payoffs to the agents (Line 17). Bothe central agent and individual agents update their replay memory (Line 18-20). Then the central agent and individual agents periodically update their network parameters (Line 21-22).

9.4 Experiments

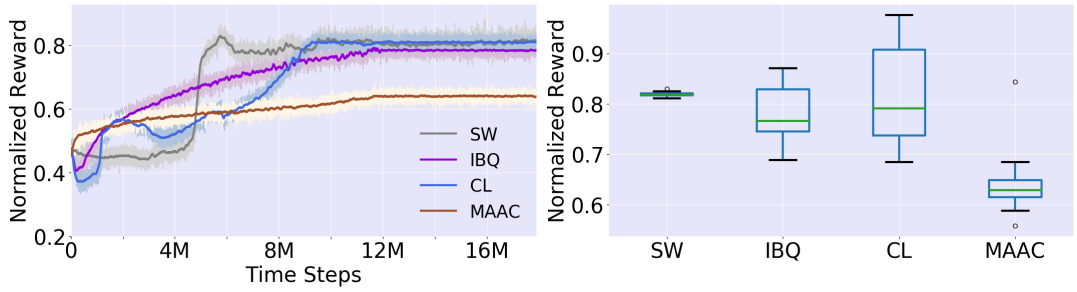
We perform experiments on the two simulators we have used in previous chapters - taxi simulator (Section 4.2.1) and online to offline service simulator (Section 4.2.2).

We compare performance of IBQ with two different relevant algorithms (1) CL (Verma & Varakantham, 2019) method presented in Chapter 8, and (2) multi-agent actor critic (MAAC) (Lowe et al., 2017). Similar to CL, MAAC is also a *centralized learning decentralized execution* algorithm which allows a centralized critic Q-function to be trained with the actions of other agents, while the actor needs only local observation to optimize its policy. As discussed in Chapter 4, we consider the action space of the central agent to be continuous for all our experimental domains. As done in previous chapters, we also compare with social welfare the (SW) policy where the all the agents execute the suggested social welfare policy cooperatively.

Similar to earlier chapters, we evaluate the performance of all learning methods on two key metrics:

- Social welfare payoff computed by aggregating payoffs of all the individual agents.
- Variation in payoff of individual agents after the learning has converged.

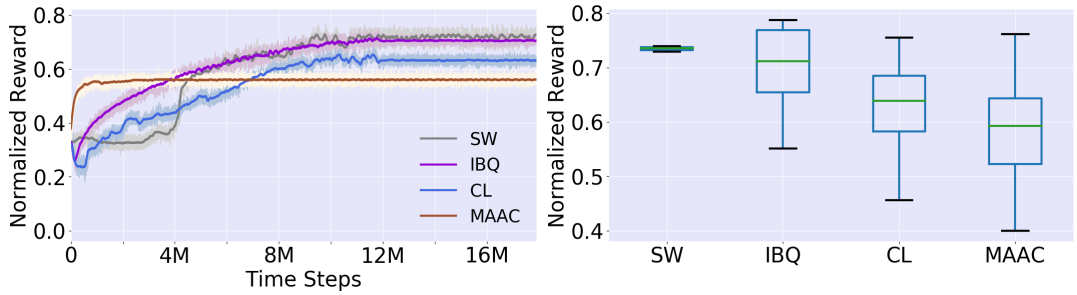
Payoff of all the agents is reset after every 1000 (1e3) time step. The graphs where social welfare of individuals has been compared provide running average of revenue over 100 evaluation periods.



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 9.1: Taxi simulator using real-world data set



(a) Social welfare of individuals

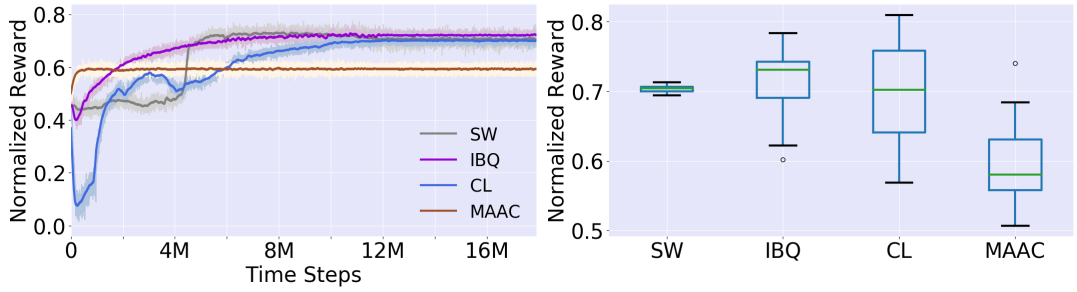
(b) Variance in individual revenue

Figure 9.2: DAR=0.4 with dynamic demand arrival rate

9.4.1 Taxi Simulator

Figure 9.1 presents the performance comparison for taxi simulator where the zone structure and demand distribution were simulated using real-world data. In Figure 9.1a we can see that IBQ's social welfare value only slightly lesser (2-3 %) than that of centralized cooperative learning and CL. As seen in previous chapters, variance is minimum for SW(Figure 9.1b). Also, variance for IBQ is lower than the other algorithms. The social welfare value of individuals is considerably low for MAAC.

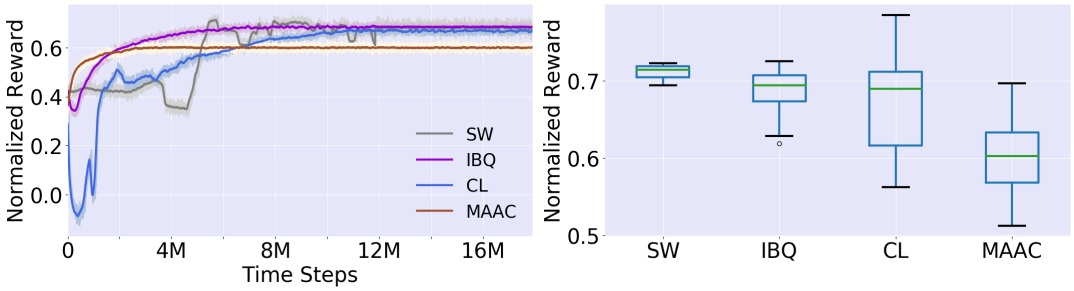
As seen in Figure 9.1a, the social welfare value of individuals for IBQ is $\approx 13-20\%$ higher than MAAC. As discussed in the last chapter, it means that there are some "lost demand" (demands that were not served) for MAAC which are being served by SW and CL and IBQ.



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 9.3: DAR=0.5 with non-uniform trips pattern



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 9.4: DAR=0.6 with uniform trip pattern.

9.4.2 Synthetic Online to Offline Service Simulator

Figures 9.2 - 9.4 presents plots for social welfare of individuals and boxplots for the variance in individual revenue for various experimental setups. The first result is for a setup with dynamic arrival rate, non-uniform trip pattern with DAR=0.4. Social welfare value for both IBQ and CL is ≈10-12% more than MAAC, NFSP and MFQ (Figure 9.2a). In Figure 9.2b we can see that variance for IBQ is slightly lesser than CL.

For setup with dynamic arrival rate and DAR=0.5 in Figure 9.3a, social welfare value for IBQ and CL is ≈8-12 % more than MAAC. Though the social welfare values are similar for IBQ and CL, the variance is significantly high for CL. As seen in box-plots of Figure 9.3b, the variance is lower for IBQ, which indicates that the joint policy of agents converge to social welfare maximizing policy. As the suggested action is sampled from the same mixed policy (social welfare policy), everyone's long term payoff is similar. We observed similar results for the setup with non-uniform trip pattern and DAR=0.6 (Figures 9.4a and 9.4b), IBQ and CL generated ≈6-10 % more social welfare

revenue than MAAC and variance for IBQ is considerably lower than that of CL.

Here are the key conclusions:

- Performances of MAAC with respect to SW vary with varying complexity of the experimental setup (with and without dynamic arrival rate and non-uniform trips). However IBQ's and CL's social welfare is consistently at par with SW's social welfare.

- Based on the complexity of experimental setup, IBQ's performance is either similar (Figures 9.1, 9.3, 9.4) or better than CL (Figure9.2).

It is notable that IBQ is able to achieve similar performance as CL even without explicitly providing any information related to the joint action.

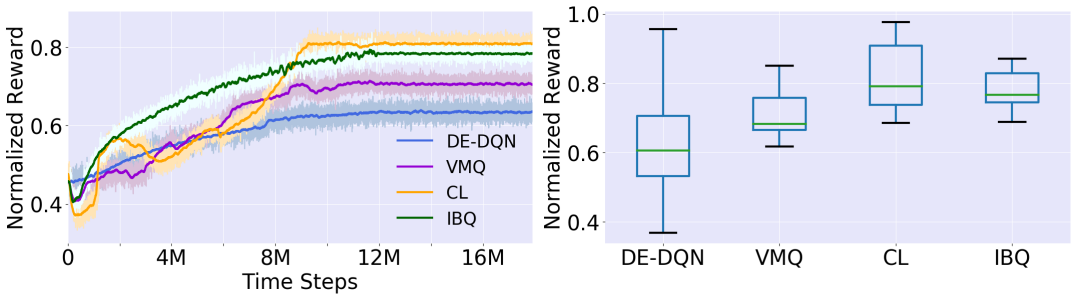
9.5 Summary

In this chapter we introduced IBQ, where central agent provides incentives based on the potential values of state. The central agent learns a social welfare policy which maximized combined values of both central and individual agents. The level of information sharing for IBQ is lesser than CL (Chapter 8) as the central agents do not suggest any policy to individuals. However, level of learning remains the same for the central agent for both IBQ and CL. Experimental results confirms that the proposed scheme of computing payment/penalties based on difference in the social welfare values for current vs. optimal joint policy is effective. IBQ is robust than CL as it does not rely on the assumption that the individuals will explore the suggested action during high exploration phase of the learning.

Chapter 10

Discussion

This thesis presents methods of learning for individual agents in a non-cooperative anonymous multi-agent settings. I have focused on aggregation systems where we can exploit the presence of a centralized entity with full view of the system. However, the objectives of the centralized entity and individual agents are not correlated hence the proposed approaches here accommodates the fact that the centralized entity is also self-interested. Based on two learning dimensions - level of extra information shared to the individual learners and level of learning done by the centralized entity, I have provided various methods of learning in my thesis. First, I provided a fully decentralized learning method where independent agents learn from the offline trajectories (ILT) by considering that others are following stationary policies. I then propose Density Entropy Deep Q-Network (DE-DQN) where the agents utilize the anonymity feature of the domain and consider the number of other agents present in their local observation to improve their learning. By increasing the levels of the two dimensions, I provide a variance minimizing Q-learning (VMQ) approach to learn ϵ-Nash equilibrium policies where the the centralized entity also learns based on the learning of the individuals. My next contribution is Correlated Learning (CL) approach, where the centralized agent learns a social welfare policy and suggests it to the individuals. Whereas the individual agents learn best response policies to the social welfare policy suggested to them. Finally, I propose an incentive based Q-learning (IBQ) approach where apart from learning social



(a) Social welfare of individuals

(b) Variance in individual revenue

Figure 10.1: Taxi simulator validated on real world data set

welfare policy, the centralized agent also provides incentives to the individuals such that they converge to a policy which improves overall system performance. Experimental results on real-world data sets and multiple synthetic data sets demonstrate that these approaches outperform other state-of-the-art approaches both in the terms of individual payoffs and overall social welfare payoff of the system.

Figure 10.1 provide a detailed comparison of the methods proposed in this thesis. We can see that as the level of information sharing and level of learning done by the centralized entity increases, the performance of the overall system starts improving with CL and IBQ achieving performance similar to a cooperative centralized learning in terms of the social welfare values. Increasing the levels of the two dimensions also help in achieving fairness (individuals earning similar amount of payoffs) in the terms of individual payoffs.

Having provided multiple learning methods to maximize revenue in aggregation systems, few natural questions arise:

- Which method is applicable in different setups of aggregation systems?
- Which method is best suited for individuals?
- Which method is best suited for an aggregation company?

It is possible that a food delivery aggregation company is not willing to provide extra assistance to the individuals. In food delivery systems, customers do not leave if a delivery boy is not readily available to deliver their food and they will wait even if the food

delivery is a little delayed. Therefore, the *lost demand* scenario does not arise frequently is this subsystem. In a setup of aggregation system where the aggregation companies are not willing to participate in the learning, the proposed density entropy based learning (DE-DQN/DE-A2C) which learns from local observation is more suitable for the individual learners.

For aggregation systems where scenarios of *lost demand* is more prevalent, it is best for the aggregation companies as well as the individuals that the company invest and participate in the learning. Though we have proposed equilibrium based solution (VMQ), other methods have shown to perform better. Furthermore, equilibrium solutions are known to be sub-optimal. Hence in such scenario, CL or IBQ is more suited. Moreover, as CL works on strong assumption that individuals will follow the suggestion during exploration, it might not be practical to use CL in real-world. Hence, if the company is willing to participate in learning, IBQ is the best suited method of learning for both individuals as well as the central agent.

Operationalization

There are few challenges in operationalizing the proposed methods in the real-world. The biggest challenge is to deal with initial learning phase when the learned policy is too random. The company will loose trust of individuals if it suggests them random actions during exploration phase. One way to mitigate this problem is to first learn a reasonable policy in simulation and use it in the field only when the learned policy has stabilized to an extent. Having a threshold value with respect to the cost involved in performing an action (for example distance/time required to travel), or a threshold on the level of confidence in the suggested action can be applied before suggesting actions to the individual agents.

One challenge with respect to individual learning is computational resources required to perform learning. The proposed methods utilize neural network framework to estimate values in continuous state space. As a results, the mobile device used by individuals for navigation may not be suitable to perform intensive computation needed for deep learning. However, cloud computing infrastructure which has become very popu-

lar recently can be used to solve this problem. The real resource intensive computation can be done in the cloud and the mobile device can be used as an interface between the individual agents and learning model. In fact, I feel that this is an exciting business idea to provide a subscription based learning platform to the individual agents.

With respect to IBQ, another challenge is to handle the incentives. Specifically during the initial learning phase when the policies are not stabilized and might provide unrealistic values as incentive. As discussed in previous paragraph, we assume that when IBQ is operationalized, the learning is not done from scratch and the company has a fair idea about the social welfare policy. Initial learning can be done in simulation and when the suggested incentive values instill some confidence, it can be implemented in the field. Furthermore, instead of using the exact value to provide incentive/charge penalties, a fraction of suggested value can be used.

Future Research

I believe the contribution of this thesis is an important progress towards applying AI learning into real-world problems where a large number of autonomous agents are present and there is a self-interested centralized entity who has the full view of the system. In the future, I would like to build upon my work by pursuing following research directions

-*Dynamic agent population*: A majority of MARL algorithms which focus on providing optimal solution in multi-agent settings assume that the number of agents is fixed. However, in reality it is seldom true. There is a need to accommodate arrival and departure of agents in the learning dynamics. Considering dynamic agent population in an environment with large number of agents is an interesting future direction for my work.

-*Mixed cooperative competitive environments*: Most of the MARL work either focus on competitive or cooperative environments. In reality, it is difficult to clearly classify real world domains into competitive or competitive. A more practical approach is provide solutions for mixed settings which acknowledge the existence of adversarial and collaborative agents together. I would like to extend my work where presence of

centralized entity is exploited in a mixed cooperative-competitive setting.

-*Domains other than aggregation systems*: There are other real-world domains such as maritime traffic control, traffic light signal control, mobile crowd sourcing etc. where presence of a central entity can be exploited to achieve higher system wide performance. However each domain are endowed with different features and and have their own limitations. Hence in future I would like to explore and extend my work for these domains.

References

Andre, D., & Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Aaai/iaai* (pp. 119–125).

Angelidakis, H., Fotakis, D., & Lianeas, T. (2013). Stochastic congestion games with risk-averse players. In *International symposium on algorithmic game theory* (pp. 86–97).

Asmuth, J., Littman, M. L., & Zinkov, R. (2008). Potential-based shaping in model-based reinforcement learning. In *Aaai* (pp. 604–609).

Babes, M., Munoz de Cote, E., & Littman, M. L. (2008). Social reward shaping in the prisoner's dilemma.

Banerjee, B., & Peng, J. (2003). Adaptive policy gradient in multiagent learning. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems* (pp. 686–692).

Bilancini, E., & Boncinelli, L. (2016). Strict nash equilibria in non-atomic games with strict single crossing in players (or types) and actions. *Economic Theory Bulletin*, *4*(1), 95–109.

Bogachev, V. I. (2007). *Measure theory* (Vol. 1). Springer Science & Business Media.

Bowling, M. (2005). Convergence and no-regret in multiagent learning. In *Advances in neural information processing systems* (pp. 209–216).

Bowling, M., & Veloso, M. (2001). Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence (ijcai)* (Vol. 17, pp. 1021–1026).

Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, *13*, 374–376.

Bulitko, V., Sturtevant, N., & Kazakevich, M. (2005). Speeding up learning in real-time search via automatic state abstraction. In *Aaai* (Vol. 214, pp. 1349–1354).

Busoniu, L., De Schutter, B., & Babuska, R. (2006). Decentralized reinforcement

learning control of a robotic manipulator. In *Control, automation, robotics and vision, 2006. icarcv'06. 9th international conference on* (pp. 1–6).

Carmel, D., & Markovitch, S. (1996). Incorporating opponent models into adversary search. In *Aaai/iaai, vol. 1* (pp. 120–125).

Carmona, R., & Delarue, F. (2014). The master equation for large population equilibriums. In *Stochastic analysis and applications* (pp. 77–128).

Castellini, J., Oliehoek, F. A., Savani, R., & Whiteson, S. (2019). The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems* (pp. 1862–1864).

Chakraborty, D., & Stone, P. (2011). Structure learning in ergodic factored mdps without knowledge of the transition function's in-degree. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 737–744).

Chau, C. K., & Sim, K. M. (2003). The price of anarchy for non-atomic congestion games with symmetric cost maps and elastic demands. *Operations Research Letters*, *31*(5), 327–334.

Chen, Y., Zhou, M., Wen, Y., Yang, Y., Su, Y., Zhang, W., ... Liu, H. (2018). Factorized q-learning for large-scale multi-agent systems. *arXiv preprint arXiv:1809.03738*.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, *1998*, 746–752.

Conitzer, V., & Sandholm, T. (2007). Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, *67*(1-2), 23–43.

Devlin, S., & Kudenko, D. (2011). Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th international conference on autonomous agents and multiagent systems-volume 1* (pp. 225–232).

Devlin, S., Kudenko, D., & Grześ, M. (2011). An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, *14*(02), 251–278.

Devlin, S. M., & Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems* (pp. 433–440).

Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2018). Learning with opponent-learning awareness. In *International conference on autonomous agents and multiagent systems (aamas)* (pp. 122–130).

Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., Whiteson, S., et al. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*.

Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2017). Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Thirty-second aaai conference on artificial intelligence.*

Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., & Spirakis, P. (2009). The structure and complexity of nash equilibria for a selfish routing game. *Theoretical Computer Science*, *410*(36), 3305–3326.

Grab. (2019). *How much is the commission charged to each completed trip. Grab*. Available: https://help.grab.com/driver/en-ph/115013565467-How-much-is-the-commission-charged-to-each-completed-trip [Last accessed: February 2020].

Greenwald, A., Hall, K., & Serrano, R. (2003). Correlated q-learning. In *International conference on machine learning (icml)* (Vol. 3, pp. 242–249).

Guéant, O., Lasry, J.-M., & Lions, P.-L. (2011). Mean field games and applications. In *Paris-princeton lectures on mathematical finance 2010* (pp. 205–266).

Guo, H., & Meng, Y. (2008). Dynamic correlation matrix based multi-q learning for a multi-robot system. In *Intelligent robots and systems, 2008. iros 2008. ieee/rsj international conference on* (pp. 840–845).

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2017). Soft actor-critic: Off-policy

maximum entropy deep reinforcement learning with a stochastic actor.

Hartman, S., & Mikusinski, J. (2014). *The theory of lebesgue measure and integration*. Elsevier.

Heinrich, J., Lanctot, M., & Silver, D. (2015). Fictitious self-play in extensive-form games. In *International conference on machine learning* (pp. 805–813).

Heinrich, J., & Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., & de Cote, E. M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hester, T., & Stone, P. (2009). Generalized model learning for reinforcement learning in factored domains. In *Proceedings of the 8th international conference on autonomous agents and multiagent systems-volume 2* (pp. 717–724).

Hu, J., & Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, *4*(Nov), 1039–1069.

Hu, J., Wellman, M. P., et al. (1998). Multiagent reinforcement learning: theoretical framework and an algorithm. In *Icml* (Vol. 98, pp. 242–250).

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical review*, *106*(4), 620.

Jong, N. K., & Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *Ijcai* (pp. 752–757).

Kapetanakis, S., & Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. *AAAI/IAAI*, *2002*, 326–331.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krichene, W., Drighes, B., & Bayen, A. M. (2014). Learning nash equilibria in congestion games. *arXiv preprint arXiv:1408.0017*.

Krichene, W., Drighès, B., & Bayen, A. M. (2015). Online learning of nash equilibria in congestion games. *SIAM Journal on Control and Optimization*, *53*(2), 1056–

1081.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Perolat, J., Silver, D., ... others (2017). A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in neural information processing systems* (pp. 4190–4203).

Lauer, M., & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In proceedings of the seventeenth international conference on machine learning.*

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for mdps. In *Isaim.*

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163).

Littman, M. L. (2001a). Friend-or-foe q-learning in general-sum games. In *International conference on machine learning (icml)* (Vol. 1, pp. 322–328).

Littman, M. L. (2001b). Value-function reinforcement learning in markov games. *Cognitive Systems Research*, *2*(1), 55–66.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems* (pp. 6379–6390).

Lyft. (2019). *Lyft serice fee. Lyft*. Available: https://help.lyft.com/hc/en-us/articles/115013081048-The-Service-Fee [Last accessed: February 2020].

Mannor, S., Menache, I., Hoze, A., & Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on machine learning* (p. 71).

Matignon, L., Laurent, G., & Le Fort-Piat, N. (2007). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *Ieee/rsj international conference on intelligent robots and systems, iros'07.*

(pp. 64–69).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Ng, A. Y., Harada, D., & Russell, S. (1999a). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml* (Vol. 99, pp. 278–287).

Ng, A. Y., Harada, D., & Russell, S. (1999b). Theory and application to reward shaping. In *Proceedings of the sixteenth international conference on machine learning*.

Nguyen, D. T., Kumar, A., & Lau, H. C. (2017). Policy gradient with value function approximation for collective multiagent planning. In *Advances in neural information processing systems* (pp. 4319–4329).

Nutz, M. (2018). A mean field game of optimal stopping. *SIAM Journal on Control and Optimization*, *56*(2), 1206–1221.

Parkes, D. C., & Singh, S. P. (2004). An mdp-based approach to online mechanism design. In *Advances in neural information processing systems* (pp. 791–798).

Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S. A., & Botvinick, M. (2018). Machine theory of mind. In *International conference on machine learning* (pp. 4218–4227).

Randløv, J., & Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Icml* (Vol. 98, pp. 463–471).

Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: monotonic value function factorisation for deep multi-agent

reinforcement learning. *arXiv preprint arXiv:1803.11485*.

Reuters. (2016). *Uber debuts self-driving vehicles in landmark pittsburgh trial. Reuters*, 14 September 2016. Available: http://www.reuters.com/article/us-uber-autonomous-idUSKCN11K12Y [Last accessed: October 2018].

Rosenthal, R. W. (1973). A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, *2*(1), 65–67.

Roughgarden, T. (2007). Routing games. *Algorithmic game theory*, *18*, 459–484.

Roughgarden, T., & Tardos, É. (2002). How bad is selfish routing? *Journal of the ACM (JACM)*, *49*(2), 236–259.

Sandholm, W. H. (2010). *Population games and evolutionary dynamics*. MIT press.

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

Sen, S., & Sekaran, M. (1998). Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, *10*(3), 333–356.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*.

Shoham, Y., Powers, R., & Grenager, T. (2003). Multi-agent reinforcement learning: a critical survey. *Web manuscript*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Simon, H. A. (1972). Theories of bounded rationality. *Decision and organization*, *1*(1), 161–176.

Singh, A. J., Kumar, A., & Lau, H. C. (2020). Hierarchical multiagent reinforcement learning for maritime traffic management. In *Proceedings of the 19th international conference on autonomous agents and multiagent systems.* International Foundation for Autonomous Agents and Multiagent Systems.

Singh, A. J., Nguyen, D. T., Kumar, A., & Lau, H. C. (2019). Multiagent decision making for maritime traffic management. In *Proceedings of the aaai conference*

on artificial intelligence (Vol. 33, pp. 6171–6178).

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*(3), 287–308.

Stanley, H. E. (1971). *Phase transitions and critical phenomena. clarendon.* Oxford.

Straitstimes. (2016). *World's first driverless taxi trial kicks off in singapore. The Straits Times*, 26 August 2016. Available: http://www.straitstimes.com/singapore/transport/worlds-first-driverless-taxi-trial-kicks-off-in-singapore [Last accessed: October 2018].

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., ... others (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems* (pp. 2085–2087).

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*(1), 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

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

Uber. (2019). *Tracking your earnings. Uber*. Available: https://www.uber.com/gh/en/drive/basics/tracking-your-earnings/ [Last accessed: February 2020].

Varakantham, P. R., Cheng, S.-F., Gordon, G., & Ahmed, A. (2012). Decision support for agent populations in uncertain and congested environments.

Verma, T., & Varakantham, P. (2019). Correlated learning for aggregation systems. *Uncertainity in Artificial Intelligence (UAI)*.

Verma, T., Varakantham, P., & Lau, H. C. (2019). Entropy based independent learning in anonymous multi-agent settings. *International Conference on Automated Planning and Scheduling (ICAPS)*.

Wang, Y., & De Silva, C. W. (2006). Multi-robot box-pushing: Single-agent q-learning vs. team q-learning. In *Intelligent robots and systems, 2006 ieee/rsj international conference on* (pp. 3694–3699).

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

Weinberg, M., & Rosenschein, J. S. (2004). Best-response multiagent learning in non-stationary environments. In *Third international joint conference on autonomous agents and multiagent systems (aamas)* (pp. 506–513).

Wiering, M. (2000). Multi-agent reinforcement learning for traffic light control. In *Machine learning: Proceedings of the seventeenth international conference (icml'2000)* (pp. 1151–1158).

Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, *19*, 205–208.

Wiewiora, E., Cottrell, G. W., & Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (icml-03)* (pp. 792–799).

Yang, T., Meng, Z., Hao, J., Zhang, C., Zheng, Y., & Zheng, Z. (2019). Towards efficient detection and optimal response against sophisticated opponents. *International Joint Conference on Artificial Intelligence*.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., & Wang, J. (2018). Mean field multi-agent reinforcement learning. In *International conference on machine learning (icml)* (pp. 5567–5576).