Markov games as a framework for multi-agent reinforcement learning*

Michael L. Littman
Brown University / Bellcore
Department of Computer Science
Brown University
Providence, RI 02912-1910
mlittman@cs.brown.edu

Abstract

In the Markov decision process (MDP) formalization of reinforcement learning, a single adaptive agent interacts with an environment defined by a probabilistic transition function. In this solipsistic view, secondary agents can only be part of the environment and are therefore fixed in their behavior. The framework of Markov games allows us to widen this view to include multiple adaptive agents with interacting or competing goals. This paper considers a step in this direction in which exactly two agents with diametrically opposed goals share an environment. It describes a Q-learning-like algorithm for finding optimal policies and demonstrates its application to a simple two-player game in which the optimal policy is probabilistic.

1 INTRODUCTION

No agent lives in a vacuum; it must interact with other agents to achieve its goals. Reinforcement learning is a promising technique for creating agents that co-exist [Tan, 1993, Yanco and Stein, 1993], but the mathematical framework that justifies it is inappropriate for multi-agent environments. The theory of Markov Decision Processes (MDP's) [Barto *et al.*, 1989, Howard, 1960], which underlies much of the recent work on reinforcement learning, assumes that the agent's environment is stationary and as such contains no other adaptive agents.

The theory of games [von Neumann and Morgenstern, 1947] is explicitly designed for reasoning about multi-agent systems. Markov games (see e.g., [Van Der Wal, 1981]) is an extension of game theory to MDP-like environments. This paper considers the consequences of using the Markov game framework in place of MDP's in reinforcement learning. Only the specific case of two-player zero-sum games is addressed, but even in this restricted version there are insights that can be applied to open questions in the field of reinforcement learning.

2 DEFINITIONS

An MDP [Howard, 1960] is defined by a set of states, S, and actions, A. A transition function, $T : S \times A \rightarrow$ PD(S), defines the effects of the various actions on the state of the environment. (PD(S) represents the set of discrete probability distributions over the set S.) The reward function, $R : S \times A \rightarrow \Re$, specifies the agent's task.

In broad terms, the agent's objective is to find a *policy* mapping its interaction history to a current choice of action so as to maximize the expected sum of discounted reward, $E\{\sum_{j=0}^{\infty} \gamma^j r_{t+j}\}$, where r_{t+j} is the reward received j steps into the future. A *discount factor*, $0 \leq \gamma < 1$ controls how much effect future rewards have on the optimal decisions, with small values of γ emphasizing near-term gain and larger values giving significant weight to later rewards.

In its general form, a Markov game, sometimes called a stochastic game [Owen, 1982], is defined by a set of states, S, and a *collection* of action sets, $A_1, \ldots, A_k$, one for each agent in the environment. State transitions are controlled by the current state and one action from each agent: $T : S \times A_1 \times \cdots \times A_k \rightarrow$ PD(S). Each agent also has an associated reward function, $R_i : S \times A_1 \times \cdots \times A_k \rightarrow \Re$, for agent i, and attempts to maximize its expected sum of discounted rewards, $E\{\sum_{j=0}^{\infty} \gamma^j r_{i,t+j}\}$, where $r_{i,t+j}$ is the reward received j steps into the future by agent i.

In this paper, we consider a well-studied specialization in which there are only two agents and they have diametrically opposed goals. This allows us to use a single reward function that one agent tries to maximize and the other, called the *opponent*, tries to minimize. In this paper, we use A to denote the agent's action set, O to denote the opponent's action set, and $R(s, a, o)$ to denote the immediate reward to the agent for taking action $a \in A$ in state $s \in S$ when its opponent takes action $o \in O$.

Adopting this specialization, which we call a two-player zero-sum Markov game, simplifies the mathematics but makes it impossible to consider important phenomena such as cooperation. However, it is a first step and can be considered a strict generalization of both MDP's (when $|O| = 1$) and matrix games (when $|S| = 1$).

As in MDP's, the discount factor, γ, can be thought of as the probability that the game will be allowed to continue after the current move. It is possible to define a notion of *undiscounted rewards* [Schwartz, 1993], but not all Markov games have optimal strategies in the undiscounted case [Owen, 1982]. This is because, in many games, it is best to postpone risky actions indefinitely. For current purposes, the discount factor has the desirable effect of goading the players into trying to win sooner rather than later.

3 OPTIMAL POLICIES

The previous section defined the agent's objective as maximizing the expected sum of discounted reward. There are subtleties in applying this definition to Markov games, however. First, we consider the parallel scenario in MDP's.

In an MDP, an optimal policy is one that maximizes the expected sum of discounted reward and is $undominated$, meaning that there is no state from which any other policy can achieve a better expected sum of discounted reward. Every MDP has at least one optimal policy and of the optimal policies for a given MDP, at least one is stationary and deterministic. This means that, for any MDP, there is a policy $\pi : S \rightarrow A$ that is optimal. The policy π is called stationary since it does not change as a function of time and it is called deterministic since the same action is always chosen whenever the agent is in state s, for all $s \in S$.

For many Markov games, there is no policy that is undominated because performance depends critically on the choice of opponent. In the game theory literature, the resolution to this dilemma is to eliminate the choice and evaluate each policy with respect to the opponent that makes it look the worst. This performance measure prefers conservative strategies that can force any opponent to a draw to more daring ones that accrue a great deal of reward against some opponents and lose a great deal to others. This is the essence of minimax: Behave so as to maximize your reward in the worst case.

Given this definition of optimality, Markov games have several important properties. Like MDP's, every Markov game has a non-empty set of optimal policies, at least one of which is stationary. Unlike MDP's, there need not be a deterministic optimal policy. Instead, the optimal stationary policy is sometimes probabilistic, mapping states to discrete probability distributions over actions, $\pi : S \rightarrow \mathrm{PD}(A)$. A classic example is "rock, paper, scissors" in which any deterministic policy can be consistently defeated.

The idea that optimal policies are sometimes stochastic may seem strange to readers familiar with MDP's or games with alternating turns like backgammon or tic-tac-toe, since in these frameworks there is always a deterministic policy that does no worse than the best probabilistic one. The need for probabilistic action choice stems from the agent's uncertainty of its opponent's current move and its requirement to avoid being "second guessed."

		Agent		
		rock	paper	scissors
Opponent	rock	0	1	-1
	paper	-1	0	1
	scissors	1	-1	0

Table 1: The matrix game for "rock, paper, scissors."

$$
\begin{array}{rlrlrll}
 & & & \pi_{\text{paper}} & - & \pi_{\text{scissors}} & \geq V \quad \textit{(vs. rock)} \\
- & \pi_{\text{rock}} & & & + & \pi_{\text{scissors}} & \geq V \quad \textit{(vs. paper)} \\
 & \pi_{\text{rock}} & - & \pi_{\text{paper}} & & & \geq V \quad \textit{(vs. scissors)} \\
 & \pi_{\text{rock}} & + & \pi_{\text{paper}} & + & \pi_{\text{scissors}} & = 1
\end{array}
$$

Table 2: Linear constraints on the solution to a matrix game.

4 FINDING OPTIMAL POLICIES

This section reviews methods for finding optimal policies for matrix games, MDP's, and Markov games. It uses a uniform notation that is intended to emphasize the similarities between the three frameworks. To avoid confusion, function names that appear more than once appear with different numbers of arguments each time.

4.1 MATRIX GAMES

At the core of the theory of games is the *matrix game* defined by a matrix, R, of instantaneous rewards. Component $R_{i,j}$ is the reward to the agent for choosing action j when its opponent chooses action i. The agent strives to maximize its expected reward while the opponent tries to minimize it. Table 1 gives the matrix game corresponding to "rock, paper, scissors."

The agent's policy is a probability distribution over actions, $\pi \in \mathrm{PD}(A)$. For "rock, paper, scissors," π is made up of 3 components: π_{rock}, π_{paper}, and π_{scissors}. According to the notion of optimality discussed earlier, the optimal agent's minimum expected reward should be as large as possible. How can we find a policy that achieves this? Imagine that we would be satisfied with a policy that is guaranteed an expected score of V no matter which action the opponent chooses. The inequalities in Table 2, with $\pi \geq 0$, constrain the components of π to represent exactly those policies—any solution to the inequalities would suffice.

For π to be optimal, we must identify the largest V for which there is some value of π that makes the constraints hold. Linear programming (see, e.g., [Strang, 1980]) is a general technique for solving problems of this kind. In this example, linear programming finds a value of 0 for V and (1/3, 1/3, 1/3) for π. We can abbreviate this linear program as:

$$V = \max_{\pi \in \mathrm{PD}(A)} \min_{o \in O} \sum_{a \in A} R_{o,a} \pi_a,$$

where $\sum_a R_{o,a} \pi_a$ expresses the expected reward to the agent for using policy π against the opponent's action o.

4.2 MDP's

There is a host of methods for solving MDP's. This section describes a general method known as *value iteration* [Bertsekas, 1987].

The *value* of a state, $V(s)$, is the total expected discounted reward attained by the optimal policy starting from state $s \in S$. States for which $V(s)$ is large are "good" in that a smart agent can collect a great deal of reward starting from those states. The *quality* of a state-action pair, $Q(s, a)$ is the total expected discounted reward attained by the non-stationary policy that takes action $a \in A$ from state $s \in S$ and then follows the optimal policy from then on. These functions satisfy the following recursive relationship for all a and s:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s') \quad (1)$$

$$V(s) = \max_{a' \in A} Q(s, a') \quad (2)$$

This says that the quality of a state-action pair is the immediate reward plus the discounted value of all succeeding states weighted by their likelihood. The value of a state is the quality of the best action for that state. It follows that knowing Q is enough to specify an optimal policy since in each state, we can choose the action with the highest Q-value.

The method of value iteration starts with estimates for Q and V and generates new estimates by treating the equal signs in Equations 1–2 as assignment operators. It can be shown that the estimated values for Q and V converge to their true values [Bertsekas, 1987].

4.3 MARKOV GAMES

Given $Q(s, a)$, an agent can maximize its reward using the "greedy" strategy of always choosing the action with the highest Q-value. This strategy is greedy because it treats $Q(s, a)$ as a surrogate for immediate reward and then acts to maximize its immediate gain. It is optimal because the Q-function is an accurate summary of future rewards.

A similar observation can be used for Markov games once we redefine $V(s)$ to be the expected reward for the optimal policy starting from state s, and $Q(s, a, o)$ as the expected reward for taking action a when the opponent chooses o from state s and continuing optimally thereafter. We can then treat the $Q(s, a, o)$ values as immediate payoffs in an unrelated sequence of matrix games (one for each state, s), each of which can be solved optimally using the techniques of Section 4.1.

Thus, the value of a state $s \in S$ in a Markov game is

$$V(s) = \max_{\pi \in \mathrm{PD}(A)} \min_{o \in O} \sum_{a \in A} Q(s, a, o)\pi_a,$$

and the quality of action a against action o in state s is

$$Q(s, a, o) = R(s, a, o) + \gamma \sum_{s'} T(s, a, o, s')V(s').$$

The resulting recursive equations look much like Equations 1–2 and indeed the analogous value iteration algorithm can be shown to converge to the correct values [Owen, 1982]. It is worth noting that in games with alternating turns, the value function need not be computed by linear programming since there is an optimal deterministic policy. In this case we can write $V(s) = \max_a \min_o Q(s, a, o)$.

5 LEARNING OPTIMAL POLICIES

Traditionally, solving an MDP using value iteration involves applying Equations 1–2 simultaneously over all $s \in S$. Watkins [Watkins, 1989] proposed an alternative approach that involves performing the updates asynchronously without the use of the transition function, T.

In this *Q-learning* formulation, an update is performed by an agent whenever it receives a reward of r when making a transition from s to s' after taking action a. The update is $Q(s, a) := r + \gamma V(s')$ which takes the place of Equation 1. The probability with which this happens is precisely $T(s, a, s')$ which is why it is possible for an agent to carry out the appropriate update without explicitly using T. This learning rule converges to the correct values for Q and V, assuming that every action is tried in every state infinitely often and that new estimates are blended with previous ones using a slow enough exponentially weighted average [Watkins and Dayan, 1992].

It is straightforward, though seemingly novel, to apply the same technique to solving Markov games. A completely specified version of the algorithm is given in Figure 1. The variables in the figure warrant explanation since some are given to the algorithm as part of the environment, others are internal to the algorithm and still others are parameters of the algorithm itself.

Variables from the environment are: the state set, `S`; the action set, `A`; the opponent's action set, `O`; and the discount factor, `gamma`. The variables internal to the learner are: a learning rate, `alpha`, which is initialized to 1.0 and decays over time; the agent's estimate of the Q-function, `Q`; the agent's estimate of the V-function, `V`; and the agent's current policy for state s, `pi[s,.]`. The remaining variables are parameters of the algorithm: `explor` controls how often the agent will deviate from its current policy to ensure that the state space is adequately explored, and `decay` controls the rate at which the learning rate decays.

This algorithm is called minimax-Q since it is essentially identical to the standard Q-learning algorithm with a minimax replacing the max.

6 EXPERIMENTS

This section demonstrates the minimax-Q learning algorithm using a simple two-player zero-sum Markov game modeled after the game of soccer.

```
Initialize:
For all s in S, a in A, and o in O,
  Let Q[s,a,o] := 1
For all s in S,
  Let V[s] := 1
For all s in S, a in A,
  Let pi[s,a] := 1/|A|
Let alpha := 1.0

Choose an action:
With probability explor, return an action uniformly at random.
Otherwise, if current state is s,
  Return action a with probability pi[s,a].

Learn:
After receiving reward rew for moving from state s to s'
      via action a and opponent's action o,
  Let Q[s,a,o] := (1-alpha) * Q[s,a,o] + alpha * (rew + gamma * V[s'])
  Use linear programming to find pi[s,.] such that:
   pi[s,.]  := argmax{pi'[s,.], min{o', sum{a', pi[s,a'] * Q[s,a',o']}}}
  Let V[s] := min{o', sum{a', pi[s,a'] * Q[s,a',o']}}
  Let alpha := alpha * decay
```

Figure 1: The minimax-Q algorithm.

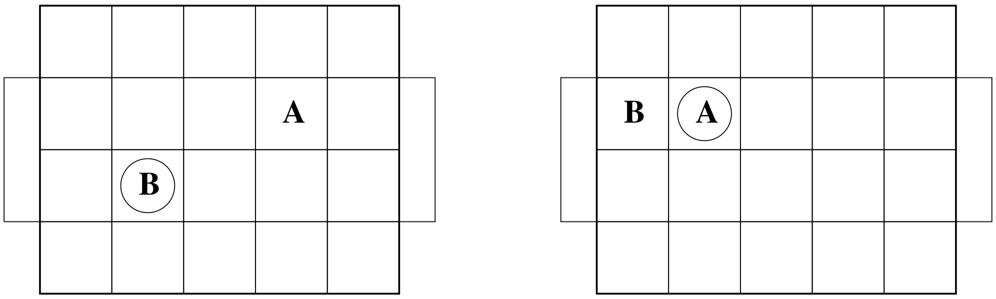

Figure 2: An initial board (left) and a situation requiring a probabilistic choice for A (right).

6.1 SOCCER

The game is played on a 4x5 grid as depicted in Figure 2. The two players, A and B, occupy distinct squares of the grid and can choose one of 5 actions on each turn: N, S, E, W, and stand. Once both players have selected their actions, the two moves are executed in random order.

The circle in the figures represents the "ball." When the player with the ball steps into the appropriate goal (left for A, right for B), that player scores a point and the board is reset to the configuration shown in the left half of the figure. Possession of the ball goes to one or the other player at random.

When a player executes an action that would take it to the square occupied by the other player, possession of the ball goes to the stationary player and the move does not take place. A good defensive maneuver, then, is to stand where the other player wants to go. Goals are worth one point and the discount factor is set to 0.9, which makes scoring sooner somewhat better than scoring later.

For an agent on the offensive to do better than breaking even against an unknown defender, the agent *must* use a probabilistic policy. For instance, in the example situation shown in the right half of Figure 2, any deterministic choice for A can be blocked indefinitely by a clever opponent. Only by choosing randomly between stand and S can the agent guarantee an opening and therefore an opportunity to score.

6.2 TRAINING AND TESTING

Four different policies were learned, two using the minimax-Q algorithm and two using Q-learning. For each learning algorithm, one learner was trained against a random opponent and the other against another learner of identical design. The resulting policies were named MR, MM, QR, and QQ for minimax-Q trained against random, minimax-Q trained against minimax-Q, Q trained against random, and Q trained against Q.

The minimax-Q algorithm (MR, MM) was as described in Figure 1 with $\texttt{explor} = 0.2$ and $\texttt{decay} = 10^{\log 0.01/10^6} = 0.9999954$ and learning took place for one million steps. (The value of `decay` was chosen so that the learning rate reached 0.01 at the end of the run.) The Q-learning algorithm (QR, QQ) was identical except a "max" operator was used in place of the minimax and the Q-table did not keep information about the opponent's action. Parameters were set identically to the minimax-Q case.

For MR and QR, the opponent for training was a fixed policy that chose actions uniformly at random. For MM and QQ, the opponent was another learner identical to the first but with separate Q and V-tables.

The resulting policies were evaluated in three ways. First, each policy was run head-to-head with a random policy for one hundred thousand steps. To emulate the discount factor, every step had a 0.1 probability of being declared a draw. Wins and losses against the random opponent were tabulated.

The second test was a head-to-head competition with a hand-built policy. This policy was deterministic and had simple rules for scoring and blocking. In 100,000 steps, it completed 5600 games against the random opponent and won 99.5% of them.

The third test used Q-learning to train a "challenger" opponent for each of MR, MM, QR and QQ. The training procedure for the challengers followed that of QR where the "champion" policy was held fixed while the challenger was trained against it. The resulting policies were then evaluated against their respective champions. This test was repeated three times to ensure stability with only the first reported here. All evaluations were repeated three times and averaged.

6.3 RESULTS

Table 3 summarizes the results. The columns marked "games" list the number of completed games in 100,000 steps and the columns marked "% won" list the percentage won by the associated policy. Percentages close to 50 indicate that the contest was nearly a draw.

All the policies did quite well when tested against the random opponent. The QR policy's performance was quite remarkable, however, since it completed more games than the other policies and won nearly all of them. This might be expected since QR was trained specifically to beat this opponent whereas MR, though trained in competition with the random policy, chooses actions with an idealized opponent in mind.

Against the hand-built policy, MM and MR did well, roughly breaking even. The MM policy did marginally better. In the limit, this should not be the case since an agent trained by the minimax-Q algorithm should be insensitive to the opponent against which it was trained and always behave so as to maximize its score in the worst case. The fact that there was a difference suggests that the algorithm had not converged on the optimal policy yet. Prior to convergence, the opponent can make a big difference to the behavior of a minimax-Q agent since playing against a strong opponent means the training will take place in important parts of the state space.

The performance of the QQ and QR policies against the hand-built policy was strikingly different. This points out an important consequence of not using a minimax criterion. A close look at the two policies indicated that QQ, by luck, implemented a defense that was perfect against the hand-built policy. The QR policy, on the other hand, happened to converge on a strategy that was not appropriate. Against a slightly different opponent, the tables would have been turned.

The fact that the QQ policy did so well against the random and hand-built opponents, especially compared to the

	MR		MM		QR		QQ	
	% won	games	% won	games	% won	games	% won	games
vs. random	99.3	6500	99.3	7200	99.4	11300	99.5	8600
vs. hand-built	48.1	4300	53.7	5300	26.1	14300	76.3	3300
vs. MR-challenger	35.0	4300						
vs. MM-challenger			37.5	4400				
vs. QR-challenger					0.0	5500		
vs. QQ-challenger							0.0	1200

Table 3: Results for policies trained by minimax-Q (MR and MM) and Q-learning (QR and QQ).

minimax policies, was somewhat surprising. Simultaneously training two adaptive agents using Q-learning is not mathematically justified and in practice is prone to "locking up," that is, reaching a mutual local maximum in which both agents stop learning prematurely (see, e.g., [Boyan, 1992]). In spite of this, some researchers have reported amazing success with this approach [Tesauro, 1992, Boyan, 1992] and it seemed to have been successful in this instance as well.

The third experiment was intended to measure the worst case performance of each of the policies. The learned policies were held fixed while a challenger was trained to beat them. This is precisely the scenario that the minimax policies were designed for because, in a fair game such as this, it should break even against even the strongest challenger. The MR policy did not quite achieve this level of performance, indicating that one million steps against a random opponent was insufficient for convergence to the optimal strategy. The MM policy did slightly better, winning against its challenger more often than did MR. It was beatable but only barely so.

The algorithms trained by Q-learning did significantly worse and were incapable of scoring at all against their challengers. This was due in great part to the fact that Q-learning is designed to find deterministic policies and every deterministic offense in this game has a perfect defense, much like rock, paper, scissors. Correctly extending Q-learning to find optimal probabilistic policies is exactly what minimax-Q was designed for.

7 DISCUSSION

This paper explores the Markov game formalism as a mathematical framework for reasoning about multi-agent environments. In particular, the paper describes a reinforcement learning approach to solving two-player zero-sum games in which the "max" operator in the update step of a standard Q-learning algorithm is replaced by a "minimax" operator that can be evaluated by solving a linear program.

The use of linear programming in the innermost loop of a learning algorithm is somewhat problematic since the computational complexity of each step is large and typically many steps will be needed before the system reaches convergence. It is possible that approximate solutions to the linear programs would suffice. Iterative methods are also quite promising since the relevant linear programs change slowly over time.

For most applications of reinforcement learning to zero-sum games, this is not an impediment. Games such as checkers [Samuel, 1959], tic-tac-toe [Boyan, 1992], backgammon [Tesauro, 1992], and Go [Schraudolph *et al.*, 1994] consist of a series of alternating moves and in such games the minimax operator can be implemented extremely efficiently.

The strength of the minimax criterion is that it allows the agent to converge to a fixed strategy that is guaranteed to be "safe" in that it does as well as possible against the worst possible opponent. It can be argued that this is unnecessary if the agent is allowed to adapt continually to its opponent. This is certainly true to some extent but any such agent will in principle be vulnerable to a devious form of trickery in which the opponent leads the agent to learn a poor policy and then exploits it. Identifying an opponent of this type for the Q-learning agent described in this paper would be an interesting topic for future research.

The use of the minimax criterion and probabilistic policies is closely connected to other current research. First, a minimax criterion can be used in single-agent environments to produce more risk-averse behavior [Heger, 1994]. Here, the random transitions of the environment play the role of the opponent. Secondly, probabilistic policies have been used in the context of acting optimally in environments where the agent's perception is incomplete [Singh *et al.*, 1994]. In these environments, random actions are used to combat the agent's uncertainty as to the true state of its environment much as random actions in games help deal with the agent's uncertainty of the opponent's move.

Although two-player Markov games are a fairly restricted class of multi-agent environments, they are of independent interest and include Markov decision processes as a special case. Applying insights from the theory of cooperative and multi-player games could also prove fruitful although finding useful connections may be challenging.

Acknowledgments

Thanks to David Ackley, Justin Boyan, Tony Cassandra, and Leslie Kaelbling for ideas and suggestions.

References

[Barto *et al.*, 1989] Barto, A. G.; Sutton, R. S.; and Watkins, C. J. C. H. 1989. Learning and sequential decision making. Technical Report 89-95, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts. Also published in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, Michael Gabriel and John Moore, editors. The MIT Press, Cambridge, Massachusetts, 1991.

[Bertsekas, 1987] Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall.

[Boyan, 1992] Boyan, Justin A. 1992. Modular neural networks for learning context-dependent game strategies. Master's thesis, Department of Engineering and Computer Laboratory, University of Cambridge, Cambridge, England.

[Heger, 1994] Heger, Matthias 1994. Consideration of risk in reinforcement learning. In *Proceedings of the Machine Learning Conference*. To appear.

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

[Owen, 1982] Owen, Guillermo 1982. *Game Theory: Second edition*. Academic Press, Orlando, Florida.

[Samuel, 1959] Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:211–229. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York 1963.

[Schraudolph *et al.*, 1994] Schraudolph, Nicol N.; Dayan, Peter; and Sejnowski, Terrence J. 1994. Using the td(lambda) algorithm to learn an evaluation function for the game of go. In *Advances in Neural Information Processing Systems 6*, San Mateo, CA. Morgan Kaufman. To appear.

[Schwartz, 1993] Schwartz, Anton 1993. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts. Morgan Kaufmann. 298–305.

[Singh *et al.*, 1994] Singh, Satinder Pal; Jaakkola, Tommi; and Jordan, Michael I. 1994. Model-free reinforcement learning for non-markovian decision problems. In *Proceedings of the Machine Learning Conference*. To appear.

[Strang, 1980] Strang, Gilbert 1980. *Linear Algebra and its applications: second edition*. Academic Press, Orlando, Florida.

[Tan, 1993] Tan, M. 1993. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts. Morgan Kaufmann.

[Tesauro, 1992] Tesauro, G. J. 1992. Practical issues in temporal difference. In Moody, J. E.; Lippman, D. S.; and Hanson, S. J., editors 1992, *Advances in Neural Information Processing Systems 4*, San Mateo, CA. Morgan Kaufman. 259–266.

[Van Der Wal, 1981] Van Der Wal, J. 1981. Stochastic dynamic programming. In *Mathematical Centre Tracts 139*. Morgan Kaufmann, Amsterdam.

[von Neumann and Morgenstern, 1947] von Neumann, J. and Morgenstern, O. 1947. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, New Jersey.

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

[Watkins, 1989] Watkins, C. J.C.H. 1989. *Learning with Delayed Rewards*. Ph.D. Dissertation, Cambridge University.

[Yanco and Stein, 1993] Yanco, Holly and Stein, Lynn Andrea 1993. An adaptive communication protocol for cooperating mobile robots. In Meyer, Jean-Arcady; Roitblat, H. L.; and Wilson, Stewart W., editors 1993, *From Animals to Animats: Proceedings of the Second International Conference on the Simultion of Adaptive Behavior*. MIT Press/Bradford Books. 478–485.