An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning

Michael Bowling Manuela Veloso

October, 2000
CMU-CS-00-165

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Learning behaviors in a multiagent environment is crucial for developing and adapting multiagent systems. Reinforcement learning techniques have addressed this problem for a single agent acting in a stationary environment, which is modeled as a Markov decision process (MDP). But, multiagent environments are inherently non-stationary since the other agents are free to change their behavior as they also learn and adapt. Stochastic games, first studied in the game theory community, are a natural extension of MDPs to include multiple agents. In this paper we contribute a comprehensive presentation of the relevant techniques for solving stochastic games from both the game theory community and reinforcement learning communities. We examine the assumptions and limitations of these algorithms, and identify similarities between these algorithms, single agent reinforcement learners, and basic game theory techniques.

This research was sponsored by the United States Air Force under Cooperative Agreements No. F30602-97-2-0250 and No. F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

Keywords: Multiagent systems, stochastic games, reinforcement learning, game theory

1 Introduction

The problem of an agent *learning to act* in an unknown world is both challenging and interesting. Reinforcement learning has been successful at finding optimal control policies for a single agent operating in a stationary environment, specifically a Markov decision process.

Learning to act in multiagent systems offers additional challenges; see the following surveys [17, 19, 27]. Multiple agents can be employed to solve a single task, or an agent may be required to perform a task in a world containing other agents, either human, robotic, or software ones. In either case, from an agent's perspective the world is not stationary. In particular, the behavior of the other agents may change, as they also learn to better perform their tasks. This type of multiagent nonstationary world creates a difficult problem for learning to act in these environments.

However, this nonstationary scenario can be viewed as a game with multiple players. Game theory has aimed at providing solutions to the problem of selecting optimal actions in multi-player environments. In game theory, there is an underlying assumption that the players have similar adaptation and learning abilities. Therefore the actions of each agent affect the task achievement of the other agents. It seems therefore promising to identify and build upon the relevant results from game theory towards multiagent reinforcement learning.

Stochastic games extend the single agent Markov decision process to include multiple agents whose actions all impact the resulting rewards and next state. They can also be viewed as an extension of game theory's simpler notion of matrix games. Such a view emphasizes the difficulty of finding optimal behavior in stochastic games, since optimal behavior depends on the behavior of the other agents, and vice versa. This model then serves as a bridge combining notions from game theory and reinforcement learning.

A comprehensive examination of the multiagent learning techniques for stochastic games does not exist. In this paper we contribute such an analysis, examining techniques from both game theory and reinforcement learning. The analysis both helps to understand existing algorithms as well as being suggestive of areas for future work.

In section 2 we provide the theoretical framework for stochastic games as extensions of both MDPs and matrix games. Section 3 summarizes algorithms for solving stochastic games from the game theory and reinforcement learning communities. We discuss the assumptions, goals, and limitations of these algorithms. We also taxonomize the algorithms based on their game theoretic and reinforcement learning components. Section 4 presents two final algorithms that are based on a different game theoretic mechanism, which address a limitation of the other algorithms. Section 5 concludes with a brief summary and a discussion of the future work in multiagent reinforcement learning.

2 Theoretical Framework

In this section we setup the framework for stochastic games. We first examine MDPs, which is a single-agent, multiple state framework. We then examine matrix games, which is a multiple-agent, single state framework. Finally we introduce the stochastic game framework which can be seen as the merging of MDPs and matrix games.

2.1 Markov Decision Processes

A *Markov decision process* is a tuple, $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, T is a transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and R is a reward function $\mathcal{S} \times \mathcal{A} \to \boldsymbol{R}$. The transition function defines a probability distribution over next states as a function of the current state and the agent's action. The reward function defines the reward received when selecting an action from the given state. Solving MDPs consists of finding a policy, $\pi : \mathcal{S} \to \mathcal{A}$, which determines the agent's actions so as to maximize discounted future reward, with discount factor γ.

MDPs are the focus of much of the reinforcement learning work [11, 20]. The crucial result that forms the basis for this work is the existence of a stationary and deterministic policy that is optimal. It is such a policy that is the target for RL algorithms.

2.2 Matrix Games

A *matrix game* or *strategic game* (see [14] for an overview) is a tuple $(n, \mathcal{A}_{1\ldots n}, R_{1\ldots n})$, where n is the number of players, $\mathcal{A}_i$ is the set of actions available to player i (and $\mathcal{A}$ is the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$), and R_i is player i's payoff function $\mathcal{A} \rightarrow \boldsymbol{R}$. The players select actions from their available set with the goal to maximize their payoff which depends on all the players' actions. These are often called matrix games, since the R_i functions can be written as n-dimensional matrices.

Unlike MDPs, it is difficult to define what it means to "solve" a matrix game. A stationary strategy can only be evaluated if the other players' strategies are known. This can be illustrated in the two-player matching pennies game. Here each player may select either Heads or Tails. If the choices are the same then Player 1 takes a dollar from Player 2. If they are different then Player 1 gives a dollar to Player 2. The matrices for this game are shown in Figure 1. If Player 2 is going to play Heads, then Player 1's optimal strategy is to play Heads, but if Player 2 is going to play Tails, then Player 1 should play Tails. So there is no optimal pure strategy, independent of the opponent.

$$R_1 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \qquad R_2 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

Figure 1: The matching pennies game.

Players can also play mixed strategies, which select actions according to a probability distribution. It is clear that in the above game there is also no optimal mixed strategy that is independent of the opponent. This leads us to define an *opponent-dependent* solution, or set of solutions:

Definition 1 *For a game, define the* best-response *function for player i, $BR_i(\sigma_{-i})$, to be the set of all, possibly mixed, strategies that are optimal given the other player(s) play the possibly mixed joint strategy σ_{-i}.*

The major advancement that has driven much of the development of matrix games and game theory is the notion of a best-response equilibrium, or *Nash equilibrium*:

Definition 2 *A* Nash equilibrium *is a collection of strategies (possibly mixed) for all players, σ_i, with,*

$$\sigma_i \in BR_i(\sigma_{-i}).$$

So, no player can do better by changing strategies given that the other players continue to follow the equilibrium strategy.

What makes the notion of equilibrium compelling is that all matrix games have a Nash equilibrium, although there may be more than one.

Types of Matrix Games. Matrix games can be usefully classified according to the structure of their payoff functions. Two common classes of games are *purely collaborative* and *purely competitive* games. In purely collaborative games, all agents have the same payoff function, so an action in the best interest of one agent is in the best interest of all the agents.

In purely competitive games, there are two agents, where one's payoff function is the negative of the other (i.e. $R_1 = -R_2$). The game in Figure 1 is an example of one such game. Purely competitive games are also called *zero-sum games* since the payoff functions sum to zero. Other games, including purely collaborative games, are called *general-sum games*. One appealing feature of zero-sum games is that they contain a unique Nash equilibrium. This equilibrium can be found as the solution to a relatively simple linear program[1] Finding equilibria in general-sum games requires a more difficult quadratic programming solution [6].

[1] The value of the equilibrium can be computed using linear programming with the following objective function, $\max_{\sigma_1 \in PD(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} R_1(s, \langle a_1, a_2 \rangle) \sigma_{a_2}$.

In the algorithms presented in this paper we use the functions Value(MG) and Solve$_i$(MG) to refer to algorithms for solving matrix games, either linear programming (for zero-sum) or quadratic programming (for general-sum) depending on the context. Value returns the expected value of playing the matrix game's equilibrium and Solve$_i$ returns player i's equilibrium strategy.

2.3 Stochastic Games

A *stochastic game* is a tuple $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, T, R_{1\ldots n})$, where n is the number of agents, $\mathcal{S}$ is a set of states, $\mathcal{A}_i$ is the set of actions available to agent i (and $\mathcal{A}$ is the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$), T is a transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, and R_i is a reward function for the ith agent $\mathcal{S} \times \mathcal{A} \rightarrow \boldsymbol{R}$. This looks very similar to the MDP framework except we have multiple agents selecting actions and the next state and rewards depend on the joint action of the agents. It's also important to notice that each agent has its own separate reward function. We are interested in determining a course of action for an agent in this environment. Specifically we want to learn a stationary though possibly stochastic policy, $\rho : \mathcal{S} \rightarrow PD(\mathcal{A}_i)$, that maps states to a probability distribution over its actions. The goal is to find such a policy that maximizes the agent's discounted future reward with discount factor γ.

SGs are a very natural extension of MDPs to multiple agents. They are also an extension of matrix games to multiple states. Each state in a SG can be viewed as a matrix game with the payoffs for each joint action determined by $R_i(s, a)$. After playing the matrix game and receiving the payoffs the players are transitioned to another state (or matrix game) determined by their joint action. We can see that SGs then contain both MDPs and matrix games as subsets of the framework.

A non-trivial result, proven by [18] for zero-sum games and by [6] for general-sum games, is that there exist equilibria solutions for stochastic games just as they do for matrix games.

Types of Stochastic Games. The same classification for matrix games can be used with stochastic games. Purely collaborative games are ones where all the agents have the same reward function. Purely competitive, or zero-sum, games are two-player games where one player's reward is always the negative of the other's. Like matrix games, zero-sum stochastic games have a unique Nash equilibrium, although finding this equilibrium is not so easy. Section 3.1 presents algorithms from game theory for finding this equilibrium.

3 Solving Stochastic Games

In this section we present a number of algorithms for "solving" stochastic games. As is apparent from the matrix game examples, unlike MDPs, there is not likely to be an optimal solution to a stochastic game that is independent of the other agents. This makes solving SGs for a single agent difficult to define. The existence of equilibria does not alleviate our problem since it's only an equilibrium if *all* agents are playing the equilibrium. There also may be multiple equilibria, so which equilibria to play is dependent on the other agents.

This problem will be discussed later, but for this section we assume that the stochastic game contains a unique equilibrium. All but one of the algorithms require the even stronger assumption that the game is zero-sum. "Solving" a stochastic game, then, means finding this unique equilibrium.

The algorithms differ on what assumptions they make about the SG and the learning process. These differences are primarily between the game theory and reinforcement learning algorithms. The main differences are whether a model for the game is available and the nature of the learning process. Most game theory algorithms require a model of the environment, since they make use of the transition, T, and reward, R_i, functions. The goal of these algorithms is also to compute *the equilibrium value* of the game (i.e. the expected discounted rewards for each of the agents), rather than finding equilibrium policies. This means they often make strong requirements on the behavior of all the agents.

In contrast, reinforcement learning algorithms assume the world is not known, and only observations of the T and R functions are available as the agents act in the environment. The goal is for the agent to find its policy in the game's equilibrium solution, and usually make little requirements on the behavior of the other agents. This fact is discussed further in Section 3.3.

3.1 Solutions From Game Theory

We present here two algorithms from the game theory community. These algorithms learn a value function over states, $V(s)$. The goal is for V to converge to the optimal value function V^*, which is the expected discounted future reward if the players followed the game's Nash equilibrium.

3.1.1 Shapley

The first algorithm for finding a SG solution was given by Shapley [18] as a result of his proof of the existence of equilibria in zero-sum SGs. The algorithm is shown in Table 1. The algorithm uses a temporal differencing technique to backup values of next states into a simple matrix game, $G_s(V)$. The value function V is then updated by solving the matrix game at each state.

1. Initialize V arbitrarily.
2. Repeat,
   (a) For each state, $s \in \mathcal{S}$, compute the matrix,
   $$G_s(V) = \left[g_{a \in \mathcal{A}} : \begin{array}{c} R(s,a)+ \\ \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V(s') \end{array} \right].$$
   (b) For each state, $s \in \mathcal{S}$, update V,
   $$V(s) \leftarrow \text{Value}\left[G_s(V)\right].$$

Table 1: Algorithm: Shapley.

Notice the algorithm is nearly identical to value iteration for MDPs, with the "$\max$" operator replaced by the "Value" operator. The algorithm also shows that equilibria in stochastic games are solutions to Bellman-like equations. The algorithm's value function V converges to V^* which satisfies the following equations,

$$\forall s \in \mathcal{S} \qquad V^*(s) = \text{Value}\left[G_s(V^*)\right].$$

3.1.2 Pollatschek & Avi-Itzhak

Just as Shapley's algorithm is an extension of value iteration to stochastic games, Pollatschek & Avi-Itzhak [25] introduced an extension of policy iteration [9]. The algorithm is shown in Table 2. Each player selects the equilibrium policy according to the current value function, making use of the same temporal differencing matrix, $G_s(V)$, as in Shapley's algorithm. The value function is then updated based on the actual rewards of following these policies.

Like Shapley's algorithm, this algorithm also computes the equilibrium value function, from which can be derived the equilibrium policies. This algorithm, though, is only guaranteed to converge if the transition function, T, and discount factor, γ, satisfies a certain property. A similar algorithm by Hoffman & Karp [25] which alternates the policy learning (rather than it occurring simultaneously) avoids the convergence problem, but requires obvious control over when the other agents in the environment are learning.

1. Initialize V arbitrarily.
2. Repeat,

$$\begin{aligned} \rho_i &\leftarrow \text{Solve}_i\ [G_s(V)] \\ V(s) &\leftarrow E\left\{\sum \gamma^t r_t | s_0 = s, \rho_i\right\}. \end{aligned}$$

Table 2: Algorithm: Pollatschek & Avi-Itzhak. The function G_s is the same as presented in Table 1.

3.2 Solutions From RL

Reinforcement learning solutions take a different approach to finding policies. It is generally assumed the model of the world (T and R) are not known but must be observed through experience. The agents are required to act in the environment in order to gain observations of T and R. The second distinguishing characteristic is that these algorithms focus on the behavior of a single agent, and seek to find the equilibrium policy for that agent.

3.2.1 Minimax-Q

Littman [13] extended the traditional Q-Learning algorithm for MDPs to zero-sum stochastic games. The algorithm is shown in Table 3. The notion of a Q function is extended to maintain the value of *joint* actions, and the backup operation computes the value of states differently. It replaces the "$\max$" operator with the "Value" operator. It is interesting to note that this is basically the off-policy reinforcement learning equivalent of Shapley's "value iteration" algorithm.

1. Initialize $Q(s \in \mathcal{S}, a \in \mathcal{A})$ arbitrarily, and set α to be the learning rate.
2. Repeat,
   (a) From state s select action a_i that solves the matrix game $\left[\, Q(s,a)_{a \in \mathcal{A}} \,\right]$, with some exploration.
   (b) Observing joint-action a, reward r, and next state s',

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')),$$

   where,

$$V(s) = \text{Value}\left(\left[\, Q(s,a)_{a \in \mathcal{A}} \,\right]\right).$$

Table 3: Algorithm: Minimax-Q and Nash-Q. The difference between the algorithms is in the Value function and the Q values. Minimax-Q uses the linear programming solution for zeros-sum games and Nash-Q uses the quadratic programming solution for general-sum games. Also, the Q values in Nash-Q are actually a vector of expected rewards, one entry for each player.

This algorithm does in fact converge to the stochastic game's equilibrium solution, assuming the other agent executes all of its actions infinitely often. This is true even if the other agent does not converge to the equilibrium, and so provides an *opponent-independent* method for learning an equilibrium solution.

Matrix Game Solver	+	Temporal Differencing	=	Stochastic Game Solver

MG	+	TD	=	Game Theory	RL
LP		TD(0)		Shapley	MiniMax-Q
LP		TD(1)		Pollatschek & Avi-Itzhak	–
LP		TD(λ)		Van der Wal[25]	–
Nash		TD(0)		–	Nash-Q
FP		TD(0)		Fictitious Play	Opponent-Modeling / JALs

LP: linear programming FP: ficitious play

Table 4: Summary of algorithms to solve stochastic games. Each contains a matrix game solving component and a temporal differencing component. "Game Theory" algorithms assume the transition and reward function is known. "RL" algorithms only receive observations of the transition and reward function.

3.2.2 Nash-Q

Hu & Wellman [10] extended the Minimax-Q algorithm to general-sum games. The algorithm is structurally identical and is also shown in Table 3. The extension requires that each agent maintain Q values for all the other agents. Also, the linear programming solution used to find the equilibrium of zero-sum games is replaced with the quadratic programming solution for finding an equilibrium in general-sum games.

This algorithm is the first to address the complex problem of general-sum games. But their algorithm requires a number of very limiting assumptions. The most restrictive of which limits the structure of all the intermediate matrix games faced while learning (i.e. $Q(s, a)$.) The largest difficulty is that it is impossible to predict whether this assumption will remain satisfied while learning. [3]

The other assumption to note is that the game must have a unique equilibrium, which is not always true of general-sum stochastic games. This is necessary since the algorithm strives for the opponent-independence property of Minimax-Q, which allows the algorithm to converge almost regardless of the other agent's actions. With multiple equilibria it's important for all the agents to play the same equilibrium in order for it to have its reinforcing properties. So, learning independently is not possible.

3.3 Observations

A number of observations can be drawn from these algorithms. The first is the structure of the algorithms. They all have a matrix game solving component and a temporal differencing component. A summary of the components making up these algorithms can be seen in Table 4.

The first thing to note is the lack of an RL algorithm with a multiple step backup (i.e. policy iteration or TD(λ).) One complication for multi-step backup techniques is that they are *on-policy*, that is they require that the agent be executing the policy it is learning. In a multiagent environment this would require that the other agents also be learning *on-policy*. Despite this limitation it is still very suggestive of new algorithms that may be worth investigating. Also, there has been work to make an off-policy TD(λ) [15, 26] that may be useful in overcoming this limitation.

A second observation is that proving convergence to an equilibrium solution is not a trivial thing. For convergence almost all the algorithms assume a zero-sum game, and the Pollatschek & Avi-Itzhak algorithm requires stringent assumptions on the game's transition function. The only other algorithm, Nash-Q, although doesn't assume zero-sum, still requires that the game have a unique equilibrium along with additional assumptions about the payoffs.

The final observation is that all of these algorithms use a closed-form solution for matrix games (i.e. linear programming or quadratic programming.) Although this provides what appears to be opponent-independent algorithms for finding equilibria, it automatically rules out stochastic games with multiple equilibria. If a game has multiple equilibria, the "optimal" policy must depend on the policies of the other agents. Values of states therefore depend on the other agents policies, and static opponent-independent algorithms simply will not work. In the next section we examine algorithms that do not use a static matrix game solver.

4 Other Solutions

We now examine algorithms that use some form of *learning* as the matrix game solving component. There are two reasons that one would want to consider a matrix game learner rather than a closed form solution. The first was mentioned in the previous section that selecting between multiple equilibria requires observing and adapting to the other players' behavior. It is important to point out that observing other players' behavior cannot be a step that follows a static learning algorithm. Equilibrium selection affects both the agent's policy at a state and that state's value. The value of a state in turn affects the equilibria of states that transition into it. In fact the number of equilibrium solutions in stochastic games can grow exponentially with the number of states, and so equilibrium selection after learning is not feasible.

The second reason is that using an opponent-dependent matrix game solver, allows us to examine the problem of behaving when the other agents have limitations, physical or rational. These limitations might prevent the agent from playing certain equilibria strategies, which the static algorithms would not be able to exploit. Handling these situations are important since solving large problems often requires generalization and approximation techniques that impose learning limitations [1, 4, 8, 23].

Here we present two similar algorithms, one from game theory and the other from reinforcement learning, that depend on the other players' behavior. Both of these algorithms are only capable of playing deterministic policies, and therefore can only converge to pure strategy equilibria. Despite this fact, they still have interesting properties for zero-sum games that have a mixed policy equilibrium.

4.1 Fictitious Play

Fictitious play [16, 25] assumes opponents play stationary strategies. The basic game theory algorithm is shown in Table 5. The algorithm maintains information about the average value of each action (i.e. $U_i(s,a)/C_i(s,a)$ is the average expected discounted reward from past experience.). The algorithm then deterministically selects the action that has done the best in the past. This is nearly identical to single agent value iteration with a uniform weighting of past experience. Like value iteration the algorithm is deterministic and cannot play a mixed strategy. Despite this limitation, when all players are playing fictitious play, they will converge to a Nash equilibrium in games that are iterated dominance solvable [7] or fully collaborative [5]. In addition, in zero-sum games the players' empirical distribution of actions selected converges to the game's equilibrium mixed strategy [16, 25].

The algorithm has a number of advantages. It is capable of finding equilibria in both zero-sum games and some classes of general-sum games. It also finds an equilibrium without keeping track of the values or rewards of the other players. It suffers from the problem that for zero-sum games, it can only *find* an equilibrium strategy, without actually *playing* according to that strategy. There is an approximation to fictitious play for matrix games called smooth fictitious play [7], which is capable of playing mixed equilibrium. It would interesting to apply it to stochastic games.

4.2 Opponent Modeling

Opponent modeling [22] or joint action learners (JALs) [5] are RL algorithms that are similar to fictitious play. The algorithm is shown in Table 6. Explicit models of the opponents are learned as stationary distributions over their actions (i.e. $C(s,a_{-i})/n(s)$ is the probability the other players will select joint action a_{-i} based on past experience.)

1. Initialize V arbitrarily, $U_i(s \in \mathcal{S}, a \in \mathcal{A}_i) \leftarrow 0$, and $C_i(s \in \mathcal{S}, a \in \mathcal{A}_i) \leftarrow 0$.
2. Repeat: for every state s, let joint action $a = (a_1, a_2)$, such that $a_i = \text{argmax}_{a_i \in \mathcal{A}_i} \frac{U^i(s,a_i)}{C_i(s,a_i)}$. Then,

$$\begin{aligned} C_i(s, a_i) &\leftarrow C_i(s, a_i) + 1 \\ U_i(s, a_i) &\leftarrow U_i(s, a_i) + R_i(s, a) + \gamma \left(\sum_{s' \in \mathcal{S}} T(s, a, s') V(s') \right) \\ V(s) &\leftarrow \max_{a_1 \in \mathcal{A}_1} \frac{U_1(s, a_1)}{C_1(s, a_1)} \end{aligned}$$

Table 5: Algorithm: Fictitious play for two-player, zero-sum stochastic games using a model.

These distributions combined with learned joint-action values from standard temporal differencing are used to select an action. Uther & Veloso [22] investigated this algorithm in the context of a fully competitive domain, and Claus & Boutilier [5] examined it for fully collaborative domains.

The algorithm has very similar behavior to fictitious play. It does require observations of the opponents' actions, but not of their individual rewards. Like fictitious play, it's empirical distribution of play may converge to an equilibrium solution, but its action selection is deterministic and cannot play a mixed strategy.

5 Conclusion

This paper examined a number of different algorithms for solving stochastic games from both the game theory and reinforcement learning community. The algorithms have differences in their assumptions, such as whether a model is available, or what behavior or control is required of the other agents. But the algorithms also have strong similarities in the division of their matrix game and temporal differencing components.

This paper also points out a number of areas for future work. There currently is no multi-step backup algorithm for stochastic games. There is also no algorithm to find solutions to general-sum games with possibly many equilibria. Variants of fictitious play that learn policies based on the behavior of the other agents may be extended to these domains. Another related issue is algorithms that can handle less information, for example, not requiring observation or knowledge of the other agents' rewards, or not requiring observation of their actions.

Another important area for future work is examining how to learn when the other agents may have some apparent physical or rational limitation (e.g. [24]). These limitations may come vfrom an incorrect belief about the actions available to other agents, or their reward function. More than likely, limitations on learning will be imposed to bias the agent for faster learning in large problems. These limitations often take the form of using a function approximator for the values or policy. Learning to cope with limited teammates or exploiting limited opponents is necessary to applying multiagent reinforcement learning to large problems.

References

[1] Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.

1. Initialize Q arbitrarily, and $\forall s \in \mathcal{S} \quad C(s) \leftarrow 0$ and $n(s) \leftarrow 0$.
2. Repeat,
   (a) From state s select action a_i that maximizes,
   $$\sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle)$$
   (b) Observing other agents' actions a_{-i}, reward r, and next state s',
   $$\begin{aligned} Q(s,a) &\leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')) \\ C(s,a_{-i}) &\leftarrow C(s,a_{-i}) + 1 \\ n(s) &\leftarrow n(s) + 1 \end{aligned}$$
   where,
   $$\begin{aligned} a &= (a_i, a_{-i}) \\ V(s) &= max_{a_i} \sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle). \end{aligned}$$

Table 6: Algorithm: Opponent Modeling Q-Learning.

[2] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on the Theoretical Aspects of Rationality and Knowledge*, pages 195–210, Amsterdam, Netherlands, 1996.

[3] Michael Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 89–94, Stanford University, June 2000. Morgan Kaufman.

[4] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*. The MIT PRESS, 1995.

[5] Caroline Claus and Craig Boutilier. The dyanmics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Menlo Park, CA, 1998. AAAI Press.

[6] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer Verlag, New York, 1997.

[7] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. The MIT Press, 1999.

[8] Geoff Gordon. *Approximate Solutions to Markov Decision Problems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1999.

[9] Ronald A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, 1960.

[10] Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, San Francisco, 1998. Morgan Kaufman.

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

[12] Harold W. Kuhn, editor. *Classics in Game Theory*. Princeton University Press, 1997.

[13] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufman, 1994.

[14] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[15] J. Peng and R. J. Williams. Incremental multi-step q-learning. *Machine Learning*, 22:283–290, 1996.

[16] Julia Robinson. An iterative method of solving a game. *Annals of Mathematics*, 54:296–301, 1951. Reprinted in [12].

[17] Sandip Sen, editor. *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*. AAAI Press, Menlo Park,CA, March 1996. AAAI Technical Report SS-96-01.

[18] L. S. Shapley. Stochastic games. *PNAS*, 39:1095–1100, 1953. Reprinted in [12].

[19] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 2000. In press.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[21] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, Amherst, MA, 1993.

[22] William Uther and Manuela Veloso. Adversarial reinforcement learning. Technical report, Carnegie Mellon University, 1997. Unpublished.

[23] William T. B. Uther and Manuela M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Menlo Park, CA, 1998. AAAI Press.

[24] Jose M. Vidal and Edmund H. Durfee. The moving target function problem in multi-agent learning. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 317–324, 1998.

[25] O. J. Vrieze. *Stochastic Games with Finite State and Action Spaces*. Number 33. CWI Tracts, 1987.

[26] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[27] Gerhard Weiß and Sandip Sen, editors. *Adaptation and Learning in Multiagent Systems*. Springer Verlag, Berlin, 1996.