ANDRIY BURKOV

Adaptive Dynamics Learning and Q-initialization in the Context of Multiagent Learning

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en Informatique
pour l'obtention du grade de Maître ès sciences, (M.Sc.)

FACULTÉ DE SCIENCES ET GENIE
UNIVERSITÉ LAVAL
QUÉBEC

MAI 2007

Résumé

L'apprentissage multiagent est une direction prometteuse de la recherche récente et à venir dans le contexte des systèmes intelligents. Si le cas mono-agent a été beaucoup étudié pendant les deux dernières décennies, le cas multiagent a été peu étudié vu sa complexité. Lorsque plusieurs agents autonomes apprennent et agissent simultanément, l'environnement devient strictement imprévisible et toutes les suppositions qui sont faites dans le cas mono-agent, telles que la stationnarité et la propriété markovienne, s'avèrent souvent inapplicables dans le contexte multiagent. Dans ce travail de maîtrise nous étudions ce qui a été fait dans ce domaine de recherches jusqu'ici, et proposons une approche originale à l'apprentissage multiagent en présence d'agents adaptatifs. Nous expliquons pourquoi une telle approche donne les résultats prometteurs lorsqu'on la compare aux différentes autres approches existantes. Il convient de noter que l'un des problèmes les plus ardus des algorithmes modernes d'apprentissage multiagent réside dans leur complexité computationnelle qui est fort élevée. Ceci est dû au fait que la taille de l'espace d'états du problème multiagent est exponentiel en le nombre d'agents qui agissent dans cet environnement. Dans ce travail, nous proposons une nouvelle approche de la réduction de la complexité de l'apprentissage par renforcement multiagent. Une telle approche permet de réduire de manière significative la partie de l'espace d'états visitée par les agents pour apprendre une solution efficace. Nous évaluons ensuite nos algorithmes sur un ensemble d'essais empiriques et présentons des résultats théoriques préliminaires qui ne sont qu'une première étape pour former une base de la validité de nos approches de l'apprentissage multiagent.

Abstract

Multiagent learning is a promising direction of the modern and future research in the context of intelligent systems. While the single-agent case has been well studied in the last two decades, the multiagent case has not been broadly studied due to its complexity. When several autonomous agents learn and act simultaneously, the environment becomes strictly unpredictable and all assumptions that are made in single-agent case, such as stationarity and the Markovian property, often do not hold in the multiagent context. In this Master's work we study what has been done in this research field, and propose an original approach to multiagent learning in presence of adaptive agents. We explain why such an approach gives promising results by comparing it with other different existing approaches. It is important to note that one of the most challenging problems of all multiagent learning algorithms is their high computational complexity. This is due to the fact that the state space size of multiagent problem is exponential in the number of agents acting in the environment. In this work we propose a novel approach to the complexity reduction of the multiagent reinforcement learning. Such an approach permits to significantly reduce the part of the state space needed to be visited by the agents to learn an efficient solution. Then we evaluate our algorithms on a set of empirical tests and give a preliminary theoretical result, which is first step in forming the basis of validity of our approaches to multiagent learning.

Preface

This work would not have been possible without the support of my family: my wife Mariya, my little daughter Catherine and my parents. I could never thank them enough for their never-ending support and worm words. Also, I would like to acknowledge my research advisor, Dr. Brahim Chaib-draa, for his comprehension and useful comments to my ideas and, of course, for the opportunity to have done my Master's work in the DAMAS laboratory, one of the most dynamical research centers in intelligent systems in Québec and in Canada. I express my special gratitude to Dr. Mario Marchand and Dr. François Laviolette, professors of the Computer Science and Software Engineering Department, for their courses in Machine Learning and Graph Theory that I took as a part of my Master's program, which were especially useful for me to accomplish this work.

Andriy Burkov

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

General Introduction

1.1 Problems

In multiagent systems (MAS) a number of autonomous agents act simultaneously in a common environment. Generally, the effect of an action of an agent does not only depend on its action as soon as the other agents' actions can also have an influence on the environment's resulting state. In this case, an agent's current behavior may be dependent not only on its preferences about the environment, but also on its beliefs about preferences and/or beliefs of the other agents acting in this environment at the same time. Thus, if certain agents in a multiagent system do not follow a fixed (stationary, non-evolving in time) policy, we say that such an environment is non-stationary, and, therefore, the techniques widely used for single-agent learning and planning are not generally applicable in this multiagent context. Another major challenge of decision making in MAS is the dimensionality of the problem's state space. Indeed, the cardinality of the state space of a multiagent system grows exponentially with the number of agents as soon as each state is composed of the individual "positions" of all the agents acting in the environment.

1.2 Environment

Multiple state environments with an agent being able to transit between states by making actions are usually modeled as Markov Decision Processes (MDPs). This is a formal model to represent an environment that can have stochastic inter-state transition

rules and where each action has a reward associated with it. Those features make this model very attractive to represent complex environments in the sequential decision making problems.

Game theory notions and concepts (Fudenberg and Tirole, 1991) are widely used in economics to find different kinds of solutions in the situations where many interested parties act in a common environment. To obtain a model of economical interactions between multiple autonomous agents in a multi-state stochastic environment, the game theory notions have been extended to MDPs. This extension gave birth to the *stochastic games* framework (Littman, 1994), the rich game theoretical background that was was first defined by Shapley (1953). In this framework, each state of the system is considered as a strategic game and there is a transition function that maps each game and the agents' joint actions taken in this game to a next game according to some probability distribution. Two principal challenges that arise in such a framework, (1) the hard predictability (called non-stationarity) of the environment and (2) the state space growth, are both a consequence of the presence of more than one agent (Littman, 1994).

1.3 Approaches

In the literature, there are a number of successful attempts to reduce state space of the problem of single-agent planning in MDPs; a variety of methods has been proposed. One of them is the so called *heuristic search* (Barto et al., 1995). Essentially, heuristic search is a set of methods based on the knowledge of a heuristic function that can estimate the real utility of any visited state. Generally, if the heuristic function is sufficiently informative and satisfies certain conditions, then the algorithm using it does not need to visit the entire state space to find the solution. Unfortunately, in multiagent systems, in most cases, an explicit search in the state space is practically impossible. Indeed, the search assumes that the properties of the environment in each state are known to the agent. This is not the case, however, when there are several, possibly adversarial, agents affecting the environment and their policies (or, at least, rationality principles) are not mutually known. Since the centralized planning in this context is not always possible, agents are usually faced with the learning or *adaptation* problem.

Classically, the usual approach to the multiagent policy learning assumes that the agents, by means of interactions and/or by using preliminary knowledge about the reward functions of all players, would find an interdependent solution called "equilibrium". One of the most widely used concepts of equilibrium is the Nash equilibrium.

In this kind of equilibrium, each player in the game plays its best response to the other players' strategies and a unilateral deviation of a player from the equilibrium strategy decreases its own utility. There are two basic approaches to find a Nash equilibrium. The first one is a game theoretic approach which supposes the complete knowledge of the reward structure of the underlying game by all the agents. In such an approach, each agent computes an equilibrium, by using mathematical programming, and then all the agents play on it. But a problem arises when there are several equivalent equilibria in a game and the agents have computed the different ones. Another problem is that the agents, when computing an equilibrium, suppose that the other agents are rational and, thus, will also follow this solution. But what if certain agents are not rational, or play a fixed strategy, or evolve according to some fixed rules. Moreover, what if some agents know (or are able to deduct) this and could exploit this knowledge to augment their utilities? As yet, there is no equilibrium concept which can answer this question.

The second approach to find an equilibrium is the adaptive one. It assumes that the agents learn by adapting to each other in self-play (i.e., all agents use the same learning algorithm) and do not know the reward structure of the game. According to this approach, the agents are only able to make actions and observe their own rewards and, in some approaches, the actions made by others. It was shown that certain algorithms of this class converge to a Nash equilibrium (or to a utility that is equivalent to the utility of a Nash equilibrium) (Singh et al., 1994; Bowling and Veloso, 2002). Among these algorithms the most outstanding are Joint-Action Learning (Claus and Boutilier, 1998), Infinitesimal Gradient Ascent (IGA)[1] (Singh et al., 1994), Policy Hill-Climbing (PHC) (Bowling and Veloso, 2002) and Adaptive Play Q-learning (APQ) (Gies and Chaib-draa, 2006) (a Q-learning based extension of the Adaptive Play algorithm (Young, 1993)). The adaptive players[2] learn their policies separately from the maintenance of the beliefs about their counterparts' future actions and make their decisions based on that policy and the current belief. These decisions can be in pure (simple actions) or in mixed (probability distribution over actions) strategies depending on the algorithm in question. Recently, certain researchers question the necessity and the validity of the concept of equilibrium as the most important multiagent solution concept (Shoham et al., 2003). They rather point out the efficiency of a particular learning algorithm versus a certain class of counterparts.

[1] IGA is the only algorithm among those listed which requires a complete knowledge of the reward structure of the game and the current strategy of the opponent to calculate the gradient.

[2] To discriminate between adaptive player as a member of a class of learning agents and Young's Adaptive Player, we will write "adaptive player" or "adaptive algorithm" (in lower case) to denote the former and Adaptive Player (with a capital letter) for the latter.

1.4 Contributions

In this Master's work we make two contributions in the area of multiagent learning. First, we propose the Adaptive Dynamics Learner (ADL) algorithm, an effective algorithm for learning in presence of adaptive counterparts. The ADL algorithm is able to learn an efficient policy over the adaptation dynamics of its opponents' rather than over simple actions and beliefs. By so doing, it is able to exploit these dynamics in order to obtain a higher utility than any equilibrium strategy would provide. We tested this algorithm on a set of repeated matrix games form GAMUT game theoretic test suite (Nudelman et al., 2004). The results obtained in these experiments indicate that ADL agent is highly effective in self-play and against APQ and IGA agents.

Our second contribution is a complexity reduction approach to multiagent learning in a particular stochastic games context, namely, in goal-directed stochastic games with action-penalty representation, one of the most important case for practical, cooperative multi-robot tasks. In this context, all agents have their respective goals and the rewards of making an action are negative in all the states except the goal state. To reduce the learning complexity, we use single-agent planning results as a heuristic function to initialize the agents' initial preferences in all unknown states of multiagent environment. The idea is to focus the multiagent learning process on a relatively small relevant region of the entire state space and, by so doing, to reduce the computation time required to learn a multiagent solution. We also proved theoretically the correctness of such an initialization. Section 4.2 of Chapter 4 contains the proofs of admissibility and monotonicity (consistence) of the proposed heuristic function. These two properties are important conditions of tractability of Q-learning.

Here we outline the chapters that follow.

In Chapter 2 we give a detailed description of the stochastic game framework, which is used as a model to represent environments in multiagent learning problems. We start by introducing agent and multiagent frameworks, MDPs, matrix games and repeated matrix games, and, finally, stochastic games, as a naturel generalization of previous models.

In Chapter 3 we observe the most important learning algorithms for stochastic games and point out their requirements and limitations.

In Chapter 4 we introduce two approaches to multiagent learning. The first one is an approach to the complexity reduction of reinforcement learning in the multiagent

context. The second is an approach of effective learning in presence of adaptive counterparts. We also present here some preliminary theoretical results.

In Chapter 5 we present the test benches we used to empirically test our algorithms. We then show the results obtained in these tests and discuss them.

In Chapter 6 we conclude by summarizing the contributions we made in this work. We analyse our approaches in the context of the related work made in similar research directions in the last several years and give a brief consideration of our future work.

Chapter 2

The Multiagent Learning Framework

2.1 Introduction

In this chapter we give a detailed overview of the *stochastic game framework*, a model used by most state-of-the-art multiagent learning algorithms to represent the environment and agent interactions. Despite the fact that the stochastic game framework is relatively simple to understand, it builds on several distinct notions: agent and multiagent frameworks, Markov Decision Processes, matrix games, and repeated games. We will then show that stochastic games is a natural generalization of these models.

2.2 Agent and Multiagent Frameworks

According to Russell and Norvig (2005), an *agent* is "anything that can be viewed as perceiving its *environment* through *sensors* and action upon that environment through *actuators*" (Figure 2.1).

It is required to note here that environment is a part of external world, which can be perceived by the agent through its sensors. At each moment of time, the environment can be in one of the finite or infinite number of *states*. The state of the environment is defined by the *values* of characteristics of the external world that can be observed (measured) by the agent's sensors.

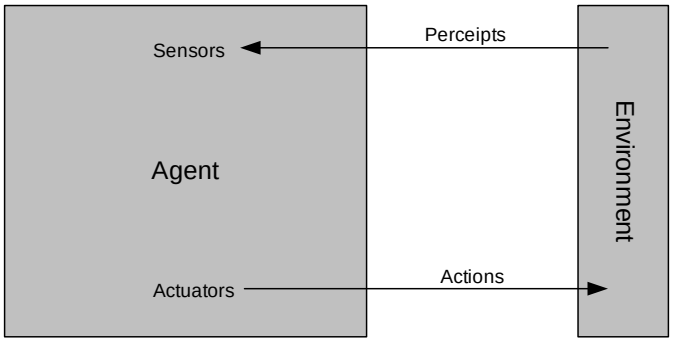


Figure 2.1: Agent framework

An agent is supposed to be capable of changing the environment's state by making *actions* though its actuators. These actions can trigger a *transition* between two states.

When there are several agents in one environment and these agents are able to observe and influence each other by making actions, such a framework is called multiagent. In the multiagent framework, the agents can have different nature, rationality principles, intentions and goals, and the problem of learning, which is the focus of this work, is to develop a *policy* of best actions in each environment's state.

A state in the multiagent framework is defined as a *joint state* of all agents in the common environment, i.e., it is a vector, each dimension of which is the environment's state from the point of view a particular agent. As soon as we are focused on the learning problem, we suppose that the set of joint states where the agents are intended to act is not known in advance to the learning agents. Furthermore, the agents cannot share any knowledge obtained during learning, i.e., they cannot communicate.

A more precise definition should be given about the observability of the environment where the agents are learning. We suppose that the environment is completely observable, i.e., agents exactly know in each joint state they are at each moment of time. Although partial observability of the environment is a more general setting (i.e., agents are always unsure where exactly they are due to the sensors' noise, for example), our main goal is to show how the multiagent setting affects learning principles and how multiagent algorithms should be designed to cope with the existence of other learning agents. Making these algorithms work in partial observable context would make these algorithms too complex or even intractable in the very small environments. First, because as yet there are no efficient algorithms of learning for partially observable environments, even for the single-agent case. Second, even if there is a model of the multiagent environment, it is known (Bernstein et al., 2003) that the complexity

of optimal decision making in a cooperative multiagent partially observable setting is NEXP-hard. Thus, we will limit ourselves to the case where the agents are able to unambiguously perceive their states in the environment.

2.3 Markov Decision Processes

2.3.1 Markov Assumption

When we presented the agent framework, we said that the agent can make actions which trigger transitions between environment's states. The goal of the agent, thus, is to find a best action to execute in each state of the environment. We say that the environment is Markovian (or stationary) if the agent's decision in a state depends on that state only and need not depend on the previously visited states or previously made actions. This assumption about the environment is called *Markov assumption* in honor of Russian mathematician, which was first to introduce and to study this kind of processes in depth.

2.3.2 MDPs

Markov Decision Process (MDP) (Russell and Norvig, 2005) is a single-agent environment, which has a Markovian inter-state transition model, additive rewards, and the state where the agent finds itself at each moment of time is fully observable, i.e., in any time the agent is sure in which state it is. More formally, an MDP is defined as a tuple, $(\mathcal{S}, \mathcal{A}, T, R)$, where,

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- T is the transition function, $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, where $T(s, a, s')$ defines the probability to finish the transition in the state s' starting in the state s and by making the action a,
- R is the reward function, $\mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, where $R(s, a)$ is the reward the agent obtains by making action a in state s
- and $s_0 \in \mathcal{S}$ is the initial state.

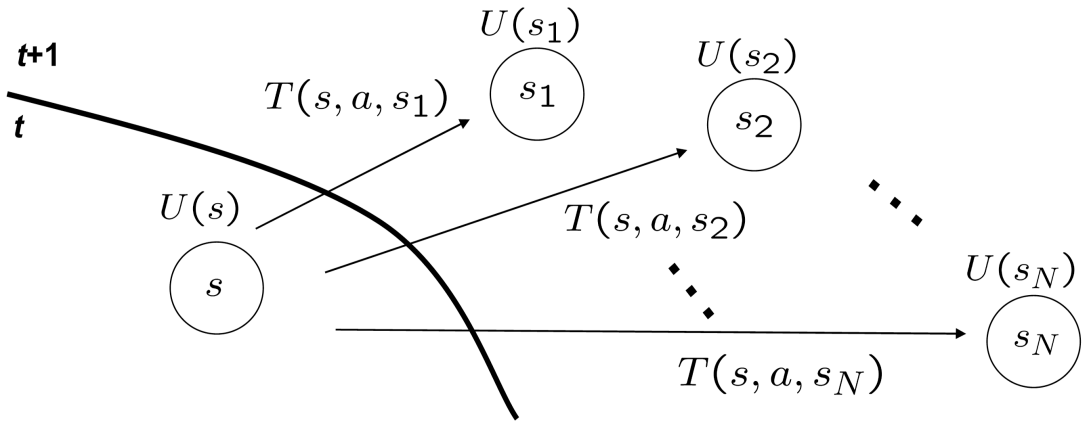


Figure 2.2: An example of a transition in Markov Decision Processes.

The transition function in an MPD is depicted in Figure 2.2. Given an action a played in a state s at time t, the agent transits to one of the states $s_1 \dots s_N$, $N = |\mathcal{S}|$, according to the probability distribution defined by the transition function T.

A solution in MDPs is called policy. A policy π assigns an action to each possible state. If we let s_t be the state the agent is in after executing π during t steps, a utility $U(\pi(s))$ of a policy π in a state s is

$$U(\pi(s)) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | s_0 = s\right] \tag{2.1}$$

where γ is the discount factor, a number between 0 and 1, which describes the preference of the agent for current reward over future rewards.

MDPs are well studied from the single-agent reinforcement learning perspective (Sutton and Barto, 1998). The most important result obtained is that in MDPs there is always a stationary (Markovian) and deterministic optimal policy, and there are reinforcement learning algorithms, such as Q-learning (Watkins and Dayan, 1992), that are guaranteed to find it. Notice that a deterministic policy assigns actions to the states. A non-deterministic, or stochastic policy assigns to each state a probability distribution over a set of actions.

2.3.3 Q-learning

Q-learning (Watkins and Dayan, 1992) is a dynamic programming method of learning in MDPs that consists of calculating the utility of an action in a state by interacting

with the environment. More formally, the goal of Q-learning is to create a function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ assigning to each state-action pair a Q-value, $Q(s, a)$, that corresponds to the agent's expected reward of executing an action a in a state s and following infinitely an optimal policy starting from the next state s':

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a} Q(a, s') \tag{2.2}$$

where $\gamma \in (0, 1]$ is the discount factor.

Equation (2.2) is often called Bellman equation in honor of mathematician Richard Bellman, celebrated for his invention of dynamic programming in 1953, and important contributions in other fields of mathematics.

Since the transition function, T, is not known for the learning agent, Q-learning consists of estimating the real Q-value $Q(s, a)$ by executing action a in state s of the environment, observing the reward $R(s, a)$ obtained and the system's next state s', using the following update rule:

$$\hat{Q}(s, a) \leftarrow (1 - \alpha)\hat{Q}(s, a) + \alpha[R(s, a) + \gamma \max_{a} \hat{Q}(s', a)] \tag{2.3}$$

where $\hat{Q}(s, a)$ is an estimated value of $\overset{\star}{Q}(s, a)$ and $\alpha \in [0, 1]$ is the learning rate. All along the learning process, the agent selects actions to execute in each state by maximizing the Q-value in that state with some stochastic exploration which decreases over time. Obviously, in that case the agent is strictly risk-neutral as soon as it tries to maximize its *expected* total reward. The convergence of the estimated Q-values, $\hat{Q}(s, a)$, to their optimal values, $Q(s, a)$, was proven by Watkins and Dayan (1992) under the conditions that each state-action pair is updated infinitely often, rewards are bounded and α tends asymptotically to 0.

2.4 Matrix Games

A *one stage matrix game* (also called "strategic game" or "normal form game") is a tuple $(n, \mathcal{A}^{1\ldots n}, R^{1\ldots n})$, where,

- n is the number of players,
- $\mathcal{A}^j$ is the strategy space of player $j, j = 1 \ldots n$, i.e., $\mathcal{A}^j$ is the set of actions $\{a_1^j \ldots a_{|}^j \mathcal{A}^j|\}$ the agent j can do.

- and the reward function $R^j : \mathbf{A} \mapsto \mathbb{R}$ defines the immediate utility for player j of a joint action $\mathbf{a} \in \mathbf{A} = \mathcal{A}^1 \times \ldots \times \mathcal{A}^n$, represented as a vector[1].

Matrix games differ according to the reward structure. There are three major classes of games that have different properties:

1. *strictly collaborative* games (or *team* games), where all players obtain the same reward when a joint action is played,
2. *strictly adversarial* (or *strictly competitive*) two-player zero-sum games, where there are only two players and, for all joint actions, the payoff of one player is the negative payoff of the other (i.e., $R^1 = -R^2$) and
3. the most wide class of matrix games, so-called *general-sum* games, which, in fact, include first two classes as special cases.

A *mixed* strategy for player j is a distribution π^j, where $\pi^j_{a^j}$ is the probability for player j to select some action a^j. A strategy is *pure* if $\pi^j_{a^j} = 1$ for some a^j. A *strategy profile* is a collection $\Pi = \{\pi^j | j = 1 \ldots n\}$ of all players' strategies. A *reduced profile for player* j, $\Pi^{-j} = \Pi \backslash \{\pi^j\}$, is a strategy profile containing strategies of all players except j, and $\Pi^{-j}_{\mathbf{a}^{-j}}$ is the probability for players $k \neq j$ to play a joint action $\mathbf{a}^{-j} \in \mathbf{A}^{-j} = \times_{-j}\mathcal{A}^{-j}$ where $\mathbf{a}^{-j}$ is $(a^k | k = 1 \ldots n, k \neq j)$. Given a player j and a reduced profile Π^{-j}, a strategy $\hat{\pi}^j$ is a *best response* (BR) to Π^{-j} if the expected utility of the strategy profile $\Pi^{-j} \cup \{\hat{\pi}^j\}$ is maximal for player j. Since a best response may not to be unique, there is a set of best responses of player j to a reduced profile Π^{-j} which is denoted as $BR^j(\Pi^{-j})$. More formally, the expected utility of a strategy profile Π for a player j is given by:

$$U^j(\Pi) = \sum_{a^j \in \mathcal{A}^j} \pi^j_{a^j} \sum_{\mathbf{A}^{-j} \in \mathbf{A}^{-j}} R^j((a^j, \mathbf{a}^{-j}))\Pi^{-j}_{\mathbf{a}^{-j}} \tag{2.4}$$

where Π is $\Pi^{-j} \cup \{\pi^j\}$ and $R^j((a^j, \mathbf{a}^{-j}))$ is the value that player j receives if the joint action $\mathbf{a} = (a^j, \mathbf{a}^{-j})$ is played by all players. In this case, a best response of player j to the reduced profile Π^{-j} is a strategy $\hat{\pi}^j$ such that:

$$U^j(\Pi^{-j} \cup \{\hat{\pi}^j\}) \geq U^j(\Pi^{-j} \cup \{\pi^j\}), \quad \forall \pi^j \neq \hat{\pi}^j$$

A solution from the game theoretic perspective if a strategy profile having some properties. One game can have different solutions depending on which properties a

[1] henceforward we will denote vectors using bold characters

strategy profile has. Adopting a best response from the game theoretic point of view refers to the notion of *equilibrium*, which is viewed as a solution concept. In general, an equilibrium in a game is a strategy profile possessing a certain degree of stability.

There may exist several equilibrium solutions in one game depending on which kind of equilibrium the players have found. We say that an equilibrium is *Pareto optimal* if it maximizes the expected value of all players in the game. Formally, a strategy profile $\bar{\Pi}$ is Pareto optimal if and only if:

$$\forall\ \Pi \neq \bar{\Pi} \quad \exists j \text{ such that } U^j(\Pi) < U^j(\bar{\Pi})$$

2.4.1 Nash equilibrium

We say that a strategy profile $\hat{\Pi}$ forms a *Nash equilibrium* if a *unilateral* deviation of a player j from $\hat{\Pi}$ does not increase its own expected utility, or, in other words, $\hat{\Pi}$ is a *Nash equilibrium* if and only if for each player j its strategy $\hat{\pi}^j \in \hat{\Pi}$ is a best response to the reduced profile $\hat{\Pi}^{-j}$, that is:

$$\hat{\pi}^j \in BR^j(\hat{\Pi}^{-j}), \quad \forall j$$

It is was shown by Nash (1950) that every matrix game has at least one Nash equilibrium in mixed strategies, but it may have no equilibrium in pure strategies or, to the contrary, to have several equilibria. An equilibrium strategy $\hat{\Pi}_1$ is said to be *Pareto dominated* by another equilibrium, say $\hat{\Pi}_2$, if the utility of $\hat{\Pi}_2$ for all players is not lower than the utility of $\hat{\Pi}_1$ and there exists a player for which $\hat{\Pi}_2$ brings a higher utility than $\hat{\Pi}_1$. More formally, if a strategy profile $\hat{\Pi}_1$ is an equilibrium, then it said to be Pareto dominated by an equilibrium strategy $\hat{\Pi}_2$ if and only if:

$$U^j(\hat{\Pi}_2) \geq U^j(\hat{\Pi}_1),\ \forall j \text{ and}$$

$$\exists j\ U^j(\hat{\Pi}_2) > U^j(\hat{\Pi}_1)$$

The examples of games of different reward structures are presented in Figure 2.3 (these games were generated using GAMUT (Nudelman et al., 2004), a test suite for game theoretic algorithms).

As soon as in the two-player case all rewards may be contained in a two-dimensional matrix, we often call the first player *row player*, since it selects its actions from the set of the rows of matrix. The second player is often called *column player*, because it usually selects its actions from the columns of the payoff matrix. In Figure 2.3, RockPaperScis-

MatchingPennies

$$R^r, R^c = \begin{bmatrix} 1,-1 & -1,1 \\ -1,1 & 1,-1 \end{bmatrix}$$

PrisonersDilemma

$$R^r, R^c = \begin{bmatrix} 0.37,0.37 & 1,-1 \\ -1,1 & -0.9,-0.9 \end{bmatrix}$$

BattleOfTheSexes

$$R^r, R^c = \begin{bmatrix} 1,0.67 & -1,-1 \\ -1,-1 & 0.67,1 \end{bmatrix}$$

CoordinationGame

$$R^r, R^c = \begin{bmatrix} 0.8,0.8 & -0.8,-0.8 \\ -0.5,-0.5 & 0.7,0.7 \end{bmatrix}$$

ShapleysGame

$$R^r, R^c = \begin{bmatrix} -1,-1 & 1,-1 & -1,1 \\ -1,1 & -1,-1 & 1,-1 \\ 1,-1 & -1,1 & -1,-1 \end{bmatrix}$$

RockPaperScissors

$$R^r, R^c = \begin{bmatrix} 0,0 & 1,-1 & -1,1 \\ -1,1 & 0,0 & 1,-1 \\ 1,-1 & -1,1 & 0,0 \end{bmatrix}$$

Figure 2.3: A set of games from GAMUT. R^r, R^c is the payoff matrix, the entries of which contain the payoffs for the row and column players respectively.

sors and MatchingPennies games are strictly adversarial and ShapleysGame is a general-sum adversarial game with the only mixed strategy equilibria; BattleOfTheSexes and CoordinationGame are general sum *coordination games*, with pure and mixed strategy equilibria. PrisonersDilemma is, probably, the most explored general sum game in Game Theory. In this game, there is a pure equilibrium strategy, $(2, 2)$, which is Pareto dominated by a non-equilibrium Pareto optimal strategy, $(1, 1)$. We will return to this game and its properties in the following section.

2.4.2 Minimax

To find an equilibrium in a two-player zero-sum game, a special technique, called *minimax*, can be used. Obviously, in such a game (when a gain of one player is the loss of the second) one player may be considered as *maximizer*, i.e., which tends to maximize its reward, and, thus, the second player is then considered as *minimizer* in the sense that it tends to minimize the reward of its adversary. In this case an equilibrium can be found if maximizer plays the *maximin* strategy and minimizer plays the *minimax* strategy:

$$\pi^j(\mathbf{s}) = \underset{\pi^j \in PD(\mathcal{A}^j)}{\operatorname{argmax}} \min_{a^{-j}} \sum_{a^j} \pi^j_{a^j} R^j(a^j, a^{-j})$$

where $PD(\mathcal{A}^j)$ denotes the set of probability distributions (possibly infinite) defined over actions in $\mathcal{A}^j$.

Von Neumann proved the Minimax theorem (Fudenberg and Tirole, 1991), stating that such strategies always exist in two-person zero-sum games and can be found by solving a set of simultaneous equations.

2.5 Repeated Matrix Games

Many new interesting properties appear when a matrix game is repeated finitely or infinitely often. A *repeated matrix game* is a matrix game in which the same players play it iteratively a number of times T, $0 < T < \infty$.

The fact of having an infinite number of plays affects the notion of equilibrium. Indeed, in this case, by speaking informally, rational players "know" that after a play there will necessarily be another play, and, thus, it may be rational to choose a strategy that maximizes the long-term utility instead of short-term reward. Let us show this on the example of the repeated PrisonersDilemma game (Poundstone, 1992). We first describe it informally.

Let's suppose that two suspects, A and B, are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal: if one testifies for the prosecution against the other and the other remains silent, the betrayer receive a minor sentence (three month) and the silent accomplice receives the full 10-year sentence. If both stay silent, the police can sentence both prisoners to only one year in jail for a minor charge. If each betrays the other, each will receive an eight-year sentence. Each prisoner must make the choice of whether to betray the other or to remain silent (Figure 2.4). However, neither prisoner knows for sure what choice the other prisoner will make. The payoff matrix for this game that was already presented in Figure 2.3 corresponds to the same game but the values in the matrix may be considered as the suspect's personal utilities of the corresponding sentences.

		Suspect B	
		Stays Silent	Betrays
Suspect A	Stays Silent	1 year, 1 year	10 years, 3 months
	Betrays	3 months, 10 years	8 years, 8 years

Figure 2.4: PrisonersDilemma game.

It is easy to see that in the *one shot* PrisonersDilemma there is the only pure strategy

Nash equilibrium (Betrays, Betrays) which brings to both players a very undesirable sentence, 8 years. The only Pareto optimal strategy, namely (Stays Silent, Stays Silent) brings them much higher utility, but it is not an equilibrium and, thus, each player have an interest of unilateral deviation from this solution to improve its own situation. But what if the game was repeated infinitely often and players could see the choice made (the strategy played) by the other player after each play? Will the equilibrium strategy of the one shot game be rational in this infinitely *repeated game*? The answer is *no*, and the reason is that the players know that after playing a single stage game they will play it again, and, thus, the decision made bay a player in the past may affect the decisions of its opponent in the future.

One of the most known stable strategies that brings the Pareto optimal utility in the infinitely repeated PrisonersDilemma is Tit-for-Tat (TFT) strategy. It consists in playing Stays Silent in the first round of the game and continuing by repeating the most recent action of the opponent in the subsequent rounds. Let us suppose that on step $t = 0$ the agents have chosen and played a strategy (Stays Silent, Stays Silent), which is Pareto optimal and non-equilibrium in the one stage game. On the next stage $t + 1$ they have a choice: to deviate from this strategy or to keep it. If one of the agents decides to deviate (i.e., plays Betrays), it augments its own immediate utility, but it knows that on the following step $t + 2$ the other agent will do the same (i.e., play Betrays) to minimize its loss, and, thus, both will deviate to the single stage game equilibrium strategy (Betrays, Betrays) which brings lower reward than the initial (Stays Silent, Stays Silent) one. Thus, in the infinitely repeated case, if the other player plays TFT, it is non rational to deviate from cooperation strategy (Stays Silent, Stays Silent). Therefore, it follows that the strategy (Stays Silent, Stays Silent) is a Pareto optimal equilibrium strategy for the repeated game.

It is important to note, that the properties of repeated matrix games are very similar to the following, more general class of games, called *stochastic games* which are a generalization of the MDPs to the multiagent case where each state of the system is considered as a repeated matrix game.

2.6 Stochastic Games

Stochastic games (SGs) combine MDPs and repeated matrix games. A stochastic game is a tuple $(n, \mathbf{S}, \mathcal{A}^{1\cdots n}, T, R^{1\cdots n})$, where,

- n is the number of agents,

- $\mathbf{S}$ is the set of joint states, $\mathbf{s} \in \mathbf{S}$, which are now represented as vectors,
- $\mathcal{A}^j$ is the set of actions, $a^j \in \mathcal{A}^j$, available to agent j,
- $\mathbf{A}$ is the joint action space, $\mathcal{A}^1 \times \ldots \times \mathcal{A}^n$,
- T is the transition function: $\mathbf{S} \times \mathbf{A} \times \mathbf{S} \mapsto [0, 1]$,
- R^j is the reward function for agent j: $\mathbf{S} \times \mathbf{A} \mapsto \mathbb{R}$
- and $\mathbf{s}_0 \in \mathbf{S}$ is the agents' (or system's) initial joint state.

Since there are multiple agents selecting actions, the agents' next state and rewards depend on the joint actions of all players[2]. It's easy to see that if, in an SG, there is only one player then this SG becomes an MDP. The goal of each agent in an SG is to maximize its expected utility of being in this game. In the stochastic game framework the "expected utility" is a combination of two expectations in the sense that the agents in an SG aim to maximize their expected utilities over other players' joint strategy in each stage game (state), and their temporal utility over all future games. Formally, for an agent j, the discounted utility U^j of a state $\mathbf{s}$ of an SG is defined as follows:

$$\begin{aligned} U^j(\Pi(\mathbf{s})) &= E\left[\sum_{t=0}^{\infty} \gamma^t u^j(\Pi(\mathbf{s}_t)) | \mathbf{s}_0 = \mathbf{s}\right] \\ &= u^j(\Pi(\mathbf{s})) + \gamma \sum_{\mathbf{s}' \in \mathbf{S}} T(\mathbf{s}, \Pi(\mathbf{s}), \mathbf{s}') U^j(\Pi(\mathbf{s}')) \end{aligned} \tag{2.5}$$

where $u^j(\Pi(\mathbf{s}))$ is the "immediate" expected utility of a stage game $\mathbf{s}_t$ for the agent j, as defined by equation 2.4. Π is the policy of joint strategies of players, which defines a strategy profile $\Pi(\mathbf{s})$ for each state $\mathbf{s} \in \mathbf{S}$.

A transition in a stochastic game is presented in Figure 2.5. Given a joint action (a^1, a^2) played in a state game $\mathbf{s}$ at time t, the transition of agents is made to one of the state games $\mathbf{s}_1 \ldots \mathbf{s}_N$, $N = |\mathbf{S}|$, according to the probability distribution defined by the transition function T.

The concept of equilibrium also extends to stochastic games. This is a non-trivial result, proven by Shapley (1953) for zero-sum stochastic games and by Fink (1964) for general-sum stochastic games. In SGs a policy Π is a Nash equilibrium if and only if in each state $\mathbf{s} \in \mathbf{S}$ the strategy profile, $\Pi(\mathbf{s})$, forms a Nash equilibrium.

[2]In the SG framework the terms *agent* and *player* mean the same.

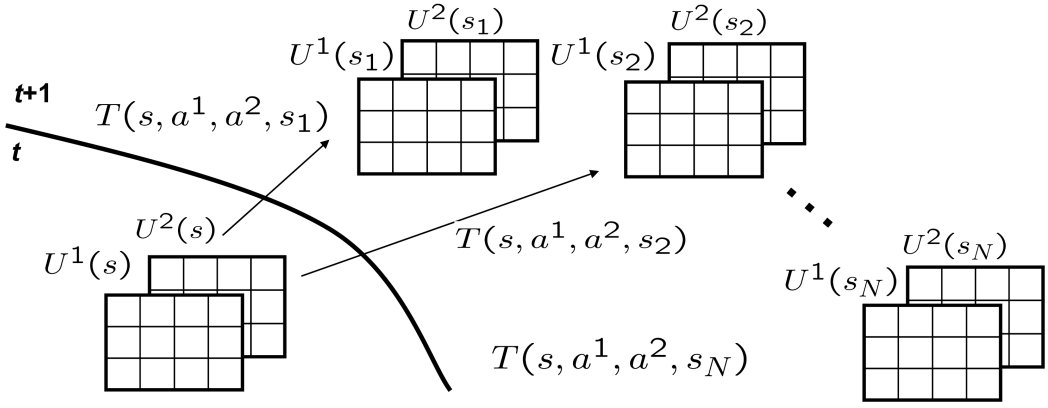


Figure 2.5: An example of a transition in a stochastic game.

2.6.1 Examples

Two-Player Grid Soccer

Littman (1994) proposed robotic soccer as an example of a two-player adversarial stochastic game (Figure 2.6). The game is played on a 4×5 grid. The players, A and B, occupy distinct cells of the grid and can choose between five available actions, *north*, *south*, *west*, *east* and *do nothing*, which, in general case, can have stochastic effect, i.e., as soon as the players have chosen and executed their actions, the they can move in the intended direction with certain probability or to stay put. The circle in Figure 2.6 represents the ball. When the player possessing the ball enter the appropriate goal cell (left for A and right for B), this player obtain a reward $+1$, the other player gain -1, and the game is reset to the initial state. When a player executes an action that can take it to the cell occupied by the other player, possession of the ball goes to the stationary player and the first player keeps its position by loosing the ball. Thus a "good" maneuver to take off the ball is to stand where the other player is supposed to go on the next time step.

Two-Player Coordination Problem

Hu and Wellman (2003) proposed a coordination problem, also called "grid world" problem or "two-robot-on-the-grid" problem, as an example of a two-player coordination stochastic game (Figure 2.7). In their example, there are two robots, 1 and 2, on a grid, each robot has its start position and pursues its own goal. Robots can

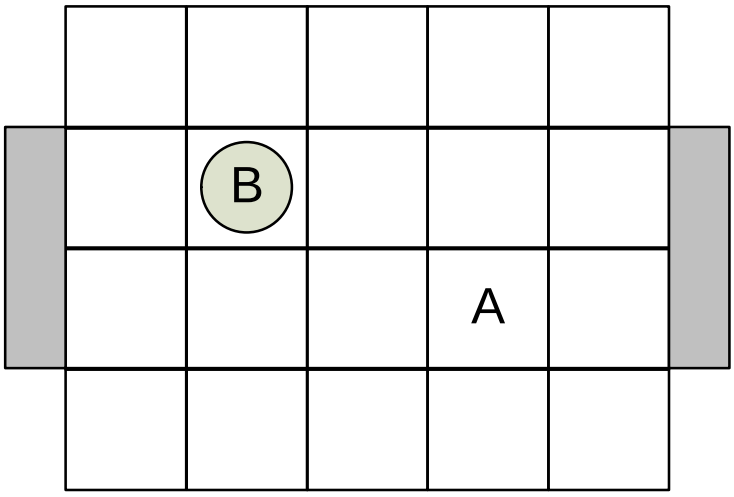


Figure 2.6: Littman's robotic soccer.

make transitions between two adjacent cells of the grid by making actions *north, south, west* and *east* which, in general case, have stochastic effect. If the action taken has success ($Pr(success) = 0.99$), robot changes its position on the grid to the intended cell, otherwise ($Pr(failure) = 0.01$) its position remains unchanged. Each action has the cost of 0.04 (reward of -0.04) in any cell except the goal cell where the cost of all actions is 0. In case of collision no transition is made and both robots lose the value of 0.1 (gain -0.1) which we will call "collision loss". Thus, robots are interested in attainment of their respective goals by making minimum number of transitions and avoiding collisions.

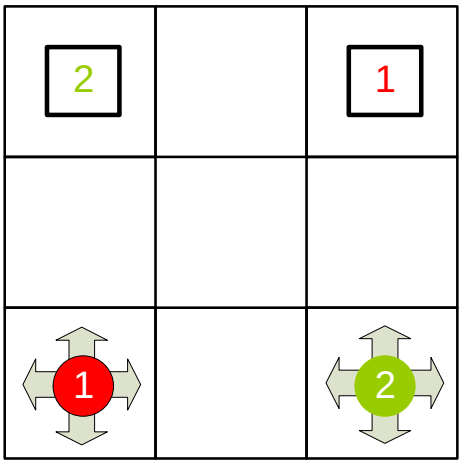


Figure 2.7: Hu and Wellman's grid world.

As it was mentioned above, stochastic games are an extension of MDPs to the multiple agents case. On the other hand, SGs can be considered as an extension of

the repeated matrix games to the multi-state context. In each state of an SG, the agents' strategies are considered as joint-actions that change the system's state with respect to some, possibly stochastic, transition rule. Indeed, if in an SG there is only one state, i.e., $|\mathbf{S}| = 1$, then this SG becomes a repeated matrix game. On the other part, if there is only one player in an SG, then this SG becomes a simple MDP. One more case where an SG may be considered as an MDP is when all the agents but one follow a fixed, stationary and, possibly, mixed policy[3]. In that case the problem for the remaining agent is restricted to an MDP because the strategies of the other agents may be considered as a part of the transition rule of the environment.

As well as in the case of matrix games, the existence of at least one Nash equilibrium was proven for stochastic games (Shapley, 1953). Similarly, stochastic games have the same classification depending on the reward structure of the problem. There are strictly collaborative (team games), strictly competitive (zero-sum) games and, obviously, general-sum games. Evidently, the finding an equilibrium strategy in an SG is a much complex task than it is for the matrix games.

2.7 Conclusion

In this chapter we presented the stochastic game framework, as a model of environment that enables us to represent multiagent interactions. Furthermore, as we will show later, this model will allows us to perform learning in presence of multiple learning agents.

We also gave a survey of previous frameworks, such as the agent framework, Markov Decision Processes and matrix games, and defined the principles of planning and learning tasks in agent-oriented and multiagent contexts.

In the next chapter we will present the most interesting, in our opinion, state-of-the-art algorithms of learning in stochastic games and will point out their advantages and major limitations.

[3] A policy in SGs corresponds to a set of strategies to play in each state.

Chapter 3

Learning in Stochastic Games

3.1 Introduction

In this chapter we describe in detail the most widely used state-of-the-art learning algorithms for stochastic games. There were a number of learning algorithms proposed for stochastic games in the last years. These algorithms have two common parts, (1) an MDP solving part and (2) a game theoretic part. A game theoretic part of an algorithm is used to take into account the existence and the strategies of the other players. A MDP solving part is typically based on a *temporal differences* algorithm (Sutton, 1988), usually Q-learning (Watkins and Dayan, 1992) or its different modifications. The goal of this MDP solving part is to obtain a multiagent algorithm to solve sequential decision problems where decisions may have a long term effect.

In this chapter we present the following state-of-the-art multiagent algorithms: Q-learning (a simple extension of the single-agent learning technique without the game theoretic part), Minimax-Q, Nash-Q, Friend-or-Foe Q-learning, Infinitesimal Gradient Ascent (IGA), Policy Hill Climbing (PHC) and (its convergent version WoLF-PHC), Joint-Action Learners (JALs), Adaptive Play Q-Learning (APQ), and the most recent PHC-Exploiter and Hyper-Q. These algorithms can be divided onto five classes, depending on their nature:

1. Single-agent techniques (such as single-agent Q-learning),
2. Equilibrium Learning Algorithms,
3. Gradient Based Algorithm,

4. Opponent Modelling Algorithms,

5. Adaptivity Modelling Algorithms.

All algorithms in these classes have both their merits and drawbacks that we will discuss in this chapter.

3.2 *Q*-learning

A *single-agent* version of Q-learning (Watkins and Dayan, 1992) was presented in chapter 2. A multiagent Q-learning differs from the single-agent one in assigning Q-values to joint states instead of single-agent states of each agent. The Q-value update in multi-agent case can be written as follows:

$$\hat{Q}^j(\mathbf{s}_t, a^j) \leftarrow (1-\alpha)\hat{Q}^j(\mathbf{s}_t, a^j) + \alpha\left(R^j(\mathbf{s}_t, a^j) + \gamma \max_{a^j} \hat{Q}^j(\mathbf{s}_{t+1}, a^j)\right) \tag{3.1}$$

As was already mentioned above, in presence of stationary opponents, an SG become an MDP and, thus, may be solved using a single-agent learning technique, such as Q-learning. However, if all the agents are learning at the same time, Q-learning is not guaranteed to converge to a "good" solution because the policies of all the leaning agents are non-stationary and, thus, the stochastic games cannot be represented as an MDP, as soon as the Markov property does not hold. Despite this fact, there were interesting attempts to directly apply the single-agent Q-learning to the multiagent context (Tan, 1993; Sen et al., 1994).

Another shortcoming of Q-learning applied to the multiagent learning context is its incapability to learn mixed (stochastic) policies. For some games, such as zero-sum matrix games and some other non-cooperative games the only "satisfactory" strategy is to play stochastically with mixed strategies. However, the Q-learning is unable to learn such strategies. On contrary, there exist single-agent learning techniques capable of learning mixed stochastic policies, but this is done to treat the problems of partial observability or the function approximation problems and cannot be applied directly to play against intelligent opponents.

To cope with the existence of multiple learning agents in a multi-state environment, the techniques of game theory, such as equilibrium calculation and the opponent's strategy estimation, have been extended to the stochastic game framework.

3.3 Equilibrium Learning Algorithms

In this subsection we present three algorithms that relate to the class of algorithms called in literature "equilibrium learners". They focus their learning interest on the direct convergence to an equilibrium by constructing in each state a matrix game from the Q-values in this state, and by updating these Q-values as to make them converge to an equilibrium.

3.3.1 Minimax-Q

One of the very first algorithms having both multiagent learning parts (an MDP solving part and a game theoretic part) was Littman's Minimax-Q algorithm for zero-sum stochastic games. Minimax-Q consists in calculating of the unique Minimax equilibrium in all the state-games composed of the estimated Q-values of the joint actions. Thereupon, all agents play on that equilibrium. Minimax-Q was proven to be convergent in zero-sum stochastic games under the assumption that all the states are visited infinitely often (Littman and Szepesvári, 1996).

The Minimax-Q algorithm differs from the single-agent Q-learning algorithm (1) by considering Q-values of joint actions instead of personal actions and (2) by using of the $Value^j(\mathbf{s})$ function instead of max_{a^j} operator in the Q-value update rule (3.1). The $Value^j(\mathbf{s})$ is given by the minimax utility for player j in the game composed of the actual Q-values of the joint actions:

$$Value^j(\mathbf{s}) = \max_{\pi^j(\mathbf{s}) \in PD(\mathcal{A}^j)} \min_{\mathbf{a}^{-j}} \sum_{a^j} \pi^j_{a^j}(\mathbf{s}) \hat{Q}^j(\mathbf{s}, (a^j, \mathbf{a}^{-j}))$$

where $PD(\mathcal{A}^j)$ denotes the set of discrete probability distributions over the set $\mathcal{A}^j$ and $\pi^j_{a^j}(\mathbf{s})$ denotes the probability to play the action a^j in state $\mathbf{s}$ according to the probability distribution $\pi^j(\mathbf{s})$.

In this case, the Q-value update rule becomes as follows:

$$\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1-\alpha)\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left(R^j_t(\mathbf{s}_t, \mathbf{a}_t) + \gamma Value^j(\mathbf{s}_{t+1}) \right)$$

Littman calculated the $Value^j(\mathbf{s})$ function using linear programming. The formal definition of the Minimax-Q algorithm is presented in Algorithm 3.1. In the lines 7-10 the learning is initialized. Until the maximum number of learning iterations is reached,

in each visited state the agent calculates the minimax equilibrium strategy of the game composed of Q-values (line 12) of this state. Then the agent plays this equilibrium strategy with some exploration (line 13). Having observed its reward, the joint action played and the new state, it updates the Q-value in previous state (lines 14-15). Finally, it updates its internal state and increments the iterations counter (lines 16-17) and goes on to the next iteration.

1: **function** MINIMAX-$Q(\mathbf{S}, \mathbf{A}, \mathbf{s}_0, t_{max})$
2: **returns:** a policy.
3: **inputs:** $\mathbf{S}$, the set of the multiagent states,
4: $\mathbf{A}$, the set of the joint actions,
5: $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
6: t_{max}, maximal number of learning iterations,

7: Initialize arbitrarily $\hat{Q}^j(\mathbf{s}, \mathbf{a}),\ \forall \mathbf{s},\ \forall \mathbf{a}$.
8: Initialize the learning rate $\alpha \in (0, 1]$.
9: Initialize the discount factor $\gamma \in (0, 1)$.
10: Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
11: **while** $t \leq t_{max}$ **do**
12: Calculate the minimax equilibrium strategy profile $\Pi(\mathbf{s})$ for the payoff matrix $[\hat{Q}^j(\mathbf{s}, \mathbf{a})_{\mathbf{a}\in\mathbf{A}}]$.
13: Play $\pi^j(\mathbf{s}) \in \Pi(\mathbf{s})$ with some exploration.
14: Observe the new state $\mathbf{s}_{t+1}$, the joint action played $\mathbf{a}_t$, and the reward obtained R_t^j.
15: Update $\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1-\alpha)\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) + \alpha\left[R_t^j + \gamma Value^j(\mathbf{s}_{t+1})\right].$$

16: Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
17: Increment the time $t \leftarrow t + 1$.
18: **return** current policy.

Algorithm 3.1: Minimax-Q algorithm for player j, adapted from(Littman, 1994).

Note that $[\hat{Q}^j(\mathbf{s}, \mathbf{a})_{\mathbf{a}\in\mathbf{A}}]$ at line 12 denotes the payoff matrix composed of Q-values because each Q-value is defined over joint actions. These joint actions define particular elements of the matrix, and the corresponding Q-values themselves can be considered as the payoffs contained in those elements.

The term "some exploration" used at line 13 refers to the using of any existing exploration technique for the reinforcement learning algorithms Thrun (1992).

As it was mentioned above, the Minimax-Q algorithm is guaranteed to converge to the equilibrium in zero-sum games if it is unique. If equilibrium is not unique, the Minimax-Q agent "converges to a policy that always achieves at least its optimal value regardless of its opponent" (Littman, 2001b).

However, the algorithm is not guaranteed to be rational in all cases. For example, in RockPaperScissors game, presented in Section 2.4, the Minimax equilibrium consists of playing *Rock*, *Paper* and *Scissors* strategies with probability $\frac{1}{3}$ and the Minimax-Q agent will find this solution even if the opponent is always playing *Rock* and hence always playing *Paper* is the rational strategy. The real applicability of the Minimax-Q algorithm is hence limited by strictly adversarial cases, such as robotic soccer stochastic game by Littman (1994).

3.3.2 Nash-Q

Nash-Q learning algorithm is an extension of the Minimax-Q algorithm to general-sum stochastic game framework, proposed by Hu and Wellman (2003). This algorithm requires that all players maintain the Q-values for all other players. The algorithm differs from the Minimax-Q in how the $Value^j$ function is calculated. In the Nash-Q algorithm, this function returns the utility of player j if a specified Nash equilibrium is played. Hu and Wellman used quadratic programming to compute Nash equilibria in all states.

The formal definition of the Nash-Q algorithm is presented in Algorithm 3.2. In the lines 9-12 the learning is initialized. Until the maximum number of learning iterations is reached the agent starts each iteration by calculating the Nash equilibrium strategy of the game composed of Q-values (line 14) and plays this equilibrium strategy with some exploration (line 15). Having observed its reward, the joint action played and the new state, it updates the Q-values in previous state (lines 16-17). Finally, it updates its internal state and increments the iterations counter (lines 18-19) and goes on to the next iteration.

It has been showed that the Nash-Q learning algorithm in a multi-agent environment converges to a Nash equilibrium policy under some conditions and assumptions about the payoff structures. In particular, it is necessary that the Nash equilibrium is unique or there are several equilibria with the same utilities. These assumptions are very hard to satisfy when designing a stochastic game of a particular reward structure because it can contain thousands of states and, thus, the equilibria in such a game will be strongly dependent on the long-term relations between the states.

function NASH-$Q(\mathbf{S}, \mathbf{A}, n, \mathbf{s}_0, t_{max}, j)$
  returns: a policy.
  inputs: $\mathbf{S}$, the set of the multiagent states,
    $\mathbf{A}$, the set of the joint actions,
    $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
    t_{max}, maximal number of learning iterations,
    n, the number of agents,
    j, the learning player's number, $j = 1 \dots n$.

  Initialize arbitrarily $\hat{Q}^i(\mathbf{s}, \mathbf{a}),\ \forall \mathbf{s},\ \forall \mathbf{a}\, \forall i$.
  Initialize the learning rate $\alpha \in (0, 1]$.
  Initialize the discount factor $\gamma \in (0, 1)$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    Calculate the Nash equilibrium strategy profile $\Pi(\mathbf{s})$ for the matrix game $[\hat{Q}^i(\mathbf{s}, \mathbf{a})^{i=1,\dots n}_{\mathbf{a}\in\mathbf{A}}]$.
    Play $\pi^j(\mathbf{s}) \in \Pi(\mathbf{s})$ with some exploration.
    Observe the new state $\mathbf{s}_{t+1}$, the joint action played $\mathbf{a}_t$, and the reward obtained by all the players $R^i_t,\ \forall i$.
    Update $\hat{Q}^i(\mathbf{s}_t, \mathbf{a}_t)$ using the following rule,

$$\hat{Q}^i(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1-\alpha)\hat{Q}^i(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left(R^i_t + \gamma Value^i([\hat{Q}^i(\mathbf{s}_{t+1}, \mathbf{a})_{\mathbf{a}\in\mathbf{A}}]) \right),\ \forall i.$$

    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$.
  return current policy.

Algorithm 3.2: Nash-Q algorithm for player j, adapted from (Hu and Wellman, 2003).

Friend-or-Foe Q-learning

As soon as the convergence conditions of the Nash-Q learning algorithm were strict enough and could be hardly satisfied in reality, Littman (2001a) developed a Friend-or-Foe Q-learning (FFQ) algorithm for the reinforcement learning in general-sum stochastic games that is based on the Nash-Q learning. The idea is that each agent in the system is identified as either "friend" or "foe" and, in consequence, each equilibrium is either coordinating or adversarial. The convergence guarantee of FFQ is in fact stronger than that of Nash-Q. In particular, it is guaranteed to converge if there is a one of these two types of equilibria regardless of opponent behavior.

This algorithm still can not be applied directly to the general-sum games because it requires satisfaction of several strict conditions:

1. the FFQ agent must know how many equilibria there are in the game and
2. an equilibrium should be known in advance to be either coordinating or adversarial.

In addition, FFQ learning does not provide a way to identify or classify a Nash equilibrium and, like Nash-Q-learning, it is not applicable to the systems where neither coordination nor adversarial equilibria exist or are known in advance. Thus, due to the fact that the algorithm has very limited application, we has decided not to describe it in more detail in our work.

3.4 Gradient Based Algorithm

The algorithms described in this section base their learning strategy on calculating or estimating the gradient of the utility of the currently playing policy. Thereupon, by using this gradient these algorithms update the current policy. They also use Q-learning to calculate the estimation of action utilities in each state of the game.

3.4.1 Infinitesimal Gradient Ascent

Singh et al. (1994) examined the dynamics of using the gradient ascent in two-player, two-action repeated games. The problem can be represented with two payoff matrices for the row and column players, r and c, as follows:

$$R^r = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \quad R^c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

The players r and c select simultaneously an action from the set $\mathcal{A}^{r,c} = \{1, 2\}$, the row player r plays an action i, the column player c selects an action j and the payoffs they obtain are R^r_{ij} and R^c_{ij} respectively.

Since this is two-action game, a mixed strategy of a player can be represented using a single value. Let $\alpha \in [0, 1]$ be a probability the player r selects the action 1 and

$1-\alpha$ the probability to play the action 2. Let, similarly, $\beta \in [0,1]$ and $1-\beta$ be the probabilities to play the actions 1 and 2 respectively by the player c. The expected utility of playing a strategy profile $\Pi = \{\alpha, \beta\}$ is then calculated as follows:

$$U^r(\{\alpha,\beta\}) = r_{11}\alpha\beta + r_{22}(1-\alpha)(1-\beta) + r_{12}\alpha(1-\beta) + r_{21}(1-\alpha)\beta$$

$$U^c(\{\alpha,\beta\}) = c_{11}\alpha\beta + c_{22}(1-\alpha)(1-\beta) + c_{12}\alpha(1-\beta) + c_{21}(1-\alpha)\beta$$

To estimate the effect of changing its current strategy, a player can calculate a partial derivative of its expected utility with respect to its mixed strategy:

$$\frac{\partial U^r(\{\alpha,\beta\})}{\partial \alpha} = \beta u - (r_{22} - r_{12})$$

$$\frac{\partial U^c(\{\alpha,\beta\})}{\partial \beta} = \alpha u' - (c_{22} - c_{21})$$

where $u = (r_{11} + r_{22}) - (r_{21} + r_{12})$ and $u' = (c_{11} + c_{22}) - (c_{21} + c_{12})$.

At each time step IGA player adjusts its current strategy in the direction of the gradient as to maximize its utility:

$$\alpha_{t+1} = \alpha_t + \eta \frac{\partial U^r(\{\alpha_t,\beta_t\})}{\partial \alpha}$$

$$\beta_{t+1} = \beta_t + \eta \frac{\partial U^c(\{\alpha_t,\beta_t\})}{\partial \beta}$$

where η is a step size, usually $0 < \eta \ll 1$. Obviously, the opponent's mixed strategy is supposed to be known by the players.

Singh and colleagues proved the convergence of IGA to a Nash equilibrium (or to the equivalent average reward), when played in self-play (i.e, when both players use the same algorithm), in the case of the infinitesimal step size ($\lim_{\eta \to 0}$), whence the name of the algorithm. However, IGA cannot be applied to a big number of real world problems because of two major factors:

1. It assumes omniscient knowledge by the agents of the opponent's current policy and

2. It was designed for the two-agent two-player case; the extension to the many-agent, many-action case is not straightforward.

3.4.2 Policy Hill-Climbing

The first practical algorithm capable to play mixed strategies that realized the convergence properties of IGA was the Policy Hill-Climbing (PHC) learning algorithm proposed by Bowling and Veloso (2002). The PHC algorithm requires neither knowledge of the opponent's current stochastic policy nor its recently executed actions. The algorithm, in essence, performs hill-climbing in the space of mixed strategies and is, in fact, a simple modification of the single-agent Q-learning technique. It is composed of two parts. The first part is the reinforcement learning part which is based on the Q-learning technique to maintain the values of the particular actions in the states. The second part is the game theoretic part, which maintains the current mixed strategy in each system's state.

The policy is improved by increasing the probability that it selects the highest valued action, by using a small step δ called learning rate 1. The Q-values are updated using the standard Q-learning value update by using the second learning rate, α, called learning rate 2. If δ is 1 the algorithm is equivalent to the single-agent Q-learning as soon as the learning agent executes the best action with probability 1 without exploration. Hence, this technique is rational and converges to the optimal solution if the other players follow a fixed (stationary) policy. However, if the other players are learning, the PHC algorithm may not converge to a stationary policy though its average reward will converge to the reward of a Nash equilibrium, as was shown in (Bowling and Veloso, 2002).

The formal definition of the PHC algorithm is presented in Algorithm 3.3. In the lines 7-12 the learning is initialized. Until the maximum number of learning iterations is reached, the agent starts each iteration by playing its current mixed strategy with some exploration (lines 14-15). Having observed its own reward and the new state, it updates the Q-values in previous state (lines 16-17). Then it updates its mixed strategy in previous state by "moving" it in the direction of the estimated gradient of the utility of its current strategy (line 18). Finally, it updates its internal state and increments the iterations counter (lines 19-20) and goes on to the next iteration.

WoLF-PHC

Bowling and Veloso (2002) have developed an important extension of the PHC algorithm, called WoLF-PHC (Algorithm 3.4). WoLF (for "Win or Learn Fast") is a technique of varying the learning rate δ depending on either the player j is winning or is

function PHC($\mathcal{A}^j, \mathbf{S}, \mathbf{s}_0, t_{max}$)
  returns: a policy.
  inputs: $\mathcal{A}^j$, the set of the actions of player j,
    $\mathbf{S}$, the set of the multiagent states,
    $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
    t_{max}, maximal number of learning iterations,

  Initialize the learning rate $\alpha \in (0, 1]$.
  Initialize the learning rate $\delta \in (0, 1]$.
  Initialize the discount factor $\gamma \in (0, 1)$.
  Initialize the Q-values $\hat{Q}^j(\mathbf{s}, a^j) \leftarrow 0,\ \forall \mathbf{s} \in \mathbf{S},\ \forall a^j \in \mathcal{A}^j$.
  Initialize current policy $\pi^j_{a^j}(\mathbf{s}) \leftarrow \frac{1}{|\mathcal{A}^j|},\ \forall a^j \in \mathcal{A}^j$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    From the state $\mathbf{s}_t$ select action a^j_t according to the strategy $\pi^j(\mathbf{s})$.
    Play a^j with some exploration.
    Observe new state $\mathbf{s}_{t+1}$ and the reward obtained R^j_t.
    Update $\hat{Q}^j(\mathbf{s}_t, a^j_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, a^j_t) \leftarrow (1 - \alpha)\hat{Q}^j(\mathbf{s}_t, a^j_t) + \alpha \left(R^j_t + \gamma \max_{a^j_{t+1}} \hat{Q}(\mathbf{s}_{t+1}, a^j_{t+1}) \right).$$

    Update current strategy, $\pi^j(\mathbf{s}_t)$, using the following rule,

$$\pi^j_{a^j}(\mathbf{s}_t) \leftarrow \pi^j_{a^j}(\mathbf{s}_t) + \Delta_{\mathbf{s}a^j},$$

    where,

$$\Delta_{\mathbf{s}a^j} = \begin{cases} -\delta_{\mathbf{s}a^j} & \text{if } a^j \neq \text{argmax}_{a'^j} \hat{Q}(\mathbf{s}_t, a'^j) \\ \sum_{a'^j \neq a^j} \delta_{\mathbf{s}a'^j} & \text{otherwise} \end{cases}$$

$$\delta_{\mathbf{s}a^j} = \min \left(\pi^j_{a^j}(\mathbf{s}_t), \frac{\delta}{|\mathcal{A}^j| - 1} \right),$$

    while constrained to a legal probability distribution.
    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$
  return current policy.

Algorithm 3.3: Policy Hill-climbing (PHC) learning algorithm for player j.

loosing. If it is winning its learning rate is small, otherwise it is big (line 16). Bowling has proved that these changes ensure the convergence of the PHC algorithm in self-play to a Nash equilibrium, which is sometimes a desired property of the learning algorithm. The same author showed that the WoLF principle aids in convergence by giving more time for the other players to adapt to changes in the player's strategy while allowing the player to adapt more quickly to other players' strategy changes when they are unfavorable. More formally, the WoLF-PHC algorithm requires two δ-learning rates, $\delta_{loosing}$ and $\delta_{winning}$ (line 4). Whether the player is loosing or winning is determined by comparing the expected value of the current policy with the expected value of the *averaged policy*, calculated at line 15 of the algorithm. If it is lower, then the player is considered as loosing and the $\delta_{loosing}$ is used, otherwise $\delta_{winning}$ is used (line 16). The averaged policy is an online mean of all played policies form the beginning of learning. The rest of the Wolf-PHC algorithm is the same as in PHC (Algorithm 3.3).

Wolf-PHC exhibits two good properties that permit it to be applied to a wide variety of stochastic games:

1. Convergence in the limit to a fixed policy (mixed or pure) and
2. Rationality in the sense of the one shot game Nash equilibrium.

However, if the reward structure of a stochastic game is resembling that of Prisoners-Dilemma, the Wolf-PHC will converge to a Nash equilibrium which is equally disastrous for all agents, while there can be a solution which, in the limit, maximizes the utility of all agents (or, at least, a subset of them). Thus, a strategy like Tit-for-Tat would be desirable to be learnt here, as soon as the stochastic game is analogous to the infinitely repeated matrix game in the sense that it is assumed to be replayed infinitely by the agents.

3.5 Opponent Modelling Algorithms

The opponent modeling learning algorithms, such as Joint-Action Learners (JALs) proposed by Claus and Boutilier (1998) and Adaptive Play Q-learning proposed by Gies and Chaib-draa (2006), are based on the *fictitious play* introduced by Brown (1951) or its modifications.

When played by all players, fictitious play is proven to converge to a Nash equilibrium in games that are iterated dominance solvable, that is, games such that it is

function WoLF-PHC($\mathcal{A}^j, \mathbf{S}, \mathbf{s}_0, t_{max}$)
  returns: a policy
  inputs: the same as in Figure 3.3.

  Initialize the learning rates $\alpha \in (0, 1]$ and $\delta_{loosing} > \delta_{winning} \in (0, 1]$.
  Initialize the discount factor $\gamma \in (0, 1)$.
  Initialize the Q-values $\hat{Q}^j(\mathbf{s}, a^j) \leftarrow 0,\ \forall \mathbf{s} \in \mathbf{S},\ \forall a^j \in \mathcal{A}^j$.
  Initialize current policy $\pi^j_{a^j}(\mathbf{s}) \leftarrow \frac{1}{|\mathcal{A}^j|},\ \forall a^j \in \mathcal{A}^j$.
  Initialize counter $C(\mathbf{s}) \leftarrow 0,\ \forall \mathbf{s}$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    From the state $\mathbf{s}_t$ select action a^j_t according to the strategy $\pi^j(\mathbf{s})$.
    Play a^j with some exploration.
    Observe new state $\mathbf{s}_{t+1}$ and the reward obtained R^j_t.
    Update $\hat{Q}^j(\mathbf{s}_t, a^j_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, a^j_t) \leftarrow (1-\alpha)\hat{Q}^j(\mathbf{s}_t, a^j_t) + \alpha \left[R^j_t + \gamma \max_{a^j_{t+1}} \hat{Q}(\mathbf{s}_{t+1}, a^j_{t+1}) \right].$$

    Update estimate of average policy, $\bar{\pi}(\mathbf{s})$, as follows,

$$C(\mathbf{s}) \leftarrow C(\mathbf{s}) + 1,$$

$$\bar{\pi}_{a'^j}(\mathbf{s}) \leftarrow \bar{\pi}_{a'^j}(\mathbf{s}) + \frac{1}{C(\mathbf{s})}(\pi^j_{a'^j}(\mathbf{s}) - \bar{\pi}_{a'^j}(\mathbf{s})),\ \forall a'^j \in \mathcal{A}^j.$$

    Update current strategy $\pi^j_{a^j}(\mathbf{s}) \leftarrow \pi^j_{a^j}(\mathbf{s}) + \Delta_{\mathbf{s}a^j}$, where,

$$\Delta_{\mathbf{s}a^j} = \begin{cases} -\delta_{\mathbf{s}a^j} & \text{if } a^j \neq \text{argmax}_{a'^j} \hat{Q}(\mathbf{s}, a'^j) \\ \sum_{a'^j \neq a^j} \delta_{\mathbf{s}a'^j} & \text{otherwise} \end{cases},$$

$$\delta_{\mathbf{s}a^j} = \min\left(\pi^j_{a^j}(\mathbf{s}), \frac{\delta}{|\mathcal{A}^j| - 1}\right),$$

$$\delta = \begin{cases} \delta_{winning} & \text{if } \sum_{a'^j} \pi^j_{a'^j}(\mathbf{s})\hat{Q}(\mathbf{s}, a'^j) > \sum_{a'^j} \bar{\pi}_{a'^j}(\mathbf{s})\hat{Q}(\mathbf{s}, a'^j) \\ \delta_{loosing} & \text{otherwise} \end{cases},$$

    while constrained to a legal probability distribution.
    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$.
  return current policy.

Algorithm 3.4: WoLF-PHC learning algorithm for player j, adapted from (Bowling and Veloso, 2002).

possible to iteratively remove dominated actions to finally obtain one action or a set of equivalent actions. Once an equilibrium is found, it is relatively easy to verify if it is a Nash equilibrium, and if it is Pareto-optimal.

3.5.1 Fictitious Play

Fictitious play is a technique that permits to find an equilibrium in pure strategies (if it exists) in a game by playing it iteratively. Players playing fictitious play maintain the *empiric beliefs* about the reduced profile of the other players' joint strategies. The idea is to learn an explicit model of other players under assumption that they play a stationary strategy, even if they actually do not. To do this, the players maintain a *history* of past joint actions of other players and calculate the empiric probability distribution of the reduced profile Π^{-j} using the following rule:

$$\Pi^{-j}(\mathbf{a}^{-j}) = \frac{C^j(\mathbf{a}^{-j})}{m}$$

where $C^j(\mathbf{a}^{-j})$ is the number of times that the joint action $\mathbf{a}^{-j}$ was played starting from the beginning of a play and m is the total number of iterations played.

Having calculated the reduced profile, agent j adopts its best response to it Π^{-j}.

The fact of considering the entire game history to calculate the estimated reduced profile of the opponents' joint strategy can provoke an error in these estimates. This is due to the fact that the opponent's strategy may not be constant and can evolve in time, as well as the player j's own strategy. To overcome this shortcoming, Young (1993) has proposed the *adaptive play*, a modification of fictitious play, which explicitly limits the memory of players available for history.

3.5.2 Adaptive Play

Formally, each player j playing the adaptive play saves in memory a history $H_t^j = \{\mathbf{a}_{t-p}^{-j}, \ldots, \mathbf{a}_t^{-j}\}$ of last p joint actions played by other players. To select a strategy to play at time $t+1$, player j randomly and irrevocably samples from H_t^j a subset, $\hat{H}_t^j = \{\mathbf{a}_{k_1}^{-j}, \ldots, \mathbf{a}_{k_l}^{-j}\}$, of examples of length l and calculates the empiric distribution $\hat{\Pi}^{-j}$ as an approximation of the real reduced profile, Π^{-j}, of strategies played by other

players, using the following:

$$\hat{\Pi}^{-j}(\mathbf{a}^{-j}) = \frac{C(\mathbf{a}^{-j}, \hat{H}_t^j)}{l} \tag{3.2}$$

where $C(\mathbf{a}^{-j}, \hat{H}_t^j)$ is the number of times that the joint action $\mathbf{a}^{-j}$ was played by other players according to the set $\hat{H}_t^j$.

Given the probability distribution over other players' actions, $\hat{\Pi}^{-j}$, the player j plays its best response to this distribution, $BR^j(\hat{\Pi}^{-j})$, with probability $1-\epsilon$, and with probability ϵ it plays another action to explore the joint action space of the game.

We will decide on two similar opponent modelling algorithms: (1) Joint-Action Learners (JALs) (Claus and Boutilier, 1998) and (2) Adaptive Play Q-learning (APQ) (Gies and Chaib-draa, 2006). They differs in how they model the behavior of their counterparts. While JALs are using fictitious play to do that, the more recent APQ is using adaptive play which has more relaxed convergence conditions as against the fictitious play.

3.5.3 Joint-Action Learners

The idea here is to learn explicitly the behavior model of the opponent by using the fictitious play probability distribution estimation technique, as if it was playing a stationary policy. In each system's state the learning player is playing its best (deterministic) response to this estimated probability distribution. The algorithm, as shown in Algorithm 3.5, has also a Q-learning based part for the optimal sequential (long term dependent) decision making. In Algorithm 3.5, at lines 7-11 learning is initialized. Until the maximum number of learning iterations is reached the agent starts each iteration by playing its best response strategy (line 13). Having observed its own reward and the new state, it updates the Q-values in previous state (lines 14-15). Next, the agent updates the counters of the opponent's actions (lines 17-20). Finally, it updates its internal state and increments the iterations counter (lines 21-22) and goes on to the next iteration.

The behavior of the Joint-Action Learners algorithm was empirically investigated for the case of cooperative (team) matrix games. Later, the case of zero-sum games was also investigated (Uther and Veloso, 2003).

function JALs($\mathbf{S}, \mathbf{A}^{-j}, n, \mathbf{s}_0, t_{max}$)
  returns: a policy.
  inputs: $\mathbf{S}$, the set of the multiagent states,
    $\mathbf{A}^{-j}$, the set of joint actions of counterpart players,
    $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
    t_{max}, maximal number of learning iterations.

  Initialize arbitrarily $\hat{Q}^j(\mathbf{s}, \mathbf{a}),\ \forall \mathbf{s} \in \mathbf{S},\ \forall \mathbf{a} \in \mathbf{A}$.
  Initialize the learning rate $\alpha \in (0, 1]$.
  Initialize the discount factor $\gamma \in (0, 1)$.
  Initialize counters $C(\mathbf{s}, \mathbf{a}^{-j}) \leftarrow 0$ and $n(\mathbf{s}) \leftarrow 0$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    Select action a^j to play by using the following rule,

$$a^j = \operatorname*{argmax}_{a^j} \sum_{\mathbf{a}^{-j}} \frac{C(\mathbf{s}, \mathbf{a}^{-j})}{n(\mathbf{s})} \hat{Q}(\mathbf{s}, (a^j, \mathbf{a}^{-j})).$$

    Observe new state $\mathbf{s}_{t+1}$, the joint action played $\mathbf{a}_t$, and the reward obtained R_t^j.
    Update $\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1 - \alpha)\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left(R_t^j + \gamma Value^j(\mathbf{s}_{t+1})\right),$$

    where $Value^j(\mathbf{s}) = \max_{a^j} \sum_{\mathbf{a}^{-j}} \frac{C(\mathbf{s}, a^{-j})}{n(\mathbf{s})} \hat{Q}(\mathbf{s}, \mathbf{a})$and $\mathbf{a} = (a^j, \mathbf{a}^{-j})$.
    Update counters:
    for all $\mathbf{a}^{-j} \in \mathbf{A}^{-j}$ **do**
      if $\mathbf{a}^{-j} = \mathbf{a}_t^{-j}$ **then**
        $C(\mathbf{s}, \mathbf{a}^{-j}) \leftarrow C(\mathbf{s}, \mathbf{a}^{-j}) + 1$
    $n(\mathbf{s}) \leftarrow n(\mathbf{s}) + 1$
    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$.
  return current policy

Algorithm 3.5: Joint-Action Learners (JALs) algorithm for player j, adapted from (Claus and Boutilier, 1998).

3.5.4 Adaptive Play Q-learning

Adaptive Play Q-learning algorithm was proposed by Gies and Chaib-draa (2006) for the stochastic robot coordination problems and was empirically tested with good results. It combines Q-learning with Young's Adaptive Play (Young, 1993), of which the

convergence conditions are more relaxed than those of fictitious play. In particular, the Adaptive Play is proven to converge to a Nash equilibrium in games having a strict equilibrium, i.e., there is no other equilibrium with the same utility. This condition is significantly less strict then the condition of the iterated dominance solvability of the game for the convergence of the fictitious play.

The APQ algorithm is presented in Algorithm 3.6. In the lines 9-15 the learning is initialized. Until the maximum number of learning iterations is reached the agent starts each iteration by playing its best response strategy (lines 17-18). Having observed its own reward and the new state, it updates the Q-values in previous state (lines 19-20). Then it updates its beliefs about the strategy of its opponent in the previous state by updating the history in that state (line 21) and by sampling new belief from the history (lines 22-23). Finally, it updates its internal state and increments the iterations counter (lines 24-25) and goes on to the next iteration.

The opponent modeling algorithms are unable to learn mixed policies, but their estimations of the opponents' strategies converge to a mixed Nash equilibrium in games where the only equilibrium is in mixed strategies. For example, in RockPaperScissors game, the only equilibrium is mixed $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$. Thus, after a sufficient number of plays, the player j will estimate the strategy of its opponent as $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ even if they both are using an opponent modelling algorithm and play, in fact, the pure strategies only. In games where there is a pure Nash equilibrium the players will converge to it if the corresponding convergence conditions are satisfied, what, in general, can not be known in advance for all stochastic games.

Thus, among the merits of the adaptive and fictitious play based algorithms are their (1) faster convergence (Claus and Boutilier, 1998) as compared to the single-agent Q-learning technique applied in multiagent context and (2) explicit modeling of the opponent's behavior that converges to the opponent's real behavior in the limit. However, these algorithms require that the agents observe the actions of each other and the field of applicability is limited to the case of team or coordination stochastic games. At the same time, the type of the game is difficult to predict in advance due to complex relations between states in multiple state systems with stochastic interstate transitions.

3.6 Adaptivity Modeling Algorithms

In this subsection we present two recent multiagent learning algorithms that differ from the previous ones in that they model and exploit the adaptive character of learning

function APQ($\mathbf{S}, \mathbf{A}^{-j}, \mathbf{s}_0, t_{max}, p, l$)
  returns: a policy.
  inputs: $\mathbf{S}$, the set of the multiagent states,
    $\mathbf{A}^{-j}$, the set of joint actions of counterpart players,
    $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
    t_{max}, maximal number of learning iterations,
    p, history size,
    l, sampling size.

  Initialize arbitrarily $\hat{Q}^i(\mathbf{s}, \mathbf{a}^{-j}),\ \forall \mathbf{s} \in \mathbf{S},\ \forall \mathbf{a}^{-j} \in \mathbf{A}^{-j}$.
  Initialize the learning rate $\alpha \in (0, 1]$.
  Initialize the discount factor $\gamma \in (0, 1)$.
  Initialize the time $t \leftarrow 0$.
  Initialize history $H_t^j(\mathbf{s}) \leftarrow \mathit{EmptySet},\ \forall \mathbf{s}$.
  Initialize reduced profile $\Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}) = \frac{1}{|\mathbf{A}^{-j}|},\ \forall \mathbf{s}$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    Select action a^j to play by using the following rule,

$$a^j = \operatorname*{argmax}_{a^j} \sum_{\mathbf{a}^{-j}} \Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}) \hat{Q}(\mathbf{s}, (a^j, \mathbf{a}^{-j})).$$

    Play action a^j with some exploration.
    Observe new state $\mathbf{s}_{t+1}$, the joint action played $\mathbf{a}_t$, and the reward obtained R_t^j.
    Update $\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1 - \alpha)\hat{Q}^j(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left[R_t^j + \gamma Value^j(\mathbf{s}_{t+1}) \right],$$

    where $Value^j(\mathbf{s}) = \max_{a^j} \sum_{\mathbf{a}^{-j}} \Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}) \hat{Q}(\mathbf{s}, \mathbf{a})$ and $\mathbf{a} = (a^j, \mathbf{a}^{-j})$.

    Update history in $\mathbf{s}_t$: $H_t^j(\mathbf{s}_t) = H_t^j(\mathbf{s}_t) \cup \{\mathbf{a}_t^{-j}\} \setminus \{\mathbf{a}_{t-p}^{-j}\}$.
    Sample from $H_t^j(\mathbf{s}_t)$ a subset $\hat{H}_t^j(\mathbf{s}_t)$ of size l.
    Calculate the new opponents' reduced profile in $\mathbf{s}_t$ using the following rule,

$$\Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}_t) = \frac{n_{\hat{H}_t^j(\mathbf{s}_t)}(\mathbf{a}^{-j})}{l},\ \forall \mathbf{a}^{-j},$$

    where $n_{\hat{H}_t^j(\mathbf{s}_t)}(\mathbf{a}^{-j})$ is the number of times that the strategy $\mathbf{a}^{-j}$ appears in the sampling $\hat{H}_t^j(\mathbf{s}_t)$.
    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$.
  return current policy.

Algorithm 3.6: Adaptive Play Q-learning algorithm for player j.

algorithms used by their counterparts.

3.6.1 PHC-Exploiter

The first to exploit the adaptivity of the opponent were Chang and Kaelbling (2001). Their PHC-Exploiter algorithm was able to outperform the PHC algorithm in an adversarial game by using the knowledge of the opponent's *adaptivity* mechanism, i.e., it played against PHC player in adversarial games, such as MatchingPennies or RockPaperScissors, and it knew how its own actions were affecting the *beliefs* of its opponent about its own strategy. This knowledge gave it the possibility to play so as to delude its opponent about its real intentions.

The PHC-Exploiter algorithm is presented in Algorithm 3.7. In the lines 8-14 the learning is initialized. Until the maximum number of learning iterations is reached, the agent starts each iteration by playing its current mixed strategy with some exploration (line 16). Having observed its own reward and the new state, it updates the Q-values in previous state (line 18). Then it updates its beliefs about the opponent's policy and learning rate in previous state by maintaining the history of past plays (lines 19-20). Then it updates its policy in previous state in a direction, which is defined by whether it is winning or loosing in that state (lines 21-24). Finally, it updates its internal state and increments the iterations counter (lines 25-26) and goes on to the next iteration.

Note that whether the PHC-Exploiter is winning or loosing (line 21 of Algorithm 3.7) is decided by using the following inequality:

$$\sum_{a'^j} \pi^j_{a'^j}(\mathbf{s})\hat{Q}(\mathbf{s}, a'^j) > Value^j((\overset{\star}{\pi}{}^j(\mathbf{s}), \Pi^{-j}(\mathbf{s})))$$

where $(\overset{\star}{\pi}{}^j(\mathbf{s}), \Pi^{-j}(\mathbf{s}))$ is a strategy profile, where player j plays a Nash equilibrium strategy $\overset{\star}{\pi}{}^j(\mathbf{s})$. $Value^j(\cdot)$ is the function returning the expected utility of a strategy profile for player j. This function was defined in previous chapter in equation 2.4.

If the above inequality is true, then PHC-Exploiter is winning, if not — it is loosing. The opponent's learning rate δ and policy $\Pi^{-j}(\mathbf{s})$ are derived from estimates using the observable history of actions (Algorithm 3.7, line 21). If one assumes the game matrix to be public information, then one can calculate directly the equilibrium strategy $\overset{\star}{\pi}{}^j(\mathbf{s})$, otherwise one can run PHC for some finite number of plays to obtain an estimate to this equilibrium strategy.

The applicability of PHC-Exploiter is strictly limited to the case of play against

function PHC-EXPLOITER($\mathcal{A}^j, \mathbf{S}, \mathbf{s}_0, t_{max}, p$)
  returns: a policy
  inputs: $\mathcal{A}^j$, the set of the actions of player j,
    $\mathbf{S}$, the set of the multiagent states,
    $\mathbf{s}_0$, the initial state, $\mathbf{s}_0 \in \mathbf{S}$,
    t_{max}, maximal number of learning iterations,
    p, the size of history.

  Initialize the learning rates $\alpha, \delta \in (0, 1]$ and the discount factor $\gamma \in (0, 1)$.
  Initialize the Q-values $\hat{Q}^j(\mathbf{s}, a^j) \leftarrow 0,\ \forall \mathbf{s} \in \mathbf{S},\ \forall a^j \in \mathcal{A}^j$.
  Initialize current policy $\pi^j_{a^j}(\mathbf{s}) \leftarrow \frac{1}{|\mathcal{A}^j|},\ \forall a^j \in \mathcal{A}^j$.
  Initialize the history $H^j_t(\mathbf{s}) \leftarrow EmptySet,\ \forall \mathbf{s}$.
  Initialize reduced profile $\Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}) = \frac{1}{|\mathbf{A}^{-j}|},\ \forall \mathbf{s}$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  while $t \leq t_{max}$ **do**
    From the state $\mathbf{s}_t$ select action a^j_t according to the strategy $\pi^j(\mathbf{s})$.
    Play a^j with some exploration.
    Observe new state $\mathbf{s}_{t+1}$ and the reward obtained R^j_t.
    Update $\hat{Q}^j(\mathbf{s}_t, a^j_t)$ using the following rule,

$$\hat{Q}^j(\mathbf{s}_t, a^j_t) \leftarrow (1-\alpha)\hat{Q}^j(\mathbf{s}_t, a^j_t) + \alpha\left(R^j_t + \gamma \max_{a^j_{t+1}} \hat{Q}(\mathbf{s}_{t+1}, a^j_{t+1})\right).$$

    Observe action $\mathbf{a}^{-j}_t$, update history $H^j(\mathbf{s}_t)$ and calculate an estimate of the opponent's policy as $\Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s})_t = \frac{n_{H^j_t(\mathbf{s}_t)}(\mathbf{a}^{-j})}{p},\ \forall \mathbf{a}^{-j}$ where $n_{H^j_t(\mathbf{s}_t)}(\mathbf{a}^{-j})$ is the number of times that the strategy $\mathbf{a}^{-j}$ appears in the history $H^j_t(\mathbf{s}_t)$.
    Estimate the PHC player's learning rate $\delta \leftarrow \frac{\left|\Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s})_t - \Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s})_{t-p}\right|}{p}$.
    if we are winning **then**

$$\pi^j_{a^j}(\mathbf{s}_t) \leftarrow \begin{cases} 1 & \text{if } a^j = \text{argmax}_{a'^j} \hat{Q}(\mathbf{s}, a'^j) \\ 0 & \text{otherwise} \end{cases}.$$

    else

$$\pi^j_{a^j}(\mathbf{s}_t) \leftarrow \pi^j_{a^j}(\mathbf{s}_t) + \begin{cases} \delta & \text{if } a^j = \text{argmax}_{a'^j} \hat{Q}(\mathbf{s}, a'^j) \\ \frac{-\delta}{|\mathcal{A}^j|-1} & \text{otherwise} \end{cases}.$$

    Update current state $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
    Increment the time $t \leftarrow t + 1$.
  return current policy.

Algorithm 3.7: PHC-Exploiter learning algorithm for player j, adapted from (Chang and Kaelbling, 2001).

PHC agent. Besides, it was not shown to perform well in general sum stochastic games, neither theoretically nor empirically. However, it suggested a good direction for future research in multiagent learning that was borne out by the following algorithms: Hyper-Q by Tesauro (2004) and Adaptive Dynamics Learner by Burkov and Chaib-draa (2007a;b).

3.6.2 Hyper-Q

Tesauro (2004) extended the idea of learning the opponents' adaptivity by introducing the Hyper-Q algorithm. He supposed that given that the agent may need to learn a mixed strategy, which may depend on the mixed strategies of other agents, an obvious idea is to make Q-values evaluate mixed strategies, rather then base actions, i.e., to include in state description the estimation of the opponents' mixed strategy. At time t, the agent j executes an action a^j according to its current mixed strategy $\pi^j(\mathbf{s})$, and then observes a payoff R_t^j, the new state $\mathbf{s}_{t+1}$ and, by observing the joint strategy executed by the opponent players, it updates its estimates of the opponents' strategy profile $\Pi^{-j}(\mathbf{s})$. The Q-value of Hyper-Q is then adjusted as follows:

$$\begin{aligned}\hat{Q}^j(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t)) \quad \leftarrow \quad & (1-\alpha)\hat{Q}^j\left(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t)\right) \\ & +\alpha\left[R_t^j + \gamma \max_{\pi^j(\mathbf{s}_{t+1})} \hat{Q}\left(\mathbf{s}_{t+1}, \pi^j(\mathbf{s}_{t+1}), \Pi^{-j}(\mathbf{s}_{t+1})\right)\right] \quad (3.3)\end{aligned}$$

Given the Q-values, the state and the other players' mixed strategy, the greedy policy for agent j is then defined by

$$\hat{\pi}^j(\mathbf{s}, \Pi^{-j}(\mathbf{s}_t)) = \operatorname*{argmax}_{\pi^j(\mathbf{s}_t)} Q(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t)) \quad (3.4)$$

To estimate the strategy of the opponents, Hyper-Q uses one of two distribution estimation techniques: (1) Exponential Moving Average (EMA) and (2) Bayesian strategy estimation. Notice that EMA is a family of similar statistical techniques used to analyze time series data, in particular, in finance analysis. Typically, EMA assumes that the recent observations are more informative than the older ones. Thus, as applied to our problem, given a new observed joint action $\mathbf{a}_t^{-j}$, agent j updates a reduced profile $\Pi_t - j$, represented as a vector $\mathbf{\Pi}_t^{-j}$, as follows:

$$\mathbf{\Pi}^{-j}(\mathbf{s})_{t+1} \leftarrow (1-\mu)\mathbf{\Pi}^{-j}(\mathbf{s})_t + \mu\mathbf{u}(\mathbf{a}_t^{-j})$$

where $\mathbf{u}(\mathbf{a}_t^{-j})$ is a unit vector representation of the action $\mathbf{a}_t^{-j}$ observed at time t. I.e., $\mathbf{u}(\mathbf{a}_t^{-j})$ is a vector, containing as many elements as the reduced strategy profile vector

$\mathbf{\Pi}_t^{-j}$ does. All elements of this vector contain 0 excepting one element containing 1, position of which corresponds to the action $\mathbf{a}_t^{-j}$. Parameter μ is a small constant, $0 < \mu \ll 1$.

A more refined alternative to EMA is a Bayesian strategy estimation technique. Let us assume a finite and discrete set of possible values of $\Pi^{-j}(\mathbf{s}_t)$ (a finite number of probability distributions over opponents' actions). In the original version of Hyper-Q, to obtain this set, Tesauro (2004) uses a discretization with a uniform grid. A probability that the opponents play a distribution $\Pi^{-j}(\mathbf{s}_t)$ given the history of observed actions, $H^j(\mathbf{s})$, can be computed using Bayes' rule:

$$P(\Pi^{-j}(\mathbf{s})|H^j(\mathbf{s})) = \frac{P(H^j(\mathbf{s})|\Pi^{-j}(\mathbf{s}))P(\Pi^{-j}(\mathbf{s}))}{\sum_{\Pi'^{-j}(\mathbf{s})} P(H^j(\mathbf{s})|\Pi'^{-j}(\mathbf{s}))P(\Pi'^{-j}(\mathbf{s}))}$$

where $P(\Pi^{-j}(\mathbf{s}))$ is the prior probability that players $-j$, i.e., all other players except j, play a strategy profile $\Pi^{-j}(\mathbf{s})$ and $\Pi'^{-j}(\mathbf{s})$ iterates over all discrete strategy values. The conditional probability of a history given a strategy, $P(H^j|\Pi^{-j}(\mathbf{s}))$, can be calculated as the following product of individual joint-action probabilities:

$$P(H^j(\mathbf{s})|\Pi^{-j}(\mathbf{s})) = \prod_{k=0}^{p} P(\mathbf{a}^{-j}(\mathbf{s})_k|\Pi^{-j}(\mathbf{s}))$$

assuming conditional independence of the individual actions and that the history length is p. If all actions in the history are considered to be equally informative regardless the time they were observed in state $\mathbf{s}$, one may write $P(\mathbf{a}^{-j}(\mathbf{s})_k|\Pi^{-j}(\mathbf{s})) = \Pi^{-j}_{\mathbf{a}^{-j}}(\mathbf{s}),\ \forall k$.

Given these notions, the equations 3.3 and 3.4 are updated as follows:

$$\begin{aligned}\hat{Q}^j(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t)) \leftarrow\ & (1 - \alpha P(\Pi^{-j}(\mathbf{s}_t)|H^j(\mathbf{s}_t)))\hat{Q}^j\left(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t)\right) \\ & + \alpha P(\Pi^{-j}(\mathbf{s}_t)|H^j(\mathbf{s}_t))\left(R_t^j + \gamma \max_{\pi^j(\mathbf{s}_{t+1})} \hat{Q}\left(\mathbf{s}_{t+1}, \pi^j(\mathbf{s}_{t+1}), \Pi^{-j}(\mathbf{s}_{t+1})\right)\right)\end{aligned}$$

$$\hat{\pi}^j(\mathbf{s}, \Pi^{-j}(\mathbf{s}_t)) = \operatorname*{argmax}_{\pi^j(\mathbf{s}_t)} \sum_{\Pi^{-j}(\mathbf{s}_t)} P(\Pi^{-j}(\mathbf{s})|H^j(\mathbf{s}))Q(\mathbf{s}_t, \pi^j(\mathbf{s}_t), \Pi^{-j}(\mathbf{s}_t))$$

The Hyper-Q algorithm was empirically compared with IGA and PHC algorithms on the example of the adversarial game RockPaperScissors. Its author has showed that the Hyper-Q algorithm is capable to outperform these two algorithms due to its ability to learn the best mixed strategy to play, versus any mixed strategy of its opponent, and to change these strategies with the object of maximizing its own long term utility. The main merit of Hyper-Q is that it is able to learn the best *long term* strategy against the

adaptive opponents by discovering their *adaptivity* and by finding an optimal *sequence of the mixed strategy changes* to turn this adaptivity to its own advantage. But the price of this is its higher computational complexity. This is due to the discretization of the continuous space of reduced strategy profiles and a need of making the Q-value update and reduced profile estimation for those discreet values at each moment of time. We will examine this question in the following subsection about the shortcomings of the presented state-of-the-art multiagent learning algorithms.

3.7 Limitations

Despite the fact that there is now a multitude of learning algorithms for the stochastic games, they have all or certain of a number of serious shortcomings, which limit their applicability to real world problems. Among them the most noticeable are the following three limitations:

1. High computational complexity,
2. Strong game structure requirements and
3. Focus on the convergence to the one shot Nash equilibrium.

3.7.1 High computational complexity

Obviously, the computational complexity of the different multiagent learning algorithms is different. There has not been much work done in the direction of the complexity estimation of the multiagent learning algorithms. The main result, on which the estimations of the computational complexity of the particular Q-learning based multiagent algorithms are made, is that of Koenig and Simmons (1996). In their paper about the complexity of Q-learning for the goal directed tasks, those authors proved that the time required for the convergence of single-agent Q-learning to the optimal solution may be exponential in the number of the states of the environment. As soon as the number of the states of multiagent system is exponential in the number of acting agents (since the multiagent system's states are joint states of all agents in the original environment), it is straightforward to suppose that the computational complexity of the Q-learning based multiagent algorithms may be at least double-exponential (in the number of the original, single-agent, states and in the number of learning agents).

Some algorithms, such as Nash-Q, perform the calculations of Q-values for the all agents, which involves additional complexity. Other algorithms, such as Hyper-Q, are even less computationally tractable because their complexity is highly dependent on the probability space discretization quality. For example, in the known realizations of the Hyper-Q algorithm, there was about of $100,000$ discrete values of the opponents strategies, $\Pi^{-j}(\mathbf{s})$, for the game with merely three opponent's base actions (*Rock*, *Paper*, *Scissors*).

3.7.2 Strong game structure requirements

As it was already mentioned above in this chapter, some learning algorithms require too much conditions about the structure of stochastic game to be satisfied to make these algorithms applicable. For example, the single-agent Q-learning requires that the other agents follow stationary policies, that is some pure or mixed policies that do not evolve in time. Minimax-Q learning algorithm is applicable in the games with two-players only and supposes that the other agent follows the same, Minimax-Q, algorithm. The Nash-Q learning algorithm requires the existence and uniqueness of the Nash equilibrium and the Friend-or-Foe Q-learning (FFQ) algorithm requires that the agents know in advance which kind of equilibrium there exists in the game, either adversarial or coordination, that cannot be easily determined for the real problems. IGA player needs to know exactly the mixed policy of its opponent and the full game matrix in advance and all along the learning process, which is not always realizable, in particular in the adversarial case or in a case with limited communication. The Adaptive Play Q-learning (APQ) requires the complete observability by the learning agents of the actions made by the other agents which limits its applicability in some cases with partial observability. In addition, APQ is not able to learn mixed strategies which does not permit it to converge in adversarial games.

The PHC and, more particularly, Wolf-PHC algorithms are devoid of many of these limitations. However they both, as well as all other learning algorithms, excepting the adaptivity modeling algorithms, have another major drawback: their focus on the convergence to the one shot Nash equilibrium.

3.7.3 Focus on the convergence to the one shot Nash equilibrium

Probably one of the most major shortcomings of all of the state-of-the-art multiagent learning algorithms is their strong focus on the convergence to the one shot game Nash equilibrium. Although Nash equilibrium is a "stable" solution, there exist a variety of cases where it is undesirable for the agents and they would prefer to play another joint action. For example, as we noted above, in the iterated PrisonersDilemma the agents would prefer the utility of the cooperative joint action instead of the Nash equilibrium. Furthermore, for the agents, such as Adaptive Play or Policy Hill-climbing, adapting to each other during learning, it would be better to converge to the Pareto optimal solution, instead of the Nash, as soon as adaptation is a bilateral process.

One more point in this discussion is that the algorithms of reinforcement learning, such as Q-learning, use discount factor, γ, to plan with infinite horizon, i.e., the policy they learn is calculated under assumption that it will be executed infinitely often from any state. In stochastic games each state of the environment is a one shot game. Thus, if an infinite horizon learning is produced in an SG, then it is also assumed that the resulting policy is to be executed infinitely often in each game/state. This corresponds to the case of infinitely repeated games, where there may exist a Pareto optimal Nash equilibrium strategies like Tit-for-Tat, as we showed on the example of the infinitely iterated PrisonersDilemma in Section 2.4. Therefore, the adaptive agents learning in such conditions, should be able to converge to this type of solution instead of a (not always desirable) one shot Nash equilibrium.

3.8 Conclusion

In this chapter we presented the most widely used state-of-the-art multiagent learning algorithms for stochastic games. These algorithms can be divided onto five principal classes:

1. Single-agent techniques adapted to multiagent context (such as Q-learning),
2. Equilibrium Learning Algorithms,
3. Gradient Based Algorithm,
4. Opponent Modelling Algorithms and

5. Adaptivity Modelling Algorithms.

While we mentioned that single-agent techniques can be applied directly to the multi-agent environment, we showed also that there are serious limitations of such an application, since single-agent learning techniques assume the environment to be stationary at all time, which is not the case when the other agents are learning as well.

We gave a detailed description for each of the selected algorithms, by pointing out their merits and demerits and by outlining their applicability to real world problems. Although nowadays there are a multitude of multiagent learning algorithms, all of them have major shortcomings related to their computational complexity, strong requirements to the environments or their tendency to converge to the one shot game Nash equilibrium even if they are learning in the infinitely repeated game context.

In the next chapter we will present two novel approaches, developed in the context of this work, to overcome some of these shortcomings. These are an algorithm that is focused on the complexity reduction of the multiagent learning and an algorithm of effective game playing in presence of adaptive counterparts.

Chapter 4

Adaptive Dynamics Learning and Q-initialization

4.1 Introduction

As we have seen in the previous chapter, the modern multiagent learning suffers from three major drawbacks:

1. High computational complexity,
2. Strong game structure requirements and
3. Focus on the convergence to the one shot Nash equilibrium.

Some algorithms, such as Wolf-PHC by Bowling and Veloso (2002) or Hyper-Q by Tesauro (2004), have only one of these drawbacks. Indeed, as we have explained in the previous chapter, in self-play, Wolf-PHC converges to the one shot Nash equilibrium. On the other hand, Hyper-Q in its current version is highly computationally complex due to the discretization of the continuous space of the other players' mixed strategies, while doing better than Nash equilibrium against Wolf-PHC in adversarial game.

However, the most of the modern multiagent learning algorithms have two or even all these demerits at once. In this chapter we present our contributions to the field of multiagent learning, which is a step forward to resolve some of the stated drawbacks. Our algorithms are using two novel approaches to the complexity reduction of

multiagent learning and to the effective learning in presence of the adaptive learning opponents, which is not focused on the Nash equilibrium.

The first algorithm, Initialized Adaptive Play Q-learning (Burkov and Chaib-draa, 2007c), permits, as we will show later, to considerably reduce the complexity of multiagent learning in a class of stochastic games, called "goal-directed stochastic games with action-penalty representation". The second one, called Adaptive Dynamics Learner (Burkov and Chaib-draa, 2007a;b), will be shown to be very effective in adversarial games against adaptive opponents, such as IGA and APQ, and to converge to a Pareto optimal strategy in general-sum games, such as the PrisonersDilemma.

4.2 Reducing the Complexity of Multiagent Learning

It is known that the complexity of the reinforcement learning algorithms (time of convergence to the optimal solution), such as Q-learning, may be exponential in the number of environment's states. As early as ten years ago it was shown by Koenig and Simmons (1996) that the learning complexity for the goal-directed problems may be substantially reduced by initializing the Q-values with a "good" approximative function. But there was no big practical progress done in that direction because hitherto there was no idea how to find this "good" function that would be equally good for a class of problems. However, in the multiagent case there exists such a good approximation for a big class of problems which are goal-directed stochastic games. These games can reflect coordination and common interest for cooperative robots, for example. For these games, the approximative function is nothing but the single-agent planning solution which can easily be found by each agent individually. In this article we show that:

1. the single-agent solution is a "good" approximation for the goal-directed stochastic games with action-penalty representation (i.e., it is admissible and monotonic);
2. the complexity is reduced when learning is initialized with this approximative function as compared to the uninformed case.

4.2.1 Introduction

In multiagent systems where several autonomous agents act simultaneously, the effect of an action of an agent does not only depend on that action as soon as the other agents' actions can also have an influence on the environment's resulting state. In this case, an agent's current policy may be dependent not only on its preferences about the environment, but also on its beliefs about preferences and/or beliefs of other agents in this environment. Thus, if some agents in a multiagent system do not follow a fixed (stationary) policy, we say that such an environment is non-stationary, and therefore the techniques widely used for single-agent learning and planning are not generally efficient in this multiagent context. Another major challenge for both multiagent learning and planning is the problem of the state space dimensionality. Obviously, the cardinality of the state space of multiagent system grows exponentially with the number of agents as soon as in such an environment each state is composed of the individual "positions" of all the agents acting in that environment.

One can divide the decision problems in MDPs in two categories: learning and planning. Learning is used when the properties of the model, such as its transition function, T, and the reward function, R, are not known to the agents. To learn a policy, the agent must *interact with the environment* and to construct a policy based on its experience. Planning is a problem of decision making when the agent knows the model of the environment where it will be acting and its task is to calculate a policy *before starting to act*.

When an agent is *learning* in an MDP, the transition function, T, of this MDP is not known to the learning agent. Thus, the agent cannot directly use the Bellman equation (2.2) to calculate the values of the actions in each state. Q-learning consists in estimating the real action's value $\overset{\star}{Q}(s,a)$ by executing action a in state s of the environment, observing the reward $R(s,a)$ obtained and the system's next state s', using the following update rule:

$$\hat{Q}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha\left(R(s,a) + \gamma \max_{a} \hat{Q}(s',a)\right) \tag{4.1}$$

where $\hat{Q}(s,a)$ is an estimated value of $\overset{\star}{Q}(s,a)$ and $\alpha \in [0,1]$ is the learning rate. All along the learning process the agent selects actions to execute in each state by maximizing the Q-value in that state with some stochastic exploration which decreases over time. Any reinforcement learning exploration technique (Thrun, 1992) can be used.

Obviously, in that case the agent is strictly risk-neutral as soon as it tries to maximize its *expected* total reward. The risk neutrality of an agent means that by maximizing

the expectation of the future rewards it is implicitly assumed that the resulting policy can be executed an infinite number of times. Thus, it is rational to maximize the expectation of future rewards. On the other hand, if the agent could assume that the policy can be executed a finite number of times, e.g. one, it could be rational for it to maximize the *immediate* rewards instead of the expected future rewards, depending on the degree of its inclination to risk (see (Russell and Norvig, 2005) for more detail).

The convergence of the estimated Q-values, $\hat{Q}(s,a)$, to their optimal values, $\overset{\star}{Q}(s,a)$, was proven by Watkins and Dayan (1992) under the conditions that each state-action pair is updated infinitely often, rewards are bounded and α tends asymptotically to 0.

It was then shown in (Koenig and Simmons, 1996) that Q-learning in general case may have an exponential computational complexity (in terms of the time required to find an optimal policy). However, they also showed that the computational complexity of Q-learning may be substantially reduced (to some small polynomial function in the size of the state space) if,

- an appropriate reward structure is chosen and
- Q-values are initialized with some "good" values.

An appropriate reward structure proposed by Koenig and Simmons (1996) is the so-called *action-penalty representation* where the agent is penalized for every executed action in every state except the goal states. In the goal states, the penalty (or reward) of any action is 0. More formally, the reward structure they propose:

$$R(s,a) = -1,\ \forall s \in \mathcal{S} - \mathcal{G},\ \forall a$$

$$R(s,a) = 0,\ \forall s \in \mathcal{G},\ \forall a$$

where $\mathcal{G}$ is the set of goal states of MDP.

In fact, the action-penalty representation is the most frequent reward structure in MDPs and the problems that are solved in such MDPs are called *stochastic shortest path* problems.

In turn, although the Q-learning technique has existed for more that a decade, the initialization of Q-values has not been explored much in the literature, substantially because a good heuristic approximation cannot be easily found for the problems where the environment is not known, as well as the location of the goal states and the reward function.

For *planning* problems, however, a variety of methods have been proposed for reducing the state space relevant to a planning task in an MDP. Most of them use a so called heuristic search, which is a set of methods based on the knowledge of a heuristic function that can estimate the real utility of any visited state (Bonet and Geffner, 2001). Generally, if that heuristic function is sufficiently informative and satisfies certain conditions, then the algorithm using it does not need to visit the entire state space to find the solution.

Unfortunately, in most cases of multiagent systems, an explicit search in the state space is practically impossible as soon as the search supposes that the properties of the environment in each state are known to the agent. That is not the case when there are several, possibly adversarial, agents affecting the environment and their policies and rationality principles are not known. Thus, since the centralized planning in that context is not always possible the agents are usually faced with the learning or adaptation problems.

As soon as it is relatively easy to define the reward function of the agents by using the action-penalty representation, the only remaining possibility to reduce the number of visited states (and, hence, the learning complexity) is to use a suitable approximation function to initialize the multiagent Q-values. Although, as it was already noticed, it is hard to find a general approach to initialization of single-agent Q-values, in the multiagent context for a big class of problems such initialization is possible if the single-agent planning solution is used for that purpose, since solving the single-agent optimal sequential decision problem is a much simpler task.

In this Section, we mainly address the problem of multiagent learning complexity reduction in goal-directed stochastic games with action-penalty representation. The problem of non-stationarity is also treated, but it is considered as a necessary condition of convergence of the algorithm based on our approach rather than an objective. Our main contribution consists in using single-agent planning results as a heuristic function to initialize the agents' multiagent Q-values in all unknown states. The idea is to focus the learning process on a relatively small relevant region of the entire state space and, by so doing, to reduce the calculation time required to learn a multiagent solution. We also show theoretically the correctness of such an initialization. To do that, we provide the proofs of admissibility and monotonicity (consistence) of such heuristic functions.

The algorithm used as a basis for our approach is Adaptive Play Q-learning (APQ) proposed by Gies and Chaib-draa (2006). Note that APQ was chosen solely to demonstrate the reduction of learning complexity, since we needed an algorithm which operates with Q-values and converges in stochastic games (or at least in a subclass of SGs). APQ

possesses these properties in particular for coordination and common interest stochastic games as soon as it is based on the Adaptive Play by Young (1993). This choice, however, is not critical for our approach and any other multiagent learning algorithms having these properties may be suitable as well.

Having this in mind we are now ready to present our approach to the complexity reduction of Q-learning in the stochastic games context. In the subsections that follow we give a detailed description of our approach, called "Initialized Multiagent Q-learning", which uses an approximation of multiagent Q-values by using single-agent optimal solution. We then demonstrate the correctness of such an approximation in goal-directed stochastic games with action-penalty representation and show the behavior of our algorithm on simple test benches: a two- and four-player versions of the grid world problem.

4.2.2 Q-values Initialization

In our approach we made several important assumptions about the model of the environment. The first assumption is that stochastic games, where agents are intended to act, are goal directed with action-penalty representation, i.e., all agents are penalized for any executed action in any state except the goal states.

We also assume that the multiagent environment applies additional restrictions on the reward and transition functions of the *underlying MDP*. By underlying MDP we mean an environment obtained by removing all agents from the original multiagent system while keeping in this environment only one agent. In this case, an MDP is obtained from the stochastic game. This MDP we call underlying MDP.

More precisely, we assume that in any joint state $\mathbf{s} = (s^1, \ldots, s^j, \ldots, s^n)$ and for any joint action $\mathbf{a} = (a^1, \ldots, a^j, \ldots, a^n)$ a penalty $R^j(\mathbf{s}, \mathbf{a})$ agent j obtains cannot be lower than its penalty $R^j(s^j, a^j)$ in the underlying MDP. In other words, the multiagent penalties for all joint state-joint action pairs may be only higher than or equal to the *corresponding* single-agent values. These corresponding values agent receives by acting alone in the environment.

The last assumption is that for any agent j, $j = 1 \ldots n$, in any joint state $\mathbf{s} = (s^1, \ldots, s^j, \ldots, s^n)$ and for any joint action $\mathbf{a} = (a^1, \ldots, a^j, \ldots, a^n)$, the transition function of the multiagent system, $T(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ relates to the transition function of the underlying MDP, $T^j(s^j, a^j, s'^j)$, in such a way that the expectation of the utility of the future state $U^j(\Pi(\mathbf{s}'))$ in multiagent system cannot be greater than the expectation of

the utility of the future state $U^j(s'^j)$ of the underlying MDP. This must be valid for any joint policy Π in the multiagent system.

More formally, we assume that

$$R^j(\mathbf{s},\mathbf{a}) \leq R^j(s^j,a^j),\ \forall \mathbf{s}=(s^j,\mathbf{s}^{-j}),\ \forall \mathbf{a}=(a^j,\mathbf{a}^{-j}) \tag{4.2}$$

where $\mathbf{s}$ is a multiagent state, $\mathbf{a}$ is a joint action, s^j and a^j correspond to j's personal state and action in the joint state $\mathbf{s}$ and joint action $\mathbf{a}$. $R^j(\mathbf{s},\mathbf{a})$ is the reward of j when $\mathbf{a}$ is played in $\mathbf{s}$. $R^j(s^j,a^j)$ is the corresponding single-agent reward that agent j obtains when it executes action a^j in the state s^j by being along in the underlying MDP. In turn, given the same rewards in multiagent and single-agent cases the multiagent transition function, $T(\mathbf{s},\mathbf{a},\mathbf{s}')$, defined for the multiagent problem, is related to the single-agent transition function, $T^j(s^j,a^j,s'^j)$, defined for the underlying MDP, by affecting the utilities as follows, for each pair $(\mathbf{s},\mathbf{a})$:

$$\underset{\mathbf{s}':T(\mathbf{s},\mathbf{a},\mathbf{s}')\neq 0}{E}\left[U^j(\Pi(\mathbf{s}'))\right] \leq \underset{s'^j:T(s^j,a^j,s'^j)\neq 0}{E}\left[U^j(\overset{\star}{\pi}{}^j(s'^j))\right]\ \forall\Pi \tag{4.3}$$

where s^j is such that $\mathbf{s}=(s^j,\mathbf{s}^{-j})$ and a^j is such that $\mathbf{a}=(a^j,\mathbf{a}^{-j})$. The policy $\overset{\star}{\pi}{}^j$ is the optimal single-agent policy of agent j in the underlying MDP. The utility $U^j(\Pi(\mathbf{s}))$ is defined using equation (2.5) and $U^j(\overset{\star}{\pi}{}^j(s^j))$ is defined using equation (2.1). As is easy to see, the single-agent problem (the underlying MDP) in that case is an appropriate *relaxation* of the multiagent learning problem, by speaking the language of the heuristic search terminology.

As mentioned in the introduction for this Chapter, in MDPs, it is not evident how to find an informative heuristic function to initialize Q-values with the purpose of reducing the time of the single-agent policy learning. But in many cases of SGs, there is such a function: a single-agent solution of the underlying MDP. An MDP may be solved with a variety of techniques (Russell and Norvig, 2005) (value iteration, reinforcement learning, heuristic search, etc). All these techniques are well known and we leave their description outside this work. Besides, as soon as the single-agent environment model is much simpler than the multiagent one, we suppose that all agents are able to calculate an optimal single-agent policy before starting to learn in multiagent context.

In order to ensure the tractability of the (single-agent) Q-learning algorithm the Q-values of all state-action pairs must be initialized with some monotonic and admissible function (Koenig and Simmons, 1996). Let's now define admissibility and monotonicity of Q-values for goal-directed single-agent Q-learning.

Definition 1 (monotonicity)**.** *Let $\mathcal{G}$ denote the set of the goal states, $\mathcal{G}\subseteq\mathcal{S}$. Q-value $\hat{Q}(s,a)$ is said to be monotonic for the goal directed Q-learning with action-penalty*

representation if and only if for each pair (s, a)

$$(1)\ \hat{Q}(s,a) = 0 \text{ if } s \in \mathcal{G} \text{ and}$$
$$(2)\ R(s,a) + E_{s':T(s,a,s')\neq 0}\left[U(\overset{\star}{\pi}(s'))\right] \leq \hat{Q}(s,a) \leq 0 \text{ if } s \notin \mathcal{G}.$$

Obviously, the monotonicity property of Q-values corresponds to the consistence of the heuristic function in the heuristic search terminology and means that the *triangle inequality* holds. In the heuristic search theory (Barto et al., 1995; Bonet and Geffner, 2001), the triangle inequality stipulates that for any state s and any successor state s' of s (according to the transition model) generated by an action a, the estimated accumulated penalty of reaching a goal from s is not greater than the immediate penalty of getting to s' from s plus the estimated (by the heuristic) accumulated penalty of reaching a goal from s'.

Definition 2 (admissibility)**.** *Q-value $\hat{Q}(s,a)$ is said to be admissible for the goal directed Q-learning with action-penalty representation if and only if each pair* (s, a)

$$(1)\ \hat{Q}(s,a) = 0 \text{ if } s \in \mathcal{G} \text{ and}$$
$$(2)\ \overset{\star}{Q}(s,a) \leq \hat{Q}(s,a) \leq 0 \text{ if } s \notin \mathcal{G},$$

where $\overset{\star}{Q}(s,a)$ is the real Q-value.

In turn, admissibility means that for all state-action pairs $(-\hat{Q}(s,a))$ never overestimates $(-\overset{\star}{Q}(s,a))$. It may be easily verified that uniformly initialized (e.g., zero-initialized) Q-values are monotonic and admissible.

According to our approach multiagent Q-values are initialized by using precalculated single-agent state utilities and single-agent transition function as follows:

$$\hat{Q}^j(\mathbf{s}, (a^j, \mathbf{a}^{-j})) \leftarrow \overset{\star}{Q}^j(s^j, a^j),\ \ \forall \mathbf{a}^{-j},\ \forall \mathbf{s}^{-j} \tag{4.4}$$

where $\mathbf{s}$ is a multiagent state, s^j is the j's component of the vector $\mathbf{s}$ (in other words, s^j is the agent j's state in the corresponding single-agent world) and $\overset{\star}{Q}^j(s^j, a^j)$ is an optimal single-agent Q-value that is calculated from the single-agent solution and the model as follows:

$$\overset{\star}{Q}^j(s^j, a^j) = R(s^j, a^j) + \gamma \sum_{s'^j} T^j_{s,a,s'} U(\overset{\star}{\pi}^j(s'^j)) \tag{4.5}$$

where $T^j_{s,a,s'}$ denotes $T(s^j, a^j, s'^j)$, the single-agent transition function, and $U(\overset{\star}{\pi}^j(s^j))$ is the utility of the single-agent state s^j according to the optimal policy $\overset{\star}{\pi}^j$.

Let's now show that in the goal directed stochastic games with action-penalty representation, the estimated multiagent Q-values for player j, $\hat{Q}^j(\mathbf{s},\mathbf{a})$, initialized with a single-agent solution are admissible and monotonic. To do that, let's prove the following theorem.

Theorem 1. *If in a goal directed stochastic game with action-penalty representation, Q-values $\hat{Q}(\mathbf{s},\mathbf{a})$ are initialized using the utilities of the corresponding single-agent state-action pairs according to equation (4.4), then these Q-values are admissible and monotonic.*

The proof of the above Theorem results from the following two Lemmas.

Lemma 1. *If in a goal directed stochastic game with action-penalty representation, Q-values of agent j, $\hat{Q}^j(\mathbf{s},\mathbf{a})$, are initialized according to equation (4.4), then these Q-values are monotonic.*

Proof. Let $\mathbf{G}$ be the set of multiagent goal states. Obviously, if $\mathbf{s} \in \mathbf{G}$ hence $\hat{Q}^j(\mathbf{s},\mathbf{a}) = 0$, $\forall \mathbf{a}$. Therefore, we must show that if $\mathbf{s} \notin \mathbf{G}$ then $0 \geq \hat{Q}^j(\mathbf{s},\mathbf{a}) \geq R^j(\mathbf{s},\mathbf{a}) + E_{\mathbf{s}':T(\mathbf{s},\mathbf{a},\mathbf{s}')\neq 0}\left[U^j(\Pi(\mathbf{s}'))\right]$. To do that, let us demonstrate that $0 \geq \overset{\star}{Q}{}^j(s^j,a^j) \geq R^j(\mathbf{s},\mathbf{a}) + E_{\mathbf{s}':T(\mathbf{s},\mathbf{a},\mathbf{s}')\neq 0}\left[U^j(\Pi(\mathbf{s}'))\right]$. In fact, since the rewards are negative in all states except the goal states, where they are 0, therefore $0 \geq \overset{\star}{Q}{}^j(s^j,a^j)$. As soon as, according to equation (4.5), $\overset{\star}{Q}{}^j(s^j,a^j)$ are defined as $R^j(s^j,a^j) + E_{s'^j:T(s^j,a^j,s'^j)\neq 0}\left[U^j(\overset{\star}{\pi}{}^j(s'^j))\right]$ and since inequalities (4.2) and (4.3) hold, hence $0 \geq \overset{\star}{Q}{}^j(s^j,a^j) \geq R^j(\mathbf{s},\mathbf{a}) + E_{\mathbf{s}':T(\mathbf{s},\mathbf{a},\mathbf{s}')\neq 0}\left[U^j(\Pi(\mathbf{s}'))\right]$ □

Lemma 2. *If in a goal directed stochastic game with action-penalty representation, Q-values of agent j, $\hat{Q}^j(\mathbf{s},\mathbf{a})$, are initialized according to equation (4.4), then these Q-values are admissible.*

Proof. Let $\mathbf{G}$ be the set of multiagent goal states. Evidently, if $\mathbf{s} \in \mathbf{G}$ therefore $\hat{Q}^j(\mathbf{s},\mathbf{a}) = 0$, $\forall \mathbf{a}$. Hence, we must demonstrate that if $\mathbf{s} \notin \mathbf{G}$ then $0 \geq \hat{Q}(\mathbf{s},\mathbf{a}) \geq \overset{\star}{Q}{}^j(\mathbf{s},\mathbf{a})$. To do that let's demonstrate that $0 \geq \overset{\star}{Q}{}^j(s^j,a^j) \geq \overset{\star}{Q}{}^j(\mathbf{s},\mathbf{a})$. Since the rewards are negative in all states except the goal states, where they are 0, therefore $0 \geq \overset{\star}{Q}{}^j(s^j,a^j)$. Since, (i) according to equation (4.5), Q-values $\overset{\star}{Q}{}^j(s^j,a^j)$ are defined as $R^j(s^j,a^j) + E_{s'^j:T(s^j,a^j,s'^j)\neq 0}\left[U^j(\overset{\star}{\pi}{}^j(s'^j))\right]$ and as long as (ii) by definition $\overset{\star}{Q}{}^j(\mathbf{s},\mathbf{a}) = R^j(\mathbf{s},\mathbf{a}) + E_{\mathbf{s}':T(\mathbf{s},\mathbf{a},\mathbf{s}')\neq 0}\left[U^j(\Pi(\mathbf{s}'))\right]$ and since (iii) inequalities (4.2) and (4.3) hold, hence, it follows that $\overset{\star}{Q}{}^j(s^j,a^j) \geq \overset{\star}{Q}{}^j(\mathbf{s},\mathbf{a})$ □

The complexity reduction of the single-agent Q-learning, proven by Koenig and Simmons (1996), suggests that in the multi-agent case, if similar conditions are satisfied, one can expect a complexity reduction as well.

More precisely, if in a stochastic game with action-penalty representation (1) the multiagent Q-learning is performed by a game-theoretic algorithm having a convergence property for this stochastic game, and (2) if it is initialized using an approximative function, which has the admissibility and monotonicity properties, one can expect that the complexity of the learning process may be reduced as compared to zero-initialized (uninformed) case.

In the next section we provide the results of the tests produced on several examples of the four-robot version of the grid world problem, which justify the above hypothesis.

The algorithm of Initialized Adaptive Play Q-learning is presented in Algorithm 4.1.

This algorithm is analogous to the APQ algorithm (Algorithm 3.6). The only difference is that the equation (4.4) is used to initialize the Q-values of unknown states (i.e., the states that has not been previously visited).

4.2.3 Conclusion

In this Section we presented the Initialized Adaptive Play Q-learning algorithm which combines Adaptive Play Q-learning with initial heuristic approximation of the multiagent Q-values by using a single-agent learning or planning solution.

We defined the admissibility and monotonicity properties of Q-values and showed that the initialization of multiagent Q-values using a single-agent planning solution preserves these properties for the case of the stochastic games with action-penalty representation. The results of the empirical evaluation of the algorithm, as well as the discussion about them, will be given in Chapter 5.

In the following section we propose an effective algorithm of learning in stochastic games. It is able to outperform the Opponent Modelling algorithms and Gradient Based algorithms in adversarial games as we will show by comparing it with Adaptive Play Q-learning and Infinitesimal Gradient Ascent algorithms. In two-player self-play, when possible, our algorithm is able to converge to a Pareto optimal strategy that maximizes the welfare of both players instead of an equilibrium strategy which can be favorable to one player only or to nobody at all.

function INITIALIZED-APQ($\mathbf{S}, \mathbf{s}_0, tr_{max}, it_{max}$)
  returns: a policy.
  inputs: tr_{max}, maximal number of trials,
    it_{max}, maximal number of iterations within one trial,
    $\mathbf{S}$, the set of the environment's states,
    $\mathbf{s}_0 \in \mathbf{S}$, the initial state.

  Current trial $tr \leftarrow 0$.
  Current iteration $it \leftarrow 0$.
  Current state $\mathbf{s}_t \leftarrow \mathbf{s}_0$.
  Previous state $\mathbf{s}_{t-1} \leftarrow null$.
  while $tr < tr_{max}$ **do**
    if $\mathbf{s}_t$ has not been visited **then**
      Initialize Q-values in $\mathbf{s}_t$ using equation (4.4).
    Sample a reduced profile $\hat{\Pi}^{-j}(\mathbf{s}_t)$ from history using equation (3.2).
    Select a best response strategy $\hat{\pi}^j(\mathbf{s}_t)$ to the profile $\hat{\Pi}^{-j}(\mathbf{s}_t)$.
    Calculate the expected utility $U^j(\hat{\Pi}(\mathbf{s}_t))$ of the strategy profile $\hat{\Pi}(\mathbf{s}_t)$ using equation 2.4.
    if $it < it_{max}$ and $\mathbf{s}_t$ is not terminal state **then**
      Execute strategy $\hat{\pi}^j(\mathbf{s}_t)$ and observe the joint action played, $\mathbf{a}^{-j}$, and the reward obtained, R^j.
      Update the history in $\mathbf{s}_t$ using $\mathbf{a}^{-j}$.
      if $\mathbf{s}_{t-1} \neq null$ **then**
        Update Q-value in $\mathbf{s}_{t-1}$ using the Q-value update rule, the reward R^j and the utility $U^j(\hat{\Pi}(\mathbf{s}_t))$.
      Update the previous internal state $\mathbf{s}_{t-1} \leftarrow \mathbf{s}_t$ and observe the new state $\mathbf{s}_t$.
      Increment iteration $it \leftarrow it + 1$.
    else
      Randomly sample the new trial's start state $\mathbf{s}_t$ from the set of visited states.
      Reset previous internal state $\mathbf{s}_{t-1} \leftarrow null$.
      Reset iteration counter $it \leftarrow 0$.
    Increment trial counter $tr \leftarrow tr + 1$.
  return $\hat{\pi}^j(\mathbf{s})$ for all visited state $\mathbf{s}$.

Algorithm 4.1: Initialized Adaptive Play Q-learning algorithm for player j.

4.3 Multiagent Learning in Adaptive Dynamic Systems

4.3.1 Introduction

Classically, an approach to the multiagent policy learning assumes that the agents, by means of interactions and/or by using preliminary knowledge about the reward functions of all players, would find an interdependent solution called "equilibrium". One of the most widely used concepts of equilibrium is the Nash equilibrium where each agent in a multiagent system (MAS) plays its best response to the other players' strategies and a unilateral deviation of a player from the equilibrium strategy decreases its own utility.

There are two basic approaches to find a Nash equilibrium. The first one is a game theoretic approach which supposes the complete knowledge of the reward structure of the underlying game by all the agents. In such an approach, each agent finds an equilibrium, by using mathematical programming, and all the agents play on it. But a problem arises when there are several equivalent equilibria in a game and the agents have calculated the different ones. Another problem is that the agents, by calculating an equilibrium, assume that the other agents are rational and, thus, they will also follow this solution. But what if certain agents are not rational, or play a fixed strategy, or evolve according to some fixed rules, and what if some agents know (or are able to deduct) this and could exploit this knowledge to augment their utilities? As yet, there is no equilibrium concept which can answer those questions.

The second approach to find an equilibrium is the *adaptive* one, which assumes that the agents learn by adapting to each other in self-play (i.e., all the agents use the same learning algorithm) and do not know the reward structure of the game and are only able to make actions and observe their own rewards and, in some approaches, the actions made by the others. As we have shown in the previous chapter, certain algorithms of this class converge to a Nash equilibrium (or to a utility that is equivalent to the utility of a Nash equilibrium). Among these algorithms, the most widely used ones are Joint-Action Learning (Claus and Boutilier, 1998), Infinitesimal Gradient Ascent (IGA)[1] (Singh et al., 1994), Policy Hill-Climbing (Bowling and Veloso, 2002) and Adaptive Play Q-learning (Gies and Chaib-draa, 2006) (a Q-learning based extension of the Adaptive Play algorithm (Young, 1993)).

[1] IGA is the only algorithm among those listed which requires a complete knowledge of the opponent's current strategy and reward structure of the game to calculate the gradient.

As we have shown, adaptive players[2] learn their policies separately from the maintenance of the beliefs about their counterparts' future actions and make their decisions based on that policy and the current belief. These decisions can be in pure or in mixed strategies depending on the algorithm in question.

Recently, certain researchers (Shoham et al., 2003) question the necessity and the validity of the concept of equilibrium as the most important multiagent solution concept. They rather point out the efficiency of a particular learning algorithm versus a certain class of counterparts. In this section we propose an efficient algorithm of learning in presence of the adaptive counterparts called Adaptive Dynamics Learner (ADL). This algorithm is able to learn an efficient policy over the opponents' adaptive dynamics rather than over the simple actions and beliefs. By so doing, ADL exploits these dynamics to obtain a higher utility than any equilibrium strategy can provide. We tested our algorithm on a representative set of repeated matrix games from the GAMUT test suit (Nudelman et al., 2004). The results show that ADL agent is highly efficient in self-play and against APQ and IGA agents[3].

We consider two adaptive learning algorithms only, Infinitesimal Gradient Ascent and Adaptive Play Q-learning, since they represent two basic subclasses of adaptive algorithms: those that are able to learn a pure strategy (APQ) and those that are able to learn a mixed one (IGA).

4.3.2 Adaptive Dynamics Learning

By adaptive dynamics we mean the dynamics of policy improvement, which is observed when the adaptive algorithms, such as PHC (Bowling and Veloso, 2002), IGA (Singh et al., 1994) and APQ (Gies and Chaib-draa, 2006) are learning in a stochastic game. These algorithms, as we have shown in Chapter 3, estimate the policy of the opponent players and improve their policy so as to be making a best response to the estimated opponents' policy.

Although the adaptive algorithms demonstrate an efficient behavior in self-play, Chang and Kaelbling (2001) showed that thay can be exploited by an agent that is in-

[2]To discriminate between adaptive player as a member of a class of learning agents and Young's Adaptive Player, we will write "adaptive player" or "adaptive algorithm" (in lower case) to denote the former and Adaptive Player (with a capital letter) for the latter.

[3]This comparison is correct as soon as APQ and IGA agents have the same or a greater quantity of available information about the world as compared to ADL, and, thus, if they behave poorer in a game, this is not because they can access less information.

formed about their properties. For example, they showed that a PHC player (Bowling and Veloso, 2002) can be exploited in adversarial games. Indeed, as we have shown in a previous section, their PHC-Exploiter agent was able to outperform the PHC player using the knowledge of the structure of PHC adaptation algorithm.

As was later shown by Tesauro (2004), it is possible to exploit the adaptive dynamics with a simple knowledge that the opponent is an adaptive player. Recall that the Hyper-Q learning algorithm (Tesauro, 2004) learned explicitly the Q-values of the mixed strategy profiles. To do that, the author discretized the probability space with a certain discretization size and empirically showed that Hyper-Q outperforms PHC and IGA players in RockPaperScissors game. But there are three major shortcomings that make this algorithm intractable in the real world. These are,

1. the discretization, which creates about 100 thousands of the virtual states for a game with merely two players and three actions, such as RockPaperScissors,
2. to obtain better results, Hyper-Q agent uses a computationally hard Bayesian belief update operation at each time step and
3. the game of total observability becomes partially observable because the beliefs about other player's strategies are represented in the form of probability distribution over all possible mixed strategies.

We propose here a much simpler adaptive dynamics learning algorithm called Adaptive Dynamics Learner (ADL), which associates a Q-value to each experienced game history H of fixed length p and a simple action $a^j \in \mathcal{A}^j$, and then learns these Q-values using a form of Q-learning. This substantially reduces the space of virtual states and actions comparatively with the Hyper-Q approach. More formally, ADL player j maintains in its memory a table H^j of histories of past joint actions, considered by it as the system's states. To each history $h^j \in H^j$ it associates a Q-value of the form $Q^j(h^j, a^j), \quad \forall a^j \in \mathcal{A}^j$. Thus we assume that our agent is able to perceive the actions made by the other players.

Being at time step t in the "state" $h_t^j = (a_{t-p}^j a_{t-p}^{-j} a_{t-p+1}^j a_{t-p+1}^{-j} \dots a_{t-1}^j a_{t-1}^{-j})$ the agent j searches in H^j the action a_t^j, which maximizes the Q-values for h_t^j. After that the agent j plays a_t^j with some exploration decreasing over time[4]. Having observed the opponents' joint action and its own reward, it updates its state at time $t+1$ by concatenating its own

[4]Although we did not studied our algorithm with different exploration strategies, we believe that any valid exploration technique (Thrun, 1992) for the reinforcement learning can be suitable here.

action a_t^j and the opponents' joint-action a_t^{-j} played at time t to h_t^j and by eliminating the first two entries, that is,

$$h_{t+1}^j = (a_{t-p+1}^j a_{t-p+1}^{-j} a_{t-p+2}^j a_{t-p+2}^{-j} \dots a_t^j a_t^{-j}) \tag{4.6}$$

Finally, the player j updates the Q-value in h_t^j corresponding to the action a_t^j by using following Q-learning update rule:

$$Q^j(h_t^j, a_t^j) \leftarrow (1-\alpha) Q^j(h_t^j, a_t^j) + \alpha \left(R^j(h_t^j, (a_t^j, a_t^{-j})) + \gamma U^j(h_{t+1}^j) \right) \tag{4.7}$$

where $U^j(h_{t+1}^j) = \max_{a^j \in \mathcal{A}^j} Q^j(h_{t+1}^j, a^j)$ and $R^j(h_t^j, (a_t^j, a_t^{-j}))$ is the reward obtained by j after playing a_t^j in the previous state h_t^j.

Notice that in the above Q-value update rule, for simplicity we omitted the joint states. Thus, we focused our attention on the repeated games only. One can augment ADL's internal states by concatenating joint states to the histories to extend its applicability to the multi-state stochastic games. However, due to the time limits this does not make part of the present work.

The complete formal definition of ADL algorithm is presented in Algorithm 4.2. The algorithm starts by initializing the current history with an empty sequence and by selecting a random action to play (lines 6-7). While the maximum number of learning iteration is not reached, it executes the selected action with some exploration (line 9), observes the reward obtained and the joint action made by the opponent players (line 10) and obtains the next state by using equation (4.6) (line 11). Then it executes a search in the table of Q-values for the next state and finds an action which maximizes the utility (line 12), updates Q-value of the previous state (line 13) and proceeds to the next iteration (lines 14-16).

4.3.3 Conclusion

In this section we have presented Adaptive Dynamics Learner, an algorithm that uses Q-learning to learn a best response strategy to the *dynamics* of its opponent instead of a best response to simple actions. The previous attempts to exploit the opponent's dynamics were limited on focusing on the specific internal structure of the opponent (PHC-Exploiter by Chang and Kaelbling (2001)) or on a computationally expensive algorithms (Hyper-Q by Tesauro (2004)).

Our approach is simpler than Hyper-Q and more general (as compared to PHC-Exploiter). In Chapter 5 we will show that it outperforms two currently best performing

function ADL(p, t_{max})
  returns: a policy.
  inputs: p, maximum history length,
    t_{max}, maximum time.

  Current time $t \leftarrow 0$.
  Current state $h_t^j \leftarrow EmptySequence$.
  $a_t^j \leftarrow RandomAction$.
  while $t \leq t_{max}$ **do**
    Play a_t^j with some decreasing exploration.
    Observe the reward R_t^j and the joint-action a_t^{-j}.
    Obtain new state h_{t+1}^j using (4.6) with h_t^j, a_t^j and a_t^{-j}.
    if h_{t+1}^j has not been visited **then**
      $\forall a^j$, initialize uniformly Q-values in h_{t+1}^j.
    Find an action a_{t+1}^j maximizing Q-values in h_{t+1}^j and calculate the utility $U^j(h_{t+1}^j)$.
    Update $Q(h_t^j, a_t^j)$ using equation (4.7) with R_t^j and $U^j(h_{t+1}^j)$.
    Update current state $h_t^j \leftarrow h_{t+1}^j$.
    Save the action to play $a_t^j \leftarrow a_{t+1}^j$.
    Increment the time $t \leftarrow t + 1$.
  return current policy.

Algorithm 4.2: Adaptive Dynamics Learner (ADL) algorithm for player j.

multiagent learning algorithms, IGA and APQ, in a set of adversarial games and is able to converge in self-play to a Pareto optimal strategy in general sum games, such as PrisonersDilemma.

4.4 Conclusion

To overcome two of three major drawbacks of the modern multiagent learning algorithms, namely, high computational complexity and focus on the convergence to the one shot Nash equilibrium, we proposed two novel approaches:

1. Q-initialization in multiagent context and
2. Adaptive Dynamics Learning.

The first algorithm, Initialized Adaptive Play Q-learning (Burkov and Chaib-draa, 2007c), permits, as we will show in the next Chapter, to considerably reduce the complexity of multiagent learning in a class of stochastic games, called "goal-directed stochastic games with action-penalty representation". The second algorithm, called ADL (for Adaptive Dynamics Learner (Burkov and Chaib-draa, 2007a;b)) will be shown to be very effective against adaptive opponents in adversarial games and to be able converge in self-play to a Pareto optimal solution, when this is possible.

In the next chapter we present the results of the experiments chosen to support of the proposed approaches. As test benches in our experiments, we use modifications of a coordination stochastic game by Hu and Wellman (2003) and a GAMUT test suite (Nudelman et al., 2004), which are commonly used by researchers when evaluating game-theoretic algorithms.

Chapter 5

Results and Discussion

5.1 Introduction

In this chapter we present the results of experiments taken over the proposed algorithms: Initialized Adaptive Play Q-learning (IAPQ) (Burkov and Chaib-draa, 2007c) and Adaptive Dynamics Learner (ADL) (Burkov and Chaib-draa, 2007a;b). The IAPQ algorithm was tested in two similar stochastic games, two- and four-player grid worlds. These two games are examples of coordination problem, which is frequently used to test cooperative robotics learning algorithms. Adaptive Dynamics Learner was tested in about twelve of the most representative repeated games from GAMUT (Nudelman et al., 2004). They include PrisonersDilemma, RockPaperScissors, MatchingPennies and others, against two adaptive learning algorithms, IGA and APQ, as well as against stationary players.

5.2 Initialized Adaptive Play Q-learning

5.2.1 Implementation Details

We defined the following values of the adjustable parameters of the IAPQ algorithm: the learning rate, α, is proper for each state-action pair and decreases gradually using the *search-then-converge* schedule suggested by Darken and Moody (1991) depending

on the number of updates of the respective Q-value:

$$\alpha_t(s, \mathbf{a}) = \frac{\alpha_0 \tau}{\tau + n_t(s, \mathbf{a})}$$

where t is the current learning time step, α_0 is the initial value of the learning rate and $n_t(s, \mathbf{a})$ is the number of times that the Q-value for the joint action $\mathbf{a}$ has been updated in state s to time t. We set $\alpha_0 = 0.5$ and $\tau = 100$. In general, the exact value of τ is not determinative for the result of the learning, but it can influence the rapidity of the process, since it determine the length of the exploration period. We have chosen this value since the structure of the problem we consider is similar to the problem considered in the paper (Barto et al., 1995), where the search-then-converge exploration technique was also used. It can be shown that this schedule satisfies the hypotheses of the theorem about the convergence of Q-learning.

The tests were conducted on a machine equipped with two processors of 2.6 GHz each and 4 GB of RAM.

5.2.2 Two-Agent Grid World

Let's first illustrate our algorithm on a simple case, where there are merely two learning agents in a grid based environment, called *two-robots-on-the-grid* or *two-player grid world* (Hu and Wellman, 2003) (Figure 5.1).

We already presented this environment in Section 2.6 but with a limited number of cells in the environment. Now we consider an environment that can be arbitrarily large. It is easy to see that in the single-agent case there are six optimal single-agent trajectories for each robot. In the multiagent case, however, some of these trajectories when used simultaneously can produce a collision.

Obviously, a Nash equilibrium in this stochastic game is a pair of trajectories when both agents make the minimum of transitions, avoid a collision and reach their respective goals. As was shown in (Gies and Chaib-draa, 2006), in the deterministic case, when actions of the agents succeed with probability 1, there are ten such equilibria. Figure 5.2 shows five of them and the other five can by obtained by symmetry. Notice that all of these Nash equilibria are Pareto optimal.

The stochastic case equilibria are more numerous because of transition failures, but the principle is the same.

The grid world we used contains $23 \times 23 = 529$ cells and, thus, the two-agent state

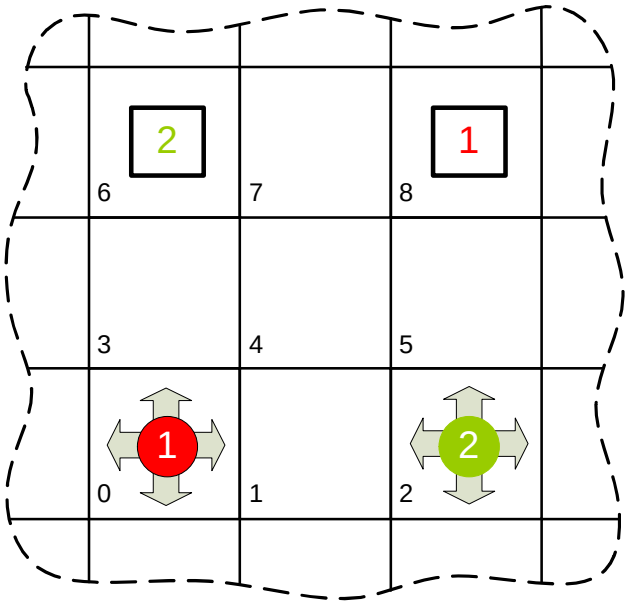


Figure 5.1: A fragment of the two-player grid world environment containing the start and goal positions of agents. The total number of cells in the grid may be arbitrarily big.

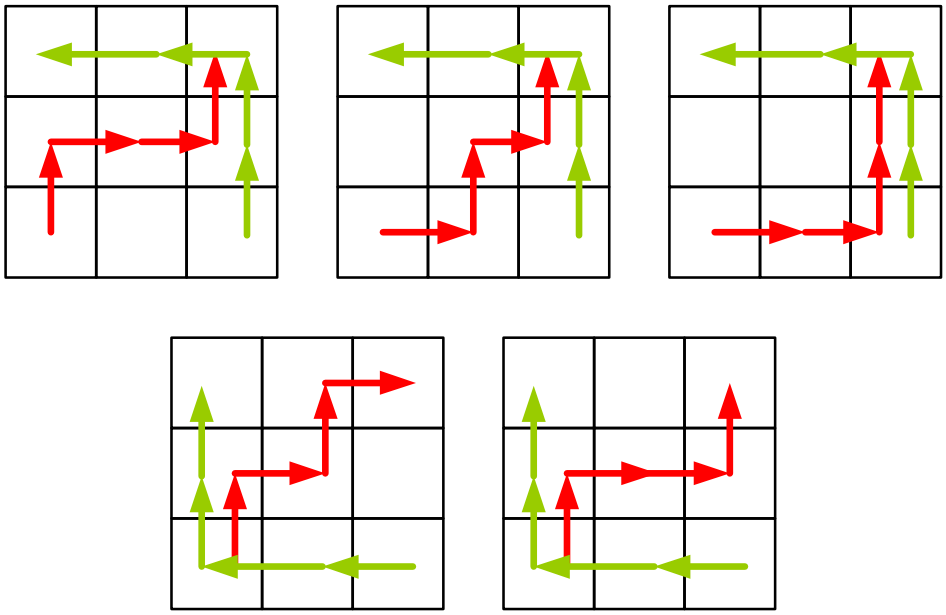


Figure 5.2: Nash equilibria for two-player grid world. Another five are obtained by symmetry.

space contains about $529^2 \backsimeq 280,000$ states. The agents were placed in the grid as shown in Figure 5.1. For the grid 3×3 which contains merely nine cells and two-agents, a good stable result was obtained after $5,000$ trials[1], as shown in (Gies and Chaib-draa, 2006). So, in our case, with a much bigger environment, if we do the basic adaptive play Q-learning and do not use the initial heuristic initialization, the learning would require about 17 millions trials, proportionally to the raise of the state space. However, if in the beginning of the learning the agents used the Q-values calculated using equation (4.4), our experiments showed that in deterministic case (when there are not action failures, or $Pr(failure) = 0$), $10,000$ trials were sufficient to learn an optimal Nash equilibrium in the 100% of the runs. During learning, the agents visited not more than $1,500$ states.

On the other hand, when the transition function was stochastic, occasionally during execution the agents got to a state having small probability to be reached starting from the initial state. Hence, the agents got to a region of the state space that was rarely visited during learning, and, therefore, contains the Q-values, which are possibly not optimal. To overcome this shortcoming, the sampling statement at line 25 of Algorithm 4.1 should be modified so as to give higher probability of selection to the states that were visited less to permit the Q-values in these states to converge to their real values. For that purpose, the Boltzmann distribution over all visited states can be used. According to these considerations, the probability that a state s will be selected as a start state at the beginning of a trial is given as follows:

$$Pr(\mathbf{s}_t = \mathbf{s}) = \frac{e^{-m(\mathbf{s})/T}}{\sum_{\mathbf{s}' \in \mathcal{V}} e^{-m(s')/T}} \tag{5.1}$$

where $\mathcal{V}$ is the set of visited states, m is the number of times the state $\mathbf{s}$ was visited during learning and T is some positive value.

Our tests showed that if modified according to equation (5.1), Algorithm 4.1 after $100,000$ learning trials in self-play found the optimal equilibrium solution in 95% of the runs, after $200,000$ learning trials the optimal equilibria were observed in 97% of the runs, and after $250,000$ learning trials the optimal equilibria were observed in 99% of runs. Thus, in stochastic case the solution seems to be $\epsilon - optimal$ with some value of ϵ, decreasing with the number of learning trials.

We compared the behavior of our algorithm with and without the initial Q-values heuristic approximation[2]. The experiments showed (Figure 5.3) that in the stochastic environment restricted to 5×5 cells zero-initialized agents ($Q_0^j(\mathbf{s}, \mathbf{a}) = 0, \ \forall \mathbf{a}, \ \forall \mathbf{s}$)

[1] A trial is a sequence of actions starting from an arbitrary state and finishing in the final state.

[2] Without Q-values approximation our algorithm behaves as the basic adaptive play Q-learning algorithm proposed by Gies and Chaib-draa (2006).

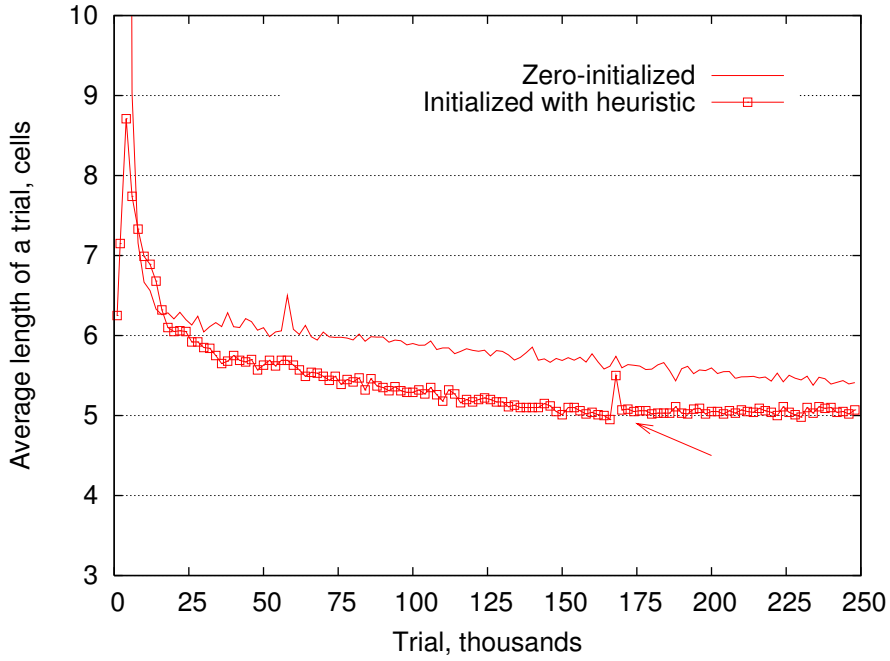


Figure 5.3: The dynamics of learning in the grid 5×5. The curves show the average length of a trial as a function of the number of learning trials. The arrow points to a trial starting with which the agents initialized with the heuristic found an optimal solution but the zero-initialized ones did not.

explored all possible 600 states and converged to an equilibrium solution in 99% of tests after $450,000$ trials, while the agents that were initialized according to our approach, explored about 570 states and converged to an equilibrium as early as after $250,000$ trials. But more impressive were the results obtained for the full sized environment, 23×23 cells. After $400,000$ trials, the zero-initialized agents explored almost every possible state ($\sim 250,000$) and had not find any equilibrium solution, while the initialized agents explored about $10,000$ states (only) and converged to an optimal equilibrium solution in 99% of tests after the same number of learning trials ($400,000$).

To observe the behavior of the algorithm with different initial values of the learning rate, α_0, and to find the optimal one, if it exists, we tested our algorithm with the values of α_0 in the range between 0.1 and 0.9. Interestingly, the number of visited states grows uniformly with the growth of α_0, but the number of trials needed to obtain the optimal solution decreases to some minimum value up to $\alpha_0 \approx 0.6$, and then starts to grow rapidly (Figure 5.4). Hence, there should exist an optimal value of α_0, with which the number of trials required to converge to a solution will be minimal.

To estimate the scalability of the algorithm with the number of the grid cells (the

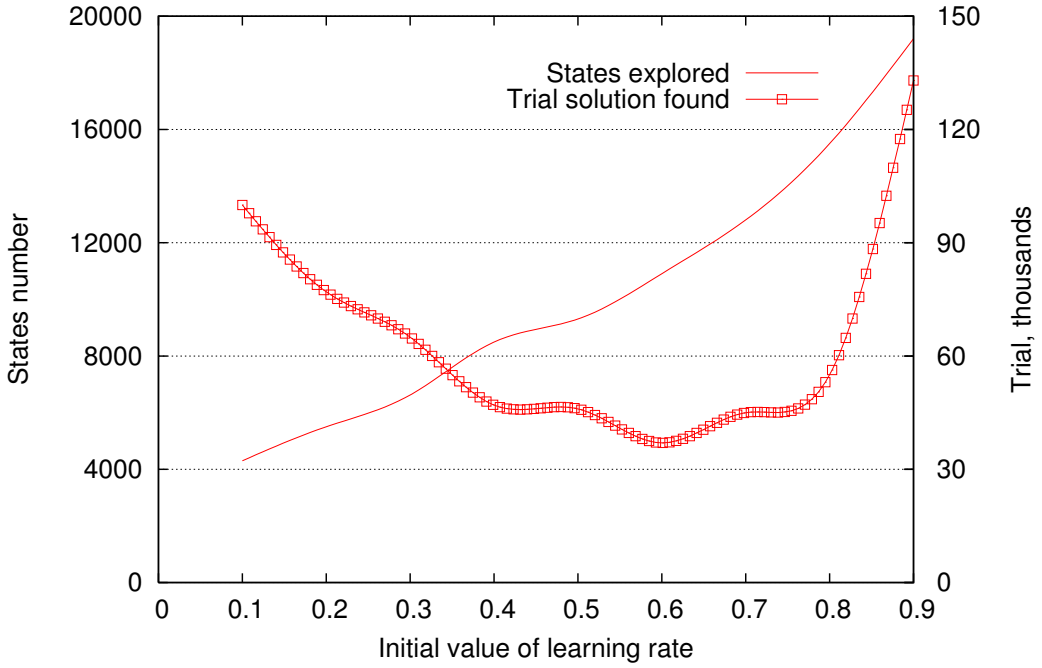


Figure 5.4: Number of states explored depending on the initial value of α.

scalability with the number of agents will be discussed in the following section) we tested its behavior on different configurations of the grid. Particularly, we studied the number of states explored by the agents as a function of the Manhattan distance between the start and goal cells[3]. We discovered, as shown in Figure 5.5, that the number of states explored by the agents grows almost linearly with the Manhattan distance to the goal. This is not surprising since the multiagent solutions differs from the corresponding single-agent trajectories by a restricted set of states used by agents to keep off collisions and to avoid mistiming.

Further, to discover how much the difference in the single-agent reward structure and multiagent one influence the learning process, we tested our algorithm on the different values of the collision loss (which is absent in the single-agent problem). We remarked in Figure 5.6 that there was slight dependence of the number of explored states during learning on the value of the collision loss, but after 0.5 the number of explored states became almost constant. The explanation is that higher values of the collision loss force the agents to explore more distant states adjacent to the states preceding to the collision. But after attainment of some "safe" distance (which depends on the dynamic of the

[3]The Manhattan distance to the goal cell is the minimal number of transitions required to perform to reach the goal cell starting in the start cell, assuming that the world is deterministic.

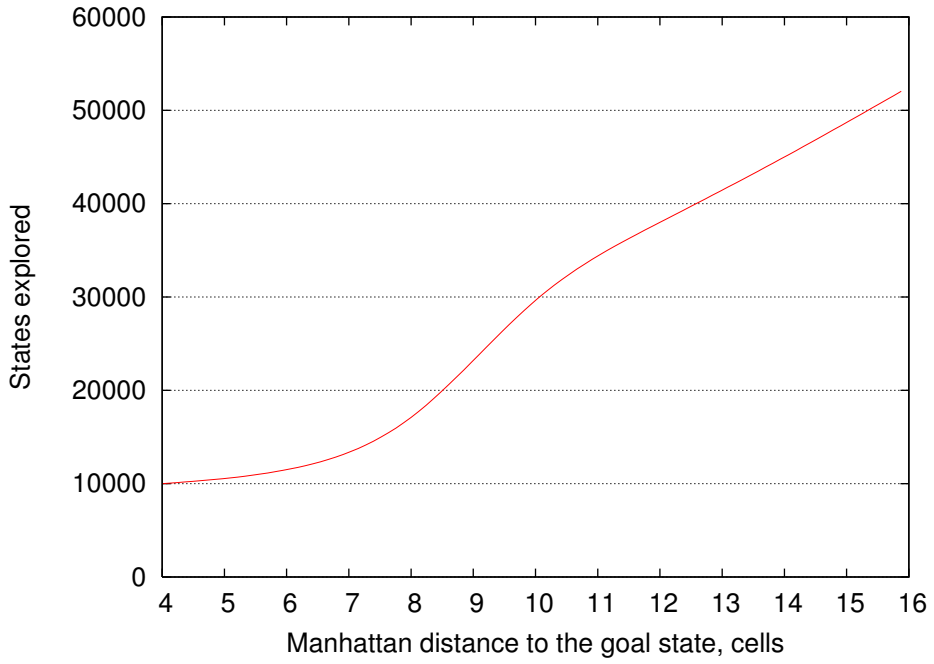


Figure 5.5: Number of states explored depending on the Manhattan distance between the start and goal states.

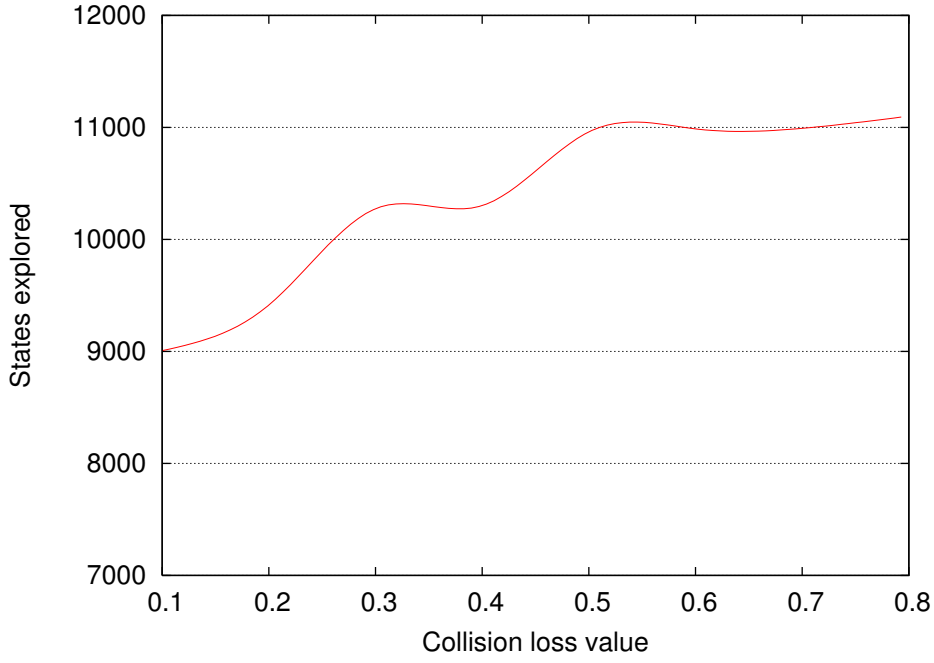
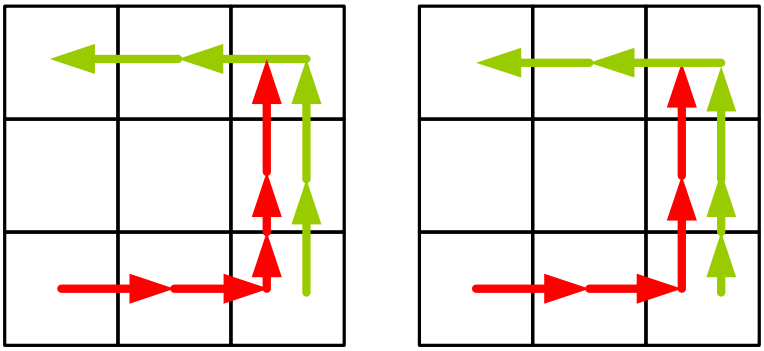


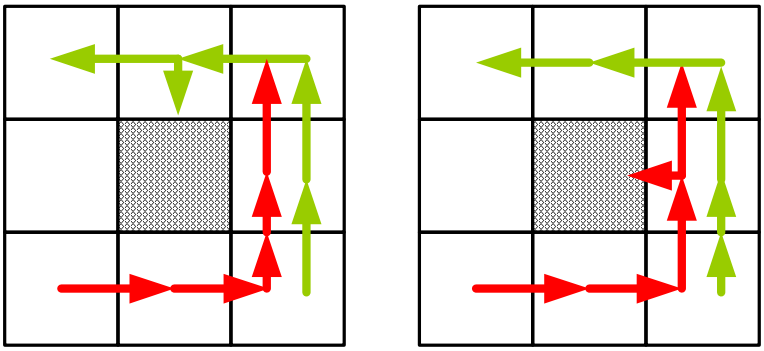
Figure 5.6: Number of states explored depending on the value of the collision loss.

environment and its stochasticity), the agents do not explore more states regardless of the value of the collision loss.

We also investigated some other interesting phenomena. For example, to show that the final solution may substantially differ from the single-agent trajectory, we set the loss of 0.15 for both agents if they reached their respective goals non-synchronously (notice, that a combination of two single-agent solutions cannot in general be appropriate multiagent solution in this case because of the nonzero probability of the action failure). We observed that if there was no wall near the goal cell (Figure 5.7 (a)), in case of occasional mistiming the agents did not attempt to synchronize, because it would require one agent to stop and "wait" another one, but there was no action "wait" in the set of actions of both agents. However, if there was a wall nearby (Figure 5.7 (b)), in case of mistiming the agent that was ahead decided to hit the wall once to stay put and "wait" another agent for one step.



(a) there is no wall



(b) there is a wall

Figure 5.7: Mistiming case equilibria. The left and right images are a cases where the agents 1 and 2 respectively are late.

Finally, we investigated some other dependencies. In short, we observed that the function of number of states explored by the agents is close to be linear in the distance

between the start and goal cells (hence in the size of the relevant subset of the state space). Furthermore, the value of the collision loss (or, essentially, the difference between single-agent and multiagent models) influences weakly the number of explored states, or, in the other words, the initial approximation of Q-values defines substantially the course of the learning process. These features allow us to draw the conclusion about the efficiency of the initial approximation of Q-values.

However, more impressive were the results obtained in the tests made with four learning agents in the same environment. As we will show in the following section, the heuristically initialized case outperformed considerably the zero-initialized one in time and state space of the learning process.

5.2.3 Four-Agent Grid World

Let us now go further and model a stochastic game with more than two learning agents, and let's demonstrate the complexity reduction in this case. To achieve that we have adopted a four-agent grid world as presented in Figure 5.8.

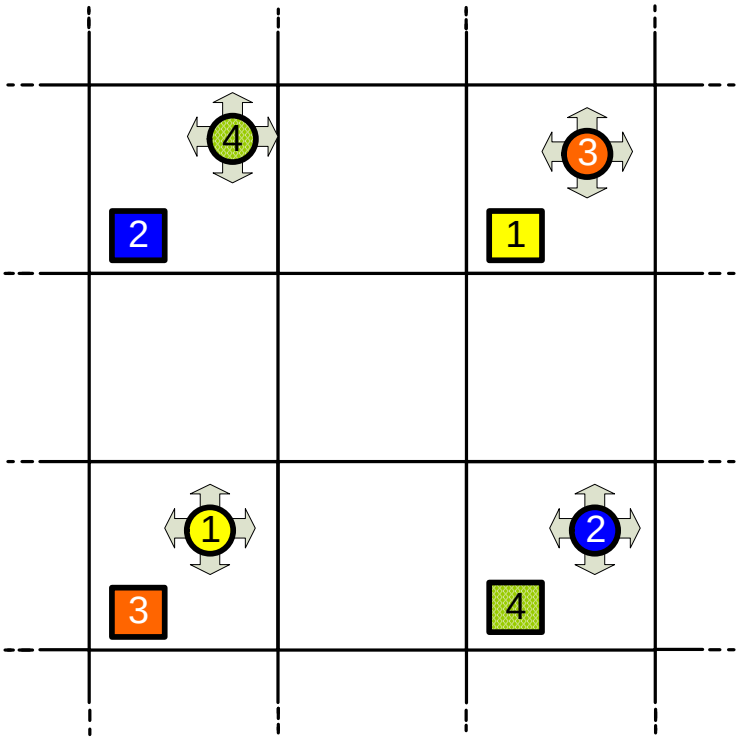


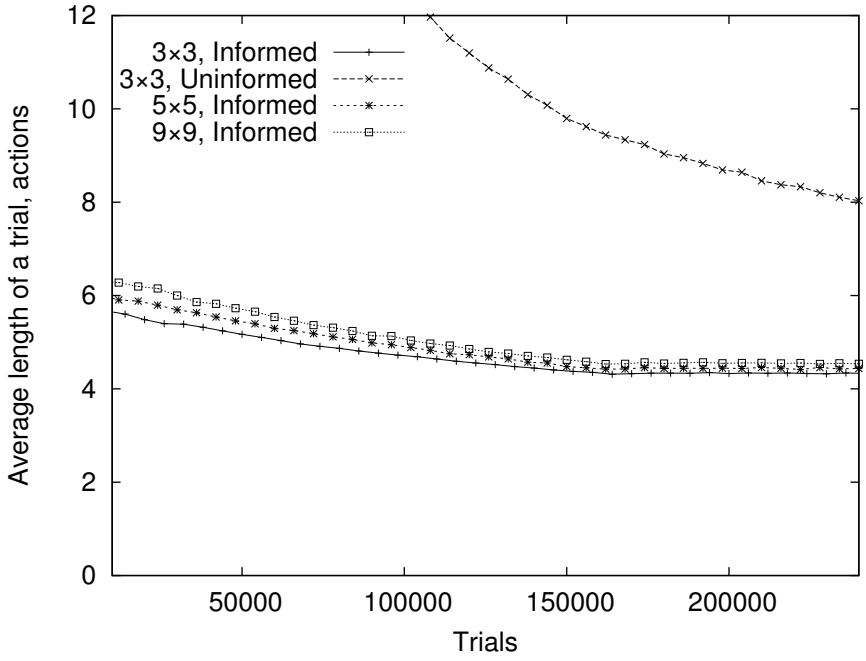
Figure 5.8: A fragment of the four-robot grid world environment containing the start and goal positions of agents.

There are four agents on a grid, each one having its own goal to reach. The dynamics of the environment and the rewards the agents obtain in similar situations are the same as they were in two-agent grid world. I.e., in each state, the probability that a transition action will be performed with success is 0.99. The reward robots obtain by performing actions is negative (-0.04) in all states except the goal states. In the goal states, the reward for each action is 0. In the case of collision, all affected robots keep their positions and receive the collision loss of (-0.1).

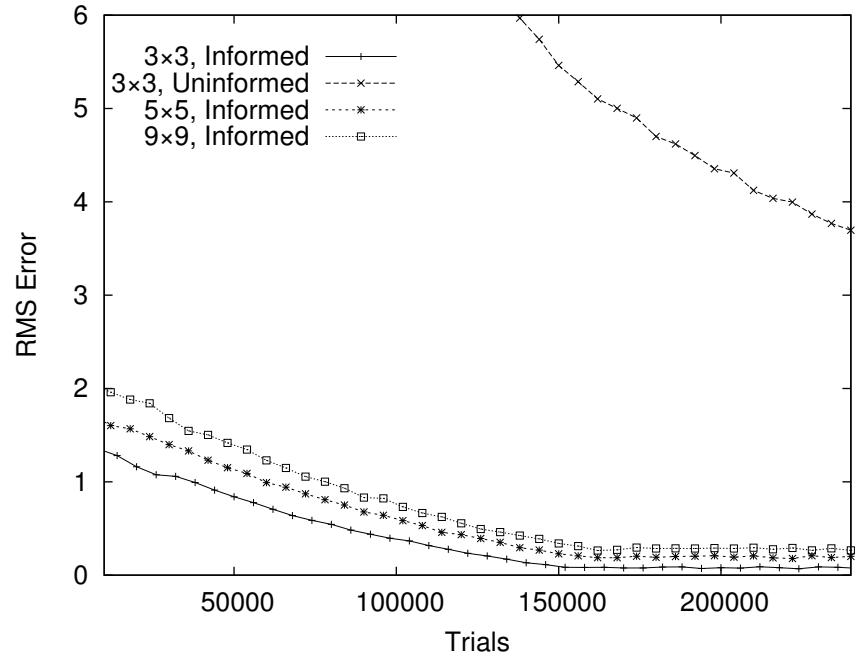
As it is easy to see, this is a goal directed coordination stochastic game with action-penalty representation. Hence, in self-play APQ initialized with monotonic and admissible Q-values must converge to an equilibrium, as it was observed in the two-agent case.

We tested our algorithm on this example in a zero-initialized (called "uninformed") case and in a case (called "informed") when Q-values were initialized using the single-agent solution, calculated via a simple value iteration. The grid sizes we considered were 3×3, 5×5 and 9×9 cells, all with the same start and goal positions. The dynamics of the learning process is presented in Figure 5.9. The first diagram (5.9(a)) illustrates the average number of actions made by the agents before they reached their goals with respect to learning trials. The second diagram (5.9(b)) represents the evolution of the root-mean-square (RMS) error of the Q-values as a function of learning trials. (Recall that in statistics the root-mean-square (RMS) error is determined by calculating the deviations of points from their true position, summing up the measurements, and then taking the square root of the sum.) The utilities and errors were averaged over 20 runs of $250,000$ trials each. In the uninformed case the results are presented for 3×3 grid only since the uninformed algorithm became intractable on our machine starting from the 5×5 cells due to the fact that it explored almost the entire state space, which is exponential in the number of agents, while the informed algorithm visited a very small relevant subset of the states.

As one can see in Figure 5.9, while the heuristically initialized Q-learning converged to an optimal solution after merely $250,000$ trials in all grids, the uninformed algorithm was still far from convergence even in a very small grid. Notice that the convergence of the informed (heuristically initialized) learning depends weakly on the dimensions of the grid, while in the uninformed case the size of the grid is critical for the tractability of the algorithm. This means that the heuristic function permits to effectively "focus" the learning process on a small subset of the total problem's state space, which is the main reason for such considerable learning complexity reduction. Additional test results are presented in Table 5.1. In this table, the infinity sign means that we did not succeed in obtaining a converged result due to intractability of the uninformed



(a)



(b)

Figure 5.9: The dynamics of the learning process.

algorithm, ISF means "Iteration Solution Found", SE is for "States Explored" and MAIL is for "Maximal Average Iteration Length".

Table 5.1: Test results for various problem settings.

	3×3		5×5		9×9	
	Uninformed	Informed	Uninformed	Informed	Uninformed	Informed
ISF	$1,150,000$	$150,000$	∞	$155,000$	∞	160,000
SE	$5,640$	$3,700$	$> 350,000$	$7,000$	$> 40,000,000$	11,000
MAIL	28.6	6.5	165.8	6.8	2645.5	7.1

5.3 Adaptive Dynamics Learner

5.3.1 Implementation Details

To compare our algorithm with adaptive ones, we programmed two adaptive learning algorithms, IGA and APQ. In the following subsections we will provide the implementation details for each of the programmed algorithms.

Adaptive Play Q-learning Agent (Gies and Chaib-draa, 2006)

The APQ agent we used has the following characteristics. The length of the history, p, is 16, the size of sampling, l, is 8, the discount factor, γ, is 0.9, the learning rate, α, is proper for each state-action pair and decreases gradually using the *search-then-converge* schedule depending on the number of updates of the respective Q-value:

$$\alpha_t(h^j, a^j) = \frac{\alpha_0 \tau}{\tau + n_t(h^j, a^j)}$$

where t is the current time step, α_0 is the initial value of the learning rate and $n_t(h^j, a^j)$ is the number of times that Q-value for the action a^j was updated in state h^j to time t. We set $\alpha_0 = 0.5$ and $\tau = 10,000$ (the same values for all programmed algorithms). We let τ be equal to $10,000$ (which is greater than the value used in the APQ algorithm), because we remarked that smaller values of τ made the convergence process slower. We observed that some histories were visited by ADL much more often than others. This can make the Q-values in these states to converge too fast, i.e. before than the agent will learn good values in the other states. We believe that a more appropriate exploration strategy could fix this problem. We will consider this in our future research.

Infinitesimal Gradient Ascent Agent (Singh et al., 1994)

IGA algorithm assumes omniscient knowledge by the agents of the mixed strategies of their opponents. However, neither ADL nor APQ agent explicitly play a mixed strategy. Their strategies, being pure for them are perceived as mixed by the IGA player as soon as they do not act in the same internal state space. Indeed, the current internal states of each agent (counterparts' actions history of APQ, concatenated joint actions of ADL and opponents' current mixed strategy of IGA) are different, though the current external state (the game played) is the same.

Thus, to make the IGA agent able to estimate the strategy of its opponents, we implemented the following well known techniques:

- Adaptive Play's technique,
- Exponential Moving Average (EMA)

We have described these techniques in Sections 3.5 and 3.6.

Adaptive Dynamics Learner (Burkov and Chaib-draa, 2007a;b)

The only interesting parameter of our ADL algorithm is p, the maximal history length. What is the length that will permit the ADL agent to learn well the dynamics of an opponent? Is there a universal value or should it be adjusted for each opponent individually? Our experiment showed that, in most cases, the history of the length 2 (that is, the only most recent joint action!) was sufficient to outperform the adaptive agents in adversarial games, but the value of 6 (three most recent actions) was sufficient to perform well regardless of the game played. Thus, in our experiments we set $p = 6$, but the question of how to determine the best value of p, if such exists, is still open.

5.3.2 Results

Let us now describe the results of experiments done with the use of our Adaptive Dynamics Learner approach. We tested the ADL algorithm in play versus IGA and APQ algorithms and in self-play on a set of games from GAMUT test suite (Nudelman et al., 2004). All the games we used were randomly generated by GAMUT with normalization

of the rewards to the values between 0 and 1. Figure 5.10 represents game matrices for the three adversarial games that are of the most interest.

MatchingPennies

$$R^r, R^c = \begin{bmatrix} 1,-1 & -1,1 \\ -1,1 & 1,-1 \end{bmatrix}$$

RockPaperScissors

$$R^r, R^c = \begin{bmatrix} 0,0 & 1,-1 & -1,1 \\ -1,1 & 0,0 & 1,-1 \\ 1,-1 & -1,1 & 0,0 \end{bmatrix}$$

ShapleysGame

$$R^r, R^c = \begin{bmatrix} -1,-1 & 1,-1 & -1,1 \\ -1,1 & -1,-1 & 1,-1 \\ 1,-1 & -1,1 & -1,-1 \end{bmatrix}$$

Figure 5.10: Three adversarial games from GAMUT. R^r, R^c is the payoff matrix each entry of which contains the payoffs for the row and column players respectively.

First, we examined the behavior of ADL against APQ player (Figure 5.11). To do this, we observed the evolution of average Bellman error, i.e., the difference between two successive updates of Q-values, and the changes in the average reward of ADL per play[4]. The rewards were averaged over each of the $10,000$ plays. It is easy to see that the process exhibited good progress toward the convergence, as suggested by progressive reducing of average Bellman error (Figure 5.11, top) and substantial positive average reward of ADL per time step (Figure 5.11, bottom).

Note that in adversarial games, the positive values gained by the ADL player mean the negative values gained by its opponent. Thus, it is unnecessary to provide the results of the other player.

Further, we examined ADL versus IGA player (Figure 5.12). ADL showed better performance against this opponent, as is seen from the average reward diagram where the converged values are higher than the corresponding values obtained in play vs. APQ agent. The average Bellman error decreased slower in that case (Figure 5.12, top), which is explained by the stochastic strategies used by IGA unlike the APQ player, which converged directly to a policy in pure strategies. Notice that we observed that in almost all runs the IGA agent that used the Adaptive Play's technique of the opponent's strategy estimation was more efficient than the one that used EMA. Therefore, in our figures, we only presented the results for the IGA agent using the Adaptive Play's strategy estimation technique.

As it is seen from the curves in Figure 5.11 and 5.12, the advantage that ADL

[4]Here plays and time steps are equivalent.

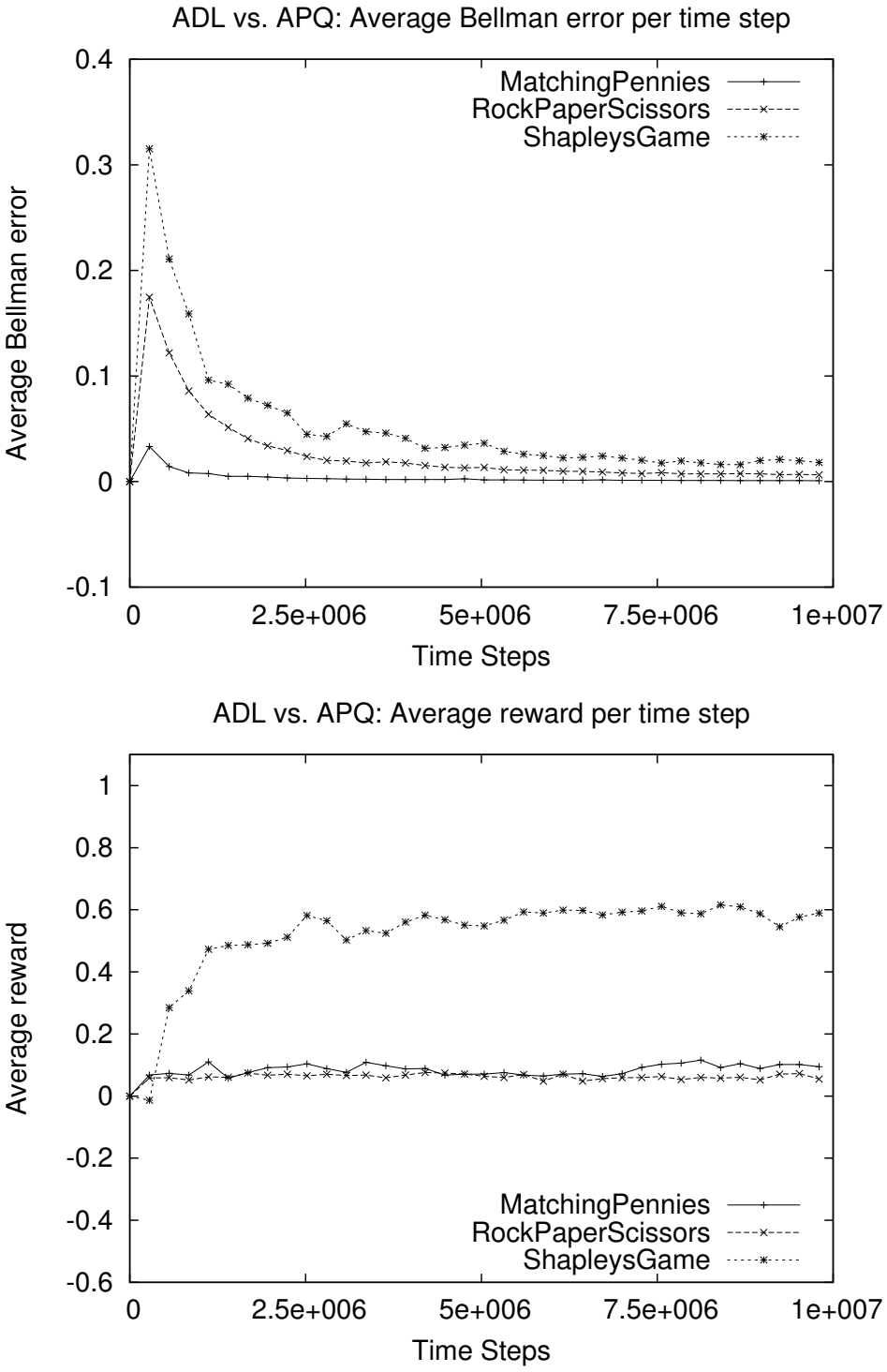


Figure 5.11: ADL vs. APQ in the adversarial games: average Bellman error and average reward of ADL per time step.

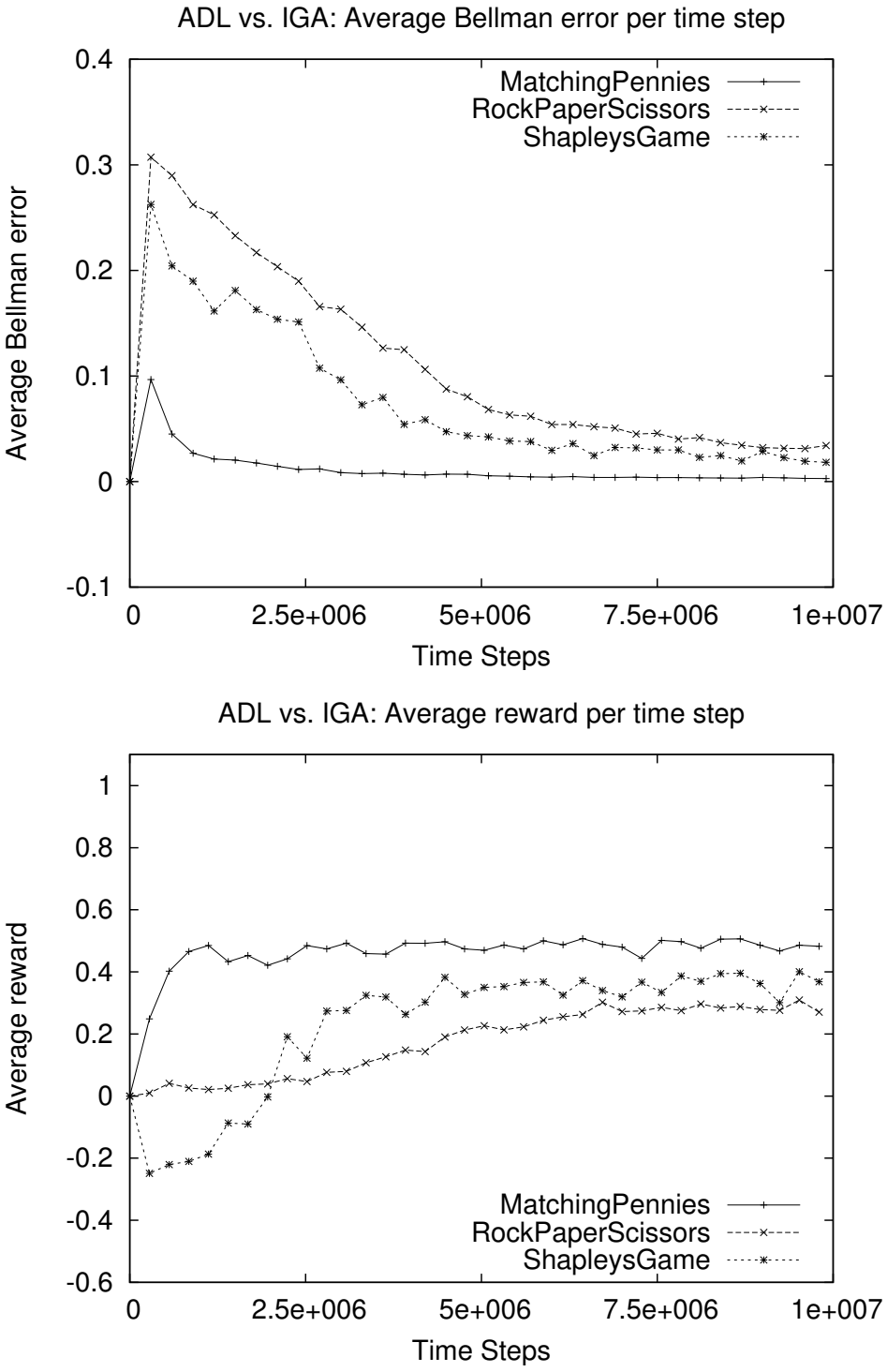


Figure 5.12: ADL vs. IGA in the adversarial games: average Bellman error and average reward of ADL per time step.

gains versus different opponents in different games is not the same. In particular, the advantage of ADL against APQ is much better in ShapleysGame than in the other games, but in play versus IGA it performs better in MatchingPennies. This particularity is still under investigations.

Finally, we verified whether ADL, by being efficient against adaptive opponents, remains rational in play against itself (so called self-play) and versus other types of opponents which do not evolve in time and follow a stationary policy, such as a mixed strategy (Figure 5.13). It was important to verify the case of stationary opponents in order to make sure that the effectiveness of our algorithm is not an effect of a particular structure of its opponents, such as adaptivity.

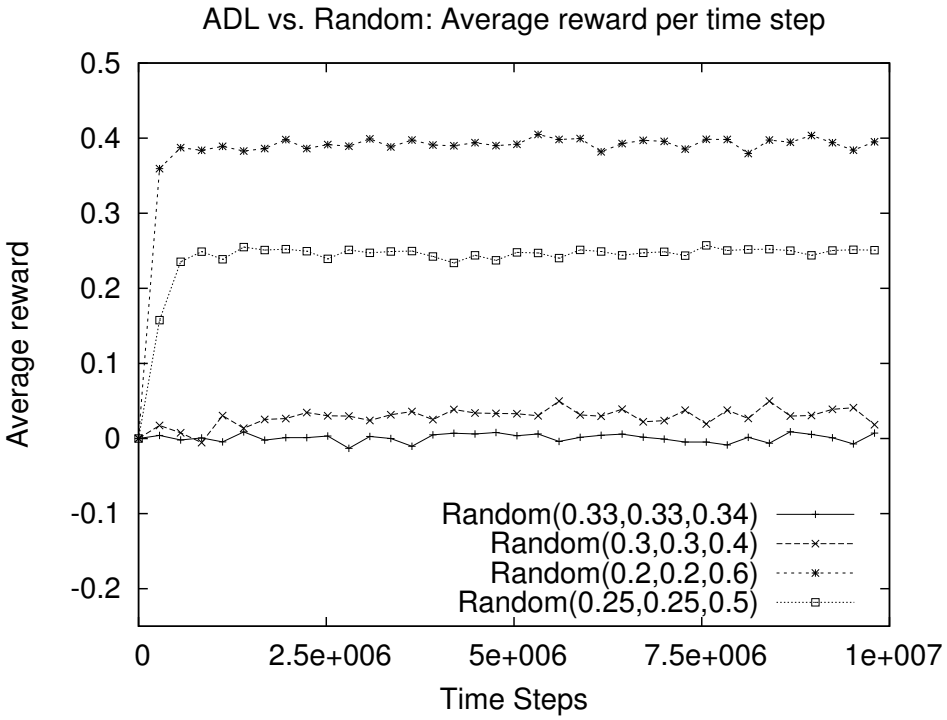


Figure 5.13: Average reward of ADL vs. stationary opponents playing a different mixed strategies in RockPaperScissors.

As for the stationary opponents, we tested the behavior of ALD against a player, which played random (mixed) strategies on the example of the RockPaperScissors game. Let $Random(x, y, z)$ denote a player playing a mixed strategy. Let $x \geq 0$, $y \geq 0$ and $z \geq 0$, such that $x + y + z = 1$, be the probabilities that the actions 1, 2 and 3 respectively will be played by that player. As expected, the ADL player had a positive average reward close to 0 against the $Random(0.33, 0.33, 0.34)$ opponent which played a strategy close to the Nash equilibrium $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, but it performed better as the random

players were more distant from the equilibrium, thus, it converged to average reward of 0.02 against $Random(0.3, 0.4, 0.4)$, to the reward of 0.25 against $Random(0.25, 0.25, 0.5)$ and to the reward of 0.4 versus $Random(0.2, 0.2, 0.6)$. Thus, an important conclusion is that ADL algorithm remained rational in that case.

As we already noticed above, we presented the dynamics of the learning process for three games only (RockPaperScissors, MatchingPennies and ShapleysGame) because, in our opinion, the adversarial case is the most interesting one. In fact, the curves of the learning dynamics in the other games are trivial: in most cases, almost from the beginning they become straight lines with some minor fluctuations. The *minimum* of the converged values of average reward of the row player over all runs for all games are presented in Figure 5.14. The bar graphs "ADL vs. IGA" and "ADL vs. APQ" show the reward of ADL, as the row player, in play versus these opponents. The "Max Nash" and "Min Nash" bar graphs reflect respectively the maximal and minimal average rewards per play, which the row player can gain if a Nash equilibrium is played.

It is important to note that in games having a Pareto optimal strategy, which is not a an equilibrium (such as PrisonersDilemma), the solution to which ADL converges in self-play is Pareto optimal (while the adaptive algorithms converge to an equilibrium that is not favorable for both players). Recall that a strategy is said to be Pareto optimal if its utility is maximal for all players. Furthermore, if there are two equilibrium outcomes, one of which is more favorable to one agent than to another one, and vice versa (such as in GrabTheDollar), then the average rewards obtained by both ADL players in self-play are a mean of these two equilibrium rewards, i.e., the welfare of both is maximized. The adaptive players are only able to find one of these equilibria, thus, as soon as an equilibrium is found, one of the agents will always have a lower utility. The dynamics of the self-play in adversarial games are not interesting enough to include in the report; as expected, the agents, by being rational, converged to a Nash equilibrium, which brings zero average reward to both players.

5.4 Conclusion

In this chapter we tested two proposed algorithms, Initialized Adaptive Play Q-learning (IAPQ) and Adaptive Dynamics Learner (ADL). The main criteria of success for the IAPQ algorithm were the scalability and the convergence speed. We have seen that it can converges much faster to a solution then the uniformly initialized Adaptive Play Q-learning agent. We also showed that it can scale better in terms of the number of system's states and in the number of learning agents.

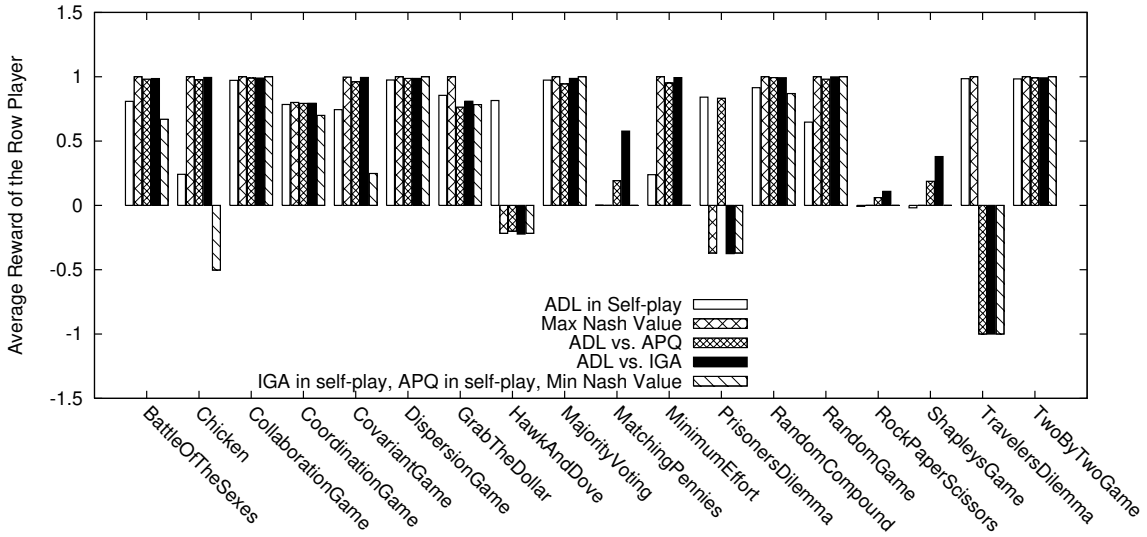


Figure 5.14: ADL, APQ and IGA players over the games from GAMUT.

As for the ADL algorithm, the principal measurement criteria were performance in terms of utility against other types of opponents in adversarial games and the utility of self-play, i.e, in the case when two or several ADL players learn simultaneously in a common environment.

As we have seen on different test examples, our approaches were very promising with respect to these criteria. In this work, however, we did not test our algorithms on the other interesting test examples, such as adversarial stochastic games (like predator-pray or robotic soccer Littman (1994)). To do that, ADL should be extended by integrating the joint state information into the Q-values, as we already mentioned in the previous chapter.

Also, we did not try to combine two our approaches into one algorithm and observe its behavior on the extended set of test benches. We keep this for future work in the field of multiagent learning.

Chapter 6

General Conclusion

Multiagent learning is a dynamically growing research domain in intelligent systems. Although much of work has been done in the last ten years, a lot of work is still remaining to do due to the complexity and non-obvious character of the problem. Indeed, in a multiagent learning environment, this latter may be stochastic, continuous and dynamic. Furthermore, there may be several learning agents, each one with its own learning algorithm, intentions, goals and rationality principles. Finally, the relations between agents may be either pure competition, either pure cooperation or it may be non-obvious how to determine these relations in advance.

From the point of view of the learning algorithm conception, there are two major challenges that any multiagent learning algorithm should overcome to be applicable in the reality. First, it should be able to learn a "good" policy in conditions of the non-stationarity (permanent and non-predictable changes of the environment), because the other agents can be learning too at the same time. Second, this algorithm would have the least possible computational complexity to be scalable to the real world domains containing thousands of states, especially when in each of these states agent may have hundreds available actions. This computational complexity problem is known as *curse of dimensionality*.

The expression "curse of dimensionality" is due to Bellman (1961) and it relates to the fact that the convergence of any estimator to the true value of a smooth function defined on a space of high dimension is very slow. In the case of the learning in MAS the Q-function is defined on a very high dimensional space. Indeed, as was shown by Koenig and Simmons (1996), a time required to learn an optimal strategy in the single-agent Q-learning is exponential in the number of the environment's states. Additionally, as we have shown in this work, in MAS, the state space of the multiagent problem is

itself exponential in the number of agents. Hence, this double exponentiality is a great challenge for the applicability of the current multiagent learning algorithms in the real life problems.

6.1 Contributions

In this work, we have shown that the initialization of multiagent Q-values using a precalculated single-agent solution permits significantly reducing the complexity (in terms of the convergence time) of the learning process. Also, we have shown that this initialization is admissible and monotonic for the problems that can be modeled as a goal-directed stochastic game with action-penalty representation. By producing a set of empirical tests on the multiagent coordination problem, we have shown that uninformed multiagent learning quickly becomes intractable. On the other hand, the informed, heuristically initialized, algorithm remains tractable with growth of the state space while being weakly sensible to that growth, due to the strict focusing on the relevant states only.

Notice that the heuristic initialization in the multiagent reinforcement learning has not been widely explored in the literature. More frequently, the researchers proposed different heuristic modifications of the Q-learning technique for the particular game structures. Three typical modifications have been considered in this context:

1. a modification of the Q-value update rule (Lauer and Riedmiller, 2000),
2. a modification of the action selection rule (Kapetanakis and Kudenko, 2002) and
3. a modification of the exploration strategy, like in (Chalkiadakis and Boutilier, 2003).

However, the emphasis of the modern research was put on the development of strategies to reach an equilibrium in the games of different reward structures (e.g., see Claus and Boutilier, 1998; Gies and Chaib-draa, 2006; Hu and Wellman, 1998). In fact, virtually all these methods suffer from a low scalability, as well as other multiagent learning techniques that restrict neither joint state space nor joint-action space of the problem during learning. To our knowledge, an explicit Q-value initialization of the type proposed in this work is a completely new approach to the complexity reduction in multiagent learning. And this approach permits reducing the computational complexity of learning in a considerably higher degree.

The second novel approach to the multiagent learning proposed in this work is called Adaptive Dynamics Learner (ADL). ADL is an algorithm of effective learning in adaptive dynamic multiagent systems. An adaptive dynamic system may be viewed as a two-player game where a goal of a player is to maximize its own long-term utility given that the other agents (considered as one whole agent) may follow an adaptive learning strategy, such as Adaptive Play (Young, 1993) or Gradient Ascent Singh et al. (1994). Our algorithm, ADL, by interacting with the opponent player, learns the Q-values of the states that are formed as an ordered set of joint actions of the past plays of limited-length history. We have empirically shown that our algorithm outperforms IGA (Singh et al., 1994) and APQ (Gies and Chaib-draa, 2006) algorithms even if a very short history length is used to form the states.

While being more general than a PHC-Exploiter by Wang and Sandholm (2002), our approach is much simpler (in terms of the amount of computations per time step) than the analogical Hyper-Q algorithm (Tesauro, 2004). Thus, it is possibly better scalable. Also, in certain games, it is able to maximize the welfare of both players in self-play. Several untested adaptive dynamic opponents, such as No-Regret (Jafari et al., 2001) and PHC (Bowling and Veloso, 2002), and certain stationary and non-stationary opponents still remain to be compared with ADL and we will give attention to that. Considerably more research is needed, however, to develop a theoretical analysis of our approach.

Also, although in this work we have shown that in order to make the ADL agent able to outperform adaptive players in adversarial context, the length of the history can have very small value, in our future work, however, we plan to study in detail the impact of the history length on the effectiveness of the learnt policy.

6.2 Future Work

We intend to extend the applicability of our complexity reduction approach to the general form stochastic games. For that, a suitable *relaxation* should be derived from the general multiagent model in such a way, that a solution of this relaxed model was (1) an admissible and monotonic approximation of the original model and (2) was easier to be calculated as compared to the solution of the original problem. Also, the advantages and limitations of practical applications of this approach to the real life problems should be investigated.

As for the adaptive dynamics learning, we studied how our approach to multiagent

learning is situated among the other recent algorithms. There have been many new algorithms proposed in the last five years, but there is still no common idea to which direction multiagent learning would progress. Three years ago, Shoham et al. (2003) published their critical survey of the modern multiagent learning trend, where the authors asked about the *question* the researchers should aim at when talking about multiagent learning. However, there is still no such a question.

We would emphasize two common directions of the modern research:

1. heuristic agent composition (Powers and Shoham, 2005b;a) and
2. direct dynamics learning and exploiting (Chang and Kaelbling, 2001; Tesauro, 2004).

The first direction consists in creating the agent, which "can" play several game theoretic techniques, such as Tit-for-Tat described in Section 2.5. Also, this agent must have a heuristic mechanism of switching between these techniques depending on the estimation of the opponent's strategy or the belief about what game-playing algorithm is used by the opponent. In our opinion, there should be a more general approach. The problem of finding this approach we consider as a major issue for further investigation.

The second direction of the modern research in the multiagent learning is one presented in this work as Adaptivity Modeling Algorithms (Section 3.6). The algorithms of this type are using the reinforcement learning as a base of their learning strategy. These algorithms are trying to "learn" (i.e., not to select from a predefined set) a best response strategy to the real *behavior* of their opponents. We believe that this direction is one of the most promising for future research in the multiagent learning.

In our future work, we will investigate a possibilities of combining the both proposed approaches, Q-initialization and Adaptive Dynamics Learning, into one algorithm. Besides, we plan to combine these approaches with the value function approximation techniques (Sutton and Barto, 1998) that appear to be very efficient in single-agent context. For example, they are able to speedup the reinforcement learning by factorizing the value or policy function.

In general, an approach to the function approximation in multiagent algorithms should be found. In our opinion, this direction along with the opponent's adaptivity learning and exploiting are two of the most promising canvas of multiagent learning for the next few years.

Bibliography

Barto, A., Bradtke, S., and Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press, New Jersey.

Bernstein, D., Givan, R., Immerman, N., and Zilberstein, S. (2003). The Complexity Of Decentralized Control Of Markov Decision Processes. *Mathematics of Operations Research*, 27(4):819–840.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33.

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

Brown, G. (1951). *Iterative Solution of Games by Fictitious Play*. Wiley, New York.

Burkov, A. and Chaib-draa, B. (2007a). Effective learning in adaptive dynamic systems. In *Proceedings of the AAAI 2007 Spring Symposium on Decision Theoretic and Game Theoretic Agents (GTDT'07)*, Stanford, California. To appear.

Burkov, A. and Chaib-draa, B. (2007b). Multiagent learning in adaptive dynamic systems. Submitted for publication to the 2007 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'07).

Burkov, A. and Chaib-draa, B. (2007c). Reducing the complexity of multiagent learning. Accepted as short paper in the 2007 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'07).

Chalkiadakis, G. and Boutilier, C. (2003). Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, Australia.

Chang, Y. and Kaelbling, L. (2001). Playing is believing: The role of beliefs in multi-agent learning. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS'01)*, Canada.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, Menlo Park, CA. AAAI Press.

Darken, C. and Moody, J. (1991). *Note On Learning Rate Schedule For Stochastic Optimisation*, volume 3, pages 832–838. Morgan Kaufmann, San Mateo, CA.

Fink, A. (1964). Equilibrium in a stochastic N-person game. *Journal of Science in Hiroshima University*, 28:89–93.

Fudenberg, D. and Tirole, J. (1991). *Game Theory*. MIT Press, Cambridge, Massachusetts.

Gies, O. and Chaib-draa, B. (2006). Apprentissage de la coordination multiagent : une méthode basée sur le Q-learning par jeu adaptatif. *Revue d'Intelligence Artificielle*, 20(2-3):385–412.

Hu, J. and Wellman, M. (2003). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069.

Hu, J. and Wellman, P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, San Francisco, CA. Morgan Kaufmann.

Jafari, A., Greenwald, A., Gondek, D., and Ercal, G. (2001). On no-regret learning, fictitious play, and Nash equilibrium. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'2001)*, San Francisco, CA. Morgan Kaufmann.

Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada.

Koenig, S. and Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22:227–250.

Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference in Machine Learning (ICML'00)*, Stanford, CA. Morgan Kaufmann.

Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, New Brunswick, NJ. Morgan Kaufmann.

Littman, M. (2001a). Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*, San Francisco, CA. Morgan Kaufman.

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

Littman, M. and Szepesvári, C. (1996). A generalized reinforcement learning model: Convergence and applications. In Saitta, L., editor, *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, Bari, Italy. Morgan Kaufmann.

Nash, J. (1950). Equilibrium points in n-person games. In *Proceedings of the National Academy of the USA*, volume 36(1).

Nudelman, E., Wortman, J., Leyton-Brown, K., and Shoham, Y. (2004). Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of Autonomous Agents and Multiagent Systems (AAMAS'04)*.

Poundstone, W. (1992). *Prisoner's Dilemma*. Doubleday, New York, NY.

Powers, R. and Shoham, Y. (2005a). Learning against opponents with bounded memory. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*.

Powers, R. and Shoham, Y. (2005b). New criteria and a new algorithm for learning in multi-agent systems. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press.

Russell, S. and Norvig, P. (2005). *Artifitial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, second edition.

Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, Seattle, Washington.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences*, 39:1095–1100.

Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: a critical survey. Technical report, Stanford University.

Singh, S., Kearns, M., and Mansour, Y. (1994). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'94)*, San Francisco, CA. Morgan Kaufman.

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

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, Amherst, MA.

Tesauro, G. (2004). Extending Q-learning to general adaptive multi-agent systems. In Thrun, S., Saul, L., and Scholkopf, B., editors, *Advances in Neural Information Processing Systems*, volume 16, Cambridge, MA. MIT Press.

Thrun, S. (1992). Efficient Exploration In Reinforcement Learning.

Uther, W. and Veloso, M. (2003). Adversarial reinforcement learning. Technical report, School of Computer Science, Carnegie Mellon University.

Wang, X. and Sandholm, T. (2002). Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'02)*, Vancouver, Canada.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Young, H. (1993). The evolution of conventions. *Econometrica*, 61(1):57–84.