R-max – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning

Ronen I. Brafman
Computer Science Department
Ben-Gurion University
Beer-Sheva, 84105 Israel

Moshe Tennenholtz
Faculty of Industrial Engineering and Management
Technion – Israel Institute of Technology
Haifa, 32000 Israel

Abstract

R-max is an extremely simple model-based reinforcement learning algorithm which can attain near-optimal average reward in polynomial time. In R-max, the agent always maintains a complete, but possibly inaccurate model of its environment and acts based on the optimal policy derived from this model. The model is initialized in an optimistic fashion: all actions in all states return the maximal possible reward (hence the name). During execution, it is updated based on the agent's observations. R-max improves upon several previous algorithms: (1) It is simpler and more general than Kearns and Singh's E^3 algorithm, covering many classes of stochastic games. (2) It has a built-in mechanism for resolving the exploration vs. exploitation dilemma. (3) It formally justifies the "optimism under uncertainty" bias used in many RL algorithms. (4) It is simpler, more general, and more efficient than Brafman and Tennenholtz's LSG algorithm for learning in single controller stochastic games. (5) It generalizes the algorithm by Monderer and Tennenholtz for learning in repeated games. (6) It is the most efficient algorithm for learning in repeated games, to date, considerably improving and simplifying previous algorithms by Banos and by Megiddo.

1 Introduction

Reinforcement learning has attracted the attention of researchers in AI and related fields for quite some time. Many reinforcement learning algorithms exist and for some of them convergence rates are known. However, Kearns and Singh's E^3 algorithm [Kearns and Singh, 1998] was the first provably optimal polynomial time algorithm for learning in Markov decision processes (MDPs). E^3 was extended later to handle single controller stochastic games (SCSGs) [Brafman and Tennenholtz, 2000] as well as structured MDPs [Kearns and Koller, 1999]. In E^3 the agent learns by updating a model of its environment using statistics it collects. This learning process continues as long as it can be done relatively efficiently. Once this is no longer the case, the agent uses its learned model to compute an optimal policy and follows it. The success of this approach rests on two important properties: the agent can determine online whether an efficient learning policy exists, and if such a policy does not exist, it is guaranteed that the optimal policy with respect to the learned model will be approximately optimal with respect to the real world.

The difficulty in generalizing E^3 to adverserial contexts, i.e., to different classes of games, stems from the adversary's ability to influence the probability of reaching different states. In a game, the agent does not control its adversary's choices, nor can it predict them with any accuracy. Therefore, it has difficulty predicting the outcome of its actions and whether or not they will lead to new information. Consequently, it is unlikely that an agent can explicitly choose between an exploration and an exploitation policy. For this reason, the only extension of E^3 to adverserial contexts used the restricted SCSG model in which the adversary influences the reward of a game only, and not its dynamics.

To overcome this problem, we suggest a different approach in which the agent never attempts to learn explicitly. Our agent always attempts to optimize its behavior, albeit with respect to a fictitious model in which optimal behavior often leads to learning. This model assumes that the reward the agent obtains in any situation it is not too familiar with, is the maximal possible reward – R_{max}. The optimal policy with respect to the agent's *fictitious* model has a very interesting and useful property with respect to the *real* model: it is always either optimal or it leads to efficient learning. The agent does *not* know whether it is optimizing or learning efficiently, but it always does one or the other. Thus, the agent will always either exploit or explore efficiently, without knowing ahead of time which of the two will occur. Since there is only a polynomial number of parameters to learn, as long as learning is done efficiently we can ensure that the agent spends a polynomial number of steps exploring, and the rest of the time will be spent exploiting.

Thus, the resulting algorithm may be said to use an *implicit* explore or exploit approach, as opposed to Kearns and Singh's *explicit* explore or exploit approach.

This learning algorithm, which we call *R-max*, is very simple to understand and to implement. The algorithm converges in polynomial-time to a near-optimal solution. Moreover, R-max is described in the context of a model that is more general than Markov Decision Processes. As a consequence, R-max is more general and more efficient than a number of previous results. It generalizes the results of Kearns and Singh [Kearns and Singh, 1998] to adverserial contexts and to situations where the agent considers a stochastic model of the environment inappropriate, opting for a non-deterministic model instead. R-max can handle more classes of stochastic games than the LSG algorithm [Brafman and Tennenholtz, 2000]. In addition, it attains a higher expected average reward than LSG, and more quickly. R-max also improves upon previous algorithms for learning in repeated games [Aumann and Maschler, 1995], such as Megiddo's [Megiddo, 1980] and Banos [Banos, 1968]. It is the only polynomial time algorithm for this class of games that we know of, and it is much simpler, too. Finally, R-max generalizes the results of Monderer and Tennenholtz [Monderer and Tennenholtz, 1997] to handle the general probabilistic maximin (safety level) decision criterion.

The approach taken by R-max is not new. It has been referred to as *the optimism in the face of uncertainty* heuristic, and was considered an ad-hoc, though useful, approach (e.g., see Section 2.2.1 in [Kaelbling *et al.*, 1996], where it appears under the heading "Ad-Hoc Techniques" and Section 2.7 in [Sutton and Barto, 1998] where this approach is called *optimistic initial values* and is referred to as a "simple trick that can be quite effective on stationary problems"). This optimistic bias has been used in a number of well-known reinforcement learning algorithms, e.g. Kaelbling's interval exploration method [Kaelbling, 1993], the exploration bonus in Dyna [Sutton, 1990], the curiosity-driven exploration of [Schmidhuber, 1991], and the exploration mechanism in prioritized sweeping [Moore and Atkenson, 1993]. More recently, Tadepalli and Ok [Tadepalli and Ok, 1998] presented a reinforcement learning algorithm that works in the context of the undiscounted average-reward model used in this paper. In particular, one variant of their algorithm, called AH-learning, is very similar to R-max. However, as we noted above, none of this work provides theoretical justification for this very natural bias. Thus, an additional contribution of this paper is a formal justification for the *optimism under uncertainty* bias.

The paper is organized as follows: in Section 2 we define the learning problem more precisely and the relevant parameters. In Section 3 we describe the R-max algorithm. In Section 4 we prove that it yields near-optimal reward in polynomial time. We conclude in Section 5.

2 Preliminaries

We present R-max in the context of a model that is called a *stochastic game*. This model is more general than a Markov decision process because it does not necessarily assume that the environment acts stochastically (although it can). In what follows we define the basic model, describe the set of assumptions under which our algorithm operates, and define the parameters influencing its running time.

2.1 Stochastic Games

A game is a model of multi-agent interaction. In a game, we have a set of players, each of whom chooses some action to perform from a given set of actions. As a result of the players' combined choices, some outcome is obtained which is described numerically in the form of a payoff vector, i.e., a vector of values, one for each of the players. We concentrate on two-player, fixed-sum games (i.e., games in which the sum of values in the payoff vector is constant). We refer to the player under our control as *the agent*, whereas the other player will be called *the adversary.*

A common description of a game is as a matrix. This is called a game in *strategic form.* The rows of the matrix correspond to the agent's actions and the columns correspond to the adversary's actions. The entry in row i and column j in the game matrix contains the rewards obtained by the agent and the adversary if the agent plays his i^{th} action and the adversary plays his j^{th} action. We make the simplifying assumption that the size of the action set of both the agent and the adversary is identical. However, an extension to sets of different sizes is trivial.

In a *stochastic game* (SG) the players play a (possibly infinite) sequence of standard games from some given set of games. After playing each game, the players receive the appropriate payoff, as dictated by that game's matrix, and move to a new game. The identity of this new game depends, stochastically, on the previous game and on the players' actions in that previous game. Formally:

Definition 1 *A* fixed-sum, two player, stochastic-game [SG] *M on states $S = \{1, \ldots, N\}$, and actions $A = \{a_1, \ldots, a_k\}$, consists of:*

- **Stage Games:** *each state $s \in S$ is associated with a two-player, fixed-sum game in strategic form, where the action set of each player is A.*
- **Probabilistic Transition Function:** *$P_M(s, t, a, a')$ is the probability of a transition from s to t given that the first player (the* agent*) plays a and the second player (the* adversary*) plays a'.*

An SG is similar to an MDP. In both models, actions lead to transitions between states of the world. The main difference is that in an MDP the transition depends on the action of a single agent whereas in an SG the transition depends on a joint-action of the agent and the adversary. In addition, in an SG, the reward obtained by the agent for performing an action depends on its action *and* the action of the adversary. To model this, we associate a game with every state. Therefore, we shall use the terms *state* and *game* interchangeably.

Stochastic games are useful not only in multi-agent contexts. They can be used instead of MDPs when we do not wish to model the environment (or certain aspects of it) stochastically. In that case, we can view the environment as an agent that can choose among different alternatives, without assuming that its choice is based on some probability distribution. In addition, the adversaries that the agent meets in each of the stage-games could be different entities.

R-max is formulated as an algorithm for learning in Stochastic Games. However, it is immediately applicable to fixed-sum repeated games and to MDPs because both these models are degenerate forms of SGs. A repeated game is an SG with a single state and an MDP is a stochastic game in which the adversary has a single action at each state.

For ease of exposition we normalize both players' payoffs in each stage game to be non-negative reals between 0 and some constant R_{max}. We also take the number of actions to be constant. The set of possible histories of length t is $(S \times A^2)^t$, and the set of possible histories, H, is the union of the sets of possible histories for all $t \geq 0$, where the set of possible histories of length 0 is S.

Given an SG, a policy for the agent is a mapping from H to the set of possible probability distributions over A. Hence, a policy determines the probability of choosing each particular action for each possible history.

We define the *value* of a policy using the *average expected reward criterion* as follows: Given an SG M and a natural number T, we denote the expected T-step undiscounted average reward of a policy π when the adversary follows a policy ρ, and where both π and ρ are executed starting from a state $s \in S$, by $U_M(s, \pi, \rho, T)$ (we omit subscripts denoting the SG when this causes no confusion). Let $U_M(s, \pi, T) = \min_{\rho \textit{ is a policy}} U_M(s, \pi, \rho, T)$ denote the value that a policy π can guarantee in T steps starting from s. If $U_M(s, \pi, T)$ is independent of s we denote it by $U_M(\pi, T)$. We define $U_M(s, \pi) = \liminf_{T\to\infty} U_M(s, \pi, T)$, and $U_M(\pi) = \liminf_{T\to\infty} U_M(\pi, T)$ when the value is independent of s.

2.2 Assumptions

We now wish to make clear the assumptions upon which R-max is based. The first assumption is that the agent always recognizes the stage-game it reached, and that after playing a game, it knows what actions were taken by its adversary and what payoffs were obtained.

The second assumption has to do with the agent's ability to reach different states. Consider the following situation: we start learning at some state s in which the optimal action is a. However, if we do not execute the action a in s, we are likely to reach some sink state s', i.e., a state from which all moves lead back to itself. Moreover, suppose that s' is a very bad state, i.e., the agent can attain only low-valued reward in s'. A learning algorithm without any prior knowledge cannot be expected to immediately guess that a should be done in s. In fact, there is no way it can learn that a is a good action without trying out some other actions and comparing them. Thus, in order to define a situation in which we can meaningfully talk about learning a near-optimal policy the agent must be able to visit state s sufficiently many times to learn about its options. Intuitively, what we would like to require is that from each state it is not too hard to get to any other state. If that is the case, then the value of different states, (i.e., $\max_\pi U_M(s, \pi)$), cannot differ too much – if the value of s is larger than the value of s' we can travel from s' to s and the losses we incur by traveling to s will be negligible compared to the long-term benefits of acting optimally from s. Therefore, our second assumption will be that $U_M(s, \pi)$ be independent of s. For instance, this holds under the conditions imposed by [Kearns and Singh, 1998] and [Brafman and Tennenholtz, 2000]. As we just noted, this will be the case when the agent can move relatively easily from one state to another.

Our third assumption is that the maximal possible reward R_{max} is known ahead of time. This assumption can be removed if we limit ourselves to integer-valued rewards within each game.[1]

2.3 Complexity parameters

An important contribution of Kearns and Singh [Kearns and Singh, 1998] is the identification of a central parameter upon which the analysis of algorithms for learning in MDPs is based – the *mixing time*. They argue that it is unreasonable to refer to the efficiency of learning algorithms without referring to the efficiency of convergence to a desired value. They defined the *ϵ-return mixing time* of a policy π to be the smallest value of T after which π *guarantees* an expected payoff of at least $U(\pi) - \epsilon$. Because we assume that $U(\pi)$ is well defined, we can use a similar definition, however, we have to pay attention to the fact that a policy's mixing time could depend on the behavior of the adversary. Thus, in the context of SGs we say that a policy π belongs to the set $\Pi(\epsilon, T)$ of policies whose ϵ-return mixing time is at most T, if after time T, π returns an expected (average, undiscounted) payoff of at least $U(\pi) - \epsilon$ for *every* possible adversary behavior. That is, no matter what the adversary does, on the average, we have to apply policy π no more than

[1] In that case, the algorithm is run repeatedly for increasing values of R_{max}.

T steps before our average accumulated reward is sufficiently close to the value of π. Notice that this means that an agent with perfect information about the nature of the games and the transition function will require at least T steps, in order to obtain an optimal value using an optimal policy π whose ϵ-return mixing time is T. Clearly, one cannot expect an agent lacking this information to perform better.

We denote by $Opt(\Pi(\epsilon, T))$ the optimal expected undiscounted average return from among the policies in $\Pi(\epsilon, T)$. When looking for an optimal policy (with respect to policies that mix at time T, for a given $\epsilon > 0$), we will be interested in approaching this value in time polynomial in T, the desired error bounds, and the description of the game.

3 The R-max algorithm

Recall that we consider a stochastic game M consisting of a set $S = \{G_1, \ldots, G_N\}$ of stage-games in each of which both the agent and the adversary have a set $A = \{a_1, \ldots, a_k\}$ of possible actions. We associate a reward matrix R^i with each game such that $R^i_{m,l}$ is a pair consisting of the reward obtained by the agent and the adversary after playing actions a_m and a_l in game G_i, respectively. In addition, we have a probabilistic transition function, P_M, such that $P_M(s, t, a, a')$ is the probability of making a transition from G_s to G_t given that the agent played a and the adversary played a'. It is convenient to think of $P_M(i, \cdot, a, a')$ as a function associated with the entry (a, a') in the stage-game G_i. This way, all model parameters, both rewards and transitions, are associated with joint actions of a particular game. Let $\epsilon > 0$. For ease of exposition, we assume throughout most of the analysis that the ϵ-return mixing time of the optimal policy, T, is known. Later, we show how this assumption can be relaxed.

The R-max algorithm is defined as follows:

Initialize: Construct the following model M' consisting of $N+1$ stage-games, $\{G_0, G_1, \ldots, G_N\}$, and k actions, $\{a_1, \ldots, a_k\}$. Here, $G_1, \ldots, G_N$ correspond to the real games, $\{a_1, \ldots, a_k\}$ correspond to the real actions, and G_0 is an additional fictitious game. Initialize all game matrices to have $(R_{max}, 0)$ in all entries.[2] Initialize $P_M(G_i, G_0, a, a') = 1$ for all $i = 0, \ldots, N$ and for all actions a, a'.

In addition, maintain the following information for each entry in each game $G_1, \ldots, G_N$: (1) a boolean value *known/unknown*, initialized to *unknown*; (2) the states reached by playing the joint action corresponding to this entry (and how many times); (3) the reward obtained (by both players) when playing the joint action corresponding to this entry. Items 2 and 3 are initially empty.

[2]The value 0 given to the adversary does not play an important role here.

Repeat the following steps:

Compute and Act: Compute an optimal T-step policy, and execute it until a new entry becomes known.

Observe and update: Following each joint action do as follows: Let a be the action you performed in G_i and let a' be the adversary's action.

- If a, a' are performed for the first time in G_i, update the reward associated with (a, a') in G_i, as observed.
- Update the set of states reached by playing (a, a') in G_i.
- If at this point your record of states reached for this entry contains $K_1 = \max((\frac{8NTR_{max}(1+16T)}{\epsilon})^3, -6ln^3(\frac{\delta}{6Nk^2})) + 1$ elements, mark this entry as *known*, and update the transition probabilities for this entry according to the observed frequencies.

As can be seen, R-max is quite simple. It starts with an initial estimate for the model parameters that assumes all states and all joint actions yield maximal reward and lead with probability 1 to the fictitious stage-game G_0. Based on the current model, the optimal policy is computed and followed. Following each joint action the agent arrives at a new stage-game, and this transition is recorded in the appropriate place. Once we have enough information about where some joint action leads to from some stage-game, we update the entries associated with this stage-game and this joint action in our model. After each model update, we recompute the policy and repeat the above steps.

4 Optimality and Convergence

In this section we provide the tools that ultimately lead to the proof of the following theorem:

Theorem 1 *Let M be an SG with N states and k actions. Let $0 < \delta < 1$, and $\epsilon > 0$ be constants. Denote the policies for M whose ϵ-return mixing time is T by $\Pi_M(\epsilon, T)$, and denote the optimal expected return achievable by such policies by $Opt(\Pi_M(\epsilon, T))$. Then, with probability of no less than $1 - \delta$ the R-max algorithm will attain an expected return of $Opt_M(\Pi(\epsilon, T)) - 2\epsilon$ within a number of steps polynomial in $N, k, T, \frac{1}{\epsilon}$, and $\frac{1}{\delta}$.*

In the main lemma required for proving this theorem we show the following: if the agent follows a policy that is optimal with respect to the model it maintains for T steps, it will either attain near-optimal average reward, as desired, or it will update its statistics for one of the unknown slots with sufficiently high probability. This can be called the *implicit* explore or exploit property of R-max: The agent does not know ahead of time whether it is exploring or exploiting – this depends in large part on the adversary's behavior which it cannot control or predict. However, it knows that it does one or the other, no matter what the adversary does. Using this result we can proceed as follows: As we will show, the number of samples required to mark a slot as known is polynomial in the problem parameters, and so is the total number of entries. Therefore, the number of T-step iterations in which non-optimal reward is obtained is bounded by some polynomial function of the input parameters, say T'. This implies that by performing T-step iterations $D = T'R_{max}/\gamma$ times, we get that the loss obtained by non-optimal execution (where exploration is performed) is bounded by γ, for any $0 < \gamma < 1$.

Before proving our main lemma we state an extension of Kearns and Singh's Simulation Lemma [Kearns and Singh, 1998] to the context of SGs that appears in [Brafman and Tennenholtz, 2000] as well.[3]

Definition 2 *Let M and $\bar{M}$ be SGs over the same state and action spaces. We say that $\bar{M}$ is an α-approximation of M if for every state s we have:*

1. *If $P_M(s,t,a,a')$ and $P_{\bar{M}}(s,t,a,a')$ are the probabilities of transition from s to t given that the joint action carried out by the agent and the adversary is (a,a'), in M and $\bar{M}$ respectively, then, $P_M(s,t,a,a') - \alpha \leq P_{\bar{M}}(s,t,a,a') \leq P_M(s,t,a) + \alpha$*
2. *For every state s, the same stage-game is associated with s in $\bar{M}$ and in M.*

Lemma 1 *Let M and $\bar{M}$ be SGs over N states, where $\bar{M}$ is an $\frac{\epsilon}{4NTR_{max}(1+16T)}$-approximation of M, then for every state s, agent policy π, and adversary policy ρ, we have that $U_M(s,\pi,\rho,T) + \epsilon \geq U_{\bar{M}}(s,\pi,\rho,T) \geq U_M(s,\pi,\rho,T) - \epsilon$.*

Next, we define the notion of an *induced SG*. The definition is similar to the definition of an induced MDP given in [Kearns and Singh, 1998] except for the use of R-max. The induced SG is the model used by the agent to determine its policy.

Definition 3 *Let M be an SG. Let L be the set of entries (s,a,a') marked* unknown*. That is, if $(s,a,a') \in L$ then the entry corresponding to the joint action (a,a') in the stage-game s is marked as* unknown*. Define M_L to be the following SG: M_L is identical to M, except that M_L contains an additional state l_0. Transitions and rewards in all states of M_L which are not in L are identical to those in M. For any entry in L or in l_0, the transitions are with probability 1 to l_0, and the reward is R_{max} for the agent and 0 for the adversary.*

Given an SG M with a set L of known states, R^{M_L}-max denotes the optimal policy for the induced SG M_L. When M_L is clear from the context we will simply use the term *R-max policy* instead of R^{M_L}-max policy.

We now prove the implicit explore or exploit lemma:

Lemma 2 *Let M be an SG, let L and M_L be as above. Let ρ be an arbitrary policy for the adversary and let $0 < \alpha < 1$. Then either (1) $|Opt(\Pi_M(\epsilon,T)) - V_{R-max}| < \alpha$, where V_{R-max} is the expected T-step average reward for the R-max policy on M; or (2) An unknown entry will be played in the course of running R-max on M with a probability of at least $\frac{\alpha}{R_{max}}$.*

In practice, we cannot determine ρ, the adversary's policy, ahead of time. Thus, we do not know whether R-max will attain near-optimal reward or whether it will reach an unknown entry with sufficient probability. The crucial point is that it will do one or the other, no matter what the adversary does.

Proof: See full paper.

We are now ready to prove Theorem 1. First, we wish to show that the expected average reward is as stated. We must consider three models: M, the real model, M'_L the actual model used, and M' where M' is an $\epsilon/2$-approximation of M such that the SG induced by M' and L is M'_L. At each T-step iteration of our algorithm we can apply the Implicit Explore or Exploit Lemma to M' and M'_L for the set L applicable at that stage. Hence, at each step either the current R-max policy leads to an average reward that is $\epsilon/2$ close to optimal with respect to the adversary's behavior and the model M' or it leads to an efficient learning policy with respect to the same model. However, because M' is an $\epsilon/2$-approximation of M, the simulation lemma guarantees that the policy generated is either ϵ close to optimal or explores efficiently. We know that the number of T-step phases in which we are exploring can be bounded polynomially. This follows from the fact that we have a polynomial number of parameters to learn (in N and k) and that the probability that we obtain a new, useful statistics is polynomial in ϵ, T and N. Thus, if we choose a large enough (but still polynomial) number of T-step phases, we shall guarantee that our average reward is as close to optimal as we wish.

The above analysis was done assuming we actually obtain the expected value of each random variable. This cannot be guaranteed with probability 1. Yet, we can ensure that the probability that the algorithm fails to attain the expected value of certain parameters be small enough by sampling it a larger (though still polynomial) number of time. This is based on the well-known Chernoff bound. Using this technique one can show that when the variance of some random variable is bounded, we can ensure that we get its average with probability $1 - \delta$ by using a sufficiently large sample that is polynomial in $1/\delta$. A more complete analysis providing the exact values of the bounds appears in the full paper.

To remove the assumptions that the ϵ-return mixing time is known, we proceed as in [Kearns and Singh, 1998]. From the proof of the algorithm we deduced some polynomial P in the problem parameters such that if T is the mixing-time, then after $P(T)$ steps we are guaranteed, with probability $1 - \delta$, the desirable return. We

[3] Formal proofs appear in a published technical report that we cannot cite because of the anonymity requirement. We will gladly supply it if needed.

repeat the execution of the algorithm for all values of $T = 1, 2, 3, \ldots$, each time performing $P(T)$ steps. Suppose that T_0 is the mixing time, then after $O(P(T_0)^2)$ steps, we will obtain the desirable return. ∎

One thing to notice is that the agent may never know the values of some of the slots in the game because of the adversary's choices. Consequently, if π is the optimal policy given full information about the game, the agent may actually converge to a policy π' that differs from π, but which yields the best return given the adversary's actual behavior. This return will be no smaller than the return guaranteed by π. The mixing time of π' will, in general, differ from the mixing time of π. However, we are guaranteed that if T_0 is the ϵ-return mixing time of π, and v is its value, after time polynomial in T_0, the agent's actual return will be at least v (subject to the deviations afforded by the theorem).

5 Conclusion

We described R-max, a simple reinforcement learning algorithm that is guaranteed to lead to polynomial time convergence to near-optimal average reward whenever the game's parameters are known and the mixing time is finite.

R-max is an optimistic model-based algorithm that formally justifies the optimism in the face of uncertainty bias. Its analysis is similar, in many respects, to Kearns and Singh's E^3 algorithm. However, unlike the E^3, the agent does not need to explicitly contemplate whether to explore or to exploit. In fact, the agent may never learn an optimal policy for the game,[4] or it may play an optimal policy without knowing that it is optimal. The "clever" aspect of the agent's policy is that it "offers" a catch to the adversary: if the adversary plays well, and leads the agent to low payoffs, then the agent will, with sufficient probability, learn something that will allow it to improve its policy. Eventually, without too many "unpleasant" learning phases, the agent will have obtained enough information to generate an optimal policy.

R-max can be applied to MDPs, repeated games, and SGs provided that the latter satisfy the conditions we stated initially. In particular, all single-controller stochastic game instances covered in [Brafman and Tennenholtz, 2000] fall into this category, and R-max can be applied to them. However, R-max is much simpler conceptually and easier to implement than the LSG algorithm described there. Moreover, it also attains higher payoff: In LSG the agent must pay an additional multiplicative factor ϕ that does not appear in R-max.

Two other SG learning algorithms appeared in the literature. Littman [Littman, 1994] describes a variant of Q-learning, called minimax Q-learning, designed for 2-person zero-sum stochastic games. That paper presents experimental results, but provides no convergence guarantees or convergence rates. Hu and Wellman [Hu and Wellman, 1998] consider a more general framework of multi-agent general-sum games. This framework is more general than the framework treated in this paper which dealt with fixed-sum, two-player games. Hu and Wellman based their algorithm on Q-learning as well. They prove that their algorithm converges to the optimal value (defined, in their case, via the notion of Nash equilibrium). However, convergence is in the limit, i.e., provided that every state and every joint action has been visited infinitely often. Note that an adversary can prevent a learning agent from learning certain aspects of the game indefinitely and that R-max's polynomial time convergence to optimal payoff is guaranteed even if certain states and joint actions have never been encountered.

The class of repeated games is another sub-class of stochastic games that satisfies the conditions of R-max. In repeated games, $T = 1$, there are no transition probabilities to learn, and we need not use a fictitious stage-game. Therefore, a much simpler version of R-max can be used. The resulting algorithm is much simpler and much more efficient than previous algorithms by Megiddo [Megiddo, 1980] and by Banos [Banos, 1968]. Moreover, for these algorithms, only convergence in the limit is proven.

[4] In a game, an agent need not play optimally to obtain an optimal reward because it may obtain this reward because of bad choices by the adversary.

References

[Aumann and Maschler, 1995] R. Aumann and M. Maschler. *Repeated Games with Incomplete Information.* MIT Press, 1995.

[Banos, 1968] A. Banos. On pseudo games. *The Annals of Mathematical Statistics*, 39:1932–1945, 1968.

[Brafman and Tennenholtz, 2000] R. Brafman and M. Tennenholtz. A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence*, 121(1–2):31–47, 2000.

[Hu and Wellman, 1998] J. Hu and M.P. Wellman. Multi-agent reinforcement learning: Theoretical framework and an algorithms. In *Proc. 15th International Conference on Machine Learning*, 1998.

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

[Kaelbling, 1993] L. P. Kaelbling. *Learning in Embedded Systems.* The MIT Press, 1993.

[Kearns and Koller, 1999] M. Kearns and D. Koller. Efficient reinforcement learning in factored mdps. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 740–747, 1999.

[Kearns and Singh, 1998] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Int. Conf. on Machine Learning*, 1998.

[Littman, 1994] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th Intl. Conf. on Machine Learning*, pages 157–163, 1994.

[Megiddo, 1980] N. Megiddo. On repeated games with incomplete information played by non-bayesian players. *International Journal of Game Theory*, 9:157–167, 1980.

[Monderer and Tennenholtz, 1997] D. Monderer and M. Tennenholtz. Dynamic Non-Bayesian Decision-Making. *J. of AI Research*, 7:231–248, 1997.

[Moore and Atkenson, 1993] A. W. Moore and C. G. Atkenson. Prioratized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

[Schmidhuber, 1991] J. H. Schmidhuber. Curious model-building control systems. In *Proc. Intl. Joint Conf. on Neural Networks*, pages 1458–1463, 1991.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Sutton, 1990] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the 7th Intl. Conf. on Machine Learning*. Morgan Kaufmann, 1990.

[Tadepalli and Ok, 1998] P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100:177–224, 1998.