Learning Dynamics and Reinforcement in Stochastic Games

by

John Edward Holler

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mathematics)
in The University of Michigan
2020

John Edward Holler

johnholl@umich.edu

ORCID iD: 0000-0002-0071-5204

Dedication

This dissertation is dedicated to Virginia Kunisch.

Acknowledgments

I would like to thank the following people for their support and friendship. Names appear in order of appearance in my life. Karen Holler, Ed Holler, Kristen Holler, Haley Hofbauer, Sam Gross, Alex Gross, Mishaal Muqaddam, Karen Smith, Igor Kriz, Robert Lutz, Joseph Kraisler, Satinder Singh, Junhyuk Oh, Janarthanan R., Tony Qin, Jieping Ji, Chenxi Wang, Xiaochen Tang, Yan Jiao, Martin Strauss, and David Leslie. You all have helped me immensely in this journey, and I am forever grateful.

Contents

List of Figures

Abstract

The theory of Reinforcement Learning provides learning algorithms that are guaranteed to converge to optimal behavior in single-agent learning environments. While these algorithms often do not scale well to large problems without modification, a vast amount of recent research has combined them with function approximators with remarkable success in a diverse range of large-scale and complex problems. Motivated by this success in single-agent learning environments, the first half of this work aims to study convergent learning algorithms in multi-agent environments. The theory of multi-agent learning is itself a rich subject, however classically it has confined itself to learning in iterated games where there are no state dynamics. In contrast, this work examines learning in stochastic games, where agents play one another in a temporally extended game that has nontrivial state dynamics. We do so by first defining two classes of stochastic games: *Stochastic Potential Games (SPGs) and* Global Stochastic Potential Games (GSPGs). We show that both games admit pure Nash equilibria, as well as further refinements of their equilibrium sets. We discuss possible applications of these games in the context of congestion and traffic routing scenarios. Finally, we define learning algorithms that

1. converge to pure Nash equilibria and
2. converge to further refinements of Nash equilibria.

In the final chapter we combine a simple type of multi-agent learning - individual Q-learning - with neural networks in order to solve a large scale vehicle routing and assignment problem. Individual Q-learning is a heuristic learning algorithm that, even in small multi-agent problems, does not provide convergence guarantees. Nonetheless, we observe good performance of this algorithm in this setting.

Chapter 1

Introductory Material

1.1 Goal

This work does not appear in chronological order. Chapter 4 corresponds to a project conducted in 2017. The work in Chapters 2 and 3 was done in 2018 and 2019. Chapter 4 conducts an empirical study of *deep reinforcement learning* (DeepRL) methods applied to multi-driver vehicle assignment and repositioning. DeepRL is a research area that combines *deep learning* and *reinforcement learning*. Deep learning consists of designing neural network architectures and training them with (stochastic) gradient descent and reinforcement learning is a learning paradigm through which one can train an agent to behave optimally in an environment. Chapters 2 and 3 can be viewed either as belonging to multi-agent systems or game theoretic control, depending on the reader's affinity. Chapter 2 examines two extensions of potential games to stochastic games. Potential games are known to admit a number of learning algorithms that converge to Nash equilibrium. Chapter 3 studies learning algorithms in the newly introduced classes of stochastic games. The next few paragraphs attempt to explain how the research in this work unfolded chronologically.

By 2017 DeepRL had seen a number of enormous successes. In 2014 researchers at Deepmind trained a DeepRL agent to play many Atari 2600 games at a superhuman level [45]. About one year later, Deepmind researchers again shocked the machine learning community when their computer Go player, dubbed "AlphaGo" [66], beat 9-dan ranked Lee Sedol in four out of five matches. The victories themselves were amazing, but were made even more incredible by the design of AlphaGo. The system was not given any dictionaries of opening moves, endgame sequences, or piece valuations. Rather, it was trained only by playing matches and learning "from scratch". That work has been carried forward and produced a new agent, "AlphaZero" [67], that plays chess, shogi, and Go at superhuman levels and can be trained in about a day.

Despite striking empirical success, there is a notable lack of theoretical results in DeepRL. Reinforcement learning on its own provides methods for learning in an environment, but many of its celebrated results only have convergence guarantees in a tabular learning setting. Tabular methods are appropriate in situations where the state and action spaces are small, so that individual states and actions may be visited and tried many times. Deep learning vastly extends the applicability of reinforcement learning by incorporating function approximation, however this comes at the price of convergence guarantees. And yet, despite the dearth of theoretical results, DeepRL has had numerous successes and continues to be an area of great interest both in academia and industry.

While the empirical progress of DeepRL has outrun reinforcement learning theory, the tabular theory of reinforcement learning developed in the 1980's and 1990's [79, 69] was paramount to these modern triumphs. Deep Q-networks (DQN) [45], deep deterministic policy gradient (DDPG) [33], and trust region policy optimization (TRPO) [62] are all examples of DeepRL algorithms whose designs are informed by tabular counterparts that preceded them by decades. Our present work is informed by the understanding that modern DeepRL owes its success to solid theoretical work in reinforcement learning. Many researchers have turned their focus towards integrating DeepRL algorithms in strategic, multi-agent interactions [75]. Yet, unlike single-agent reinforcement learning, the theory of tabular learning in the multi-agent settings is not well understood. Certain negative results are known, for example, that agents independently engaging in tabular Q-learning do not necessarily reach Nash equilibrium, even when the game has only one state [cite].

Chapter 4 presents empirical work in which DeepRL methods are applied to driver dispatching and repositioning problems at DiDi Taxi, a ridesharing company. Every few seconds, DiDi must take stock of available drivers in an area, requesting customer orders, and decide which drivers to dispatch to which orders. With an appropriately selected neural network architecture one may approach the problem with DeepRL techniques. One may either cast the problem as a single-agent problem in which the agent is DiDi, or a multi-agent problem in which the agents are drivers. Over the course of experimentation it became clear that while the single-agent learning method could sometimes outperform the multi-agent method in final performance, the multi-agent methods trained much faster, and generally was more stable. Specifically, the single-agent algorithm required more and more training time as the number of drivers increased, while the multi-agent training time was essentially constant. This suggests that decomposing a large single-agent problem into multiple smaller single-agent problems can be a powerful technique. However, from a theoretical perspective, the large single-agent problem enjoys convergence guarantees in a tabular setting while the multi-agent approach does not.

This gap in empirical success and lack of theoretical justification motivated the work in Chapters 2 and 3. Much work in game theory and multi-agent reinforcement learning has explored learning in normal form games. The types of games in which players may learn to play Nash equilibria through uncoupled dynamics (which closely resemble reinforcement learning) remains an open area of research. However, a number of such games have been identified: zero sum games, team games, potential games, and supermodular games [24]. Very few games are known to converge to Nash equilibria under Q-learning. As far as that author is aware, the only positive results known are for zero sum games and 2-player partnership games [31].

The DiDi work, as well as most domains of interest in reinforcement learning, involve problems where there is a state that evolves in time as a result of the decisions of an agent. Normal form games on the other hand are stateless. The appropriate extension of game theory to problems with state are known as stochastic games. Unlike learning in normal form games, learning in stochastic games is still quite fledgling. Very few subclasses of games that are amenable to learning methods have been identified in the literature. Some pioneering work has been done in the case of stochastic zero-sum games [1] and stochastic team games [2], but this area of research could use more attention. Chapter 2 will examine two distinct ways that we can extend the definition of normal form potential games to the stochastic setting. Chapter 3 presents extensions of two normal form game learning methods, joint strategy fictitious play (JSFP) with inertia and log-linear learning (LLL), to these stochastic games.

The preceding discussion suggests a *prescriptive* motivation for studying stochastic games, with a view towards applying stochastic game learning methods as a computational tool. However, there is also a *descriptive* motivation for understanding such methods. There is an enormous amount of research in psychology and neuroscience which suggests that mammalian learning resembles a form of reinforcement learning [85, 53, 56]. On the other hand, we know that humans and animals learn in an environment filled with other learning agents. A careful study of learning in stochastic games may uncover explanations for observed animal and human behavior.

In conclusion, the study of learning in stochastic games has at least two distinct motivations. First, there are real world single-agent problems that may become more computationally tractible when decomposed into several simpler interrelated learning problems. Second, understanding the possibilities and limitations of learning in stochastic games may help us better understand animal and human behavior. Despite these motivations, theoretical studies on learning in stochastic games have remained scarce. Possible directions of research include identifying subclasses of stochastic games amenable to learning algorithms, proving convergence results in such games, and characterizing various equilibria sets for such games. This thesis embarks on such a study by extending potential games in two different ways into the class of *stochastic potential games* (SPGs) and *stochastic global potential games* (SGPGs). For these classes of games we study their structure, equilibria sets, and identify several convergent learning algorithms. In the final chapter we shift to an empirical study of the power of applying DeepRL to multi-agent decision problems.

1.2 Markov Decision Processes

Markov Decision Processes (MDPs) provide a straightforward means of modelling discrete-time optimization problems in which the actions of a single decision maker within an environment result in rewards and determine how the state of the environment changes in time. When there is a single decision maker we will refer to them as an agent. A typical example is an agent placed inside a maze; the agent takes actions to navigate the maze (forward, backward, left, right), and receives a constant reward of -1 after each decision. The environment terminates when the agent reaches the end of the maze. Given this reward

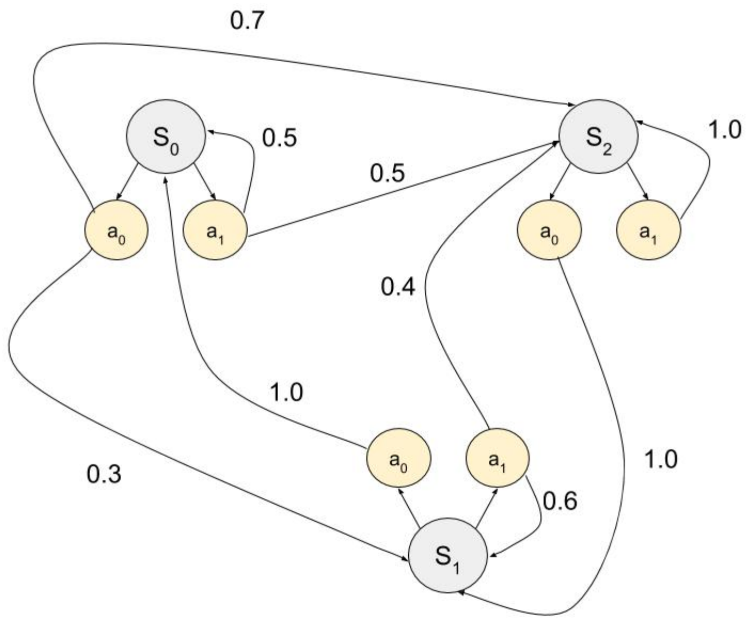

Figure 1.1: A finite MDP with 3 states and 2 actions in each state. The edges are labelled with the action-dependent probability transition.

signal, if an agent learns to maximize the sum of their rewards this is equivalent to saying that the agent has learned the quickest route to the end of the maze.

Formally, an MDP consists of a state space $\mathcal{S}$ and an action spaces $\mathcal{A}_s$ for each state $s \in \mathcal{S}$. In situations where the action spaces are all the same we drop the subscript. The agent "enters" the environment according to an initial probability distribution μ_0 over states. At each timestep t, the agent finds themselves in a state s_t and must select and action $a_t \in \mathcal{A}_{s_t}$. They transition to the next state s_{t+1} according to a transition probability distribution $P_{s_t s_{t+1}}(a) = p(s_{t+1}|s_t, a_t)$ and also receive a reward according to a reward distribution $R(s_t, a_t) = p(r_t|s_t, a_t)$. A *finite* MDP is one in which the state space is finite, action spaces are finite, and the possible rewards received at each timestep are finite. An example of a finite MDP is shown below:

Often, to deal with MDPs that last for arbitrary amounts of time, we introduce a *discount* *factor* $\gamma < 1$ that weights future rewards less than immediate rewards. This is to guarantee that infinite sums converge.

In Chapters 2 and 3 we will specifically be dealing with *fixed-time episodic MDPs*. In such MDPs, the state of an agent is "reset" using μ_0 every T timesteps where T is a fixed constant, and it is unnecessary to perform reward discounting, so we set $\gamma = 1$. When it is clear, we will drop γ entirely from equations. Time plays as important role in the decision-making process of the agent in fixed-time episodic MDPs. Take for example a basketball player dribbling the ball up across halfcourt in the first quarter of a game. When there are 2 minutes left in the quarter, the player should continue to dribble the ball up and run a play with his teammates to maximize the chances of a scoring a field goal. Under essentially no circumstances should the player shoot the ball at halfcourt. In contrast, in the exact same situation, except with 2 seconds left in the quarter, the player should almost certainly shoot the ball. We should really consider these two different situations as separate states, even if everything else on the court is identical. We can view the state space of a fixed-time episodic MDP as branching at every timestep, and never returning to the same state or earlier states in a given episode. This structure is shown in Figure 1.2.

"Time" can become overloaded in this discussion, and so from now on we will refer to these T timesteps as *layers*. A state being in layer k means that it may only be encountered at the k^{th} timestep in the MDP.

Next, we introduce some definitions that are important in discussing agent behavior in an MDP.

mydef]thmA *policy* π is a complete specification for how an agent should select actions in each state. It consists of a collection of probability distributions π_s, indexed by state s, where each π_s is a distribution over actions.

mydef]thmA policy π is called deterministic if for each distribution π_s there exists an action $a \in \mathcal{A}_s$ such that $\pi_s(a) = 1$.

mydef]thmThe *state value function* of a policy π is the function

$$V_\pi : \mathcal{S} \to \mathbb{R} \tag{1.1}$$

which maps a state to the expected sum of rewards that the agent will receive starting in

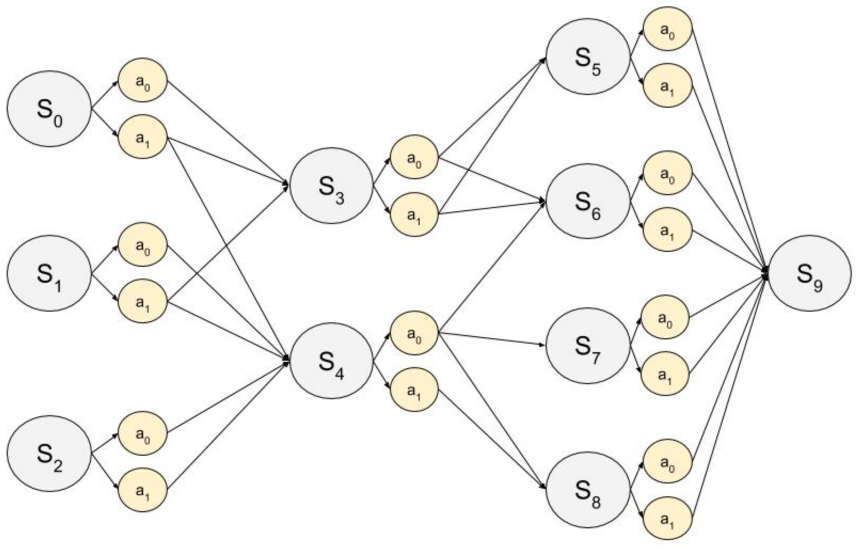

Figure 1.2: A finite 4 layer fixed-time MDP. There are 10 states, and two actions in every state except the final absorbing state. Transition probabilities are omitted to more clearly illustrate structure.

state s and behaving according to policy π. Explicitly it is given by the following equation:

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=k}^{T} R_t(a_t, s_t)|s_k = s\right] \tag{1.2}$$

mydef]thmA policy π^* is called *optimal* or *a solution to the MDP* if

$$\pi^* \in \text{argmax}_\pi \mathbb{E}_{s\sim\mu_0}\left[V_\pi(s)\right]$$

In other words, a policy π^* is optimal if it maximizes the expected sum of rewards that an agent encounters during an episode. The following is a well-known fact about optimal policies in an MDP:

Theorem 1.2.1. *Every MDP admits a* deterministic *optimal policy* π^*.

Chapter 4 adapts the well known Q-learning algorithm to a novel neural network architecture. To discuss Q-learning we must first introduce the action-value function.

mydef]thmThe *action-value* or *Q-value* function of a policy π is the function which maps a state s and state action $a \in \mathcal{A}_s$ to the expected sum of rewards that the agent will receive starting in state s, taking action a, and then behaving according to policy π. Explicitly it is given by the following equation:

$$V_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=k}^{T} R_t(a_t, s_t)|s_k = s, a_k = a\right] \tag{1.3}$$

There are two broad approaches to solving MDPs:

1. Model-based approaches such as dynamic programming
2. Model-free approaches, such as Q-learning and policy gradients

These two approaches differ in their assumptions regarding agent sophistication. Model-based methods make the assumption that an agent has access to transition probabilities and reward functions, meaning they may use these quantities in calculating how to behave.

They are so named because agents have "access to the model". In contrast, model-free methods do not make this assumption, and assume that agents only have knowledge of *their own experience*. Model-free methods are attractive as they represent a more realistic framework for "learning from scratch" as compared to model-based methods. The empirical study in Chapter 4 uses model-free methods, while our theoretical study of learning in stochastic games in Chapter 3 makes use of model-based approaches.

1.3 Reinforcement Learning

The previous section discussed MDPs, which describe a type of sequential optimization problem. In this section we will discuss methods of solving MDPs. This background is relevant to the later chapters for two reasons. First, the empirical DeepRL work in Chapter 4 uses Q-learning which is an off-policy model-free reinforcement learning algorithm. Second, the multiagent learning algorithms in Chapter 3 will in various ways draw analogies to several model-based and model-free learning algorithms.

1.3.1 Policy Evaluation

Policy evaluation is a model-based method by which a player can calculate the value function $V_\pi(s)$ for a given policy π. It is a solution to the first question one might want to answer en route to finding an optimal policy: how can an agent compute the performance of a given policy? The algorithm relies on the fact that $V_\pi(s)$ satisfies the *Bellman equation*:

$$V_\pi(s) = \sum_a \pi(a|s) \left[r(s,a) + \gamma \sum_{s'} P_{ss'}(a) V_\pi(s') \right] \tag{1.4}$$

In fact it is the *unique* solution, meaning this equation completely characterizes $V_\pi(s)$. One can calculate $V_\pi(s)$ via the recursive update

$$v_{k+1}(s) = \sum_a \pi(a|s) \left[r(s,a) + \gamma \sum_{s'} P_{ss'}(a) v_k(s') \right] \tag{1.5}$$

It can be shown that $\lim_{k\to\infty} v_k(s) = V_\pi(s)$.

1.3.2 Policy Improvement

The next natural question one can ask is how to improve on a given policy π once it has been evaluated. This can be accomplished through policy improvement, which is a model-based learning method.

Suppose we have arrived at the state-value function $V_\pi(s)$, perhaps through policy evaluation. Given access to the model, we can immediately compute the action-value function $Q_\pi(s,a)$ via

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} P_{ss'}(a) V_\pi(s') \tag{1.6}$$

From here we define a new policy π' according to the decision rule

$$\pi'(a|s) = \mathrm{a}rgmax_a Q_\pi(s,a) \tag{1.7}$$

The *Policy Improvement Theorem* guarantees that

$$V_{\pi'}(s) \geq V_\pi(s) \qquad \text{for all states } s \tag{1.8}$$

1.3.3 Policy Iteration

We now have all of the requisite ingredients to perform model-based reinforcement learning. We can:

1. Evaluate π (policy evaluation)
2. Improve π once it has been evaluated (policy improvement)

Policy iteration simply alternates between these two steps. That is, in each epoch, the algorithm begins with a policy π, evaluates it using policy evaluation, and derives a new improved policy π' from policy improvement. In the next epoch, π' takes the place of π. This process is guaranteed to converge to an optimal policy π^*.

1.3.4 Q-learning

Q-learning [79] is the quintessential model-free reinforcement learning algorithm. It leads to an optimal policy while making minimal assumptions about an agent's knowledge of the environment. In particular Q-learning only assumes the agent observes trajectories ie state-action-reward sequences from the environment. Q-learning is an *off-policy* learning algorithm, meaning that the agent follows one policy while estimating the value of a different policy. The agent can essentially follow an arbitrary policy (so long as it is sufficiently exploratory) and gain knowledge about the action-values of an optimal policy. This makes Q-learning quite flexible. For example, the agent may explore the environment according to any number of schemes. Also, the agent can incorporate experience from various sources during Q-learning, for instance, it may utilize experience from expert players [**?**, **?**].

The goal of Q-learning is to learn the Q-values $Q_*(s,a)$ of an optimal policy π^*. Note that, while π^* may not be unique, all optimal policies have the same Q-values. This follows from the fact that Q^* is the unique fixed point of a contraction operator. Pseudocode for Q-learning is shown in Algorithm 1.

Algorithm 1: Tabular Q-learning algorithm.

```
Initialize $Q(s,a)$ arbitrarily for non-terminal states and $Q(s,a) = 0$ for terminal states.
$n = 0$
while 1 do
    Initialize starting state $s$
    while state is not terminal do
        Take action $a$ chosen according to exploratory behavior policy $\pi$
        Receive reward $r$ and next state $s'$
        $Q_{n+1}(s,a) \leftarrow Q(s,a) + \alpha_n \left[r + \gamma max_a Q_n(s',a) - Q_n(s,a)\right]$
        $n \leftarrow n+1$
    end while
end while
```

The quantity α_n is positive and sometimes called the learning rate. For Q-learning to converge, the sequence $\{\alpha_n\}$ must satisfy

$$\sum_n \alpha_n = \infty \qquad \text{and,}$$
$$\sum_n \alpha_n^2 < \infty$$

A standard choice is $\alpha = \frac{1}{n}$, which makes the update equation into a running average.

1.4 Neural Networks

Artificial neural networks are computational models for learning that have been around since at least the 1940's [40, 22]. Their popularity has waxed and waned over the last 7 decades, but in the last 10 years interest has grown dramatically as various computational components such as convolutional networks [29, 27, 45], recurrent networks [41, 20, 44], and attention mechanisms [21, 74, 81] have shown impressive performance in a variety of classification, generation, and control tasks. The modern study of artificial neural networks is called *deep learning* to emphasize the use of deep networks with many layers of neurons. Deep learning has become a vast subfield of machine learning, with lengthy books and monographs dedicated to its study [19, 61]. In this section we will briefly touch on some of the key ideas in deep learning that are used in the work found in Chapter 4.

1.4.1 Overview and feedforward networks

A neural network is a parametrized function $y = f(x; w)$ where x is an input vector and w is a vector of parameters for the function. What gives a neural network its distinctive characteristic is the modular and layered way in which its parameters are arranged in the computation process. A representation of a small neural network is shown below.

The input vector x is represented at the bottom by the 5 green blocks, indicating that the input is a 5-dimesional vector. The input vector undergoes 4 successive layers of computation. Each arrow in the diagram corresponds to a single parameter (also known as a *weight*) in the network. In the first layer, the input is transformed into a 3-dimensional vector. The result is then transformed into a 2-dimensional vector, a 3-dimensional vector,

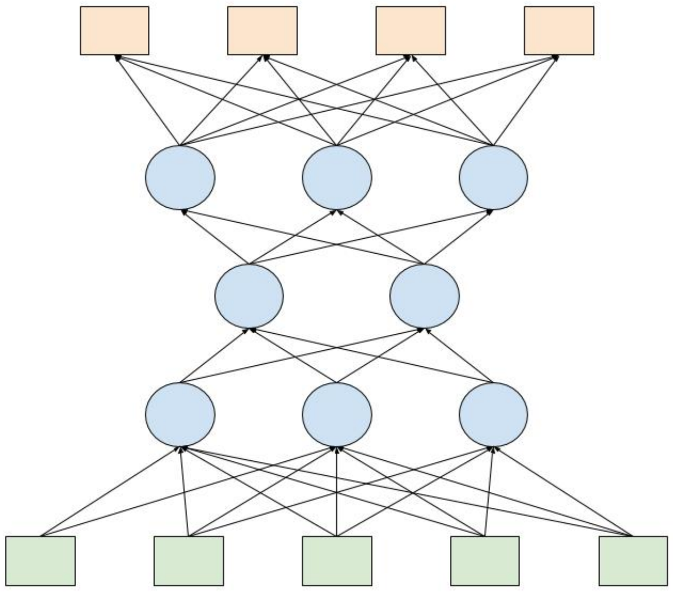

Figure 1.3: A small neural network. The network has a 5 dimensional input, 4 dimensional output, and 3 hidden fully connected layers.

and finally the output (shown as orange blocks) which is a 4-dimensional vector. We say this network has 3 hidden layers, corresponding to the layers of intermediate vectors displayed as blue circles. We denote these by the vectors h^1, h^2, and h^3. The computational components in each layer may vary from network to network; we will describe one example using sigmoid activation functions.

Each arrow in the network corresponds to a weight. We let w_{ij}^k denote the weight connecting the i^th component of the layer k vector with the j^{th} component of the layer $k+1$ vector. By convention we call the input vector the 0^{th} layer. The first hidden layer h^1 is calculated from the input x by

$$h_j^1 = \sigma\left(\sum_i w_{ij} x_i\right) \tag{1.9}$$

Where σ is the sigmoid function

$$\sigma(x) = \frac{e^x}{e^x + 1}$$

Once we computesh^1, we can in turn move on to computing h^2 and so on, until we at last calculate the output vector. This process is often called the *forward pass* of a neural network; it is the sequential process by which one computes outputs y from inputs x.

1.4.2 Training neural networks

The process of training a neural network begins by initializing all of the weights of the network. Often, this is done through random sampling, although some work begins by initializing the network to a known "good" set of weights. After initialization, the network weights are adjusted slowly based on how well its output fits to a particular task. This "goodness of fit" is characterized by an error function $E(x; w)$.

In the case of supervised learning, one starts with a set of training data $\{(x_i, y_i)\}$ and the error may be taken to be the mean squared distance between the neural network outputs and the true outputs given by the data:

$$E(x, w) = \sum_i (f(x_i, w) - y_i)^2 \tag{1.10}$$

Backpropagation [58] is an efficient algorithm for computing the partial derivative of error with respect to network weights, $\frac{\partial E}{\partial w_{ij}^k}$. Once these partial derivatives are calculated, one can adjust the weights of the network by following the negative gradient of the network weights. Typically one does not compute the error with respect to all of the training data but only a small subset, called a minibatch, at a time. This introduces randomness into the gradient descent process that has been shown empirically to result in networks that generalize to unseen data better [26, 54].

1.4.3 Deep Q-learning

Only in the last 5 years have neural networks and reinforcement learning algorithms seen consistent and successful integration. In their groundbreaking work [45] researchers at DeepMind were able to train a feedforward neural network using a gradient-based version of Q-learning which they dubbed Deep Q-learning (DQN) to play Atari 2600 games. The nature of training a network to learn Q-values via experience rather than a supervised learning task forced several innovations to how data is processed and fed to networks when performing reinforcement learning tasks.

Q-learning is built on the idea of learning a guess from a better guess. While interacting with the enviornment (in their case, a video game), the system collects one step trajectories (s_t, a_t, r_t, s_{t+1}). The network takes state s as input and outputs one scalar quantity for each action. The output of the neural network is interpreted as action-values $Q_w(s, a)$, parametrized by the set of weights w. Given a one step trajectory, the system can calculate the one step error

$$\left[r_t + \gamma \max_a Q_{w_{targ}}(s_{t+1}, a)\right] - Q_w(s_t, a_t) \tag{1.11}$$

and perform backpropagation using this error function. Without special care this kind of training can be unstable, as the network is using its own calculations as targets in the error function. The researchers used two strategies for stabilizing the learning process. First, they obtained minibatch samples by maintaining one step trajectories in a large replay buffer and sampling randomly from the buffer. Second, when calculating the target for the error function, they used an old and slow updating set of weights w_{targ} for $Q_{w_{targ}}(s_{t+1}, a)$, which they called “target weights”. These techniques have been employed in a vast number of followup papers, and indeed they are employed in the Chapter 4 work.

1.5 Normal Form Games

Normal form games are one of the foundational models of strategic interaction that is studied in game theory. A finite n-player normal form game consists of a set of players $\{1, 2, \ldots, n\}$. Each player i has a finite set of actions $\mathcal{A}^i$. We define the set of joint actions to be

$$\mathcal{A} = \times_i \mathcal{A}^i \tag{1.12}$$

In addition to an action set, each player is endowed with a utility function

$$u^i : \mathcal{A} \to \mathbb{R} \tag{1.13}$$

A player’s utility serves to specify their goal within the strategic confines of the game. A player wishes to maximize their utility, yet the utility is determined by both action of the player *and* the actions of other players.

Normal form games can be represented as tensors, and the special case of 2 player games can be represented as a bimatrix. For example, as a bimatrix the standard prisoner’s dilemma [28] is given by

$$\begin{array}{cc} & \begin{array}{cc} \text{c} & \text{d} \end{array} \\ \begin{array}{c} \text{c} \\ \text{d} \end{array} & \begin{pmatrix} -1,-1 & -3,0 \\ 0,-3 & -2,-2 \end{pmatrix} \end{array}$$

Utility structure in the Prisoner’s dilemma (1.14)

The data of the game may be read directly from the bimatrix. For example, the utility of player 1 when player one cooperates (c) and player two defects (d) is the first entry in the (c, d) position of the matrix.

In Chapters 2 and 3 there will be examples of games in which there are 3 players. This can be represented by a collection of trimatrices, one for each action of the third player. For example, in the 3-player matching pennies game, player one wins by selecting the same action as player two, player two wins by selecting a different action from player one, and player 3 bets on the winner and is rewarded if they are correct. This is represented in the following two trimatrices:

bet 1

$$\begin{array}{cc} & \begin{array}{cc} \text{h} & \text{t} \end{array} \\ \begin{array}{c} \text{h} \\ \text{t} \end{array} & \begin{pmatrix} 1,-1,1 & -1,1,-1 \\ -1,1,-1 & 1,-1,1 \end{pmatrix} \end{array}$$

bet 2

$$\begin{array}{cc} & \begin{array}{cc} \text{h} & \text{t} \end{array} \\ \begin{array}{c} \text{h} \\ \text{t} \end{array} & \begin{pmatrix} 1,-1,-1 & -1,1,1 \\ -1,1,1 & 1,-1,-1 \end{pmatrix} \end{array}$$

(1.15)

The first trimatrix gives the utilities that each player receives when player three bets on player one. The second trimatrix gives the utilities if player 3 bets on player two.

Game theory seeks to understand the kinds of strategies one would expect to arise when rational agents engage in a given game. The most fundamental notion of equilibrium behavior is that of Nash equilibrium. Before we explain this concept, we introduce notation and vocabulary related to actions within a game.

Let $\Delta\mathcal{A}^i$ denote the simplex of discrete probability distributions over $\mathcal{A}^i$. We will refer to an element x^i of this distribution as a player strategy, and a *pure* strategy if the support of x^i is a single action a^i. Otherwise we call the strategy *mixed*. By a small abuse of notation we will call such pure strategies by their action name a^i. We may linearly extend the domain of each utility function to the domain $\times_{j\in\{1,\ldots,n\}}\mathcal{A}^j$, and we will generally think of the u^i's as having this enlarged domain.

A list of strategies, one per player: $x = (x^1, \ldots, x^n)$ is called a *joint strategy*. Often, it is useful to consider two joint strategies that differ by one or two players' actions. For notational convenience, we define the joint action $(x^1, \ldots, x^{i-1}, y^i, x^{i+1}, \ldots, x^n)$ by $x\backslash y^i$ and the joint action $(x^1, \ldots, x^{i-1}, y^i, x^{i+1}, \ldots, x^{j-1}, y^j, x^{j+1}, \ldots, x^n)$ by $x\backslash y^i y^j$. A joint pure strategy is one where all players' strategies are pure. In this case we will often use a and b rather than x and y.

mydef]thmFor a game $G = (\mathcal{A}^i, u^i)$ and player i, a strategy $x^i \in \Delta\mathcal{A}^i$ is said to be a best response to $\{x^j\}_{j\neq i}$ if

$$u^i(x^1, \ldots, x^i, \ldots, x^n) \geq u^i(x^1, \ldots, y^i, \ldots x^n)$$

for any $y^i \in \mathcal{A}^i$.

We now introduce the important concept of Nash equilibrium. Simply put, a Nash equilibrium is a joint strategy such that no player can unilaterally deviate and receive a larger utility.

mydef]thmA Nash equilibrium is a set of strategies $x^i \in \Delta\mathcal{A}^i$ such that for all i, the player strategy x^i is a best response to $\{x^j\}_{j\neq i}$

A *pure* Nash equilibrium is a Nash equilibrium such that each player strategy is a pure strategy. Otherwise we call the equilibrium *mixed*. A priori, a game need not have a pure Nash equilibrium. For instance, the following is the original 2-player matching pennies game

$$\begin{array}{c c} & \begin{array}{cc} h & t \end{array} \\ \begin{array}{c} h \\ t \end{array} & \begin{pmatrix} 1,-1 & -1,1 \\ -1,1 & 1,-1 \end{pmatrix} \end{array}$$

(1.16)

As we can see, no pure joint strategy will have players in simultaneous best response. However, if both player selects their action uniformly at random, then their behavior will be in equilibrium. In his original on the subject, Nash shows that all finite games contain at least one Nash equilibrium.

There are a variety of other equilibrium and dominance concepts that have been devised, however these are outside of the scope of the present work.

1.6 Learning in Iterated Normal Form Games

1.6.1 General Background

The traditional view of game theory views equilibrium as the end result of computation done by rational agents with complete knowledge of a game. For example, when we looked at the example of matching pennies in the previous section, we noted that there is no pure Nash equilibrium, and that there is a single mixed strategy equilibrium in the game. We did this without "playing" the game. We simply noted the structure of the utility functions and came to this conclusion theoretically. We could have done a formal analysis showing that the mixed strategy uniquely simultaneously maximizes the utility functions.

Learning in games takes an alternative, experiential perspective on equilibrium. It views equilibrium as the end result or limit of a process in which players repeatedly face an instance of a game and adapt their strategy over time. Depending on which players are adapting, and how much information each knows about the game, various situations and questions can be fit into this paradigm. These subjects fill entire books [16]; we provide a small sampling below.

The following is the general set up that we mean by *learning in iterated games*. We begin with a fixed n-player game G and play proceeds sequentially at discrete timesteps. At each timestep t, players are faced with an instance (we will refer to this as an *iterate*) of the game G. Each player selects a strategy at time t. Let h_t denote the history of actions and received utilities for all players up to time t. A learning method $\mathcal{L}$, or adaptive strategy, is a function which maps a play history, utility functions u_t, and player index i to a player strategy x_t^i:

$$\mathcal{L}(h_t, u_t, i) = x_t^i \tag{1.17}$$

It is of course not necessary for the learning method to make use of all input information. For example, some adaptive strategies do not assume that players have knowledge of utility functions, so that the calculation of x_t^i does not use u_t, perhaps they only have knowledge of their own utility function (so only u_t^i may appear in the calculation). A very simple learning method simply assigns a constant strategy, x^i to player i regardless of history and utilities:

$$\mathcal{L}(h_t, u_t, i) = x^i \tag{1.18}$$

Let's consider a few examples that fit into the iterated game learning paradigm. For example, first suppose only one player is able to adapt their strategy across these instances, while all other players' strategies are fixed. That is, for all players other than i, the learning method is constant. In this case, the player that is adapting is faced with an MDP with only a single state. As such, there are a number of adaptive strategies that the player may employ to optimize their utility. First, suppose the player has complete knowledge of its own utility function and the strategies of all other players. In this case, the player may directly compute

$$\mathcal{L}(h_t, u_t, i) = \text{argmax}_{a^* \in \mathcal{A}^i} \mathbb{E}_{a^{-i}} \left[u^i(b^i, a^{-i}) \right] \tag{1.19}$$

This player "learns" to play optimally in one step. On the other end of the spectrum, suppose that the player only has access to its own history of actions and the resulting

timestep	player 1 action	player 2 action
0	H	T
1	T	T
2	T	H
3	H	H
4	H	T
5	T	T

Figure 1.4: Plays in matching pennies under best response dynamics.

realized utility. In this situation, the player cannot immediately compute **??**, however they may learn it over multiple timesteps via a model-free reinforcement learning such as Q-learning. Note that in both cases described above, the limiting joint strategy is likely not a Nash equilibrium, since only one player has been allowed to update their strategy.

Best response dynamics

Now suppose that all players adapt their strategy as they iterate G. If each player has knowledge of their utility function, and of the actions played by all other players at the previous timestep, then they may employ *best response dynamics*. The best response dynamic is one of the first studied [49] adaptive rules for learning in games and it has an exceptionally simple description. In the first timestep, players select their actions arbitrarily, leading to an initial joint action a_0. For $t > 0$, player i will select an action uniformly from the set of best responses to the previous opponent action a^{-i}_{t-1}. We can write down the best response learning dynamic $\mathcal{L}_{BR}$ as follows

$$\mathcal{L}_{BR}(h_t, u_t, i) = \text{unif}\left(\mathcal{BR}(a^{-i}_{t-1})\right) \tag{1.20}$$

where unif() is the uniform distribution over the set of best responses. Depending on the structure of the game, best response dynamics may or may not converge to Nash equilibrium. For example, in matching pennies, suppose player 1 begins by playing H and player 2 begins by playing T. 1.4 shows the first 6 timesteps of best response dynamics.

As we can see, the strategies cycle every four timesteps. At no point in time do

timestep	player 1 action	player 2 action
0	c	c
1	d	d
2	d	d
3	d	d

Figure 1.5: Convergence of best response dynamics in the prisoner's dilemma.

the player strategies constitute a Nash equilibrium - this is immediately clear since their strategies at each timestep are deterministic and the only Nash equilibrium is mixed. However, the *empirical frequencies* of play do approach the unique Nash equilibrium, since both players will play H half the time and T the other half of the time. We call this *empirical frequency convergence*, and consider it a less desirable outcome than if the play itself *directly converged* to Nash equilibrium. There are games for which best response dynamics do not even converge in empirical frequency.

In contrast, consider best response dynamics applied to the Prisoner's dilemma problem. Suppose that players initially both cooperate with one another (ie they both start with strategy c). 1.5 shows the immediate convergence of best response dynamics.

In this case the strategies directly converge to the unique pure Nash equilibrium $[d, d]$ very quickly.

Despite being a well known and studied learning method, it can still be hard to determine whether best response dynamics will converge in a given game.

Fictitious play

A closely related learning algorithm is fictitious play (FP) [8]. Fictitious play uses the entire history of opponent strategies, rather than only using the strategy at the previous timestep. At time t player i calculates the empirical frequency of each other player j as

$$z_t^j = \frac{1}{t}\sum_{k=0}^{t-1} a_k^j$$

Player i will then choose their action by uniformly selecting a best response to z_t^{-i}:

$$\mathcal{L}_{BR}(h_t, u_t, i) = \text{unif}\left(\mathcal{BR}(z_t^{-i})\right)$$

FP and BR dynamics can behave slightly differently, however, they are *almost* the same process. A way of formalizing this is to consider continuous time variants of both FP and BR, where the learning method is defined by a differential equation rather than an iterative formula. One can explicitly compute the gradients of the two processes and see that they are the same up to a time transformation.

Stochastic fictitious play and Smooth best response

Stochastic fictitious play (SFP) [17] and smooth best response (SBR) dynamics [23] are variants of FP and BR that allow for the possibility of direct convergence to mixed equilibria. Like FP and BR, there are few rules for determining whether they will converge in a given game.

1.6.2 Learning in iterated potential games

Chapter 3 will focus on extending two learning methods for learning in iterated potential games into the stochastic game setting. These are joint strategy fictitious play (JSFP) with inertia [38] and log-linear learning (LLL) [84, 39]. The details of both methods will be explained in Chapter 3, but we will summarize the main characteristics of each below.

Joint strategy fictitious play with inertia

Requirements: JSFP with inertia requires that all players be able to compute their own utility function. JSFP also requires that players observe the sequence of opponent actions. All players update their strategy at every timestep.

Convergence Results: JSFP is guaranteed to converge with probability one to a pure strategy Nash equilibrium in potential games. This strategy need not maximize the potential function.

Log linear learning

Requirements: LLL requires that all players be able to compute their own utility function. LLL also requires that players observe the sequence of opponent actions. Only one player updates their strategy at each timestep.

Convergence Results: LLL is a perturbed markov process, so it describes a family of Markov processes $\mathcal{P}^\epsilon$ controlled by a single parameter ϵ. For a fixed member of the family with ϵ sufficiently close to zero, the process is guaranteed to converge to near-potential maximizing joint actions in potential games. This means that the process selects for the specific subset of Nash equilibria that maximize the potential function.

1.7 Potential Games

The formal study of potential games began with [46], although the use of potential theory for studying certain types of games was initiated decades prior by Rosenthal [57]. Potential games can be viewed as a natural extension of team games. In a team game, all players possess the same utility function. In potential games, players may have different utility functions, but these utility functions are all linked together by a shared *potential function*. Potential games possess a wide number of applications in economics and engineering [9, 10, 32, 37, 59, 63, 68, 60].

Potential games are a class of normal-form games that admit pure Nash equilibria, with several learning methods that are guaranteed to converge to Nash equilibria. Their equilibrium sets are also interesting because they admit natural refinements of Nash equilibria.

mydef]thmA finite potential game is an finite n-player game $G = (\mathcal{A}^i, u^i)$ together with a function (called a *potential function*)

$$\varphi : \mathcal{A} \longrightarrow \mathbb{R}$$

such that for any joint strategy $x = (x^1, \ldots, x^n)$ and player strategy $y^i \in \mathcal{A}^i$,

$$\varphi(x\backslash y^i) - \varphi(x) = u^i(x\backslash y^i) - u^i(x)$$

This equation says that when a player unilaterally changes the joint strategy (that is, they change their strategy and all other players maintain the same strategy), the change in that player's utility is equal to the change in the potential function. This means that potential functions are not unique: if φ is a potential for the game G then clearly so is $\varphi + c$ where c is any constant. It's also not hard to see that this completely characterizes the set of potential functions that exist for a potential game G.

We can view the existence of a potential function as a graph-theoretic condition on the graph of joint strategies as follows. Let G be a finite game, and let Γ be a weighted directed graph whose vertices correspond to the set of joint strategies in G. This graph will have edges between strategies $a \to b$ if and only if a and b differ by exactly one player strategy. That is, $b = a\backslash b^i$ for some player i (this also means that there is an edge $b \to a$). The weight on edge $a \to b$ is $u^i(b) - u^i(a)$, and the weight on edge $b \to a$ is $u^i(a) - u^i(b)$. The *distance* of a directed path $\mathcal{P}$ in Γ is the sum of weights along the directed edges that make up $\mathcal{P}$.

prop]thmA game G is a potential game if and only if for all pure joint strategies a_1 and a_2, all paths between a_1 and a_2 have the same distance.

In fact, any pair of paths $\mathcal{P}_1$ and $\mathcal{P}_2$ between a_1 and a_2 can be decomposed as a sum of simple commutative squares in Γ. So, one needs only check local commutativity conditions to guarantee the existence of a potential function:

cor]thm[46] Let G be a normal-form game. Then G is a potential game if and only if for any pure joint stragegy a, players i and j, and pure player strategies $b^i \in \mathcal{A}^i$ and $b^j \in \mathcal{A}^j$

$$u^i(a\backslash b^i b^j) - u^i(a\backslash b^j) + u^j(a\backslash b^j) - u^j(a) = u^j(a\backslash b^i b^j) - u^j(a\backslash b^i) + u^i(a\backslash b^i) - u^i(a)$$

This corollary will play an important role in much of the work generalizing potential games to stochastic games.

1.7.1 Examples of potential games

The following is a sampling of some two and three player potential games with discussion on their equilibrium sets.

Let's start with the simplest possible potential game. The "trivial" game

$$\begin{array}{c|cc} & a & b \\ \hline a & 0,0 & 0,0 \\ b & 0,0 & 0,0 \end{array}$$

(1.21)

is a potential game with trivial potential function $\varphi \equiv 0$. Any strategies in this game will constitute a Nash equilibrium, which includes both pure and mixed strategies.

A game may have nonzero utilities but still admit a trivial potential function. Consider the following game:

$$\begin{array}{c|cc} & a & b \\ \hline a & 0,0 & 1,0 \\ b & 0,1 & 1,1 \end{array}$$

(1.22)

In this game, utilities are not trivial, however, players are unable to influence their own utilities. That is, the action of player 1 determines player 2's utility and vice versa. This game still has a trivial potential function $\varphi \equiv 0$, and all strategies are in Nash equilibrium.

One more complete example of a potential games is:

$$\begin{array}{c c} & \begin{array}{cc} \text{a} & \text{b} \end{array} \\ \begin{array}{c} \text{a} \\ \text{b} \end{array} & \begin{pmatrix} 1,1 & 0,1 \\ 1,0 & 1,1 \end{pmatrix} \end{array}$$

(1.23)

This game is interesting from the perspective of its pure Nash equilibria. It is a potential game with potential function:

$$\varphi(a,a) = 0$$
$$\varphi(a,b) = 0$$
$$\varphi(b,a) = 0$$
$$\varphi(b,b) = 1$$

The potential maximizing pure joint strategy $[b, b]$ is a Nash equilibrium, and so is the pure strategy $[a, a]$. The other two pure strategies are not Nash equilibria. It is also interesting to note that the two equilibria have identitical utilities but different potential function values.

Any *team game* G is a potential game. A team game is a game where there is a single utility function $u = u^i$ that is shared for all players. In this case, G is a potential game with potential function $\varphi = u$.

Here is one more important class of potential games. Consider a system of n interacting players, by which we mean a set of n players, where each player i has a strategy set $\mathcal{A}^i$. This is not yet a game because we have not specified the utility functions. Now suppose we are given a *welfare function*

$$W : \mathcal{A} \to \mathbb{R} \tag{1.24}$$

which is meant to be some global measure of the "goodness" of a joint strategy. From this welfare function we may derive the *Wonderful Life Utility* (WLU) [80]. Let a_0 be some baseline fixed joint strategy and define

$$u^i(a) = W(a) - W(a \backslash a_0^i) \tag{1.25}$$

A game with these utilities will be a potential game, and the welfare function W is a potential function for this game.

1.7.2 Equilibrium sets in potential games

In the examples above we saw several potential games with different equilibrium set characteristics. An important unifying fact about potential game equilibria is

Theorem 1.7.1. *[46] Let Γ be a potential game with potential function φ. Any pure joint strategy x that maximizes φ is a Nash equilibrium.*

There of course is always such a joint strategy, and therefore potential games are guaranteed to admit at least one pure Nash equilibrium. As we saw in one of the preceding examples, this theorem cannot be extended to an if and only if statement, that is, there can be pure Nash equilibria that do not maximize the potential function.

1.7.3 Generalizations of potential games

While the present work only makes use of the strict definition of potential games, it is worth describing drawbacks of this definition as well as some extensions of potential games that overcome such drawbacks. These more general versions of potential games may serve as the bedrock of future research into “nice” classes of stochastic games.

First, consider strategic situations where player utilities are aligned but operate at different scales. A contrived but concrete example would be a team game where players are using different units to measure their utility. This can be represented by a modified version of the standard 2-player coordination game:

$$\begin{array}{ccc} & \text{a} & \text{b} \\ \text{a} & 1,2 & 0,0 \\ \text{b} & 0,0 & 1,2 \end{array}$$

(1.26)

It is to both players' advantages to coordinate, and this closely resembles a team game, however it does not admit a potential function. This game is however a *weighted potential game*:

mydef]thmA finite weighted potential game is an finite n-player game $\Gamma = (\mathcal{A}^i, u^i)$ together with a function (called a *potential function*)

$$\varphi : \mathcal{A} \longrightarrow \mathbb{R}$$

along with player-specific weights $w_1, \ldots, w_n$ such that for any joint strategy $x = (x^1, \ldots, x^n)$ and player strategy $y^i \in \mathcal{A}^i$,

$$\varphi(x\backslash y^i) - \varphi(x) = w_i \left[u^i(x\backslash y^i) - u^i(x)\right]$$

In our example, we can make the game a weighted potential game with $w_1 = 2$ and $w_2 = 1$.

Weighted potential games are able to handle player utilities that are at different *multiplicative* scales. An even more general game is an *ordinal potential game*, where utility difference and potential function difference merely have the same sign:

mydef]thmA finite ordinal potential game is an finite n-player game $\Gamma = (\mathcal{A}^i, u^i)$ together with a function (called a *potential function*)

$$\varphi : \mathcal{A} \longrightarrow \mathbb{R}$$

such that for any joint strategy $x = (x^1, \ldots, x^n)$ and player strategy $y^i \in \mathcal{A}^i$, $\varphi(x\backslash y^i) - \varphi(x) > 0$ if and only if $u^i(x\backslash y^i) - u^i(x) > 0$

A *generalized ordinal potential game* only has one side of the implication, that is, $u^i(x\backslash y^i) - u^i(x) > 0 \implies \varphi(x\backslash y^i) - \varphi(x) > 0$.

1.8 Stochastic Games

In this section we will introduce the definition of a stochastic game. A stochastic game is like an MDP, except instead of a single agent interacting with an environment there are multiple agents. To more closely match game theoretic language, we will call these agents *players*.

mydef]thmAn n-player stochastic game G consists of a state space $\mathcal{S}$, state-dependent action sets $\mathcal{A}^i_s$ for each player, and games $\{G_s\}_{s\in\mathcal{S}}$ with utilities $u^i_s(a)$, called "stage games", and transition probabilities $P_{ss'}(x)$ for each state pair $s, s' \in \mathcal{S}$ and joint strategy x. The players begin the game by entering a state determined by an initial probability distribution μ_0. The game G progresses by agents simultaneously selecting actions $a^i_s \in \mathcal{A}^i$ in state s. Player i receives utility $u^i_s(a_s = \{a^1_s, \dots, a^n_s\})$ and the state transitions to s' according to the transition probability distribution.

In this work we will always assume that $|\mathcal{S}|$ is finite. We will also assume that the action sets are finite and shared across all states, ie $\mathcal{A}^i_s = \mathcal{A}^i_{s'}$ for all states s, s'. In this case we will simply refer to the action set as $\mathcal{A}^i$.

The stochastic game framework lies at the intersection of game theory and control theory. By introducing a state space and transition dynamics it extends the normal-form game from game theory. Specifically, a normal-form game can be viewed as a stochastic game with one state and trivial transition dynamics. On the other hand, it extends the Markov decision process (MDP) framework to the multi-agent setting. An MDP is a stochastic game where $n = 1$.

It will also be useful to restrict the form of stochastic games for later analysis in the paper. For this purpose we introduce the following definition:

mydef]thmA stochastic game G is said to be "fixed finite-time" If the underlying MDP (that is, considering the game as an MDP with 1 agent representing all players) is a fixed finite-time MDP.

A different way of stating this is that there exists a time T and a partition of the state space $\mathcal{S} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_T$ such that for $s_i \in \mathcal{S}_i$ and $s_j \in \mathcal{S}_j$ we have

$$P_{s_i s_j} = 0$$

if $j \neq i + 1$.

In normal-form games, each player chooses a player strategy. In stochastic games, a player must choose a strategy in every state. We refer to such a choice as a *behavior*, and denote it by $\pi^i \in \times_{s \in \mathcal{S}} \mathcal{A}^i$. The specific strategy employed in state s will be denoted by π^i_s. A *joint behavior* is a collection of behaviors for all players $\pi = (\pi^i, \ldots, \pi^n)$. It is useful to introduce notation for joint behaviors that differ by one or two players' behaviors. Let τ^i be a behavior for player i. We denote the joint behavior $(\pi^1, \ldots, \pi^{i-1}, \tau^i, \pi^{i+1}, \ldots \pi^n)$ by $\pi \backslash \tau^i$. Similarly, if we change two players behaviors, we denote the joint strategy by $\pi \backslash \tau^i \tau^j$. Finally, we will often modify a joint behavior π by changing a player's strategy in a single state s from pure strategy a^i_s to b^i_s. We will denote the modified joint behavior by $\pi \backslash b^i_s$ or $\pi \backslash b^i$ when the particular state is clear.

In normal-form games, players wish to maximize their utilities. In stochastic games, players wish to maximize their returns. A return is simply the sum of utilities from the stage games the player encounters, and so it is analogous to the notion of return in MDPs. In analogy with the reinforcement learning literature, we let $V^i_\pi(s)$ denote the expected return for player i starting from state s when players employ the joint behavior π. We let $Q^i_\pi(s, a)$ denote the expected return for player i starting from state s when they first take action a in state s while other players play π_s, and then all players play π in subsequent states.

As in normal form games, stochastic games have a notion of Nash equilibrium. A joint behavior is in Nash equilibrium if no individual player can increase their expected return by changing their strategy. All stochastic games admit at least one Nash equilibrium, although this fact is nontrivial [14, 65].

At this point we have introduced three distinct models of environment and agent (player) interaction: MDPs, normal form games, and stochastic games. The differences in language between the three models are summarized in 1.6.

MDP	normal form game	stochastic game
agent	player	player
reward	utility	utility (in stage game)
return	–	return
–	(player) strategy	(player) behavior
policy	–	joint behavior
action	joint strategy	joint strategy (in stage game)
optimal policy	Nash equilibrium	Nash equilibrium
deterministic	pure	pure

Figure 1.6: Language differences in MDPs, normal form games, and stochastic games.

1.9 Learning in Stochastic Games

The stochastic game framework is a very flexible model for studying learning because stochastic games encompass and extend two already large bodies of problems: the Markov decision process framework of single agent learning and the normal form game framework of strategic multi-player interaction. Researchers at the turn of the century realized the possibility of approaching game theoretic problems with reinforcement learning solutions [34, 25, 7]. However, it became clear that one could not simply use existing reinforcement learning methods in stochastic games and expect any type of convergence to equilibrium [6] despite earlier claims to the contrary [11]. Ultimately this period of research produced a few convergent "reinforcement style" algorithms in stochastic games [7, 34, 25, 35], however their applicability was limited. For example, some apply only in zero sum games, 2-person 2 action games, or require specific conditions apply throughout the learning process.

The question of how to learn in stochastic games remained somewhat dormant from 2005 until 2015. With the incredible success of combining reinforcement learning and neural networks, some researchers began taking this empirical success and applying it to multi-agent environments. In fact, one of the monumental successes of DeepRL, Alphago [66, 67], was a result of applying DeepRL in a zero sum game, *not* a stationary single-agent environment. More concretely multi-agent DeepRL papers include [15, 55, 3, 83, 70, 52, 73, 36]. These works are primarily empirical, but inspired by older, theoretical work in game theory and reinforcement learning. For example, counterfactual policy gradients

[15] are closely related to the wonderful life utility in collective intelligence [80], and mean field Q-learning [83] is informed by the study of mean field games [30].

There has also been some recent theoretical work on learning on stochastic games [1, 2]. These two works study learning in stochastic zero-sum games [1] and stochastic team games [2]. The work in chapters 2 and 3 most naturally belong in this family of work, as we define two other classes of stochastic games and perform a theoretical study of learning methods in these games.

Chapter 2

Stochastic Extensions of Potential Games

2.1 Introduction

In its early years, game theory focused on the static concepts of equilibria in normal-form games, proving their existence, and computing their behavior. From the period of the 1980's until the early 2000's there was a shift in research focus towards *learning* equilibria. This was motivated both by descriptive applications in economics as well as prescriptive applications in engineering. This led to a more in-depth study of simple learning rules such as best response, fictitious play, log-linear learning, and joint strategy fictitious play. A number of papers presented convergence results for special subclasses of games. Two types of games that are of particular interest are zero-sum games [48] and potential games [46]. Both classes of games admit pure Nash equilibria, and furthermore a number of the simple learning rules listed above converge to Nash equilibria in these types of games.

While a great deal of work has been done on learning to play equilibria strategies in normal-form games, the same cannot be said for stochastic games. The defining feature of stochastic games is that they introduce state dynamics, with each state corresponding to a normal-form "stage" game. Stochastic games are of interest both from a theoretical and practical point of view. They are the natural generalization of Markov decision processes

(MDPs) to the multiagent setting. As such they extend the modelling capacity of MDPs, and can be used to model markets, bargaining, and routing problems for example.

In this chapter we will study two classes of stochastic games that are motivated by their connections to normal form potential games. They are *stochastic potential games* (SPGs) and *stochastic global potential games* (SGPGs). We will study the properties of their equilibrium sets, transition dynamics, and stage games. In the next chapter we will follow this up by considering learning methods in these games.

2.2 Definitions

In this section we will present the two basic types of stochastic games that will be studied in the next two chapters: SPGs and SGPGs. Before stating these definitions we will give some motivation for each of them.

Motivation for SPGs

In [1] the authors study the convergence properties of best response dynamics in *zero sum* stochastic games. For this purpose they study *continuation games*. Given a stochastic game G with state space $\mathcal{S}$ and a set of continuation vectors $z^i = \{z^i_s\}_{s\in\mathcal{S}}$, one for each player i, one can define the following (normal form, *not* stochastic) continuation games $G_s(z)$:

$$u^i_s(a) + \sum_{s'\in\mathcal{S}} P_{ss'}(a) z^i_{s'} \tag{2.1}$$

These define normal form games for every state s. This equation is closely related to the Bellman equation in MDPs. Specifically, if we choose the continuation vectors to be the value functions $V^i_\pi(s')$ for a joint behavior π, then this equation is exactly the recursive definition of the value function. The point of defining a continuation game is that it can capture both short term (utility) and long term (continuation payoff / value) considerations in a single normal form game.

One of the learning processes - “stopping time best response dynamics” - defined in [1], makes use of a two phase learning process. In the first phase, at learning step t, players are presented with zero sum continuation games at each state, defined by continuation payoffs $z_s^i(t)$. Players converge to minimax behavior (or more accurately, nearly converge) $\tilde{\pi}_s$ in all of these continuation games using normal best response dynamics on each game. In the second phase, the continuation value $z_s^i(t+1)$ becomes the minimax value of the continuation game from the first phase. This algorithm, and specifically phase one, a priori relies on the fact that each time a new continuation game is defined, that game will be zero sum. This is a somewhat special property of zero sum games: if every stage game is a zero sum game, then automatically each continuation game one encounters will also be zero sum.

By comparison, suppose we have a stochastic game where every stage game is a potential game. In this case, one cannot expect every continuation game to be a potential game. To force this to be true, we must also make assumptions about the transition probabilities in the game. It will be these transition probabilities that define SPGs.

Motivation for SGPGs

One traditional motivation for the study of potential games is that the potential function may be a kind of "system-wide" welfare function. As discussed in Chapter 1, if one starts with a welfare function, one can always construct a potential game where the welfare function is its potential function. From this perspective, the subset of *potential maximizing* equilibria in potential games are important, since they are stable behaviors that maximize the potential function. In SPGs, all continuation games will be potential games, but we will give examples of SPGs that have no notion of a "global potential", that is, a potential function whose inputs are player behaviors. SGPGs will be defined by the existence of such a global potential. As a result, SGPGs will have a notion of equilibrium that is completely analogous to potential maximizing equilibria in potential games.

2.2.1 Definition of stochastic potential games

mydef]thmWe say that a stochastic game G has *modular dynamics* if, for any two players i and j, any joint action $a \in \times_i \mathcal{A}^i$, and any states $s, s' \in \mathcal{S}$:

$$P_{ss'}(a) + P_{ss'}(a \backslash b^i b^j) = P_{ss'}(a \backslash b^i) + P_{ss'}(a \backslash b^j) \tag{2.2}$$

mydef]thmWe say that a stochastic game G is a stochastic potential game if

Definition 2.2.0. 1. Every stage game G_s is a potential game and

2. G has modular dynamics

We would like to show that the two properties of SPGs guarantee that every continuation game is a potential game. We will make use of the following lemma from [46]:

lem]thmLet G and H be two potential games with potential functions φ and ψ respectively. The games $G+H$ and $G-H$, defined by adding and subtracting payoffs respectively, are potential games with potential functions $\varphi + \psi$ and $\varphi - \psi$ respectively.

Theorem 2.2.1. *Consider a stochastic game G. Every continuation game is a potential game if and only if G is a stochastic potential game.*

Proof. Suppose G is a stochastic potential game. Fix z, a collection of continuation vectors for each player and consider the continuation game $G_s(z)$ with payoffs

$$u^i_s(a) + \sum_{s' in \mathcal{S}} P_{ss'}(z^i_{s'})$$

The stage game G_s is a potential game since Γ is a stochastic potential game, and so by the lemma it suffices to show that the game $D_s(z) = G_s(z) - G_s$ is a potential game. This game has utilities

$$\sum_{s' \in \mathcal{S}} P_{ss'}(a) z^i_{s'}$$

We will refer to $D_s(z)$ as the "delay game".

By , it suffices to check that for any players i and j, any joint action a, and any player actions $b^i \in \mathcal{A}^i$ and $b^j \in \mathcal{A}^j$ we have

$$\begin{aligned}
&\sum_{s'} P_{ss'}(a \backslash b^i b^j) z^i_{s'} - \sum_{s'} P_{ss'}(a \backslash b^j) z^i_{s'} + \sum_{s'} P_{ss'}(a \backslash b^j) z^j_{s'} - \sum_{s'} P_{ss'}(a) z^j_{s'} \\
=& \sum_{s'} P_{ss'}(a \backslash b^i b^j) z^i_{s'} - \sum_{s'} P_{ss'}(a \backslash b^i) z^j_{s'} + \sum_{s'} P_{ss'}(a \backslash b^i) z^i_{s'} - \sum_{s'} P_{ss'}(a) z^i_{s'}
\end{aligned}$$

which can be rearranged to

$$\begin{aligned}&\sum_{s'}\left[P_{ss'}(a)+P_{ss'}(a\backslash b^i b^j)-P_{ss'}(a\backslash b^i)-P_{ss'}(a\backslash b^j)\right] z^i_{s'}\\ =&\sum_{s'}\left[P_{ss'}(a)+P_{ss'}(a\backslash b^i b^j)-P_{ss'}(a\backslash b^i)-P_{ss'}(a\backslash b^j)\right] z^j_{s'}\end{aligned} \tag{2.3}$$

And, since, the dynamics are modular, all terms in both the left-hand and right-hand sum are zero. Therefore the continuation game is a potential game.

Now, assume every continuation game is a potential game. Then in particular this is true when z is identically zero, and therefore the stage games of G are potential games. Finally we need to show that this implies G has modular dynamics.

Fix a players i and j, states s and s', joint strategy a and player strategies b^i and b^j. Let z^j be the zero vector, and z^i be zero everywhere except at state s'. Then equation **??** reduces to

$$P_{ss'}(a)+P_{ss'}(a\backslash b^i b^j)-P_{ss'}(a\backslash b^i)-P_{ss'}(a\backslash b^j)=0$$

Hence G has modular dynamics.

□

2.2.2 Stochastic Global Potential Games

Notice that while our definition of SPGs places a potential at each state, there is not necessarily a unified potential function defined over the stochastic game. A different approach to extending potential games is to draw the analogy as follows: that the definition of a normal form potential game uses a potential function over strategies, and so a stochastic version should define a potential function over *behaviors*. We will call this a *stochastic global potential game* (SGPG).

mydef]thmA stochastic global potential game is a stochastic game Γ together with a *global potential function*:

$$\Phi : \mathcal{S} \times \Pi^1 \times \cdots \Pi^n \to \mathbb{R}$$

such that for every state s, joint behavior π, and player i with behavior τ^i,

$$\Phi(s, \pi \backslash \tau^i) - \Phi(s, \pi) = V^i_{\pi \backslash \tau^i}(s) - V^i_\pi(s)$$

Stochastic global potential games are amenable to reinforcement learning methods because they admit acyclic "strict better reply graphs". This essentially means that if players sequentially make improvements to their own behaviors, their joint behavior will eventually constitute a Nash equilibrium. However, determining whether a stochastic game is an SGPG is not straightforward. A priori, there is no way to check whether a global potential function exists other than to construct one, and the temporally extended nature of the global potential function may make this difficult. In contrast, SPGs are characterized by temporally local conditions on stage games and one-step transition probabilities.

2.3 Stochastic Potential Games

In this section we aim to answer three questions. First, is the class of SPGs distinct from SGPGs? In other words, are there really SPGs that do not have global potential functions? The answer is yes, and we will provide a simple example in this section. Second, to what extent are SPGs applicable in the real world? We will answer this question by examining modular dynamics further, and leveraging the equivalence between potential games and congestion games to examine a "realistic" subclass of SPGs, which we call *routing/construction games*. Finally, we will discuss some properties of the set of Nash equilibrium in SPGs.

Let's begin with an example of an SPG that has no global potential function.

eg]thmConsider a 2-player stochastic game $\mathcal{G}$ with 3 states: s_1, s_2, and s_3. We will call the corresponding stage games G_1, G_2, and G_3 for ease of notation. Episodes of the game always start in s_1. Both players have two actions, a and b, in each state. The stage games in s_1 and s_3 are trivial, that is, all actions yield zero utility for both players. The stage game in s_2 is given by the following bimatrix:

$$\begin{array}{c c} & \begin{array}{cc} a & b \end{array} \\ \begin{array}{c} a \\ b \end{array} & \begin{pmatrix} 4,2 & 2,2 \\ 2,1 & 2,3 \end{pmatrix} \end{array}$$

Finally, the transition probabilities are

$$P_{12} = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

and then necessarily

$$P_{13} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 0 \end{pmatrix}$$

It is straightforward to see that these transition probabilities are modular. G_1 and G_3 are trivially potential games with potential function $\varphi \equiv 0$. G_2 is a potential game with potential function:

$$\begin{array}{c c} & \begin{array}{cc} a & b \end{array} \\ \begin{array}{c} a \\ b \end{array} & \begin{pmatrix} 0 & 0 \\ -2 & 0 \end{pmatrix} \end{array}$$

Since all stage games are potential games and the dynamics are modular, $\mathcal{G}$ is an SPG. Now, let

$$\begin{aligned} \pi^1 &= \pi^2 = [a, a, a] \\ \tilde{\pi}^1 &= [b, a, a] \\ \tilde{\pi}^2 &= [a, b, a] \end{aligned}$$

Here we use the notation $[x, y, z]$ as shorthand for the deterministic player behavior:

Example 2.3.0. "Take action x in state s_1, action y in state s_2, and action z in state s_3."

It is a simple exercise to calculate the value difference of the following joint behaviors:

$$\begin{aligned}
V^1_{s_1}(\tilde{\pi}^1, \pi^2) - V^1_{s_1}(\pi^1, \pi^2) &= 2 \\
V^2_{s_1}(\tilde{\pi}^1, \tilde{\pi}^2) - V^2_{s_1}(\tilde{\pi}^1, \pi^2) &= 1 \\
V^2_{s_1}(\pi^1, \tilde{\pi}^2) - V^2_{s_1}(\pi^1, \pi^2) &= 1 \\
V^1_{s_1}(\tilde{\pi}^1, \tilde{\pi}^2) - V^1_{s_1}(\pi^1, \tilde{\pi}^2) &= 1
\end{aligned}$$

Because these returns do not commute (ie $2 + 1 \neq 1 + 1$) there cannot exist a global potential function. In the next section we will describe all of the constraints associated with SGPGs and write them down explicitly. At that point, we will be able to more clearly identify why the above example has no global potential function. For now we simply point out that in the above example the commutativity issue arose when we changed the players' actions at different states, namely s_1 and s_2. In particular, these states occur at different layers in the finite MDP over joint actions.

2.3.1 Modular dynamics with a view towards applications

In the previous subsection we confirmed that SPGs are a distinct class of stochastic games, and that they may not have global potential functions. In this section we will explore the definition of SPGs and characterize modular dynamics in a way that suggests possible real-world applications. We will end this section by identifying a subclass of SPGs, which we call *routing/construction games* which may be of interest in engineering and economics applications.

Recall that a stochastic game G has modular dynamics if for all state pairs s and s', joint action a, players i and j with alternative actions b^i and b^j, the following equation holds:

$$P_{ss'}(a) + P_{ss'}(a\backslash b^i b^j) = P_{ss'}(a\backslash b^i) + P_{ss'}(a\backslash b^j) \tag{2.4}$$

This equation defines the transition probabilities of SPGs in terms of constraints. The following sequence of results transforms this into a constructive characterization of transition probabilities.

If $\mathcal{G}$ has modular dynamics then for any player i, states s and s', joint actions a and b with $a^i = b^i$, and player action c^i,

$$P_{s'}(a) - P_{ss'}(a\backslash c^i) = P_{s'}(b) - P_{ss'}(b\backslash c^i)$$

Proof. We know this is true when a^{-i} and b^{-i} differ by a single player's action since modular dynamics tells us that

$$P_{ss'}(a) - P_{ss'}(a\backslash b^i) = P_{ss'}(a\backslash b^j) - P_{ss'}(a\backslash b^i b^j)$$

For general a and b with $a^i = b^i$, we simply chain together single action changes, maintaining equality throughout the process, ie

$$\begin{aligned}
P_{ss'}(a) - P_{ss'}(a\backslash c^i) &= P_{ss'}(a\backslash b^1) - P_{ss'}(a\backslash c^i b^1) \\
&= P_{ss'}(a\backslash b^1 b^2) - P_{ss'}(a\backslash c^i b^1 b^2) \\
&\vdots \\
&= P_{ss'}(a\backslash b^1 \cdots b^{i-1} b^i b^n) - P_{ss'}(a\backslash b^1 \cdots b^{i-1} b^{i+1} b^n c^i) \\
&= P_{ss'}(b) - P_{ss'}(b\backslash c^i)
\end{aligned}$$

□

lem]thmIf $\mathcal{G}$ is a stochastic game with modular dynamics then for any pair of states s and s', there exists a nonnegative *constant $c_{ss'}$ and* nonnegative *functions*

$$f^i_{ss'} : \mathcal{A}^i \to \mathbb{R}$$

for each player such that

$$P_{ss'}(a) = c_{ss'} + \sum_{i=1}^{n} f^i_{ss'}(a^i)$$

Proof. Fix player i and joint action b such that

$$b^i \in \text{argmin}_{b^i} P_{ss'}(b^i, b^{-i})$$

Define the following functions

$$\begin{aligned} f^i_{ss'}(a^i) &:= P_{ss'}(a^i, b^{-i}) - P_{ss'}(b^i, b^{-i}) \\ g^i_{ss'}(a^{-i}) &:= P_{ss'}(b^i, a^{-i}) \end{aligned}$$

Notice that by construction

$$P_{ss'}(a^i, a^{-i}) = f^i_{ss'}(a^i) + g^i_{ss'}(a^{-i})$$

Since we can do this for all players, we can decompose $P_{ss'}(a)$ as

$$P_{ss'}(a) = c_{ss'} + \sum_{i=1}^{n} f^i_{ss'}(a^i)$$

Furthermore, since $b^i \in \text{argmin}_{b^i} P_{ss'}(b^i, b^{-i})$, all of the functions $f^i_{ss'}$ are nonnegative, and reach their minimum value of 0 when $a^i = b^i$. Note that there are possibly other minima.

Finally, we need to show that $c_{ss'}$ is nonnegative. But, based on our previous statement, we can certainly choose a joint action c such that all of the functions $f^i_{ss'}$ are simultaneously zero. Then

$$P_{ss'}(c) = c_{ss'}$$

□

Theorem 2.3.1. *Let $\mathcal{G}$ be a stochastic game. Then $\mathcal{G}$ has modular dynamics if and only if for every state s there exist state dependent player weights w^i_s, action-conditional probability distributions over next states $\{p^i_{ss'}(a^i)\}$, and "natural weights" $w^0_{ss'}$, such that*

$$P_{ss'}(a) = w^0_{ss'} + \sum_i w^i_s p^i_{ss'}(a^i)$$

Proof. Suppose $\mathcal{G}$ has modular dynamics. Then by the previous lemma we have the decomposition

$$P_{ss'}(a) = c_{ss'} + \sum_{i=1}^{n} f^i_{ss'}(a^i)$$

where $c_{ss'}$ and $f^i_{ss'}(a^i)$ are all nonnegative. Since $P_{ss'}(a)$ is a probability distribution over next states s',

$$\sum_{s'} \sum_{i=1}^{n} f^i_{ss'}(a^i) = \sum_{s'} P_{ss'}(a) = 1$$

Suppose we fix a player j and alter their action to b^j while keeping all other actions fixed. Then

$$\sum_{s'} \sum_{i=1}^{n} f^i_{ss'}(a^i) = 1 = \sum_{s'} f^j_{ss'}(b^j) + \sum_{s'} \sum_{i=1, i\neq j}^{n} f^i_{ss'}(a^i)$$

The terms not related to player j cancel on both sides leaving

$$\sum_{s'} f^j_{ss'}(a^j) = \sum_{s'} f^j_{ss'}(b^j)$$

This equality holds for any pair of player j actions and so it defines an action independent quantity $w^j_s := \sum_{s'} f^j_{ss'}(a^j)$. For a given action b^j let

$$p_{ss'}(b^j) = \frac{f^j_{ss'}(b^j)}{w^j_s}$$

This defines a probability distribution over next states s' that depends on the current state s and player action b^j. Letting $w^0_{ss'} = c_{ss'}$ yields

$$P_{ss'}(a) = w^0_{ss'} + \sum_{i} w^i_s p^i_{ss'}(a^i)$$

Now we need to show the reverse implication. Suppose G has transition probabilities that can be decomposed as

$$P_{ss'}(a) = w^0_{ss'} + \sum_i w^i_s p^i_{ss'}(a^i)$$

Let b^i and b^j be alternative actions for players i and j. Then

$$P_{ss'}(a) - P_{ss'}(a\backslash b^i) = w^i_s p^i_{ss'}(a^i) - w^i_s p^i_{ss'}(b^i)$$

and similarly

$$P_{ss'}(a\backslash b^j) - P_{ss'}(a\backslash b^i b^j) = w^i_s p^i_{ss'}(a^i) - w^i_s p^i_{ss'}(b^i)$$

So

$$\begin{aligned} P_{ss'}(a) - P_{ss'}(a\backslash b^i) &= P_{ss'}(a\backslash b^j) - P_{ss'}(a\backslash b^i b^j) \\ P_{ss'}(a\backslash b^j) + P_{ss'}(a\backslash b^j) &= P_{ss'}(a) + P_{ss'}(a\backslash b^i b^j) \end{aligned}$$

Therefore $\mathcal{G}$ has modular dynamics. □

The above theorem allows us to describe the mechanics of an SPG $\mathcal{G}$ in plain language as follows. In a give state, players are endowed with "voting shares" specified by the w^i_s's. The various actions that a player i can take determine how that player's voting share is distributed across next states. A priori, player i does not have full control in distributing w^i_s across next states, but rather is constrained by their action set. For example, suppose that the action set $\mathcal{A}^i_s$ of a layer k state s exactly corresponds to $\mathcal{S}_{k+1}$, the set of layer $k+1$ states. So an element of $\mathcal{A}^i_s$ can be identified with a next state s'. Furthermore, suppose that $p^i_{ss'}(s') = 1$. In this kind of game, players can only distribute their voting share unilaterally on a single next state and cannot "manage risk" by placing voting shares on multiple next states.

In addition to each player having a voting share, nature is also endowed with a state-dependent voting share $w^0_s = \sum_{s'} w^0_{ss'}$. In the extreme case $w^0_s = 1$, in which case players have *no* control on transition probabilities and should therefore act myopically in state s.

As a first example, let's reexamine . There are two state dependent weights

$$w_1^1 = 0.5$$
$$w_1^2 = 0.5$$

The natural weights are all zero. Furthermore the probability distributions are simply

$$p_{12}^i(a) = 0$$
$$p_{13}^i(a) = 1$$

and

$$p_{12}^i(b) = 1$$
$$p_{13}^i(b) = 0$$

For both players $i = 1$ and $i = 2$. This is an example not only of an SPG in general, but of the special case noted in the paragraph above. Namely, each player has two actions that correspond exactly with the number of next states. Action a casts a vote for state s_3 and action b casts a vote for s_2. Both players have equal voting shares, and the natural weights are zero.

2.3.2 Routing/construction games

In the pioneering paper on potential games [46], Monderer and Shapley show that the class of potential games is equivalent to the class of *congestion games*. They do so through explicitly constructing potential functions for arbitrary congestion games, and conversely constructing a congestion game out of an arbitrary potential function. The link between congestion games and potential functions can be traced further back to the work of Rosenthal [57] where congestion games are defined and the existence of pure Nash equilibrium is shown through the construction of a potential function.

A congestion game arises when a set of *homogeneous* players have to select from a set of resources, and where the payoff of a player depends on the number of players selecting each resource [46]. Congestion games form a natural model for traffic routing, both in the context of automobile traffic [78] and internet packet traffic [18]. Below is a precise definition of a congestion game:

mydef]thmA (finite) congestion game consists of the following data:

Definition 2.3.1.
1. A (finite) set of n players
2. A (finite) set of resources M
3. For each player i a (finite) set of actions $\mathcal{A}^i$ where each action $a \in \mathcal{A}^i$ corresponds to a subset of M.
4. For each resource $m \in M$ a delay function $d_m : \mathbb{N} \rightarrow \mathbb{R}$

For a joint action a and resource m let $C_m(a)$ be the number of player actions that contain resource m. The payoff of player i is defined to be

$$u^i(a) = \sum_{m \in a^i} d_m(C_m(a)) \tag{2.5}$$

Routing games form a special subclass of congestion games. A routing game is a congestion game where the resources M consist of edges in a fixed graph Γ and the actions of player i consist of paths in the graph that all have the same starting and ending location. The iterated version of this problem models the strategic aspect of morning commutes. Each day drivers must decide amongst multiple routes between their home and workplace. These routes have different trip times that are dependent on the number of other players utilizing the roadways.

Figure 2.1 shows a simple two player congestion game. There are two players, and they each need to travel from vertex A to vertex B. The edges are labelled with pairs of numbers (a, b) where a is the delay cost when one player utilizes the edge and b is the delay cost when two players utilize the edge. There are two pure Nash equilibria in this game. Either player one takes the upper route and player two takes the lower route or vice versa.

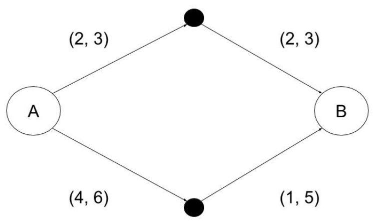

Figure 2.1: A simple routing game.

Connecting routing games and SPGs

Recall that the first property of SPGs is that each stage game is a potential game. Since every potential game is equivalent to a congestion game, we can instead think of the SPG as being made up of a collection of congestion games with modular transition probabilities. A priori there may be no "physical" relationship between the stage congestion games in an SPG. Below we describe *routing/construction games* which are a subset of SPGs in which the stage congestion games (specifically, routing games) are connected in a physical way.

In the traditional routing game story, one imagines a network of roads, and a set of commuters with various start and end locations. Each road has an associated delay function which relates the number of commuters on a road to the expected time it will take a given commuter to traverse that road. This can depend on a number of road characteristics such as speed limit, lanes, and road quality. It is the "job" of commuters (players) to select how to *route* themselves.

We are going to modify this story by adding in a new actor which we will call "the government". At each timestep, the government observes the strategy of commuters and elects to take on *construction* projects to fix roads, add lanes, etc. based which roads the commuters chose at the last timestep. These choices are reflected in modifications to the road delay functions. It is natural to assume that the government is more likely to fix roads that are used by a large number of drivers, and this is where the "voting" interpretation of modular dynamics is employed. In order to fit the SPG model to this situation, the government must modify the roads probabilistically. We provide the full definition of routing/construction games below.

mydef]thmAn n-player routing/construction game consists of a directed graph $\Gamma = (V, E)$. The graph has an initial set of delay functions

$$d_e^0 : \{1, \ldots, n\} \to \mathbb{R}$$

for each $e \in E$ which defines an initial stage routing game. It is also equipped with "construction" functions

$$c_e(d_e, b) : \mathbb{R}^n \times \{0, 1\} \to \mathbb{R}^n$$

which describe how the delay functions change from state to state. At state s, let d_e^s denote the delay function on edge e. From this state, the government will be able to strengthen one edge, and all other roads will decay. Suppose players select joint action a. This action defines a probability distribution over *edges* as follows. Each player has identical voting share $w_s^i = \frac{1}{n}$. If player i's action is made up of a subset of edges $e_1^i, \ldots, e_r^i$, then their corresponding vote is a uniform distribution over these edges. A sample is drawn from this distribution. For the selected edge, the new delay function is

$$d_e^{s'} = c_e(d_e^s, 1)$$

and for all other edges f, the new delay function is

$$d_f^{s'} = c_f(d_f^s, 0)$$

The game $\mathcal{G}$ proceeds for a fixed number of layers k and then terminates. Intuitively, construction on an edge should *reduce* the values of the delay function while decay should *increase* values of the delay function, so that

$$d_e^{s'} = c_e(d_e^s, 1) \le d_e^s \le d_e^{s'} = c_e(d_e^s, 0)$$

By construction, routing/construction games are SPGs since we have described their transition dynamics in terms of state and player dependent weights and action conditional probability distributions.

2.3.3 Equilibria

The set of Nash equilibria in a potential games has two immediately interesting features. First, every potential games admits at least one *pure* Nash equilibrium. Second, potential games admit further specialized equilibria which are called "potential maximizing equilibria". These are joint behaviors that maximize the potential function. In some applications, the potential function is interpreted as a welfare function, and one desires to find learning algorithms that converge to potential maximizing behavior. We say that a learning algorithm has "equilibrium selection properties" if it is guaranteed to converge to a certain subset of Nash equilibria. So, an algorithm that converges to the set of potential maximizing equilibria in a potential game has an equilibrium selection property. Log linear learning [39, 84] is an example of such a learning algorithm, and we will discuss it further in the next chapter.

mydef]thmLet G be an SPG. A joint behavior π is called a *simultaneously potential maximizing* if, for each state s, the joint behavior π_s maximizes the potential function that has utilities

$$u^i_s(a) + \sum_{s'} P_{ss'} V^i_\pi(s') \tag{2.6}$$

The three main results of this section are

Theorem 2.3.2. *Every stochastic potential game admits a pure Nash equilibrium.*

Theorem 2.3.3. *Every simultaneously potential maximizing equilibrium is a Nash equilibrium.*

Theorem 2.3.4. *Every stochastic potential game admits a simultaneously potential maximizing equilibrium.*

We will prove all of these results together.

Proof. Let G be an n-player stochastic potential game. As usual denote the state layers by $\mathcal{S}_1, \dots, \mathcal{S}_T$, the stage potential games by G_s with potential φ_s and utilities u^i_s.

We will construct a pure simultaneously potential maximizing Nash equilibrium π by a backwards iterative method through the layers of G.

First, for each state $s \in \mathcal{S}_T$ let a_s be a pure potential maximizing Nash equilibrium for the stage game G_s. Set $\pi_s := a_s$ and then $V^i_\pi(s) = u^i_s(a_s)$. Note that we haven't fully defined π yet, but it is okay to talk about $V^i_\pi(s)$ for states $s \in S_T$ since we have described the behavior of π at layer T.

Next, for each state $s \in \mathcal{S}_{T-1}$ consider the continuation game $G_s(V^i_\pi)$ with utilities

$$u^i_s(a) + \sum_{s' \in \mathcal{S}_T} P_{ss'}(a) V^i_\pi(s')$$

Since G is a stochastic potential game, each $G_s(V^i_\pi)$ is a potential game and hence admits a pure potential maximizing Nash equilibrium a_s. For the states in $\mathcal{S}_{T-1}$ set $\pi_s := a_s$ and then $V^i_\pi(s) = u^i_s(a_s) + \sum_{s' \in \mathcal{S}_T} P_{ss'}(a_s) V^i_{s'}$.

Iterating this process, for each state $s \in S_j$ consider the continuation game $G_s(V^i_\pi)$ with utilities

$$u^i_s(a) + \sum_{s' \in \mathcal{S}_T} P_{ss'}(a) V^i_\pi(s')$$

This game admits a pure Nash equilibrium a_s. We set $\pi_s := a_s$ for all $s \in \mathcal{S}_j$ and $V^i_\pi = u^i_s(a_s) + \sum_{s' \in \mathcal{S}_T} P_{ss'}(a_s) V^i_\pi(s')$.

Eventually this process terminates, at which point it defines a pure joint strategy in every state, that is, a pure joint behavior π. By construction it is clear that π is simultaneously potential maximizing. Finally, we need to show that π is a Nash equilibrium.

Fix a player i whose behavior under π is π^i and consider an alternative behavior τ^i. Let t_0 be the last layer where π^i and τ^i differ. That is, the largest t_0 such that there exists $s_0 \in \mathcal{S}_{t_0}$ with$\pi^i_{s_0} \neq \tau^i_{s_0}$. Then the behaviors π^i and $\pi \backslash \tau^i$ correspond to strategies in the game $G_{s_0}(z(t_0 + 1))$. But, since we know that π_s is a Nash equilibrium for the game $G_{s_0}(V^i_\pi)$, it must be the case that

$$u^i_s(\pi_s) + \sum_{s' \in \mathcal{S}_{t_0+1}} P_{ss'}(\pi_s) V^i_\pi(s') \geq u^i_s(\pi_s \backslash \tau^i) + \sum_{s' \in \mathcal{S}_{t_0+1}} P_{ss'}(\pi_s \backslash \tau^i) V^i_\pi(s')$$

Therefore π is a Nash equilibrium.

□

2.4 Stochastic Global Potential Games

In the previous section we found that not all SPGs are SGPGs. Is the opposite true? That is, does the existence of a global potential imply stage games are potential games and that transition probabilities are modular? We will see that the existence of a global potential function is in some ways more restrictive than the assumption of modular dynamics, although it does not imply modular dynamics. In this section we will attempt to characterize the structure of SGPGs by looking at the constraints created by a global potential function. We will begin by deriving a constraint for SGPGs that resemble modular dynamics. Then, we will examine additional restrictions on SGPGs. We will conclude this section by discussing equilibrium properties of SGPGs.

The following is a class of games that are SGPGs but not SPGs:

eg]thmLet G be any stochastic team game (ie all players receive the same utility in every state) with non-modular dynamics. By definition it cannot be an SPG. However, it admits a global potential function:

$$\Phi(s, \pi) = V_{\pi}^{i}(s)$$

Note that the right hand side is independent of the choice of player i since all players receive the same utilities and hence have the same V functions.

2.4.1 A gap between SGPGs and SPGs

Now that we've established that SPGs and SGPGs are distinct classes of stochastic games, we'll examine their relationship in more detail.

The definition of SPG is derived from a consistency equation based on the desire for continuation games to be potential games. The same kind of consistency equation can be derived from a global potential in SGPGs.

Let Φ be the global potential function for a SGPG G.

Consider a state s, a joint action a, a joint pure behavior π with $\pi_s = a$, players i and j, and $b^i \in \mathcal{A}^i$ and $b^j \in \mathcal{A}^j$ alternative actions for players i and j in state s. The global potential function must satisfy

$$\Phi(s, \pi \backslash b^i b^j) - \Phi(s, \pi \backslash b^i) + \Phi(s, \pi \backslash b^i) - \Phi(s, \pi) = \Phi(s, \pi \backslash b^i b^j) - \Phi(s, \pi \backslash b^i) + \Phi(s, \pi \backslash b^i) - \Phi(s, \pi)$$

We can replace these with Q-values, and write the Q-values using the one step Bellman equation:

$$\begin{aligned}
&\left[u^j(a \backslash b^i b^j) + \sum_{s'} P_{ss'}(a \backslash b^i b^j) V_\pi^j\right] && - \left[u^j(a \backslash b^i) + \sum_{s'} P_{ss'}(a \backslash b^i) V_\pi^j\right] \\
+&\left[u^i(a \backslash b^i) + \sum_{s'} P_{ss'}(a \backslash b^i) V_\pi^i\right] && - \left[u^i(a) + \sum_{s'} P_{ss'}(a) V_\pi^i\right] \\
=&\left[u^i(a \backslash b^i b^j) + \sum_{s'} P_{ss'}(a \backslash b^i b^j) V_\pi^i\right] && - \left[u^i(a \backslash b^j) + \sum_{s'} P_{ss'}(a \backslash b^j) V_\pi^i\right] \\
+&\left[u^j(a \backslash b^j) + \sum_{s'} P_{ss'}(a \backslash b^j) V_\pi^j\right] && - \left[u^j(a) + \sum_{s'} P_{ss'}(a) V_\pi^j\right]
\end{aligned}$$

Rearranging terms and simplifying yields

$$\begin{aligned}
&\left[u^j(a \backslash b^i b^j) - u^j(a \backslash b^i) + u^i(a \backslash b^i) - u^i(a)\right] - \left[u^i(a \backslash b^i b^j) - u^i(a \backslash b^j) + u^j(a \backslash b^j) - u^j(a)\right] \\
&= \sum_{s' \in \mathcal{S}_{k+1}} \left[P_{ss'}(a \backslash b^i b^j) - P_{ss'}(a \backslash b^i) - P_{ss'}(a \backslash b^j) + P_{ss'}(a)\right] \left(V_\pi^i(s') - V_\pi^j(s')\right)
\end{aligned}$$

There is a lot to unpack in this equation. First, the left hand side of this equation is the consistency equation on the stage game G_s. That is, if the right hand side of the equation is zero, one can conclude that the G_s is a potential game. Note that only the right hand side of this equation depends on π. So, if there is any choice of π where the right hand size is zero, then G_s will be a potential game. In contrast, suppose we start with the assumption that G_s is a potential game, so that the left hand side of the equation is zero. In this case we have

$$\begin{aligned}
&\sum_{s' \in \mathcal{S}_{k+1}} \left[P_{ss'}(a\backslash b^i b^j) - P_{ss'}(a\backslash b^i) - P_{ss'}(a\backslash b^j) + P_{ss'}(a)\right] V^i_\pi(s') \\
= &\sum_{s' \in \mathcal{S}_{k+1}} \left[P_{ss'}(a\backslash b^i b^j) - P_{ss'}(a\backslash b^i) - P_{ss'}(a\backslash b^j) + P_{ss'}(a)\right] V^j_\pi(s')
\end{aligned}$$

This equation resembles equation **??** except that continuation vectors z^i are replaced by expected rewards $V^i_\pi(s)$. When we had the freedom to choose z^i arbitrarily, we could construct them such that the only way to satisfy the equations was to have modular dynamics. In contrast, the $V^i_\pi(s)$ are given as part of the stochastic game, and their differences $V^i_\pi(s') - V^j_\pi(s')$ may span a comparatively low-dimensional subspace of an $|\mathcal{S}|$-dimensional space. This is particularly clear in the case of stochastic team games, where $V^i_\pi \equiv V^j_\pi$ for all i and j so that their difference spans a zero-dimensional space.

2.4.2 Further constraints on SGPGs

In the preceding analysis, we made use of consistency equations 2.7 for SGPGs in which the changes in player i and player j's strategies occur in the same state s. However, if we vary the states that the strategy changes occur, we can derive a much larger set of consistency equations that restrict the form of SGPGs even further.

Suppose alternative actions b^i and b^j occur in adjacent layers, that is, in states $s_1 \in \mathcal{S}_k$ and $s_2 \in \mathcal{S}_{k+1}$ respectively. Fix a baseline joint pure behavior π, and for notational simplicity let $a_1 = \pi_{s_1}$ and $a_2 = \pi_{s_2}$. As usual, we start from the consistency equation

$$\begin{aligned}
&\Phi(s_1, \pi\backslash b^i b^j) - \Phi(s_1, \pi\backslash b^i) + \Phi(s_1, \pi\backslash b^i) - \Phi(s_1, \pi) \\
&= \Phi(s_1, \pi\backslash b^i b^j) - \Phi(s_1, \pi\backslash b^j) + \Phi(s_1, \pi\backslash b^j) - \Phi(s_1, \pi)
\end{aligned}$$

Replacing each difference with a difference in values, and then simplifying yields:

$$\Phi(s_1, \pi\backslash b^i b^j) - \Phi(s_1, \pi\backslash b^i) =$$
$$u^j_{s_1}(a_1\backslash b^i) + \sum_{s'\in\mathcal{S}_{k+1}} P_{s_1 s'}(a_1\backslash b^i)V^j_{\pi\backslash b^i b^j}(s') - u^j_{s_1}(a_1\backslash b^i) - \sum_{s'\in\mathcal{S}_{k+1}} P_{s_1 s'}(a_1\backslash b^i)V^j_{\pi\backslash b^i b^j}(s')$$

Notice that changing actions in earlier layer states has no effect on the value of later states so

$$V^j_{\pi\backslash b^i b^j}(s') = V^j_{\pi\backslash b^j}(s')$$

Furthermore, when $s' \neq s_2$,

$$V^j_{\pi\backslash b^j}(s') = V^j_{\pi}(s')$$

So most terms above cancel, leaving

$$\Phi(s_1, \pi\backslash b^i b^j) - \Phi(s_1, \pi\backslash b^i) = P_{s_1 s_2}(a_1\backslash b^i)\left[Q^j_\pi(s_2, a_2\backslash b^j) - Q^j_\pi(s_2, a_2)\right]$$

Following a similar process for the other differences in the consistency equation we get

$$\Phi(s_1, \pi\backslash b^i) - \Phi(s_1, \pi) = u^i_{s_1}(a_1\backslash b^i) - u^i_{s_1}(a) + \left[P_{s_1 s_2}(a_1\backslash b^i) - P_{s_1 s_2}(a)\right] V^i_\pi(s_2)$$

$$\Phi(s_1, \pi\backslash b^i b^j) - \Phi(s_1, \pi\backslash b^j) = u^i_{s_1}(a_1\backslash b^i) - u^i_{s_1}(a) + \left[P_{s_1 s_2}(a_1\backslash b^i) - P_{s_1 s_2}(a)\right] Q^i_\pi(s_2, a_2\backslash b^j)$$

$$\Phi(s_1, \pi\backslash b^j) - \Phi(s_1, \pi) = P_{s_1 s_2}(a_1)\left[Q^j_\pi(s_2, a_2\backslash b^j) - V^j_\pi(s_2)\right]$$

Substituting these into the original consistency equation and rearranging terms results in

$$\begin{aligned}&\left[P_{s_1 s_2}(a) - P_{s_1 s_2}(a\backslash b^i)\right]\left(Q^i_\pi(s_2, a_2\backslash b^j) - Q^i_\pi(s_2, a_2)\right)\\ &= \left[P_{s_1 s_2}(a) - P_{s_1 s_2}(a\backslash b^i)\right]\left(Q^j_\pi(s_2, a_2\backslash b^j) - Q^j_\pi(s_2, a_2)\right)\end{aligned}$$

For a fixed player i, this implies one of two possibilities. Either

Statement 1:

$$P_{s_1s_2}(a) - P_{s_1s_2}(a\backslash b^i) = 0 \tag{2.7}$$

for all choices of a_1, b^i, and s_1, or

Statement 2:

$$Q^i_\pi(s_2, a_2\backslash b^j) - Q^i_\pi(s_2, a_2) = Q^j_\pi(s_2, a_2\backslash b^j) - Q^j_\pi(s_2, a_2) \tag{2.8}$$

For all choices of j, a_2, b^j, and π.

If Statement 1 is false for every player i, then $Q^i_\pi = Q^j_\pi$ for all players, which would mean that G is a team game. If Statement 1 is true for some player i it means that player i has no control with respect to transitioning to state s_2. This condition is quite restrictive.

2.4.3 Equilibria

The existence of a single potential function makes SGPGs more closely analogous to normal form potential games. In potential games, potential maximizing strategies are examples of Nash equilibria. Similarly, in SGPGs, potential maximizing behaviors are examples of Nash equilibria. This is obvious, as any change in a single players behavior from a potential maximizing joint behavior will negatively impact that players returns. Hence:

Theorem 2.4.1. *Every SGPG admits at least one pure Nash equilibrium.*

Chapter 3

Learning Methods in Stochastic Potential Games and Stochastic Global Potential Games

3.1 Introduction

In the previous chapter we defined two classes of stochastic games: stochastic potential games (SPGs) and stochastic global potential games (SGPGs). Via their analogies with normal form potential games, these two classes of games admit extra structure on their Nash equilibrium sets. Namely, both SPGs and SGPGs admit pure Nash equilibrium behaviors. Furthermore, each admits a distinguished class of Nash equilibrium. In the case of SPGs we called these "simultaneously potential maximizing equilibria." SGPGs on the other hand admit "potential maximizing equilibria," so named because they follow essentially the same definition as potential maximizing equilibria in normal form potential games.

The structure of their Nash equilibrium sets immediately raise the question of whether there are convergent learning methods that converge to Nash equilibria, pure Nash equilibria, or the specialized Nash equilibria in each class of game. This chapter will focus on two learning methods: joint strategy fictitious play (JSFP) with inertia and log-linear learning (LLL). These two learning methods converge to pure Nash equilibria and potential maximizing equilibria respectively in potential games. For each of these learning methods we will examine how they may be extended to SPGs and SGPGs in order to define convergent learning dynamics in both.

3.2 Joint Strategy Fictitious play with Inertia in Stochastic Games

3.2.1 Background on JSFP for repeated games

In standard Fictitious play (FP), agents learn by selecting best responses to the empirical past play of their opponents. If a_t is the joint action played at stage t, then a^i_{t+1} is obtained as a best response

$$a^i_{t+1} \in \mathrm{BR}(x^{-i}_t)$$

to the (typically mixed) opponent actions x^j_t where

$$x^j_t = \frac{1}{t} \sum_{k=1}^{t} a^j_k$$

Here we identify an action a^i_k with the discrete probability distribution with all probability concentrated on a^i_k. A different way of describing the choice of a^i_{t+1} is that it is an action which maximizes $u^i(\cdot, x^{-i}_t)$ where u^i is the utility function of player i.

The essential difference between FP and JSFP is that, from the perspective of player i, JSFP treats all other opponents as if they were a single entity. This manifests in the way that player i calculates empirical frequencies of opponent play. JSFP will calculate a^i_{t+1} as a best response

$$a^i_{t+1} \in \mathrm{BR}(z^{-i}_t)$$

to the empirical joint opponent play

$$z_t^{-i} = \frac{1}{t}\sum_{k=1}^{t} a_k^{-i}$$

As in FP, the action choice a_{t+1}^i can also be viewed as maximizing $u^i(\cdot, z_t^{-i})$. On initial inspection, it seems that JSFP has a very costly memory constraint compared to FP since each player needs to maintain empirical joint strategy counts. However, it is in fact not necessary for players to keep track of z_t^{-i}, but instead each player can instead track and maximize $u^i(\cdot, z_t^{-i})$ directly. By linearity,

$$u^i(y, z_t^{-i}) = \frac{1}{t}\sum_{k=1}^{t} u^i(y, a_t^{-i})$$

At each timestep, this quantity can be updated recursively:

$$u^i(y, z_{t+1}^{-i}) = \frac{1}{t+1}u^i(y, a_{t+1}^{-i}) + \frac{t}{t+1}u^i(y, z_t^{-i})$$

At time t, player i can now calculate $\text{argmax}_{y\in\mathcal{A}} u^i(y, z_t^{-i})$ and therefore can perform JSFP without storing the empirical joint distribution of opponent play.

JSFP with Inertia In [38] the authors modify JSFP using an inertial condition to guarantee convergence to pure Nash equilibrium. We will now describe JSFP with inertia and then state the main theorem of [38].

At time t player i plays according to the following rule:

1. If a_{t-1}^i is a best response to z_t^{-i}, then $a_t^i = a_{t-1}^i$. That is, player i repeats their action.
2. Otherwise, player i will repeat their action with probability $\alpha^i(t)$ or will select a different action according to the probability distribution $\beta_i(t)$ with probability $1-\alpha_t^i$. $\beta_i(t)$ can be any distribution that is supported on a subset of the best responses. We assume that $\alpha^i(t)$ is bounded away from 0 and 1.

mydef]thmA player is indifferent between two strategies a_1 and a_2 if there exists a joint opponent play a^{-i} such that

$$u^i(a_1, a^{-i}) = u^i(a_2, a^{-i})$$

Note that the player actions do not have to yield the same utilities for all opponent plays;

one will suffice.

Theorem 3.2.1. *[38] In any finite generalized ordinal potential game in which no players are indifferent between two strategies the joint actions a_t generated by JSFP with inertia will converge to a pure Nash equilibrium almost surely.*

3.2.2 Extension to Finite-time SPGs

The almost-sure nature of convergence allows us to make a straightforward extension of the JSFP with inertia algorithm to the finite-time SPG setting. This establishes that there exists a decoupled learning method that converges to a pure Nash equilibrium in SPGs.

Before we get into the algorithmic details we first need be clear about the learning setting: At each time t, each player i must choose an action $a^i_s(t)$ for every state $s \in \mathcal{S}$. This is a bit different from the sequential decision making problem, where at each timestep the agent is only faced with a decision in a single state, that is typical when working with MDPs. The way we have formulated the learning problem allows for a more stylized argument for convergent dynamics, however there is no significant barrier to rephrasing it as a sequential decision making algorithm.

Let G be an SPG. We let $z_s(t)$ denote the empirical joint play in state s at time t, and $z^{-i}_s(t)$ denote the empirical joint play of opponents against player i in state s at time t. We begin by defining an extension of JSFP to stochastic games and after this define the full algorithm which employs inertia.

Stochastic Game JSFP (SG-JSFP) At each timestep, players will update two state-dependent quantities: their behavior $a^i_s(t)$ and continuation payoffs $c^i_s(t)$. At time t, player i computes a best response $a^i_s(t+1)$ to z^{-i}_s in the game $G_s(c^i(t))$ which has utilities

$$u^i_s(a) + \sum_{s' \in \mathcal{S}} P_{ss'}(a) c^i_{s'}(t)$$

For ease of notation we will simply refer to $G_s(c^i(t))$ as $G_s(t)$. Once the actions $a_s(t+1)$ are computed, the continuation payoff is updated to be the utility player i receives in game $G_s(t)$ when joint action $a_s(t+1)$ is taken. Explicitly:

$$c_s^i(t+1) = u_s^i(a_s(t+1)) + \sum_{s' \in \mathcal{S}} P_{ss'}(a_s(t+1))c_{s'}^i(t)$$

As in JSFP, players can perform SG-JSFP without explicitly maintaining the empirical joint play $z_s(t)$ in every state. At each timestep player i must select an action which is a best response to $z_s^{-i}(t)$ in game $G_s(t)$. In the same way as JSFP, player i can update their expected immediate utility for playing action y against the empirical joint opponent play via

$$u_s^i(y, z_s^{-i}(t+1)) = \frac{1}{t+1}u_s^i(y, a_s^{-i}(t+1)) + \frac{t}{t+1}u_s^i(y, z_s^{-i}(t)).$$

Then player i can calculate the utility in game $G_s(t)$ of any action y according to

$$u_s^i(y, z_s^{-i}(t+1)) + \sum_{s' \in \mathcal{S}} P_{ss'}(a)c_{s'}^i(t+1)$$

Note that this requires the player to know the structure of the game. In standard JSFP, knowing the structure of the game simply means that a player can calculate any value of their utility function. In the stochastic game setting this knowledge requirement is stronger—players must also be able to calculate all transition probabilities $P_{ss'}(a)$.

SG-JSFP with inertia We now introduce SG-JSFP with inertia. At each timestep player i must select an action in each state. We wish to impose additional decision-making rules on top of SG-JSFP which guarantee convergence to pure Nash equilibrium. They are:

1. If $a_s^i(t-1)$ is a best response to z_t^{-i} in the game $G_s(t)$ then $a_s^i(t) = a_s^i(t-1)$.
2. Otherwise, player i will repeat their action in state s with probability $\alpha^i(t)$ or will select a different action according to the probability distribution $\beta_i(t)$ with probability $1 - \alpha^i(t)$. $\beta_i(t)$ can be any distribution with support on best responses and $\alpha^i(t)$ must be bounded away from 0 and 1.

The last point we must address before the theorem is "indifference to distinct strategies". This is needed to show that JSFP with inertia converges to a pure Nash equilibrium in (repeated) potential games. We will need a similar kind of indifference to guarantee our theorem.

mydef]thmWe say that a stage game $G_s(c)$ with continuation payoff c is indifferent to distinct strategies of one can find two joint plays a_1 and a_2 differing only in player i's behavior such that the utility of player i is the same. That is,

$$u_s^i(a_1) + \sum_{s'} P_{ss'}(a_1)c_{s'}^i = u_s^i(a_1) + \sum_{s'} P_{ss'}(a_1)c_{s'}^i$$

mydef]thmWe say that a continuation payoff c (indexed by states and players) is "grounded" or "grounded in reality" if it is equal to a value function of some joint behavior. That is, there exists a joint behavior π such that

$$c_s^i = V_\pi^i(s)$$

mydef]thmWe say that a stochastic game Γ is "indifferent to distinct strategies of grounded continuations" if $G_s(c)$ is indifferent to distinct strategies whenever c is grounded in reality.

Theorem 3.2.2. *In any finite-time SPG that is indifferent to distinct strategies of grounded continuations, the behaviors $a^i(t)$ converge to a pure Nash equilibrium almost surely.*

Proof. This proof relies on two simple facts. First, JSFP with inertia converges to a pure Nash equilibrium in potential games. Second, for any continuation payoff c, the game $G_s(c)$ is a potential game.

To show almost sure convergence, we can show that with probability one there is a finite time T such that a_T consititutes a pure Nash equilibrium in the stochastic game. Once such an action is played, the inertial condition (1) guarantees that it will continue to be played forever.

Let L be the number of layers/times that elapse in the game before an episode ends. Consider states s that arise in the final layer/time of the game. The games played in these states are each potential games, and are never modified with a continuation payoff since the episode ends after an action is taken in any of these states. Hence $G_s(t) = G_s$ for all times. It is clear that SG-JSFP with inertia reduces simply to JSFP with inertia in these states, and therefore by 3.2.1, with probability one there is a finite time $T_1(s)$ for each state in this layer such that the action profiles constitute a Nash equlibrium. By the inertial condition (1), since the stage game $G_s(t)$ is not changing, players will play the same action in state s forever once it is played once. Since the number of states is finite, this means that with probability one there is a time T_1 where all action profiles in all layer L states are simultaneously Nash equilibria.

Now consider states in layer $L - 1$. Initially, the stage games $G_s(t)$ are changing as the payoffs in layer L change. However, after time T_1, these continuation payoffs stabilize to a constant value, and therefore $G_s(t) = G_s(t')$ for times $t, t' > T_1$. As a result, eventually SG-JSFP with inertia is identical with JSFP with inertia in the repeated games $G_s(T_1)$. Hence, since we are in an SPG and all continuation games are potential games, behavior in each layer-$(L - 1)$ state s converges to a pure Nash equilibrium of $G_s(T_1)$ with probability one in finite time $T_2(s)$. Again, since the number of states is finite, with probability one there is a time T_2 such that all layer-$(L - 1)$ states are playing Nash equilibria to their continuation games.

We may repeat this argument moving backwards step-by-step through all layers of the stochastic game and conclude that with probability one, there is a finite time T_L such that for all $t > T_L$ and states s, $a_s(t)$ is a Nash equilibrium for the game $G_s(T_L)$. Furthermore, $G_s(T_L)$ is the "true" continuation game. That is, the continuation payoff of player i for the game $G_s(T_L)$ is $V_a^i(s)$. Therefore a_t is a pure Nash equilibrium for the stochastic game.

□

3.2.3 SG-JSFP with inertia for SGPGs

Let's return for a moment to the proof of theorem 3.2.2 and consider the learning process in an intermediate layer k. In terms of learning, nothing that happens in the first T_{L-k} steps has any bearing on what behaviors eventually converge to in this layer. In particular, only two facts were important when establishing convergence to pure Nash equilibrium in this layer:

1. Eventually the behavior in layer $k + 1$ converged to a fixed pure Nash equilibrium, and therefore the continuation payoffs $c(t)$ also converged.
2. The final continuation games $G_s(t) = G_s(c(T_{L-k}))$ are potential games

In order to guarantee the second point, we assumed that our stochastic game was an SPG. This guarantees that *any* continuation game is a potential game, and in particular, $G_s(c(T_1))$ is a potential game. However, these final continuation games are special: by construction their continuation payoff is equal to the true expected return ie

$$c_s^i(T_{L-k}) = V_\pi^i(s),$$

where π is a joint pure behavior where $\pi_s^i = a_s^i(T_{L-k})$ when s is in a layer greater than k, and π_s^i is arbitrary for any earlier layers.

But, as we showed in section 2.4, such continuation games will be potential games for SGPGs. Hence, the argument for SPGs will also work for SGPGs and we can conclude the following theorem:

Theorem 3.2.3. *In any finite-time SGPG that is indifferent to distinct strategies of grounded continuations, the behaviors $a^i(t)$ converge to a pure Nash equilibrium almost surely.*

3.2.4 Conclusion on SG-JSFP with inertia

The benefits of SG-JSFP is that it offers a convergent decoupled learning algorithm that converges to a pure Nash equilibrium in both SPGs and SGPGs. Furthermore, this algorithm has a small memory requirement which is similar to the memory requirement in regular JSFP (with inertia). One significant drawback of SG-JSFP with inertia is that it requires players to know the "structure of the game", in that they must be able to compute arbitrary stage game utility functions *and* transition probabilities.

3.3 Log-linear Learning

The goal of this section is to define a learning algorithm for stochastic games that resembles the log-linear algorithm used in iterated normal form games. We will begin by reviewing log-linear learning (LLL) and stating the main convergence results in iterated games. We will then describe a variant that converges to potential maximizing equilibrium in SGPGs. Finally we will discuss the challenge of extending LLL to SPGs.

3.3.1 Log-linear Learning in Repeated Normal-form Games

Log-linear learning is a learning algorithm that provably converges to pure Nash equilibria in potential games. That is, if players repeatedly play the same normal-form game, adjusting their strategies at each timestep according to LLL, then their play will approach a pure Nash equilibrium in probability. In fact, one can say even more: LLL will converge to *potential–maximizing* pure Nash equilibria. This is referred to as an "equilibrium-selection property" in that the algorithm converges to a particular type of Nash equilibrium. In the context of potential games, the potential function often characterizes some notion of social/global utility, and so it can be valuable to have an algorithm that will converge to such behavior.

LLL proceeds as follows. At each timestep t, players simultaneously select actions $a^i \in \mathcal{A}^i$ and receive utility $u^i(a)$. The players select these actions according to probability distributions $x^i \in \mathcal{A}^i$ over their action sets. The players update these distributions as follows. At time t, one player i is selected uniformly at random and generates a distribution x^i_t. All other players will play the same action they did at the previous timestep, ie $a^j_t = a^j_{t-1}$ for players $j \neq i$. In contrast, player i will play by sampling from the distribution

$$x^i_t(a) = \frac{e^{u^i(a,a^{-i}_{t-1})}/\tau}{\sum_{b\in\mathcal{A}^i} e^{u^i(b,a^{-i}_{t-1})}/\tau},$$

where a^{-i}_{t-1} is the list of actions played at time $t-1$ by players other than player i, $x^i_t(a)$ is the probability of selecting action a under the distribution x^i_t, and τ is a *temperature parameter*. Notice that as the temperature parameter tends to zero, the probability distribution concentrates on best responses to the other players' actions at the last timestep. In [5] they show that the stationary distribution μ of joint strategies is

$$\mu(a) = \frac{e^{\varphi(a)/\tau}}{\sum_b e^{\varphi(b)/\tau}},$$

where b varies over the set of joint actions $b \in \times_i \mathcal{A}^i$.

From this explicit form of the stationary distribution it follows that as the temperature is brought to zero, the joint distribution concentrates on potential maximizing joint behaviors.

In [39] the authors provide an alternative proof method using the theory of resistance trees in Markov processes. This allows them to characterize the stochastically stable states, which are exactly the potential maximizing joint actions, without explicitly writing down the stationary distribution.

Theory of resistance trees

Let P^0 denote the probability transition matrix for a finite state Markov chain over the state space Z. We refer to P^0 as the unperturbed process. Consider a perturbed process where the size of the perturbation is indexed by a scalar $\epsilon > 0$, and let P^ϵ be the associated transition matrix. The process P^ϵ is called a regular perturbed Markov process if P^ϵ is ergodic for sufficiently small ϵ and P^ϵ approaches P^0 at an exponentially smooth rate. Specifically, the latter condition means that for all $z, z' \in Z$,

$$\lim_{\epsilon \to 0^+} P^\epsilon_{z \to z'} = P^0_{z \to z'}$$

and,

$$P^\epsilon_{z \to z'} > 0 \text{ for some } \epsilon > 0 \text{ implies } \lim_{\epsilon \to 0^+} \frac{P^\epsilon_{z \to z'}}{\epsilon^{R(z \to z')}}$$

for some nonnegative real number $R(z \to z')$, which we call the *resistance* of the transition $z \to z'$ under the perturbed process.

Consider a complete directed graph with $|Z|$ vertices for each state. The vertex corresponding to state z_j will be called j. The weight on the directed edge $i \to j$ is the resistance $R(z_i \to z_j)$. A j-tree, or "tree rooted at j", is a directed tree T such that all paths terminate at j. The resistance of T is the sum of the edge resistances composing it, and the *stochastic potential*, γ_j, of z_j is the minimum resistance among all j-trees.

Theorem 3.3.1. *[84] Let P^ϵ be a regular perturbed Markov process, and for each $\epsilon > 0$ let μ^ϵ be the unique stationary distribution of P^ϵ. Then* $\lim_{\epsilon \to 0^+} \mu_\epsilon$ *exists and the limiting distribution μ^0 is a stationary distribution of P^0. The stochastically stable states (i.e. the support of μ^0) are precisely those states with minimal stochastic potential. Furthermore, if a state is stochastically stable then the state must be in a recurrent class of the unperturbed* *process P^0.*

In [39] the authors show that log-linear learning converges to potential maximizing behavior by proving the following sequence of results:

lem]thmLog-linear learning induces a regular perturbed Markov process where the unperturbed Markov process is an asynchronous best reply process and the resistance of any feasible transition $a \rightarrow b = (b^i, a^{-i})$ is

$$R(a \rightarrow b) = \max_{a_*^i \in \mathcal{A}^i} u^i(a_*^i, a^{-i}) - u^i(b)$$

lem]thmConsider any finite n-player potential game with potential function $\varphi : \mathcal{A} \rightarrow \mathbb{R}$ where all players adhere to log-linear learning. For any feasible action path

$$\mathcal{P} = \{a^0 \rightarrow a^1 \rightarrow \cdots \rightarrow a^m\}$$

and its reverse path

$$\mathcal{P}^R = \{a^m \rightarrow a^{m-1} \rightarrow \cdots \rightarrow a^0\},$$

the difference in the total resistance across the paths is

$$R(\mathcal{P}) - R(\mathcal{P}^R) = \varphi(a^0) - \varphi(a^m).$$

Finally,

prop]thmConsider any finite n-player potential game with potential function $\varphi : \mathcal{A} \rightarrow \mathbb{R}$ where all players adhere to log-linear learning. The stochastically stable states are the set of potential maximizers, i.e., $\{a \in \mathcal{A} : \varphi(a) = \max_{a^* \in \mathcal{A}} \varphi(a^*)\}$

3.3.2 Generalization to Stochastic Games, and Convergence in Stochastic Global Potential Games

This section will generalize log-linear learning to the stochastic setting and show that, in the case of stochastic global potential games, the stochastically stable states of generalized log-linear learning are precisely those joint behaviors that maximize the global potential function.

Generalized log-linear learning proceeds as follows. At each timestep t, players select pure behaviors π^i. The players update their behavior choices as follows. At time t, a player i and a state s are selected uniformly at random. Player i will generate a distribution x^i_s. All other players select the same behavior as they did at the previous timestep, i.e. $\pi^j_t = \pi^j_{t-1}$ for players $j \neq i$. In contrast, player i will play by sampling from the distribution

$$x^i_s(a) = \frac{e^{V^i_{\pi_{t-1}\backslash a}(s_0)/\tau}}{\sum_{b\in\mathcal{A}^i} e^{V^i_{\pi_{t-1}\backslash b}(s_0)/\tau}},$$

where π_{t-1} is the joint behavior played at time $t-1$ and s_0 is the initial state of the stochastic game. $x^i_s(a)$ is the probability of selecting action a in state s under the distribution x^i_s, and τ is a *temperature parameter*.

It is important to note that this learning method is not based solely on payoff outcomes. It requires that players are able to compute their state values in order to update their strategy.

lem]thmGeneralized log-linear is a regular perturbed Markov process where the resistance of any feasible transition $\pi \longrightarrow \tau = (a^i_s, \pi^{-i})$ is

$$R(\pi \longrightarrow \tau) = \max_{a^i_*\in\mathcal{A}^i} V^i_{\pi\backslash a^i_*}(s_0) - V^i_\tau(s_0)$$

Proof. The unperturbed process behaves as follows for each timestep $t > 0$. Choose a player i and state s, both uniformly at random. Change player i's action in that timestep to an element of $a_* \in \arg\max_a V^i_{\pi_{t-1}\backslash a}(s_0)$. Keep the behavior of player i in all other states unchanged, and keep the behaviors of all other players the same.

Let $\epsilon = e^{-1/\tau}$. Under the perturbed process P^ϵ, the probability of transitioning from π to τ is

$$P^\epsilon_{\pi\rightarrow\tau} = \frac{1}{n}\frac{1}{|\mathcal{S}|}\frac{\epsilon^{-V^i_{\pi\backslash a^i_s}(s_0)}}{\sum_{b\in\mathcal{A}^i}\epsilon^{-V^i_{\pi\backslash b}(s_0)}}$$

Let $W_i(s,\pi) = \max_{a\in\mathcal{A}^i} V^i_{\pi\backslash a}(s_0)$. Multiply the numerator and denominator by $\epsilon^{W_i(s,\pi)}$ yields

$$P^{\epsilon}_{\pi \to \tau} = \frac{1}{n} \frac{1}{|\mathcal{S}|} \frac{\epsilon^{W_i(s,\pi) - V^i_{\pi \backslash a}(s_0)}}{\sum_{b \in \mathcal{A}^i} \epsilon^{W_i(s,\pi) - V^i_{\pi \backslash a}(s_0)}}$$

So then,

$$\lim_{\epsilon \to 0^+} \frac{P^{\epsilon}_{\pi \to \tau}}{\epsilon^{W_i(s,\pi) - V^i_{\pi \backslash a}(s_0)}} = \lim_{\epsilon \to 0^+} \frac{1}{n|\mathcal{S}|} \frac{1}{\sum_{b \in \mathcal{A}^i} \epsilon^{W_i(s,\pi) - V^i_{\pi \backslash a}(s_0)}}$$

The summands in the denominator will approach either 0 or 1 depending on if b is in $\arg\max V^i_{\pi \backslash a}(s_0)$. So this limit is equal to

$$\frac{1}{n|\mathcal{S}|| \arg\max V^i_{pi \backslash a}(s_0)|}$$

Therefore generalized log-linear learning is a regular perturbed Markov process with resistances

$$R(\pi \to \tau) = \max_{a^i_* \in \mathcal{A}^i} V^i_{\pi \backslash a^i_*}(s_0) - V^i_{\tau}(s_0)$$

□

In order to relate the graph-theoretic notion of stochastic potential to the stochastic game-theoretic notion of global potential we must define feasible paths and reverse paths. A feasible path is a sequence of joint behaviors $\mathcal{P} = \{\pi_0 \to \pi_1 \to \cdots \to \pi_m\}$ such that two adjacent behaviors differ by the action of a single player in a single state. The resistance $R(\mathcal{P})$ of a path is the sum of edge resistances

$$R(\mathcal{P}) = \sum_{i=1}^{n} R(\pi_{i-1} \to \pi_i)$$

The reverse path consists of the same joint behaviors, but with arrows travelling in the opposite direction. We can relate path resistances with changes in global potential with the following lemma.

lem]thmConsider any finite n-player SGPG with potential function $\Phi : \mathcal{A} \to \mathbb{R}$ *where* *all players adhere to log-linear learning. For any feasible action path*

$$\mathcal{P} = \{\pi_0 \to \pi_1 \to \cdots \to \pi_m\}$$

and its reverse path

$$\mathcal{P}^R = \{\pi_m \to \pi_{m-1} \to \cdots \to \pi_0\}$$

the difference in the total resistance across the paths is

$$R(\mathcal{P}) - R(\mathcal{P}^R) = \varphi(s_0, \pi_0) - \varphi(s_0, \pi_m)$$

Proof. Consider a single edge $\pi_k \to \pi_{k+1}$ in the path and its reverse $\pi_{k+1} \to \pi_k$. By the previous lemma we have

$$R(\pi_k \to \pi_{k+1}) = W_i(s, \pi_k) - V^i_{\pi_{k+1}}(s_0) R(\pi_{k+1} \to \pi_k) = W_i(s, \pi_{k+1}) - V^i_{\pi_k}(s_0)$$

Since π_k and π_{k+1} only differ by player i's action in state s we have that:

$$W_i(s, \pi_{k+1}) = W_i(s, \pi_k)$$

So then

$$R(\mathcal{P}) - R(\mathcal{P}^R) = V^i_{\pi_k}(s_0) - V^i_{\pi_{k+1}}(s_0) = \Phi(s_0, \pi_k) - \Phi(s_0, \pi_{k+1})$$

Adding these differences up over all pairs in the path gives the desired result. □

prop]thmConsider any finite n-player SGPG G with potential function $\Phi : \mathcal{A} \to \mathbb{R}$ *where all players adhere to log-linear learning. The stochastically stable states are the set of potential maximizers, i.e.,* $\{\pi : \Phi(\pi) = \max_{\pi_*} \Phi(\pi_*)\}$

Proof. Let π be a behavior with minimum stochastic potential, and let Γ be a π-tree that realizes this stochastic potential. Let π_* be a behavior that minimizes $\Phi(s_0, \cdot)$.

Within the graph Γ there is a path $\mathcal{P}$ beginning at π_* and ending at π. Let Γ_* be the graph obtained by removing $\mathcal{P}$ from Γ and replacing it with $\mathcal{P}^R$. So Γ_* is a tree rooted at

π_*. By the previous lemma,

$$R(\Gamma) - R(\Gamma_*) = \Phi(s_0, \pi) - \Phi(s_0, \pi_*).$$

Since π has minimum stochastic potential, the right hand side of the equation is greater than or equal to zero. Hence

$$\Phi(s_0, \pi) \leq \Phi(s_0, \pi_*)$$

and so π must also minimize $\Phi(s_0, *)$. □

3.3.3 Log-linear learning in SPGs

The first question we need to answer when addressing log-linear learning for SPGs is one of goals. In repeated potential games, log-linear learning is attractive as a method because it guarantees potential maximizing behavior. Potential maximizers are a subset of pure Nash equilibria in potential games, and therefore log-linear learning exhibits an "equilibrium selection" property, in that it selects Nash equilibria with an additional restriction (namely those that maximize potential). When we extended log-linear learning to SGPGs there was still a clear goal: define a learning algorithm whose stationary distribution is supported on Nash equilibria that maximize the global potential function. Generalized log-linear learning selects for these more restrictive Nash equilibria in SGPGs. In contrast, an SPG does not necessarily have a global potential function, and so first we need to describe the kind of equilibria we would like log-linear learning to select for in SPGs. The kind of equilibria we are interested in are *simultaneously potential–maximizing*.

mydef]thmLet G be an SPG. A joint behavior π is simultaneously potential maximizing if for all states s, the joint strategy π_s is a potential maximizing strategy in the continuation stage game $G_s(V_\pi)$.

Recall that for an SPG G, any continuation game $G_s(c)$ is a potential game. As a result we are able to talk about potential maximizing strategies for $G_s(V_\pi)$.

We've remedied one issue that the lack of global potential presents us. However, there is a deeper issue: without a global potential function we do not have a global way to characterize resistances in log-linear learning for SPGs. As a result, a "naive" attempt to use log-linear learning in SPGs will fail. We will illustrate this with an example of an SPG that does not converge under a naive log-linear learning rule. Like all of our previous discussions of SPGs, we will be in the fixed finite-time setting.

Naive log-linear learning. At each timestep t all players select actions $a_s^i(t)$ in all states simultaneously. Let $a_s(t)$ denote the joint action taken in state s at time t, and $a(t)$ denote the entire behavior at time t. At time t, a player i and state s are selected uniformly at random. Player i selects an action according to the distribution

$$pr(a^i) = \frac{e^{Q^i_{a(t-1)}(s,a^i,a_s^{-i}(t-1))/\tau}}{\sum_{b^i} e^{Q^i_{a(t-1)}(s,b^i,a_s^{-i}(t-1))/\tau}}$$

where τ is a temperature parameter. The action taken in all other states is the same as at time $t-1$, and the action taken by all other players in state s is the same as at time $t-1$. We are interested in understanding the stochastically stable states of this process. Namely we'd like to answer

1. In an SPG, are the stochastically stable states of this process simultaneously potential maximizing Nash equilibria?
2. In an SPG, are the stochastically stable states of this process Nash equilibria?

The following example will show that neither of these statements is necessarily true.

eg]thmConsider the 3 state, 2-horizon game G described as follows. G is a two player game, and in each state both players have two actions a and b. Every episode begins in state s_1 and transitions either to state s_2 or s_3 where the players take a final move and the episode ends. The stage games for s_1 and s_3 are trivial (that is, all utilities are zero) and the stage game for s_2 is defined by the bimatrix:

$$\begin{array}{cc} & \begin{array}{cc} a & b \end{array} \\ \begin{array}{c} a \\ b \end{array} & \begin{pmatrix} 100, 100 & -100, 100 \\ 100, -100 & -100, -100 \end{pmatrix} \end{array}$$

Notice that by construction, all stage games are potential games with trivial potential functions $\varphi \equiv 0$. The transition matrices are

$$P_{12} = \begin{pmatrix} 0 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

and

$$P_{13} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 0 \end{pmatrix}$$

One can describe the pure Nash equilibria of this game as follows. Both players may act arbitrarily in state s_3 and s_2 (since all actions are potential maximizing in those states). However, the action to be taken in state s_1 is determined based on the action in s_2. The equilibria are:

s_1	s_2	s_3
(a, a)	(a, a)	$(\cdot, \cdot)$
(a, b)	(a, b)	$(\cdot, \cdot)$
(b, a)	(b, a)	$(\cdot, \cdot)$
(b, b)	(b, b)	$(\cdot, \cdot)$

Where $(\cdot, \cdot)$ denotes that the players may perform any actions in s_3. Now, suppose players adhere to log-linear learning as described above. Our goal is to show that the unperturbed process may move the behavior out of Nash equilibrium. Suppose at time t the joint action is

$$a^1(t) = [a, a, a]$$
$$a^2(t) = [a, a, a]$$

With positive probability, the following will occur at time t:

Example 3.3.1.
- state 2 is selected
- player 2 is selected
- player 2 changes their action from a to b

But then the new joint action is

$$\begin{aligned} a^1(t) &= [a, a, a] \\ a^2(t) &= [a, b, a] \end{aligned}$$

which does not simultaneously maximize potentials, and is not a Nash equilibrium.

The problem that this example brings forward is that an action change in a layer k state may alter the continuation games played in layer $l < k$ states such that their behaviors are no longer Nash equilibria. This is exactly what happens in our example—asynchronous best response dynamics can alter the behavior in state s_2, which in turn changes the continuation game played at s_1 and what constitutes Nash equilibrium.

3.4 Conclusion

In this chapter we reviewed two learning methods that converge to Nash equilibrium in iterated potential games: JSFP with inertia and LLL. We were able to extend JSFP with inertia to SG-JSFP with inertia, a stochastic game learning algorithm that converges to pure Nash equilibria in both SPGs and SGPGs. We also extended LLL to an algorithm that converges to potential maximizing behavior in SGPGs. Finally, we discussed the obstruction to naively extending LLL to SPGs. This study has been far from comprehensive; with their added structure stochastic games offer a multitude of choices in designing learning algorithms. This chapter merely served as a first push into the study of learning in SPGs and SGPGs.

Chapter 4

Q-Learning Approaches to Dynamic Multi-Driver Dispatching and Repositioning

The driver management system at a ride-sharing company must make decisions both for assigning available drivers to nearby unassigned passengers (hereby called orders) over a large spatial decision-making region (e.g., a city) and for repositioning drivers who have no nearby orders. Such decisions not only have immediate to short-term impact in the form of revenue from assigned orders and driver availability, but also long-term effects on the distribution of available drivers across the city. This distribution critically affects how well future orders can be served. The need to address the exploration-exploitation dilemma as well as the delayed consequences of assignment actions makes this a Reinforcement Learning (RL) problem.

Recent works [4, 50] have successfully applied Deep Reinforcement Learning techniques to dispatching problems, such as the Traveling Salesman Problem (TSP) and the more general Vehicle Routing Problem (VRP), however they have primarily focused on *static* (i.e., those where all orders are known up front) and/or *single-driver* dispatching problems. In contrast, for use in ride-sharing applications, one needs to find good policies that can accommodate a *multi-driver* dispatching scheme where demands are not known up front but rather generated *dynamically* throughout an episode (e.g., a day). We refer to this problem as a multi-driver vehicle dispatching and repositioning problem (MDVDRP).

We define an MDVDRP as a continuous-time semi-Markov decision process with the following state-reward dynamics. At any time, state is given by a set of requesting *orders* $o_t^i \in \mathcal{O}_t$, a set of drivers *drivers* $d_t^i \in \mathcal{D}_t$, and time of day t. There is also a set of repositioning movements $m^j \in \mathcal{M}$, which are not part of state but will be part of the action space. The size of $\mathcal{O}_t$ will change in time due to orders stochastically appearing in the environment and disappearing as they are served or canceled. Orders are canceled if they do not receive a driver assignment within a given time window. $\mathcal{D}_t$ can be further subdivided into *available drivers* and *unavailable drivers*. A driver is unavailable if and only if they are currently fulfilling an order. An action is a pairing of an available driver with either a requesting order or repositioning movement. An order is characterized by a pickup location (where the passenger is located), and end location (where the customer wants to go), a price, and the amount of time since the order arrived in the system. A driver is described by her position if she is available, and her paired order or reposition movement if she is unavailable. A reposition movement is described by a direction and duration, e.g. "Travel west for three minutes". When a driver is assigned to an order, the agent receives a reward equal to the price of that order, and the driver becomes unavailable until after it has picked up its order at the start location and dropped it off at the end location. When a driver is assigned a repositioning movement, the agent receives no reward and repositions until it either reaches the maximum repositioning duration or is assigned an order. When any action is taken in the environment, time advances to the next time that there is a driver that is available and not repositioning. Note that if there are multiple non-repositioning drivers at time t and multiple requesting orders, then time will not advance after the first action is taken since there will still be at least one available non-repositioning driver and order pairing.

Heuristic solutions construct an approximation to the true problem by ignoring the spatial extent, or the temporal dynamics, or both, and solve the approximate problem exactly. One such example is myopic pickup distance minimization (MPDM), which ignores temporal dynamics and always assigns the closest available driver to a requesting order [86]. We illustrate this below in two simple dispatching domains that capture the essence of these suboptimalities, and demonstrate that our methods can overcome such issues.

In this paper we construct a global state representation along with a neural network (NN) architecture that can take this global state as input and produce action-values (Q-values). Due to the variable number of orders, which appear both as part of state and part of actions, we make use of attention mechanisms both for input and output of the NN. We then present two methods for training this network to perform dispatching and repositioning: single-driver deep Q-learning network (SD-DQN) and multi-driver deep Q-learning network (MD-DQN). Both approaches are based on the deep Q-learning network (DQN) algorithm [45], but differ from each other in the extent to which individual driver behaviors are coordinated. SD-DQN learns a Q-value function of global state for single drivers by accumulating rewards along *single-driver* trajectories. On the other hand, MD-DQN uses *system-centric* trajectories, so that the Q-value function accumulates rewards for all drivers. We find that MD-DQN can learn superior behavior policies in some cases, but that in practice SD-DQN is competitive in all environments and scales well with the number of drivers in the MDVDRP while MD-DQN performs poorly in real-data domains. Empirically we compare performance of SD-DQN and MD-DQN on a static assignment problem, illustrative MDVDRPs, and a data-driven MDVDRP designed using real world ride-sharing dispatching data.

4.1 Related Work

There have been several recent approaches to solving dispatching and routing problems. Some examples include Bello et al. ([4]), Nazari et al. ([50]), and Vinyals et al. ([76]). All of these works use an encoding-decoding scheme, where first information is processed into a global-context, and then from this context actions are decoded sequentially. Pointer networks [76] offer an approximate solution to TSPs by encoding cities (in our terminology, orders) sequentially with a recurrent network, and then producing a solution by sequentially "pointing" to orders using an attention mechanism [44]. The network is trained with a supervised loss function by imposing a fixed ordering rule on decoded orders. Bello et al.([4]) build off of the pointer network architecture, but train the network with policy gradients and so may dispense with the fixed ordering during the decoding phase.

While related to our work, the above papers typically focus on problems that are *static* and/or *single-driver* (in the sense that there is only one driver that must be controlled). In contrast, we are interested in making dispatching decisions for many drivers in a dynamic environment. Furthermore, the sequential encoding in Bello et al. and Vinyals et al. introduces an unnecessary forced ordering of inputs into the neural network by encoding using a recurrent network. Instead we follow an architecture more closely related to Nazari et al.([50]), which uses an attention mechanism [44] for encoding, however they only apply their architecture to static and single-driver vehicle routing problems. We depart from their architecture further by dispensing with the recurrent network used for decoding. Instead, we construct a global representation of state that is encoded and decoded in a completely feed-forward fashion.

Next we discuss a previous work on a problem more closely related to MDVDRPs. Oda et al. [51] offer a *value-based* approach to the dynamic fleet management problem, which is a strict subproblem of the MDVDRP. In dynamic fleet management, one is concerned only with *repositioning* available drivers, while driver-order assignments are handled via hard-coded rules (in their case, minimizing pickup distance). A MDVDRP combines fleet management with driver-order matching.

Another thread of related work comes from the multi-agent reinforcement learning literature. Specifically, our two training approaches, SD-DQN and MD-DQN, are analogous to the multi-agent reinforcement learning (MARL) algorithms of "independent Q-learning" [11] and "team-Q learning" [35] respectively. The tradeoffs between these two approaches to team problems are broadly known but not well understood. Team-Q has the benefit of theoretical justification, but its action set grows exponentially in the number of agents. On the other hand, while mostly lacking in theory, independent Q-learning has been leveraged successfully in a number of problems [12, 51]. Claus and Boutilier ([11]) conjecture that independent Q-learners will converge to a Nash equilibrium under appropriate assumptions in team games, however this claim remains unproven. Our results reflect the theme that SD-DQN works well in practice at a variety of scales in team problems despite the dearth of convergence results, while MD-DQN has considerable difficulty scaling beyond a small number of agents.

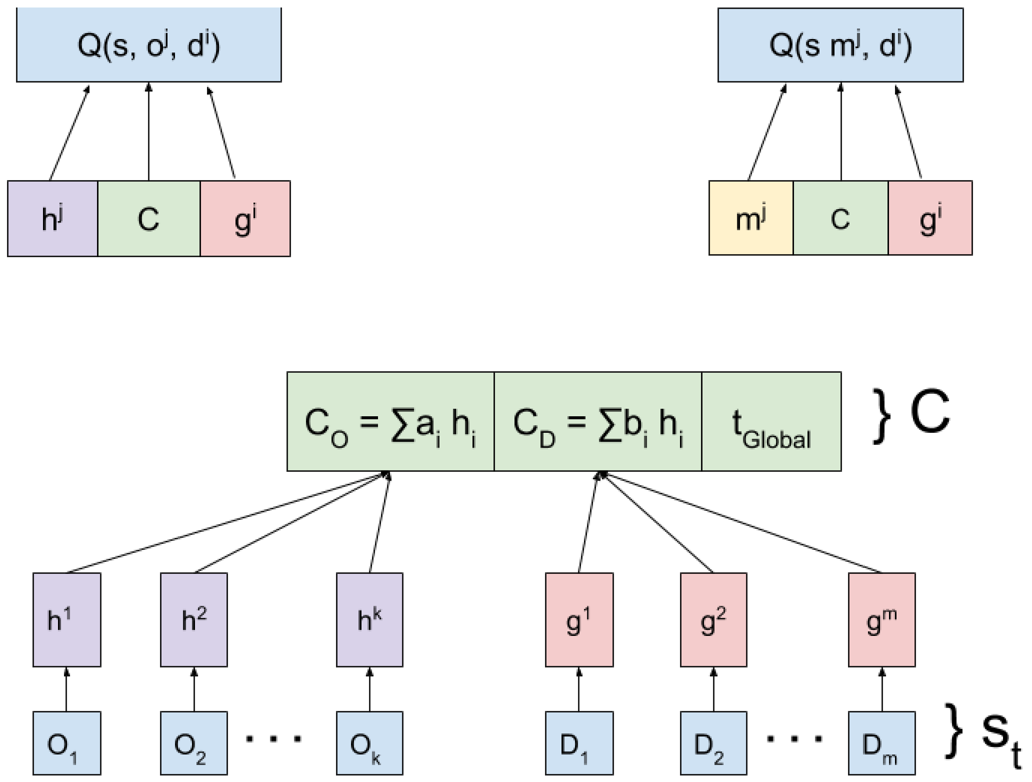

Figure 4.1: The neural network architecture used in SD-DQN and MD-DQN. Order and driver embeddings are attended to for global context. The network then uses global context to query order/driver pairs (represented by the matrix in the top left) and reposition/driver pairs (represented by the matrix in the top right) to produce Q-values for all possible actions.

4.2 Our Approach

4.2.1 Neural Network Overview

At time t there is a collection of orders $o_t^i \in \mathcal{O}_t$, drivers $d_t^j \in \mathcal{D}_t$, and subset $\mathcal{D}_t^{\text{avail}} \subset \mathcal{D}_t$ of available drivers. A driver is available if it is not currently serving an order. State is given to the neural network as $s_t = (\mathcal{O}_t, \mathcal{D}_t, \mathcal{D}_t^{\text{avail}}, t)$. At time t, the action space is made up of available driver-order pairings or non-repositioning driver-repositioning pairings.

Input representation

The exact vector representation of o_t^i and d_t^j depend on the environment. In the static assignment problem [47], we are only concerned with the position of drivers and *start positions* of orders. Therefore drivers and orders are each represented as two-dimensional vectors of their x and y coordinates. In MDVDRP, orders are given as six-dimensional vectors consisting of starting x position, starting y position, ending x position, ending y position, price, and time waiting, where time waiting is the difference between the current time and the time that the order first began requesting a driver. A driver is represented by a six-dimensional vector: an x position, y position, time to completion, repositioning x and y coordinates, and reposition counter. If the driver is available, its x and y position are given by its actual location, and time to completion is 0. If the driver is not available, the x and y position are the ending location of the order it is servicing, and the time to completion is the time it will take to finish the order. If a driver is repositioning, the direction of repositioning is reflected in the repositioning coordinates, and reposition counter counts down from the maximum repositioning time.

Embedding and global context

The network first embeds orders $\{o_t^i\}_i$ and drivers $\{d_t^j\}_j$ into memory cells $\{h_t^i\}_i$ and $\{g_t^j\}_j$ respectively found in the purple and red boxes in **??**. Then, a single round of attention is performed over each to produce a *global order context* C_t^O and *global driver context* C_t^D. These contexts as well as global time are concatenated to produce a *global context vector* C_t^G, which is the system's internal global representation of state.

Action-value calculation

The network then computes two types of Q-values: those for driver-order pairs and those for driver-reposition pairs. Each of these two processes can be viewed as their own attention mechanisms over pairs, with the attention coefficients interpreted as Q-values. More precisely, for available driver-order pairs we construct a small fully connected neural network Att_o such that $Q(s_t, d_t^j, o_t^i) = Att_o(C_t, g_t^j, h_t^i)$. Similarly, for available driver-reposition pairs, we construct another small neural network Att_r, so that $Q(s_t, d_t^j, m_t^i) =$

$Att_r(C_t, g_t^j, m_t^i)$ where m_t^i is a vector representation of a reposition action. The top left of **??** represents a generic example of Att_o while the top right represents Att_r. In all repositioning experiments, there are 9 reposition actions consisting of 8 cardinal directions and a stationary action (no movement). The m_t^i's are represented as one-hot vectors, and displayed as yellow boxes in **??**.

4.2.2 Single-driver and multi-driver Q-values

We take two approaches to training the above network: single-driver DQN (SD-DQN) and multi-driver DQN (MD-DQN). Each can be viewed as a 1-step bootstrapping approach like in the standard DQN, but they differ from one another in the form of one step data that is given to the network. As a result, the learned Q-values in each approach have distinct semantics.

In SD-DQN, we use *driver-centric* transitions. At time t, the system is in global state $s_t = (O_t, \mathcal{D}_t, \mathcal{D}_t^{avail}, t)$ and a driver-order or driver-reposition action is selected, yielding reward r_t. Let d be the selected driver, and a_t denote the action. We then proceed to make dispatching and repositioning decisions for other drivers as they become available, until eventually driver d is available again in state $s_{t'} = (O_{t'}, \mathcal{D}_{t'}, \mathcal{D}_{t'}^{avail}, t')$. The change in time $t' - t$ is the time it takes for driver d to complete a single trip or repositioning, which is typically between 10 and 30 minutes for trips or $2 - 3$ minutes for repositionings. In SD-DQN, this yields a transition $(s_t, a_t, r_t, s_{t'})$. To train our network in the SD-DQN setting, we update the outputs of the network using the target:

$$\widehat{Q}(s_t, a_t; \theta_t) = r_t + \gamma^{t'-t} \cdot \max_{a'} Q^T(s_{t'}, a'; \theta'_t), \tag{4.1}$$

where θ_t are the current network weights and θ'_t are the weights of the target network, which are slow updating weights that sync with the network's current weights at fixed intervals. To account for the fact that transitions occur over a variable time interval, future value is discounted continuously with discount factor γ. The network is trained to reduce the Huber loss between $\widehat{Q}(s_t, a_t)$ and $Q(s_t, a_t)$. Intuitively, SD-DQN updates the network towards driver-centric Q-values, only accounting for the discounted return associated to a single driver. We collect driver-centric transitions from each driver into a single shared replay buffer. The drawback of this method is that, as we learn the single driver Q-values, we update the behavior of *all drivers*, and so one perspective is that the driver, as an agent, is learning in a nonstationary MDP. This has the potential to cause instability issues in training, though we did not observe such issues in our experiments. Another issue with SD-DQN is that it may train single drivers to behave selfishly. That is, drivers might learn to choose actions that are best for themselves rather than actions that promote the greatest overall reward for the system. This issue is raised in work on learning in collectives [80, 72], however the interaction of individual learning and system optimization is not well understood.

In MD-DQN we use *system-centric* transitions. At time t, the system is in global state $s_t = (\mathcal{O}_t, \mathcal{D}_t, \mathcal{D}_t^{\text{avail}}, t)$ and we select action a_t, yielding reward r_t. When the next new driver becomes available, we transition to state $s_{t'} = (\mathcal{O}_{t'}, \mathcal{D}_{t'}, \mathcal{D}_{t'}^{\text{avail}}, t')$. In contrast to the SD-DQN transition, the change in time $t' - t$ is the time it takes for the next available driver to appear, which is on the order of fractions of a second in large cities. As in the SD-DQN case, this yields a 1-step transition $(s_t, a_t, r_t, s_{t'})$. We use the same target and update procedure from equation (1), but with this different transition as input data. MD-DQN will update the network towards a global, system-centric Q-value that sums the discounted rewards of all drivers together. This means the Q-values produced by MD-DQN will be approximately n times larger than those of SD-DQN, where n is the number of drivers. Ignoring issues related to function approximation MD-DQN provides a correct learning signal for solving an MDVDRP. However, as a practical matter, the Q-values learned in MD-DQN are based on many more transitions, and therefore should be harder to learn. This has been our empirical experience. In the results section we describe steps we needed to take in order to stabilize MD-DQN learning, but the primary change we make is to do n-step Q-learning [43] with n equal to the number of drivers in the system.

4.2.3 Architecture details

For ease of exposition, in the following section we omit subscript t's that had denoted time in previous sections.

Memory embedding. We begin with a global state s, consisting of order and drivers and time. All orders and drivers are both embedded into length 128 memory cells using a single layer network with 128 output units followed by RELU nonlinearities.

Attention for global context. Given a set of N order memories h^i and M driver memories g^j. The attention mechanism for orders/drivers are given by the following equations:

$$C^O = \sum_{i=1}^{N} a_i h^i$$

$$C^D = \sum_{j=1}^{M} b_j g^j$$

where

$$a_i = \sigma(v_o \cdot \tanh(W^O \cdot h^i)), b_j = \sigma(v_d \cdot \tanh(W^D \cdot g^j)) \tag{4.2}$$

and σ is a sigmoid activation function, W^O and W^D are 128 dimensional square matrices, and v^O and v^D are 128 dimensional vectors so that a_i and b_j are scalars. Both W's and v's are trained.

We concatenate the two contexts, along with the episode time t, to produce a 257 dimensional global context vector

$$C_G = [C_o | C_d | t], \tag{4.3}$$

Q-values. To compute a driver-order Q-value $Q(s, d^j, o^i)$, we concatenate the global context with the order's memory embedding h^i and driver's memory embedding g^j, and pass this through a fully connected layer with 64 units and RELU activation, and finally pass this to a single linear unit. We use the same network architecture for driver-repositioning pairs, but we *do not* share weights between the driver-order network and driver-repositioning network.

Further training details

In all experiments, we use a replay memory that is initialized with random behavior transitions. The size of replay memory as well as how frequently the target network is synced are both environment dependent, and are specified in the appendix. We also use a target network for setting Q-value targets. During training, each training loop begins by taking one step in the environment. Behavior is ϵ-greedy with respect to the network's Q-values, where ϵ is linearly annealed in all experiments as specified in the appendix. For both SD-DQN and MD-DQN, one new transition will be added to replay memory (though they differ in what this one step transition is). Then, we sample a batch of 32 transitions from replay memory, and perform a Q-value update using equation (1). For all experiments, $\gamma = 0.99$. Gradients are applied using the Tensorflow implementation of RMSProp with gradients clipped at 100.

4.3 Empirical Results

Results are presented in table format. Entries with error bars are computed as follows. We run the learning method (SD-DQN or MD-DQN) 4 times, with each experiment differing only in random seed. For each run, once learning has appeared to converge, we average reward, pickup distance, and orders served percentages over 20 episodes. We then compute the average and error bars across the four runs. If no error bars are included, this means the table is showing results over a single run, with entries averaged over 20 episodes.

4.3.1 Static multi-driver assignment problem

The assignment problem [47] is a combinatorial optimization problem defined by a *cost matrix*, where the $i - j^{\text{th}}$ entry represents the cost of assigning the i^{th} row object to the j^{th} column object. The goal is to produce an assignment that minimizes the sum of costs. In the context of driver assignment, the assignment cost is given by the Euclidean distance between order o^i and driver d^j. The assignment problem is the core subproblem for a dispatching problem with fixed windowing and a distance-minimization objective. An episode is initialized by a uniform random distribution of k orders and k drivers over the unit square. At each step, we choose a random driver, and the agent selects an order to match with the given driver. The reward associated to this action is the negative Euclidean distance between the driver-order pair. We do not perform discounting due to the static nature of the problem.

policy	total pickup distance
Random	10.25
SD-DQN	4.83 ± .06
MD-DQN	4.12 ± .03
Optimal	3.82

Table 4.1: 20 driver 20 order static assignment problem

The assignment problem is a particularly good environment for demonstrating the potential miscoordination of SD-DQN. For the single-driver approach, each transition ends in a terminal state, meaning that Q-learning reduces to one–step reward prediction. Therefore a policy which is greedy with respect to single driver Q-values will be suboptimal since it does not learn any coordination between drivers. On the other hand, an MD-DQN agent is concerned with maximizing the aggregate return across all drivers, and so should be capable of learning a better policy.

Results are summarized in **??**. We show total distance traveled (that is, the sum of the distances of all assignments) for SD-DQN and MD-DQN when there are 20 orders and 20 drivers. We compare them to optimal solutions as well as *Random assignments*. The results reflect our intuition regarding the shortcomings of SD-DQN in sensitive coordination problems. Namely, SD-DQN performs quite a bit worse than MD-DQN. This emphasizes the innate desirability of MD-DQN—it is capable of learning more complex coordination behavior.

4.3.2 Dynamic Multi-Driver Dispatching Problems

The remaining experiments will focus on *dynamic* dispatching problems. In the dynamic setting, orders arrive at different times throughout an episode. Additionally, we are focused on ride-sharing, so orders now are defined by pickup *and* dropoff locations. When a driver is assigned to an order, it must first navigate to the pickup location and then travel to the dropoff location. We first present results on small domains where myopic policies are demonstrably suboptimal. Then, we apply the SD-DQN and MD-DQN approaches to a large-scale dispatching simulator derived from real-world data collected from DiDi Chuxing. We find that SD-DQN outperforms all other methods in the realistic simulators, but is occasionally worse than MD-DQN in illustrative domains.

Illustrative domains with no repositioning

In this group of experiments, we use a simple dispatching simulator to show that both SD- and MD-DQN can learn good policies in two key scenarios where myopic behavior fails. The city is represented by the unit square. In these domains, at the start of a new episode, drivers are all located at a "dispatching depot" at position $[0.5, 0.5]$. Drivers travel at a constant speed of .1, and travel along straight lines from their initial position to the order pickup location, and then from order pickup to order dropoff location. Order arrivals are generated according to a Poisson distribution, with controllable parameter κ. In the following experiments, κ is set to either 3 or 10 (that is, average order arrivals per unit time are either 3 or 10) to simulate "low demand" and "high demand" environments. The pickup and dropoff locations as well as the reward for an order are specified below for two different environment settings. An episode lasts 5000 timesteps.

Surge domain

The Surge domain illustrates an explicit, temporal effect caused by order pricing that cannot be exploited by myopic dispatchers. In the Surge domain, there are three regions: left, center, and right. One quarter of all orders go from the center to the upper-left region, one quarter from center to bottom-right, one quarter from upper-left to center, and one quarter from bottom-right to center. All orders yield a reward of 2 except those that go from right to center, which yield a reward of 4. For this domain, the best policy first assigns drivers to travel to the bottom-right region, and once they are there, assign them to the bonus reward trips back from right to center. A policy that minimizes pickup distance will fail to value trips to the bottom-right more than trips to top-left, and therefore yield suboptimal behavior. On the other hand, a policy which is greedy with respect to rewards will always select the bonus order regardless of driver location. In effect the policy "skips" the price 2 order that will ferry a driver out to the bottom-right region, and is therefore also suboptimal. An illustration of the surge domain can be found in the appendix.

Hot/Cold domain

In the Surge domain, the advantage of traveling to the bottom right region is clear; it is directly tied to the price of orders found in that region (4 vs. 2). In the Hot/Cold domain, the agent must learn a more subtle advantage. Order pickup locations are located uniformly along the top edge of the unit square, called the "hot region". Half of the orders end uniformly along the bottom edge of the unit square, called the "cold region" and half end uniformly in the hot region. Order price is given by the *Euclidean distance from order pickup to order dropoff locations*. The hot region can be thought of as a busy area of downtown, while the cold region represents surrounding suburbs. Despite orders to the cold region having higher price (since they are longer), it is generally more advantageous for drivers to stay in the hot region, since they can quickly pick up new orders. In other words, the advantage is entirely *temporal*. An illustration of the Hot/Cold domain can be found in the appendix.

We compare SD-DQN and MD-DQN with 3 other algorithms: myopic revenue maximization (MRM), myopic pickup distance minimization (MPDM), and local policy improvement (LPI). MRM always assigns the highest value order to an available driver. MPDM always assigns the closest order to an available driver. LPI [82] discretizes the environment into a 20x20x144 spatiotemporal grid and performs tabular TD(0). We report average revenue, pickup distance, and served percentages with error bars over 100 episodes. Each episode lasts 5000 time units, which allows each driver to serve approximately 1000 orders.

Results can be found in **??** and **??**. There are a few key takeaways from these results. First, all learning methods, including LPI, outperform myopic strategies across the board. SD-DQN and MD-DQN also typically improve over LPI, though the margin is considerably smaller. Finally, it is not clear what situations favor SD-DQN vs. MD-DQN. For instance, one might expect SD-DQN to do comparatively better in high demand situations, where there would seem to be a reduced need for coordination, however this is not the case.

	Low Demand			High Demand		
Algorithm	Revenue	Pickup distance	Served %	Revenue	Pickup distance	Served %
MRM	29447	.33	73.6	35939	.54	18.1
MPDM	32279	.178	86.3	42842	.016	34.2
LPI	31112	.245	80.0	50147	.046	33.6
SD-DQN	31860 ± 448	.279 ± .0005	78.2 ± .19	50323 ± 87	.045 ± .02	33.54 ± .09
MD-DQN	33700 ± 225	.177 ± .0001	88.23 ± .28	49031 ± 130	.056 ± .0012	32.79 ± .05

Table 4.2: Surge Domain

	Low Demand			High Demand		
Algorithm	Revenue	Pickup distance	Served %	Revenue	Pickup distance	Served %
MRM	50953	1.04	31.5	52094	1.11	8.7
MPDM	56546	.535	53.8	58287	.508	16.5
LPI	58173	.45	60.64	76840	.1545	30.06
SD-DQN	58580 ± 124	.4609 ± .007	59.26 ± .13	78552 ± 212	.1108 ± .003	39.25 ± .04
MD-DQN	58893 ± 181	.5158 ± .008	52.97 ± .027	78860 ± 285	.111 ± .012	33.625 ± 1.16

Table 4.3: Hot/Cold Domain

Illustrative Domains with Repositioning

Whereas the previous experiments only deal with dispatching, we now examine our methods on domains where drivers can both be dispatched and repositioned.

Repositioning Hot/Cold Domain

The first such environment is the same as the previous Hot/Cold domain, except we impose a *broadcasting radius* d_{bcast} on drivers. This means that drivers may only pair with orders if they are within d_{bcast} units of the driver. Otherwise, the driver may only take a repositioning action. For this domain we set $d_{bcast} = 0.3$ If a driver matches to an order that ends in the cold region, the agent must learn to reposition that driver from the cold region towards the hot region (which consists of approximately 4 consecutive "move up" repositioning actions) so the driver can pair with additional orders. As with the dispatch-only experiments, we present results for high and low demand regimes.

	Low Demand			High Demand		
Algorithm	Revenue	Pickup distance	Served %	Revenue	Pickup distance	Served %
MRM-random	932	.199	4.2	911	.177	1.8
MPDM-random	939	.174	8.1	936	.161	2.5
MRM-demand	4861	.180	34.1	4902	.178	8.1
MPDM-demand	5234	.1624	53.2	5644	.164	15.9
SD-DQN	5478 ± 188	.1615 ± .03	57.5 ± .31	7387 ± 41	.0781 ± .008	33.8 ± .43
MD-DQN	5717 ± 213	.1879 ± .05	54.5 ± .25	7309 ± 56	.1519 ± .04	24.2 ± .22

Table 4.4: Hot/Cold with repositioning

We compare our methods against two versions of MPDM with repositioning: MPDM-random and MPDM-demand. If a driver has no orders within broadcast distance, MPDM-random *randomly* selects a repositioning action, whereas MPDM-demand repositions the driver towards the nearest order. As we can see in **??**, SD-DQN and MD-DQN both maintain their advantage over baselines when required to learn repositioning and dispatching together.

Distribute Domain

While Hot/Cold with repositioning tests an important aspect of learning—namely, the ability of MD-DQN and SD-DQN agents to reposition drivers to locations where they can pick up new orders, this repositioning behavior is quite simple in that it is *uniform across drivers*. This means that the agent can always reposition drivers in the same manner (i.e. "if in cold region, go to hot region"). In order to test whether our methods can learn non-uniform repositioning behavior, we introduce a class of “Distribution environments” where drivers must be repositioned so as to match their spatial distribution with a fixed future order distribution. A Distribute domain operates in two phases. In the first phase, the environment resets with k drivers and no orders in the system, so drivers may only reposition during this phase. In the second phase, k orders appear according to a fixed spatial distribution, and drivers can match to orders if they are within a given broadcast radius d_{bcast}. The second phase only lasts long enough to allow drivers to reposition one more time before all orders cancel and the environment is reset. We alter the reward function so that each order–matching action receives +1 reward. Order destinations are designed to be far away from start locations so that each driver may only serve one order per episode. As a result, the episodic return is proportional to the number of orders served, so we may interpret the episode score as a measure of how well the agent arranges driver supply in phase 1 with order demand in phase 2.

In our experiments, the distribution of orders always consists of two small patches in the top left and bottom right parts of the unit square. The order start locations are sampled uniformly within each patch. The total number of orders in each patch is fixed across episodes, and we denote it fractionally. An even order split between patches (eg 10 orders in both patches) is denoted 50/50. If 80 percent of orders are in the first patch and 20 percent are in the second patch, we denote it as 80/20. Visualizations of the Distribute domain are in the appendix.

Results for 20-driver Distribute domains are shown in **??**. We include the optimal served percentage (which is 100 %) and the "uniform optimal" served percentage. This quantity reflects the maximum score one can obtain if the repositioning behavior is uniform across drivers. SD- and MD-DQN are able to get near optimal test scores when the demand is balanced. However, in the 80/20 task, only SD-DQN was able to escape the uniform optimum. For all experiments it was critical to allow for sustained high exploration. Specifically, in all experiments we used an ϵ-greedy behavior policy where ϵ was linearly annealed epsilon from 1.0 to 0.2 over the first 1000 episodes. Test performance is averaged over 10 episodes. Also, we ran each experiment 4 times changing only the random seeds. We found that final performance across seeds was nearly identical in all experiments. Learning curves can be found in the appendix.

We also used the distribute domain to test the saliency of global state information in the learning process of SD-DQN. Traditionally, independent learning in games assumes that agents only have a partial view of state at decision time [16]. In contrast, SD-DQN receives full state information as input. We demonstrate the salience of global state through a small distribute domain in which there are 4 drivers and a 75/25 split i.e., three orders appear in one region and 1 order appears in the other. We then trained SD-DQN with and without the inclusion of global context. Without global context, the network becomes stuck in the uniform optimal strategy that sends all drivers to the three–order region. A graph comparing served percentage with and without global state can be found in the appendix.

Algorithm	50/50 Served %	80/20 Served %
Optimal	100%	100%
Uniform Optimal	50%	80%
SD-DQN	$96 \pm .13\%$	$92 \pm .72\%$
MD-DQN	$95 \pm .11\%$	$80 \pm 3.42\%$

Table 4.5: Distribute Domain with 20 Drivers

Non-repositioning Historical-Statistics Real-World Domain

Finally, we test our method in more realistic dispatching environments. We refer to the first of these as the Historical-Statistics domain, because it derives distributional order and driver generation parameters from historical data. This first realistic simulator *does not include repositioning*. Specifically, we used 30 days of dispatching data from DiDi Chuxing's GAIA dataset [13], which contains spatial and temporal information for tens of millions of trips in one of the largest cities in China. To build the simulator from this data, we first covered the city in a square 20 by 20 grid, and extracted Poisson parameters $\kappa_{x,y,t}$ where x is an order start tile, y is an order end tile, and t is the time of day in hours. This results in $400 \times 400 \times 24 = 3.84$ million parameters which we use to specify an order generation process. In addition to these, we also extract the average *ETA*, as well as its variance, for each (x, y, t) triple. When a driver is assigned to an order, the simulator computes a Gaussian sample (using the historical mean and variance) of the ETA t_1 from the driver's position to the order's start location, and another sample of the ETA t_2 for the order's start location to the order's end location. The driver will become available at the order's end location in time $t_1 + t_2$. Orders price is equal to $\max(5, t_2)$, where t_2 is given in minutes. Driver entry and exit parameters are also derived from data. For each tile-hour triple (x, y, t) we compute the poisson parameter from driver arrival, and the duration that the driver remains in the system is given by a poisson parameter that is a function only of t.

	.1% scale			1% scale		
Algorithm	Revenue	Pickup ETA	Served %	Revenue	Pickup ETA	Served %
MRM	10707	22.74	20.9	117621	22.32	20.16
MPDM	11477	11.99	31.6	134454	6.1	36.79
SD-DQN	12085 ± 19	19.15 ± .16	24.96 ± .11	146182 ± 244	15.07 ± .11	27.64 ± .09
MD-DQN	11145 ± 78	21.77 ± .62	21.38 ± .32	122671 ± 698	19.50 ± .52	22.14 ± .76

Table 4.6: .1% and 1% scaled real data

Algorithm	Revenue	Pickup ETA	Served %
MRM	1112340	22.37	20.04
MPDM	1333180	6.2	29.4
SD-DQN	1391141	17.28	25.3
MD-DQN	1161780	20.05	23.17

Table 4.7: 10% scaled real data

To control computational costs we control the *scale* of the MDVDRP via a scaling parameter $0 < \lambda \leq 1$. All order and driver generation parameters are multiplied by λ. For example, a 1% scaled environment means that we multiplied all generation parameters by 0.01. We present results for 3 scale regimes: .1%, 1% (**??**), and 10% (**??**). For .1% and 1% we report values with standard errors across 100 episodes, and for 10% we report values with standard error across 10 episodes.

Across all scales, SD-DQN outperforms both myopic baselines, while MD-DQN generally only performs slightly above myopic revenue maximization.

Repositioning Historical-Order Real-World Domain

We also experiment with a simulator that uses historical days of orders instead of generating orders randomly from historical statistics. The GAIA dataset provides 30 days of orders in

Algorithm	Revenue	Pickup ETA	Served %
MRM	414	2.60	44
MPDM	511	1.1	79
MPDM-random	494	.9	73
MPDM-demand	502	.8	76
SD-DQN	542	1.2	75
MD-DQN	474	1.8	53

Table 4.8: 10% region real data

the city of Chengdu. We first found a small spatial region of Chengdu for which 10% of orders both start and end in that region. This region defined the historical data simulator. We then created 30 *order generation schemes*. Precisely, when the environment is reset, it randomly selects one of the 30 days, and generates orders exactly according to how orders arrived on that day. We used a fixed number of drivers (100), and a fixed speed (40 km/h). An illustration of this environment can be found in the appendix. For SD-DQN and MD-DQN, we impose a 2 kilometre broadcast radius. We compare performance against the standard non-repositioning baselines of myopic revenue maximization (MRM) and myopic pickup distance minimization (MPDM), both of which have no broadcast distance and no repositioning. We also compare against MPDM-random and MPDM-demand. The broadcast distance used in these repositioning baselines is 2 kilometers.

We obtain the similar relative performance as the historical statistics domain, with MD-DQN performance above MRM but below MPDM, and SD-DQN outperforming all myopic strategies.

4.3.3 Conclusion

We performed a detailed empirical study of two reinforcement learning approaches to multi-driver vehicle dispatching and repositioning problems: single-driver Q-learning and multi-driver Q-learning. Both approaches leverage a global representation of state processed using attention mechanisms, but differ in the form of Q-learning update used. We found that, while one can construct environments where MD-DQN is superior, typically SD-DQN is competitive. Furthermore we applied these methods to domains built from real dispatching data, and found that SD-DQN is able to consistently beat myopic strategies across scales, as well as with and without repositioning actions.

4.4 Environment descriptions and visualizations

The following figures show and describe MDVDRPs in general, as well as particular visualizations for all dispatching environments.

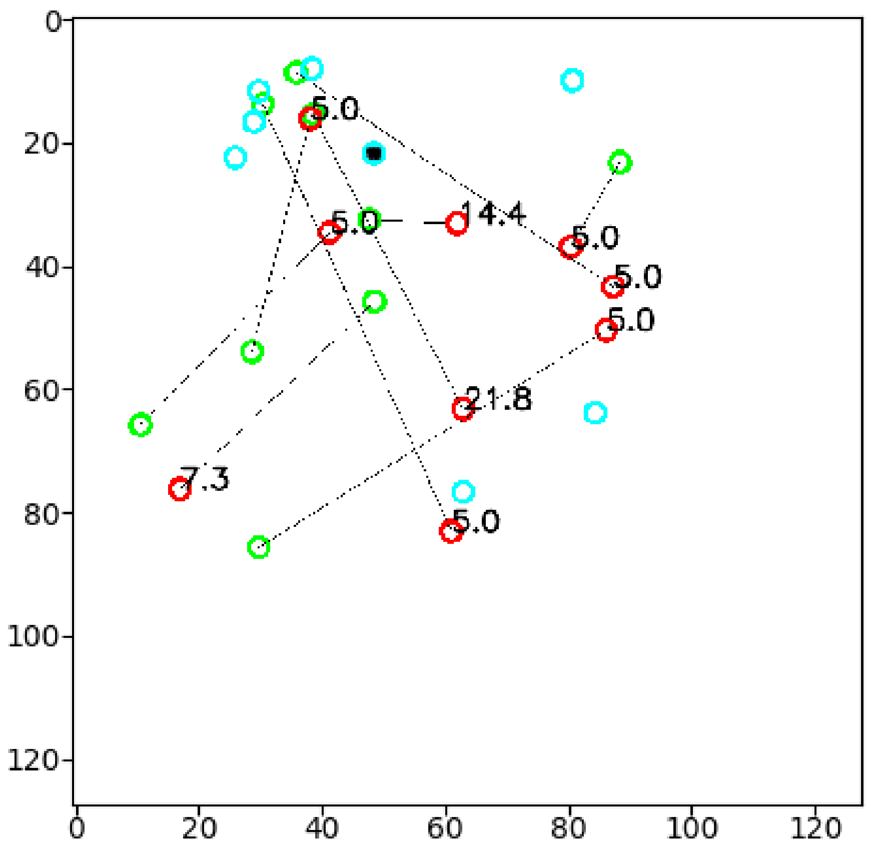

Figure 4.2: A visual representation of a dispatching state. Blue dots represent drivers. The black centered driver located at position (60, 20) is available, and all others are dispatched. Orders start at red dots and end at the connected green dot. Order prices are denoted above their pickup location

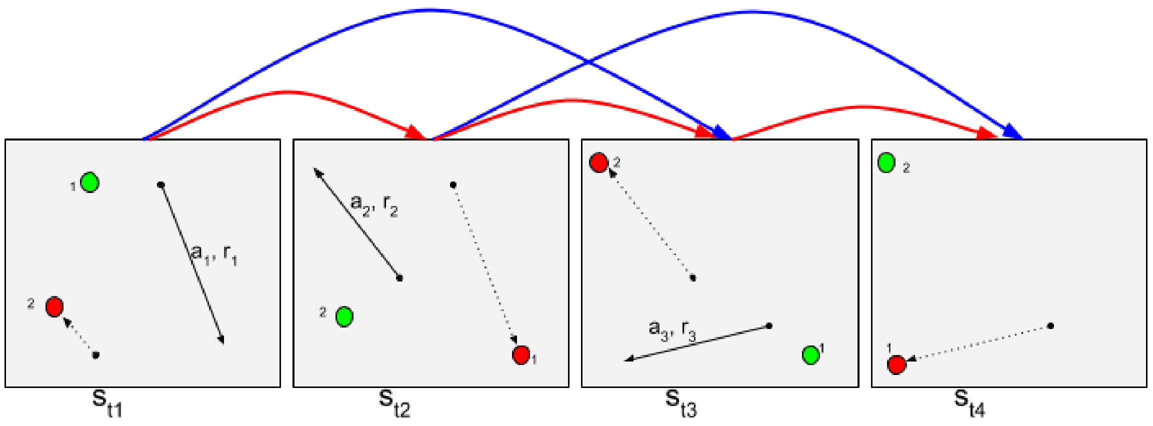

Figure 4.3: The above image shows a length 4 trajectory. The currently available driver is green, dispatched driver is red, and the order that the available driver accepts at time t_i is a_i and has price r_i. The accepted order at time t_i is labeled by its action name and price, (a_i, r_i) and travels from the solid black dot to the terminal arrow. SD-DQN transitions are indicated by blue arrows above state, e.g. transition $(s_{t1}, a_1, r_1, s_{t3})$, which is driver-centric with respect to driver 1. MD-DQN transitions are indicated by red arrows e.g. transition$(s_{t1}, a_1, r_1, s_{t2})$, which transitions from a state where driver 1 is available to a state where driver 2 is available.

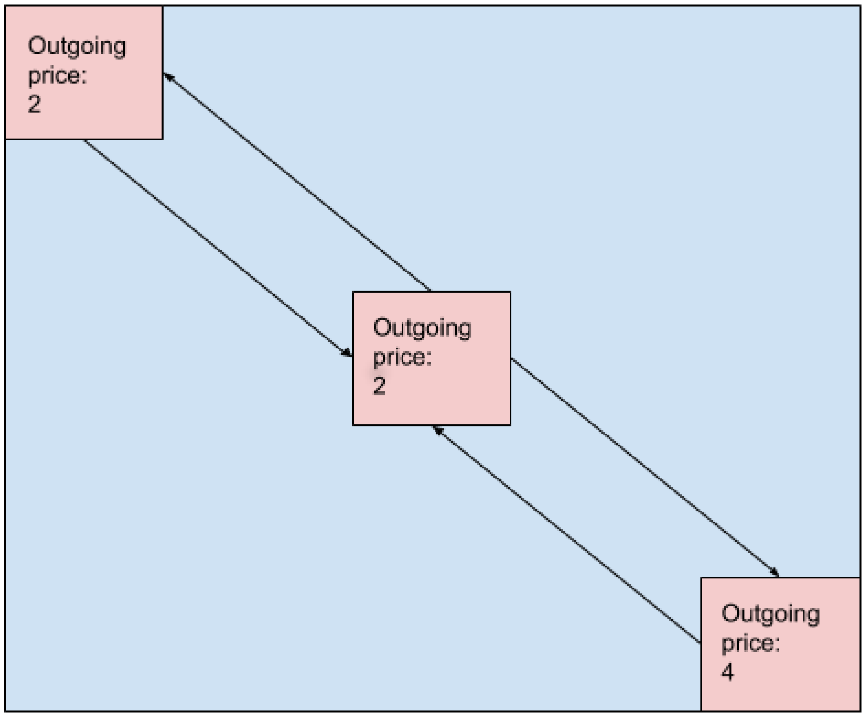

Figure 4.4: Surge domain. Orders travel between the three red squares. Each square is labeled with its outgoing order value. Within each square, order start and end locations are generated uniformly randomly.

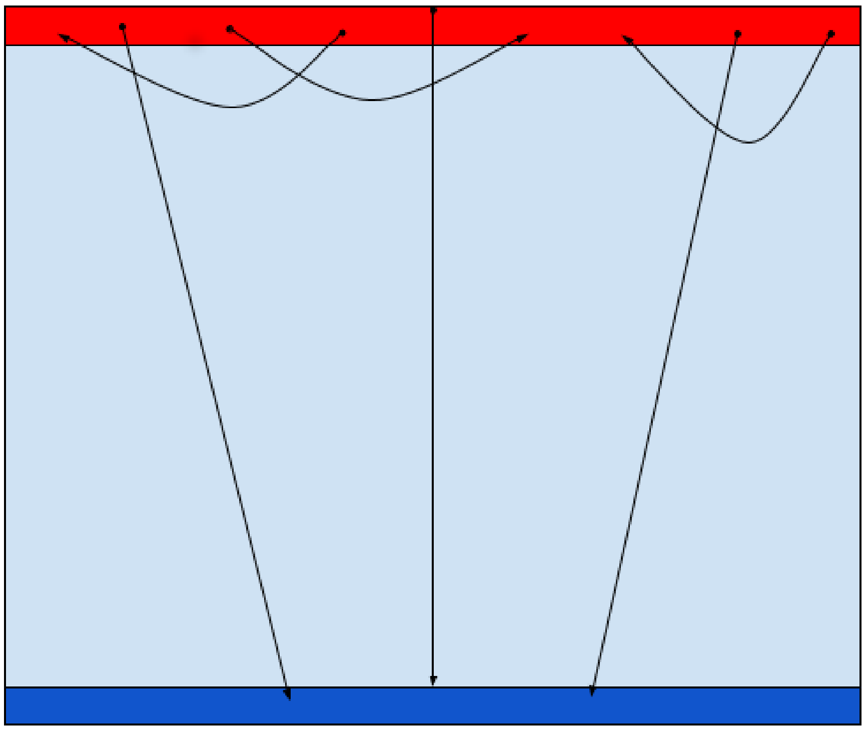

Figure 4.5: Hot/Cold domain. Orders all begin in the red bar, with their positions generated uniformly randomly. For the destination, a fair coin is flipped to decide whether the order ends in hot or cold, and then the exact position is sampled uniformly randomly in the designated region.

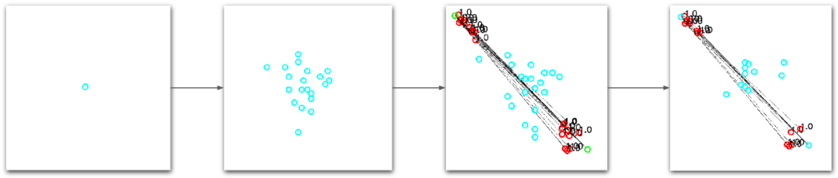

Figure 4.6: Distribute domain. Drivers begin in the center of the region. They then proceed with 5 steps of repositioning. At the 6^{th} timestep, orders appear in the two corners. Drivers that are within .3 units of an order start, denoted by a red circle, are assigned. All orders end in the opposing cornerâĂŹs green dot so that trips are long enough that a single driver can only satisfy one order per episode. After two timesteps, all remaining orders cancel and the environment resets.

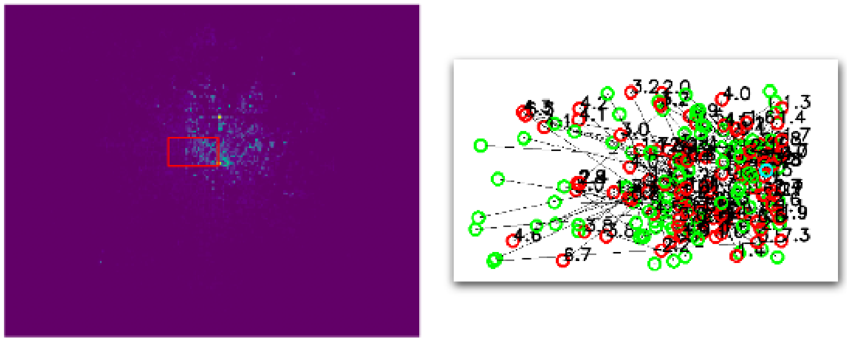

Figure 4.7: Historic data order start distribution and corresponding simulator rendering. The red box indicates the spatial region that is selected for the simulator. an average of 10 percent of orders start and end in this region, which roughly correspond to an edge of downtown and some outlying areas.

4.4.1 Graphs

In the following graphs, the units of the x-axis are in *episodes*. For the nonrepositioning illustrative domains, an episode lasts 5000 time units. In the Repositioning Hot/Cold domain an episode is 500 time units. The distribute domain lasts 7 time units. The realistic domains last for 1440 time units in the realistic simulator. Graphs for Distribute domains show served percentage on the y-axis, while all other curves are environment reward.

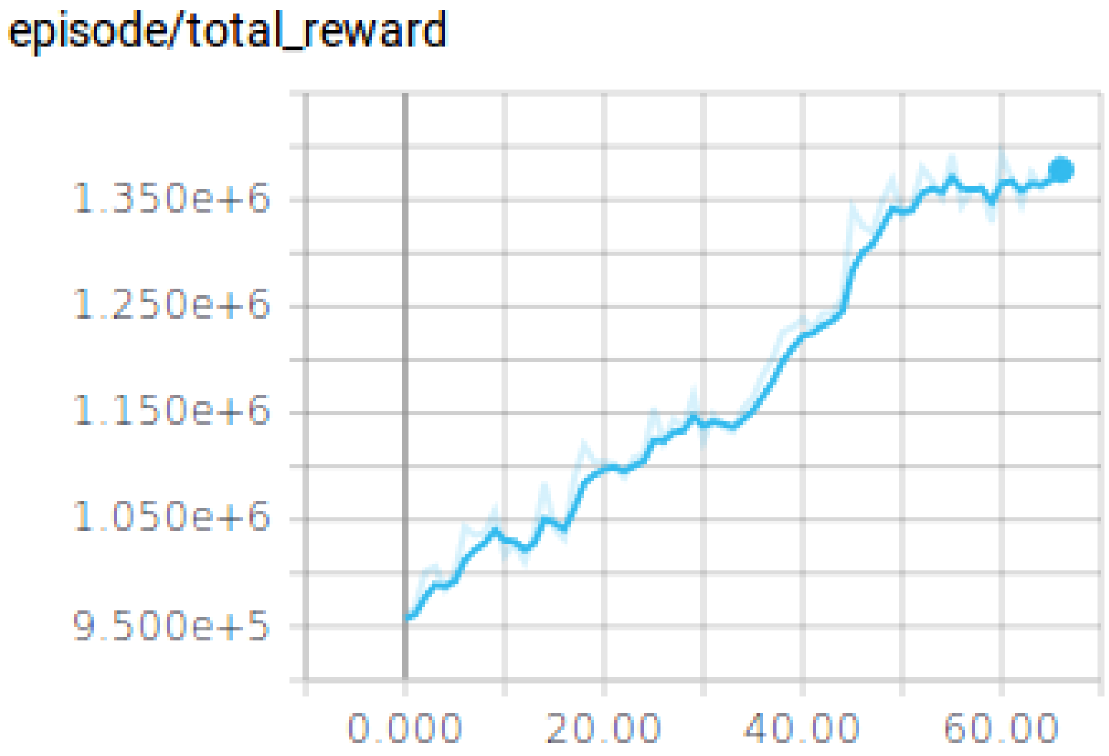

Figure 4.8: SD-DQN on 10% historical statistics simulator

4.4.2 Distribute Domains

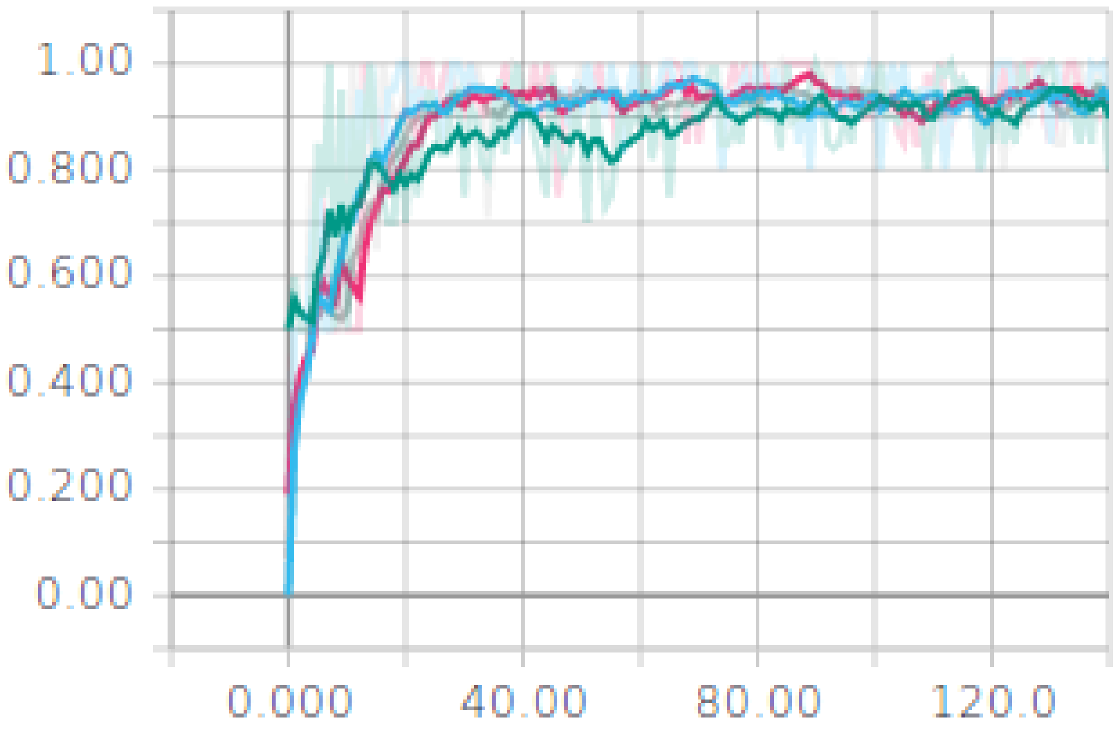

Figure 4.9: SDDQN served percentage on 20 driver 50/50 distribute domain

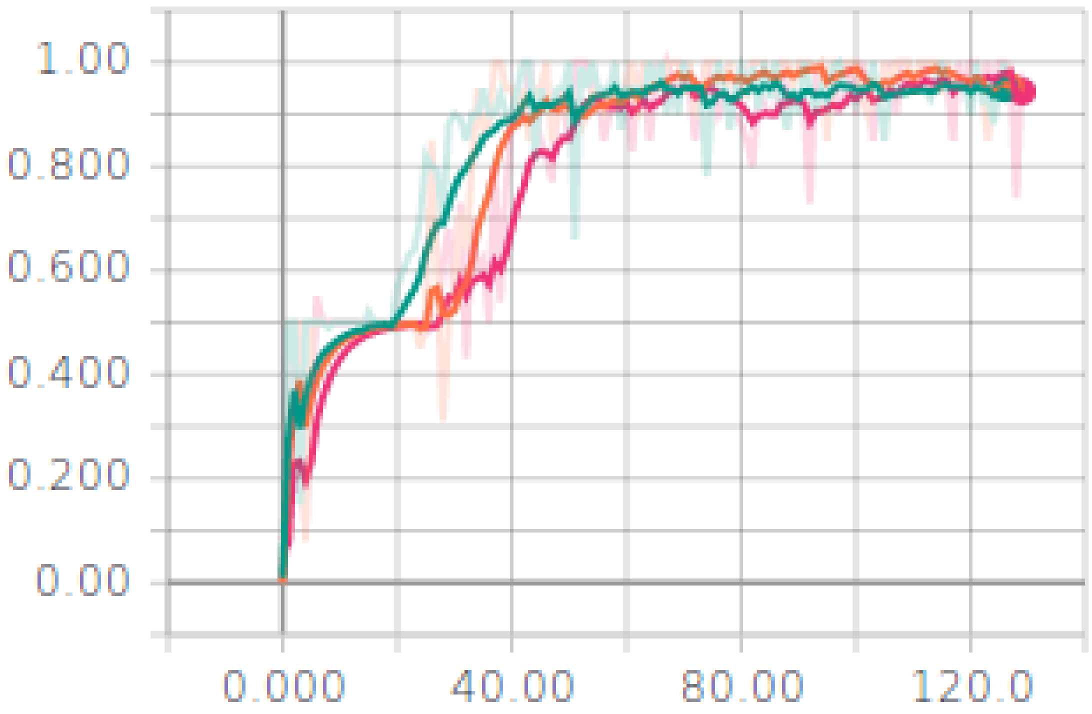

Figure 4.10: MDDQN served percentage on 20 driver 50/50 distribute domain

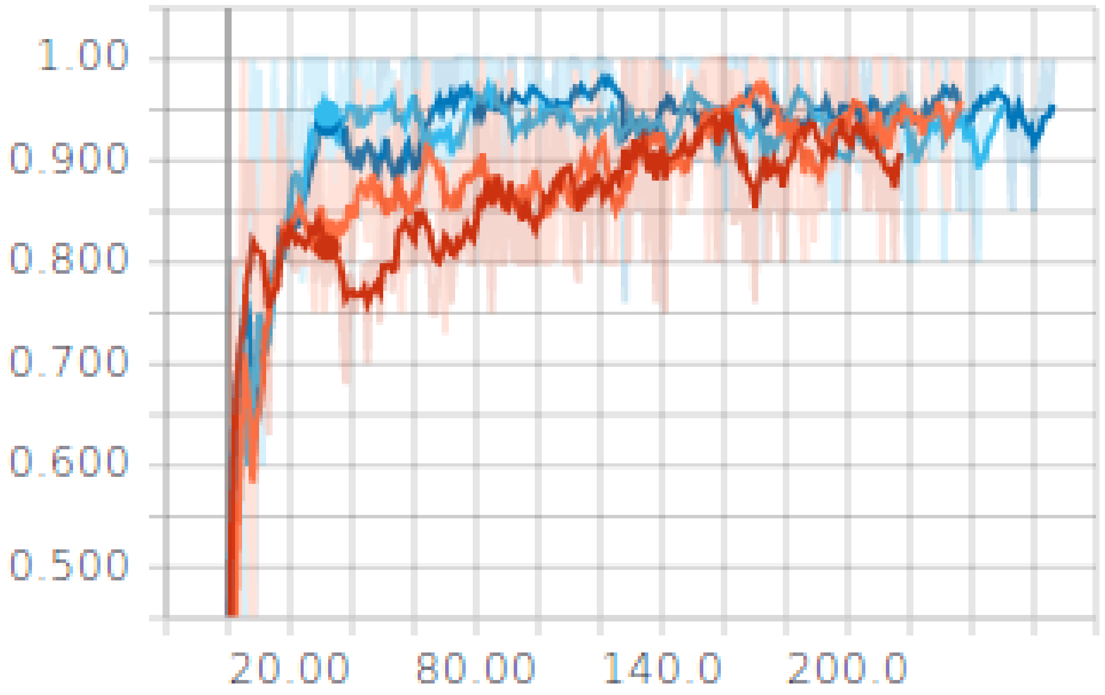

Figure 4.11: SDDQN served percentage on 20 driver 80/20 distribute domain

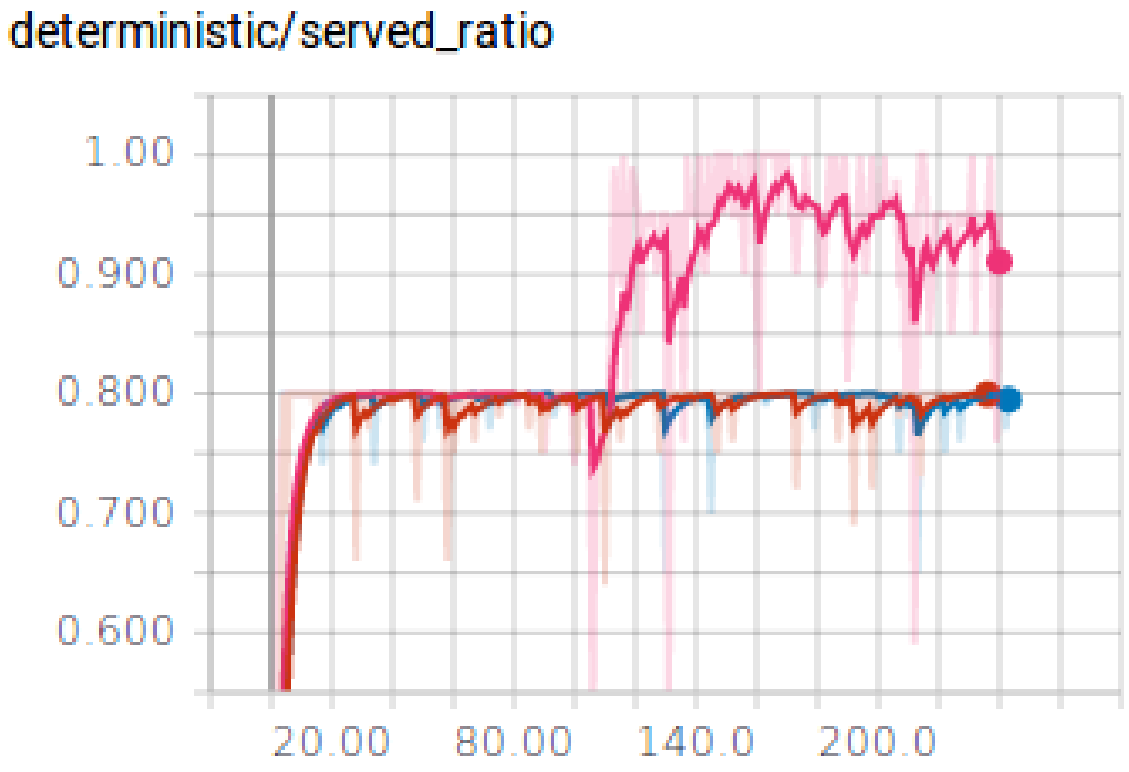

Figure 4.12: MDDQN served percentage on 20 driver 80/20 distribute domain

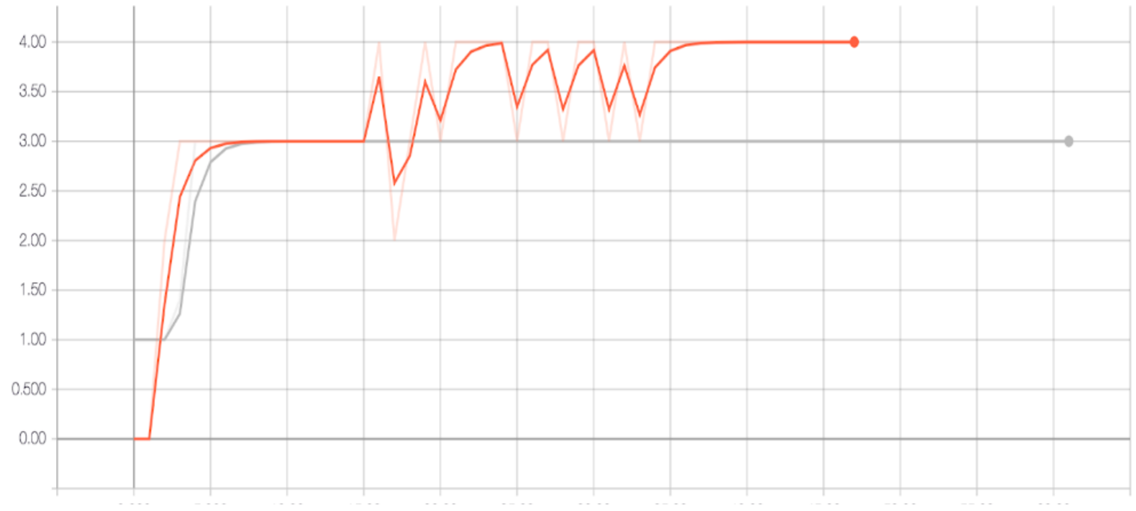

Figure 4.13: SDDQN served percentage on 4 drivers 75/25 distribute domain. The orange curve shows SD-DQN performance when global context is included during the Q-value querying while the gray curve does not include global context. SDDQN without global context learns a policy that is uniform across drivers, and so it never escapes the uniform optimum.

4.4.3 Historical Data Domain

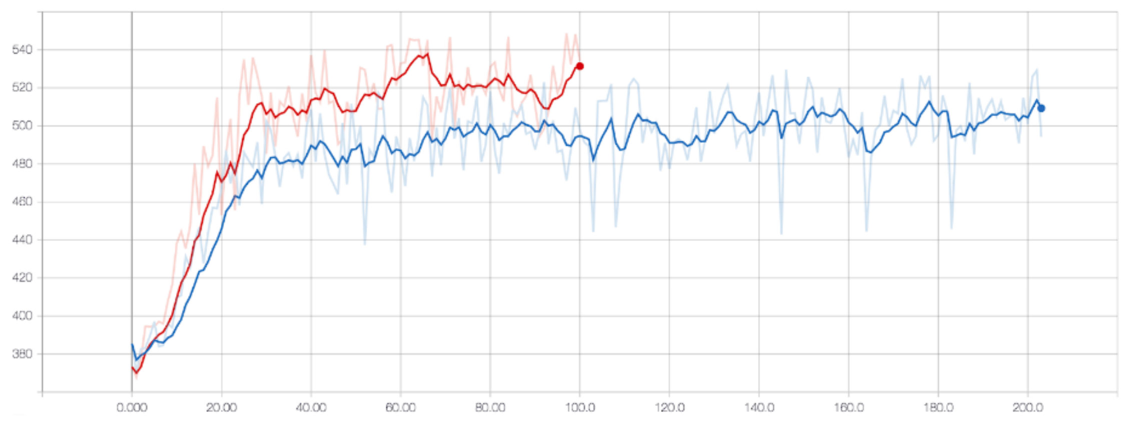

Figure 4.14: SD-DQN revenue on 10% historical simulator. The blue curve shows learning when revenue itself is used as the reward function. The red curve shows revenue when negative pickup distance is used as the reward function.

4.4.4 Training Details

20 Driver Static Assignment Problem

ϵ-exploration is annealed linearly over $10,000$ episodes from 1.0 to 0.03, and the target network is updated every 500 training steps. The replay buffer maintains the 10000 most recent transitions, and is initialized with 5000 samples. We used a learning rate of 0.001. Unlike the following domains, the reward function is given as the negative Euclidean distance between the selected driver-order pair.

All other illustrative domains

For SD-DQN we used a size 20000 replay buffer, learning rate 0.001, and we annealed ϵ-exploration linearly over 100 episodes. We update the target network every 500 training updates, and perform a training update every environment step. For MD-DQN we used a size 20000 20-step Q-learning, a learning rate of 0.0001, and annealed ϵ-exploration linearly over 100 episodes. We found it critical to stability to reduce the learning rate and perform *n*-step Q-learning in MD-DQN.

Chapter 5

Future Work

Chapters 2 and 3 of this work served as first expedition into "potential-like" stochastic games. It leaves open several corridors to be explored in future work. Four of these are briefly summarized below.

5.1 Learning Methods and Learning by Reinforcement

The learning methods we presented in Chapter 3 were somewhat stylized and had significant knowledge requirements for players. For example, we made the assumption in SG-JSFP that all players select all actions in all states simultaneously. For both SG-JSPF and SG-LLL we also assumed that players could calculate their utility functions and that all players could observe the actions of all other players at every timestep. The first assumption for SG-JSFP can be dispelled with, but the second two assumptions are necessary for the implementation of both algorithms.

Are there learning methods for SPGs and/or SGPGs that do not require players to compute their utility functions? These would resemble reinforcement-style algorithms, where players merely *estimate* their stage game utility functions from experience. Regret matching [64] is a learning method in iterated games that does not require computing utility functions, however its convergence guarantees are weaker than the learning methods we've discussed. In particular, strategies converge to the *set* of *coarse correlated equilibria*. So, one concrete research direction is to explore if regret matching can be extended to stochastic games, and if so, whether one can make universal convergence guarantees. This may not be straightforward - recall that we were unable to extend LLL to SPGs because the set of stochastically stable states contained more than one strategy. For the same reason it may be difficult to extend regret matching algorithms to stochastic games.

5.2 Other Subclasses of Stochastic Games

SPGs were motivated by the desire to make all continuation games be potential games. This required the stochastic game to have modular dynamics. In contrast, stochastic zero sum games automatically have this feature, without any requirements on their transition probabilities. We may use this general template the construct new classes of stochastic games:

1. Start with an interesting class C of normal form games.
2. Identify constraints on a stochastic games such that all of its continuation games belong to C.
3. Use these constraints to define a subclass of stochastic games.

One class of games that could be studied in this way are supermodular games. These games are interesting from a learning perspective as there are learning methods that converge to Nash equilibrium [24]. Furthermore, supermodular games have interesting equilibria sets. They also have a number of economic applications [71, 77, 42].

5.3 Infinite Horizion SPGs

It was important in our analyses of SPGs and SGPGs that we make them finite fixed-time. In particular this mean that states were visited only once in an episode. For SPGs we made use of this fact in arguing for both the existence of simultaneously potential maximizing equilibria, and the convergence of SG-JSFP to a pure Nash equilibrium. To briefly summarize its use: At a state in layer k, modification of behavior in this state can only change the continuation payoffs in layers less than k. So we were able to make arguments that iterate backwards through time, and be sure that once behavior stabilizes in layers $l > k$, the continuation payoffs for these layers will also remain constant.

Do simultaneously potential maximizing equilibria exist in SPGs? Are there even pure Nash equilibria? And are there convergent learning algorithms? These questions are not addressed by the current work. The author suspects however that these could be approached through the average reward formulation of MDPs.

5.4 Learning Methods for Simultaneously Potential Maximizing Equilibria

As far as we are aware, log-linear learning (LLL) is the only known learning algorithm that converges to potential maximizing behavior in potential games. So, it offers a promising way to learn simultaneously potential maximizing behavior in SPGs. However, as we saw in Chapter 3, our naive extension of LLL to stochastic games does necessarily converge to Nash equilibrium, let alone simultaneously potential maximizing equilibrium. So, it remains an open question to describe learning dynamics that converge to simultaneously potential maximizing equilibria in SPGs.

Bibliography

[1] Best response dynamics in zero-sum stochastic games.

[2] G. Arslan and S. Yüksel. Decentralized q-learning for stochastic teams and games. *IEEE Transactions on Automatic Control*, 62(4):1545–1558, 2016.

[3] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel. The mechanics of n-player differentiable games. *arXiv preprint arXiv:1802.05642*, 2018.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[5] L. E. Blume. The statistical mechanics of strategic interaction. *Games and economic behavior*, 5(3):387–424, 1993.

[6] M. Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *ICML*, pages 89–94, 2000.

[7] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, volume 17, pages 1021–1026. Lawrence Erlbaum Associates Ltd, 2001.

[8] G. W. Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[9] S. Buzzi, G. Colavolpe, D. Saturnino, and A. Zappone. Potential games for energy-efficient power control and subcarrier allocation in uplink multicell ofdma systems. *IEEE Journal of Selected Topics in Signal Processing*, 6(2):89–103, 2011.

[10] D. Cheng. On finite potential games. *Automatica*, 50(7):1793–1801, 2014.

[11] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 1998, pages 746–752, 1998.

[12] R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine learning*, 33(2-3):235–262, 1998.

[13] Didi Chuxing. Didi launches gaia initiative to facilitate data-driven research in transportation. URL: `http://www.didichuxing.com/en/press-news/bgcitgfc.html`, 2017.

[14] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[15] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] D. Fudenberg, F. Drew, D. K. Levine, and D. K. Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[17] D. Fudenberg and D. M. Kreps. Learning mixed equilibria. *Games and Economic Behavior*, 5(3):320–367, 1993.

[18] R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35(12):1969–1985, 1999.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[20] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[21] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[22] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[23] J. Hofbauer and E. Hopkins. Learning in perturbed asymmetric games. *Games and Economic Behavior*, 52(1):133–152, 2005.

[24] J. Hofbauer and W. H. Sandholm. On the global convergence of stochastic fictitious play. *Econometrica*, 70(6):2265–2294, 2002.

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

[26] S. Jastrzębski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. Finding flatter minima with sgd. 2018.

[27] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[28] D. M. Kreps, P. Milgrom, J. Roberts, and R. Wilson. Rational cooperation in the finitely repeated prisoners' dilemma. *Journal of Economic theory*, 27(2):245–252, 1982.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] J.-M. Lasry and P.-L. Lions. Mean field games. *Japanese journal of mathematics*, 2(1):229–260, 2007.

[31] D. S. Leslie and E. J. Collins. Individual q-learning in normal form games. *SIAM Journal on Control and Optimization*, 44(2):495–514, 2005.

[32] N. Li and J. R. Marden. Designing games for distributed optimization. *IEEE Journal of Selected Topics in Signal Processing*, 7(2):230–242, 2013.

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

[34] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

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

[36] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[37] J. R. Marden, G. Arslan, and J. S. Shamma. Connections between cooperative control and potential games illustrated on the consensus problem. In *2007 European Control Conference (ECC)*, pages 4604–4611. IEEE, 2007.

[38] J. R. Marden, G. Arslan, and J. S. Shamma. Joint strategy fictitious play with inertia for potential games. *IEEE Transactions on Automatic Control*, 54(2):208–220, 2009.

[39] J. R. Marden and J. S. Shamma. Revisiting log-linear learning: Asynchrony, completeness and payoff-based implementation. *Games and Economic Behavior*, 75(2):788–808, 2012.

[40] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[41] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[42] P. Milgrom and J. Roberts. Rationalizability, learning, and equilibrium in games with strategic complementarities. *Econometrica: Journal of the Econometric Society*, pages 1255–1277, 1990.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[44] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[45] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[46] D. Monderer and L. S. Shapley. Potential games. *Games and economic behavior*, 14(1):124–143, 1996.

[47] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[48] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[49] J. F. Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[50] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč. Deep reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

[51] T. Oda and C. Joe-Wong. Movi: A model-free approach to dynamic fleet management. *IEEE INFOCOM*, 2018.

[52] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017.

[53] M. Pessiglione, B. Seymour, G. Flandin, R. J. Dolan, and C. D. Frith. Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans. *Nature*, 442(7106):1042, 2006.

[54] T. Poggio and Q. Liao. *Theory II: Landscape of the empirical risk in deep learning*. PhD thesis, Center for Brains, Minds and Machines (CBMM), arXiv, 2017.

[55] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[56] R. A. Rescorla, A. R. Wagner, et al. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

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

[58] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[59] W. Saad, Z. Han, H. V. Poor, and T. Başar. Game theoretic methods for the smart grid. *arXiv preprint arXiv:1202.0452*, 2012.

[60] W. H. Sandholm. Potential games with continuous player sets. *Journal of Economic theory*, 97(1):81–108, 2001.

[61] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[62] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[63] G. Scutari, S. Barbarossa, and D. P. Palomar. Potential games: A framework for vector power control problems with coupled constraints. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 4, pages IV–IV. IEEE, 2006.

[64] H. Sergiu and M.-c. Andreu. *Simple adaptive strategies: from regret-matching to uncoupled dynamics*, volume 4. World Scientific, 2013.

[65] L. S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

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

[67] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[68] C. Soto, B. Song, and A. K. Roy-Chowdhury. Distributed multi-target tracking in a self-configuring camera network. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1486–1493. IEEE, 2009.

[69] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[70] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[71] D. M. Topkis. Equilibrium points in nonzero-sum n-person submodular games. *Siam Journal on control and optimization*, 17(6):773–787, 1979.

[72] K. Tumer and A. K. Agogino. Time-extended policies in multi-agent reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1338–1339. IEEE Computer Society, 2004.

[73] E. Van der Pol and F. A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

[74] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[75] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[76] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[77] X. Vives. Nash equilibrium with strategic complementarities. *Journal of Mathematical Economics*, 19(3):305–321, 1990.

[78] X. Wang, N. Xiao, T. Wongpiromsarn, L. Xie, E. Frazzoli, and D. Rus. Distributed consensus in noncooperative congestion games: An application to road pricing. In *2013 10th IEEE International Conference on Control and Automation (ICCA)*, pages 1668–1673. IEEE, 2013.

[79] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[80] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific, 2002.

[81] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[82] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913. ACM, 2018.

[83] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

[84] H. P. Young. The evolution of conventions. *Econometrica: Journal of the Econometric Society*, pages 57–84, 1993.

[85] D. H. Zald, I. Boileau, W. El-Dearedy, R. Gunn, F. McGlone, G. S. Dichter, and A. Dagher. Dopamine transmission in the human striatum during monetary reward tasks. *Journal of Neuroscience*, 24(17):4105–4112, 2004.

[86] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2151–2159. ACM, 2017.