Thresholded-Rewards Reinforcement Learning

Colin McMillen

May 3, 2006

1 Background

Markov Decision Processes (MDPs) are a powerful tool for planning in the presence of uncertainty [1, 4]. MDPs provide a theoretically sound means of achieving optimal rewards in uncertain domains. The standard MDP problem is to find a policy $\pi : S \to A$ that maps states to actions such that the cumulative long-term reward is maximized according to some objective function. Over an infinite time horizon, the objective function is typically a sum of discounted rewards or the average reward rate as $t \to \infty$ [5, 8]. Over a finite time horizon, a discount factor is not needed, and the objective function is typically the sum of the rewards achieved at each time step.

In previous work [12], we have presented an alternative objective function, known as "thresholded rewards" for finite-horizon MDPs. In a thresholded-rewards problem, we are given an MDP M, a number of time steps k, and a desired reward r. We wish to find the policy π that maximizes the probability of receiving a reward of at least r in M within k time steps. It is important to note that the optimal policy for the thresholded-rewards problem is not necessarily the policy that maximizes rewards according to one of the other standard objective functions (such as discounted future rewards or average reward rate).

The thresholded-rewards objective function is motivated by the domain of robot soccer, specifically the RoboCup four-legged league, in which two teams of four Sony AIBO robots play a twenty-minute game of soccer [3, 6]. Robot soccer is a zero-sum, finite-horizon, stochastic game [2, 13, 14], in which the outcome—win or loss—is far more important than the final score. Therefore, a team that is losing near the end of the game should play aggressively to try to even the score even if an aggressive strategy has other weaknesses. Our team has the ability to change strategy as a game progresses, based on high-level features such as the time remaining in the game and the current score. [9, 10, 11]. For the RoboCup 2005 competition, we used hand-coded rules to determine the strategy, such as "play very aggressively if we are losing and there is one minute left." The thresholded-rewards objective function arose out of our desire to make decisions of this sort in a theoretically sound way.

2 Previous Results

Given an MDP M (including the definitions of the transition and reward functions), we have shown how to find the optimal thresholded-rewards policy for M [12].

To do so, we construct a new MDP M' such that finding the policy that maximizes reward in M' is equivalent to finding the policy that maximizes the probability of achieving reward of at least r in M. We can then apply standard MDP solution techniques to find the policy that maximizes reward in M', and use this result to derive the optimal strategy for the original MDP M.

We present a brief outline of our MDP-conversion algorithm: every state s in the original MDP M becomes a 3-tuple $s' = (s, t, ir)$, where s is the state from M, t is a number of time steps remaining, and ir is a possible value for the *cumulative intermediate reward* we could have accumulated within the first $k - t$ time steps. (For simplicity, assume that the rewards for each state s in M are drawn from $\{-1, 0, +1\}$.) Every time we take an action a in M', we end up in a new state s_2 (with probability determined by the probability of the transition $T(s, a, s_2)$ in M), get some intermediate reward, and have one less timestep remaining. So

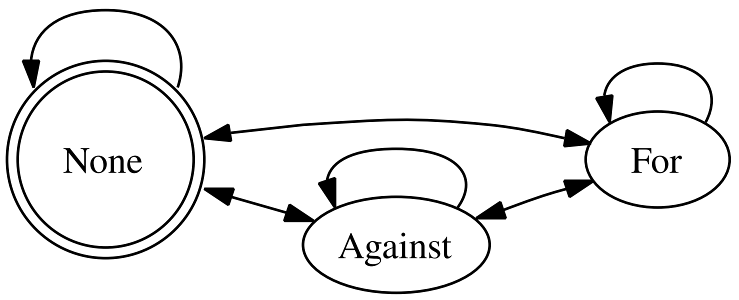

Figure 1: Example MDP M, inspired by robot soccer.

a	$T(*, a, For)$	$T(*, a, Against)$	$T(*, a, None)$
balanced	0.05	0.05	0.9
offensive	0.25	0.5	0.25
defensive	0.01	0.02	0.97

Figure 2: Transition probabilities for M. The transition probabilities are the same from every state.

all the transitions look like: $(s, t, ir) \rightarrow (s_2, t-1, ir + R(s, a))$. Due to the fact that t decreases by 1 every time we take a step, the states of M' will be partitioned into $k+1$ layers, where each layer consists of all states with a given value of t. We assign reward only to the $t = 0$ layer: a state (s, t, ir) gets a reward of $+1$ if ir is at least as big as our desired reward r, and -1 otherwise. Depending on the domain, we may want to assign a reward of 0 to those states that exactly hit the reward threshold; this is what we would do for a zero-sum game (such as robot soccer) where it is possible to tie the opponent.

To illustrate the conversion algorithm, a simple MDP M with 3 states and 3 actions is shown in Figure 1. The three states are For, which corresponds to our team's scoring a goal, and has a reward of $+1$; $Against$, which corresponds to the opponents' scoring a goal, and has a reward of -1; and $None$, which corresponds to no score occuring, and has a reward of 0. Our "agent" is actually a team of robots, and each action choice corresponds to a strategy the team can adopt. The probability of scoring is encoded in the transition function, which is summarized in Figure 2. The result of the applying the conversion algorithm to M (with $k = 3$) is shown in Figure 3.

By the definition of M', the policy which maximizes reward in M' is exactly the policy that maximizes the probability of achieving reward of at least r in M. So finding the optimal policy in M' will give us the optimal policy for M, which will be non-stationary, since the optimal action for any given state depends not just on the state, but also on the time steps remaining and the total intermediate reward accumulated so

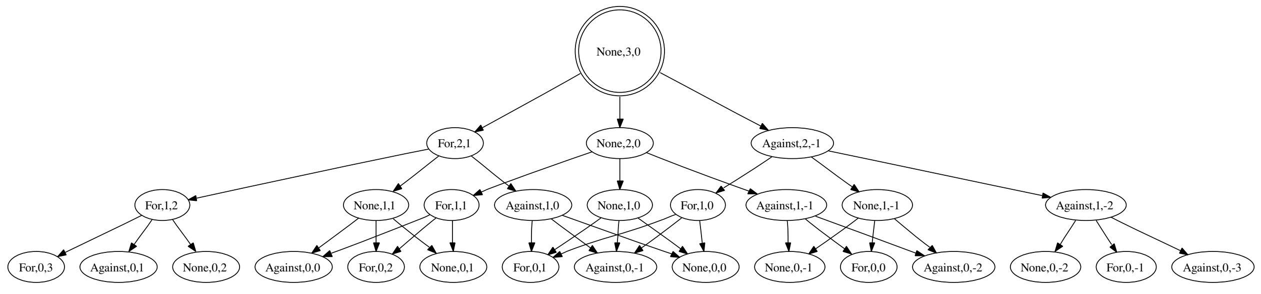

Figure 3: The MDP M' returned by the thresholded-rewards conversion algorithm given the MDP M presented in Figure 1 and $k = 3$. Each state is a tuple (s, t, ir), where s is a state from the MDP M, t is the number of time steps remaining, and ir is the cumulative intermediate reward obtained.

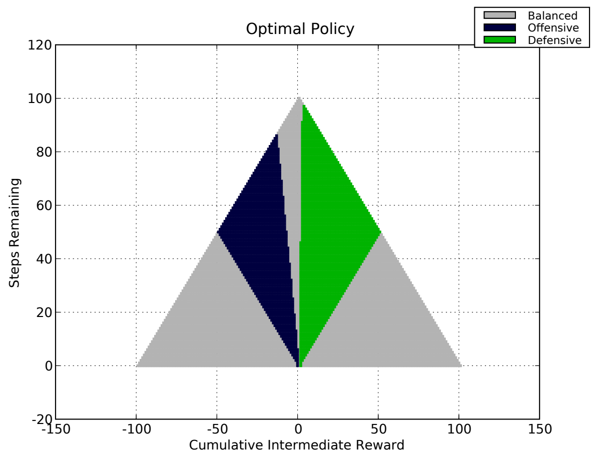

Figure 4: The optimal policy for M, with $k = 100$.

far.

We can find the optimal policy for M' by doing straightforward value iteration for k steps to compute the value V^k of every state. The number of states in each layer t of M' is $|S|(2(k-t)+1) = O(|S|k)$, and the total number of states in the $k+1$ layers of M' is upper bounded by $O(|S|k^2)$. The naïve value iteration procedure turns out to use $O(|A||S|^2k^5)$ time and $O(|S|k^2)$ space. By exploiting the layered structure of the MDP M', we can achieve much tighter bounds. Basically, at iteration t of value iteration, it is only possible to update the values of the states in layer t of M', so an efficient algorithm can just start with the layer at the bottom and propagate the rewards upwards until the initial state is reached. Then each of the $O(|S|k^2)$ states is updated only once, and each update takes time $O(|A||S|)$, leading to a time bound of $O(|A||S|^2k^2)$. We can also save on space: when updating the values of one layer, only the values of the previous layer will be needed, so we only need to hold space for two layers at a time (and the optimal policy for one layer can be written to disk before that layer is flushed from main memory.) The space requirement is then $O(|S|k)$. Using these techniques, we can efficiently find the optimal thresholded-reward policy for problems with relatively large MDPs and a reasonably large number of timesteps. The optimal policy for our example MDP M is shown in Figure 4. As expected, the optimal policy depends on both the time steps remaining and the amount of reward the agent has actually gained during execution.

3 Thresholded-Rewards Reinforcement Learning

For this project, I looked at extending the thresholded-rewards model to reinforcement learning. In the reinforcement learning scenario, our agent is operating in an MDP world, but the transition function and rewards are unknown. The goal is to find a good strategy according to some objective function. We will use the thresholded-rewards objective function; that is, we would like the agent to maximize the probability of achieving some desired reward r during the course of executing k actions in the world. However, it's not clear how to do this in a general sense: to maximize the probability of achieving a given reward, we need to take optimal actions from the beginning, but it's not possible to take optimal actions until our agent has taken sufficient exploratory actions to gain information about the world.

We therefore relax the problem slightly. Rather than finding the optimal policy, we would like to find a *sufficient policy* – one that achieves reward r with sufficiently high probability p (which is specified as input to the algorithm).

There are three main questions that have to answered when designing a reinforcement learning algorithm:

1. Which learning algorithm should we use?

2. When should the learning algorithm explore?

3. How should the learning algorithm explore?

To perform well in a given domain, all of these questions need to be answered satisfactorily. For this project, we mainly focus on the second concern—the so-called "exploration vs. exploitation tradeoff"—because the thresholded-rewards objective function gives us a particularly appealing way of deciding when to explore. Intuitively, we will explore until a policy is found such that the reward threshold r is exceeded with probability at least p. After we have found such a policy, we will "exploit" that policy—choosing the optimal actions based on our current Q estimates for each state—for the remainder of the time steps. We might also resume exploration if we find out new information that indicates that our policy is not actually sufficient.

We chose to use Q-learning [15] as the learning algorithm, because it is well-known, easy to implement, and was discussed in class. We also chose a very simple exploration strategy: when the agent chooses to explore, it does so by choosing a random action out of those available at the current state. Many reinforcement learning algorithms and exploration strategies have been proposed in the literature (see [5] for a survey), and there are undoubtedly techniques more suited to the domains presented in this paper. With a better learning algorithm, we could find a sufficient policy sooner, and may therefore be able to find policies with a higher probability of success p, since we waste less time exploring. However, we believe that the thresholded-rewards objective function provides a natural answer to the exploration-vs.-exploitation dilemma for any reinforcement learning algorithm and any exploration strategy: "quit when you've found something good enough."

The remaining question is: how do we know when we've found a policy that's good enough? The Q-values computed during Q-learning don't give us a good way of evaluating the reward we expect to get in the remaining k time steps. Instead, we build a model of the MDP as we explore, treating our experiences as estimates of the rewards and transition probabilities of each $(state, action)$ pair. To evaluate the probability of achieving a given reward, we use a Monte Carlo approach: run multiple simulations in which we start at the current state and follow the currently-optimal policy for k steps. If the fraction of simulations where we achieve the desired reward $\geq p$, we assume we've found a sufficient policy.

An alternative exploration-vs.-exploitation strategy, commonly used in the literature, is known as ϵ-greedy. At each time step, the ϵ-greedy strategy chooses to explore with probability ϵ and chooses to exploit with probability $1 - \epsilon$. Unfortunately, the performance of Q-learning depends greatly on how ϵ is chosen. Q-learning is guaranteed to converge if all $(state, action)$ pairs are sampled an infinite number of times in an infinite run. [15] Choosing any $\epsilon > 0$ satisfies this, but the actual long-term reward achieved by such an agent will not be maximized, because the agent will continue taking random actions even after the true optimal policy has been found. To maximize reward over the long term, we need to have $\epsilon \rightarrow 0$ as $t \rightarrow \infty$. Therefore, it is common to "cool down" epsilon over time—based on either the total number of actions the agent has taken or the number of times the agent has previously been in the current state. A commonly-used function that satisfies these constraints is $\epsilon = 1/(\delta \times visits(s) + 1)$, where δ is a constant that controls the rate of decay.

4 Experimental Results

To test the usefulness of the proposed exploration-vs.-exploitation strategy, we tested it in a couple of simple grid-world domains. For each domain, the number of time steps used was 1000. (This number was chosen for being reasonable in the robot soccer domain: a 20-minute soccer match is 1200 seconds long, so making 1000 action choices corresponds to choosing an action about once a second.) Four movement actions, "up", "down", "left", and "right", correspond to the four cardinal directions; a fifth action, "stay," corresponds to the agent deciding to stay in place.

The first domain is a 20×1 grid (see Figure 5), where the agent begins fully to the right. There are several nearby squares with reward 1/2; the leftmost state has reward 1. There is no action noise in this domain. It is therefore easy to find a policy that obtains a total reward of almost 500 in this domain, but

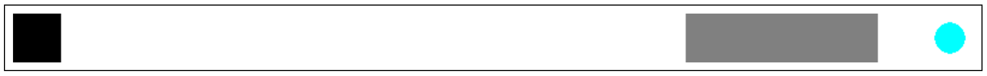

Figure 5: 20×1 grid world used in experiments. Grey states have reward $= 1/2$; the black state has reward 1. The circle shows the initial position of the agent.

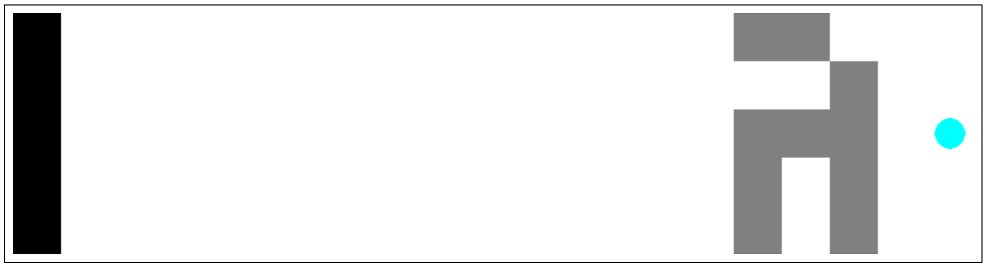

Figure 6: 20×5 grid world used in experiments. Grey states have reward $= 1/2$; black states have reward 1. The circle shows the initial position of the agent.

very difficult to find a policy that obtains a reward of 500 or more, as such a policy needs to explore until it encounters the leftmost state and then spend most of the remaining time exploiting that state.

The second domain is a 20×5 grid (see Figure 6), with a similar initial setup—the agent begins on the right, there are several nearby states offering reward 1/2, and the states all the way to the left offer a reward of 1. This domain also includes action noise: whenever the agent chooses a movement action, the agent will move in the desired direction with 70% probability and move in one of the other three directions each with 10% probability. (The "stay" action still has no noise.) Due to the increased size of the state space and the addition of action noise, finding policies with reward ≥ 500 is even more difficult in this domain.

In each domain, we tested the performance of ϵ-greedy agents and the performance of agents that use the reward threshold for their exploration policy. We ran several different ϵ-greedy agents, with different values of δ, as appropriate to each domain. The thresholded-rewards agents were run with multiple different reward thresholds: from 0 to 1000 in increments of 100. Each agent was run in each domain for 10,000 independent trials.

Figure 7 shows the cumulative distribution of the rewards obtained by the ϵ-greedy agents in the 20×1 domain. Each line shows the performance of a single agent, with the associated δ value. It is clear from this graph that it is relatively easy for agents to achieve a reward of about 400, regardless of the choice of ϵ, however, very few trials result in agents scoring above 500. Figure 8 shows another view of the same data. In this plot, each line represents a given threshold and the graph shows the fraction of trials that achieve the threshold for each setting of δ. This plot clearly shows one of the drawbacks of the ϵ-greedy approach: the optimal value for δ appears to depend on the reward we actually want to obtain. For instance, if we want reward ≥ 500, $\delta = 0.012$ is best (achieving the desired reward approximately 11% of the time); if we want reward ≥ 600, $\delta = 0.020$ is best (achieving the desired reward approximately 9% of the time.)

Figures 9 and 10 show the same data for the thresholded-rewards reinforcement learner. The performance of the thresholded-rewards strategy is clearly superior: when the threshold is set to 500, 38% of trials actually achieve this threshold (compared to 11% for the best parameter settings for ϵ-greedy). When the threshold is set to 800, 8% of trials achieve the threshold (compared to 3%). Even for "easy" goals, like a reward of 400, the thresholded-rewards strategy is superior, meeting the threshold in practically 100% of trials (compared to 97% for the best tested settings of ϵ-greedy).

The results of the ϵ-greedy agents in the 20×5 world are shown in Figures 11 and 12. The results are similar in shape to the results for the 20×1 world; however, the δ parameters that lead to good performance are an order of magnitude apart. For instance, the value of δ that maximizes the probability of achieving reward 500 is 0.012 in the 20×1 world, but 0.25 in the 20×5 world. This shows another weakness of the ϵ-greedy strategy: over a finite horizon, the best choice of parameters depends heavily on the environment.

In this case, it is true even though the two environments are qualitatively very similar.

In contrast, the performance of the thresholded-rewards reinforcement learners (shown in Figures 13 and 14) is still quite high. For a threshold of 500, 27% of trials still manage to achieve the threshold. The larger state space and addition of action noise have hurt the performance of the learner slightly, but the overall performance is still much higher than the ϵ-greedy strategy, without the need for any additional parameter tweaking.

5 Discussion / Future Work

In this paper, we have shown that a thresholded-rewards objective function provides a natural answer to the "exploration-vs.-exploitation" tradeoff in machine learning. Preliminary experimental results show that explicitly taking the desired reward into account can greatly increase the performance of a simple reinforcement learning algorithm. In addition, the thresholded-rewards approach is parameter-free, in that there are no domain-specific parameters to be tweaked—the user only has to specify the desired reward and the desired probability of achieving this reward.

Though the results presented here are promising, there are also many possible avenues for future work. First, model-free approaches such as Q-learning are somewhat notorious for making inefficient use of the available data. Since we are concerned about acting in a finite-horizon environment, the amount of available data will always be fixed, so it seems that a more data-efficient method would perform better in these domains. Since we already need to estimate the transition probabilities and rewards to do policy evaluation, it seems appropriate to try using a model-based reinforcement learning approach.

One possible shortcoming of our current approach is that the agent has no explicit preference for choosing high-reward policies over low-reward policies, as long as it has found a policy that achieves the desired reward. For instance, if the threshold is set to 200 in the 20×1 world, a perfectly valid policy involves alternating between a state with reward 1/2 and a state with reward 0. The expected value of such a behavior is slightly less than 250 (depending on how many exploration steps were needed to find this policy), which satisfies the reward threshold. However, it seems our agent should prefer the strategy of just staying on the square with reward 1/2, which will also satisfy the reward threshold with high probability, and has the added bonus of achieving a higher expected reward. But if our agent never explores after finding the "alternation" policy, the higher-reward strategy will never be found.

Another potential shortcoming with our approach is that we only evaluate the optimal policy found by the Q-learner. This policy is optimal only in the sense of maximizing expected rewards—not necessarily in the sense of optimizing the probability of achieving a given reward. It's possible to construct worlds where the policy that maximizes the probability of achieving a given reward is not the policy that maximizes expected reward. So there is a chance that our agent could have the information needed to find a sufficient policy, yet not actually find the policy.

Another possible avenue of future research would be exploring the use a thresholded-rewards objective function for reinforcement learning in partially observable MDPs (POMDPs). Unfortunately, existing theoretical results seem to discourage this avenue—it is known that the problem of deciding whether an infinite-horizon POMDP has a policy that achieves reward $\geq r$ with probability > 0 is undecidable. [7]

References

[1] Richard E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

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

[3] RoboCup Technical Committee. Sony four legged robot football league rule book, 2006.

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

[5] Leslie P. Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 1996.

[6] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot World Cup initiative. In *Proc. of the First Intl. Conf. on Autonomous Agents (Agents '97)*, 1997.

[7] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the American Association for Artificial Intelligence*, 1999.

[8] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1-3):159–195, 1996.

[9] Colin McMillen, Paul Rybski, and Manuela Veloso. Levels of multi-robot coordination for dynamic environments. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III*, pages 53–64. Kluwer Academic Publishers, 2005.

[10] Colin McMillen and Manuela Veloso. Distributed, play-based coordination for robot teams in dynamic environments. In *RoboCup 2006: Robot Soccer World Cup X*, 2006 (in press).

[11] Colin McMillen and Manuela Veloso. Distributed, play-based role assignment for robot teams in dynamic environments. In *Proceedings of Distributed Autonomous Robotic Systems*, 2006 (in press).

[12] Colin McMillen and Manuela Veloso. Thresholded-rewards finite-horizon Markov Decision Processes: Temporal planning with uncertainty. In *Proceedings of Uncertainty in Artificial Intelligence*, 2006 (under review).

[13] T. E. S. Raghavan and J. A. Filar. Algorithms for stochastic games–A survey. *Mathematical Methods of Operations Research (ZOR)*, 1991.

[14] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39:1095–1100, 1953.

[15] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4), 1992.

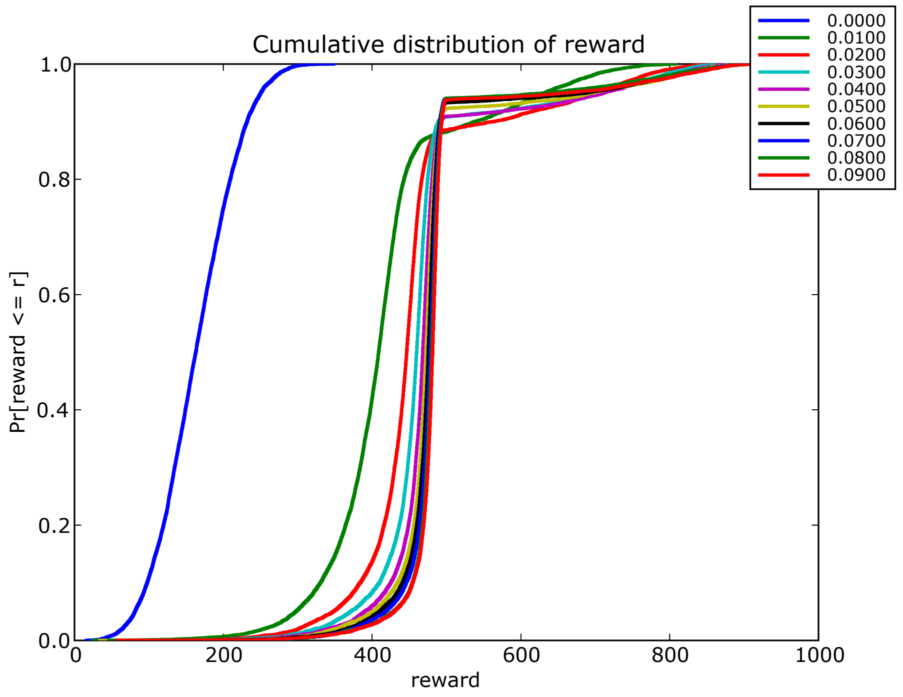

Figure 7: CDF of rewards for ϵ-greedy agents in the 20×1 world. Each line represents a single setting of the δ parameter.

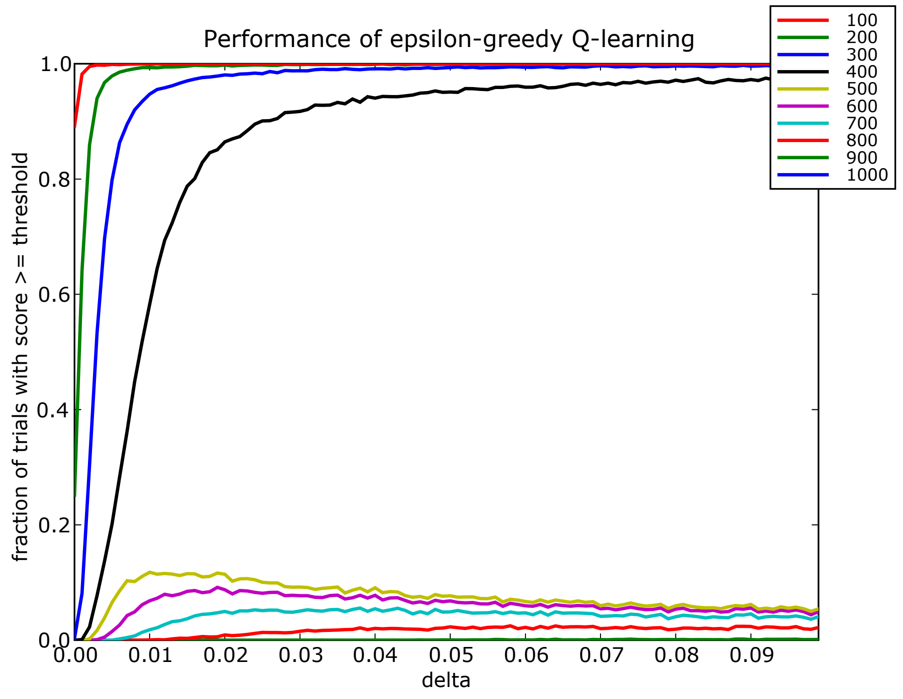

Figure 8: Performance of ϵ-greedy agents in the 20×1 world for various values of the reward threshold r.

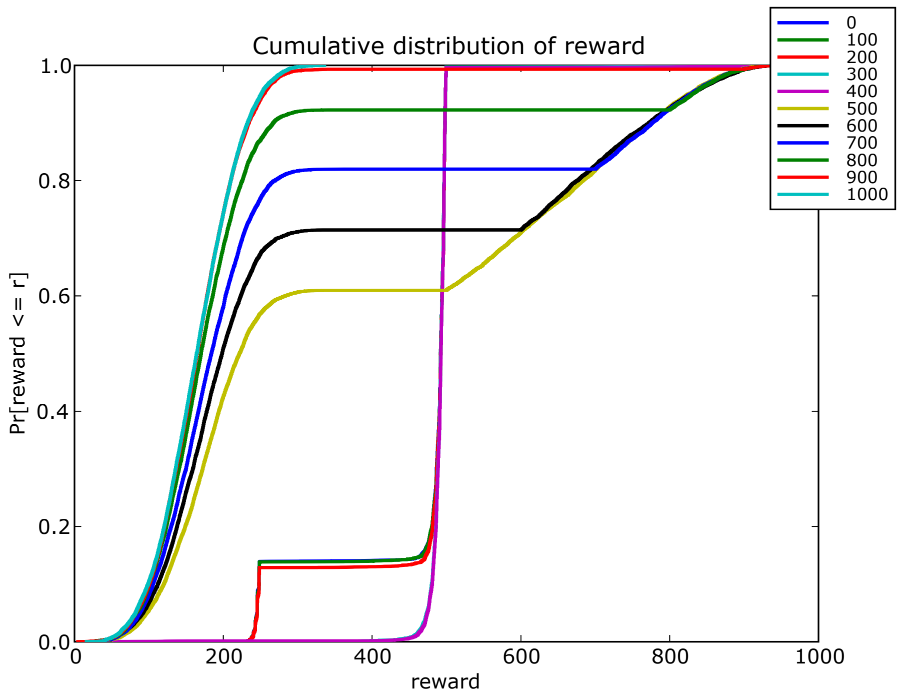

Figure 9: CDF of rewards for thresholded-rewards agents in the 20×1 world. Each line represents a single setting of the “desired reward” parameter.

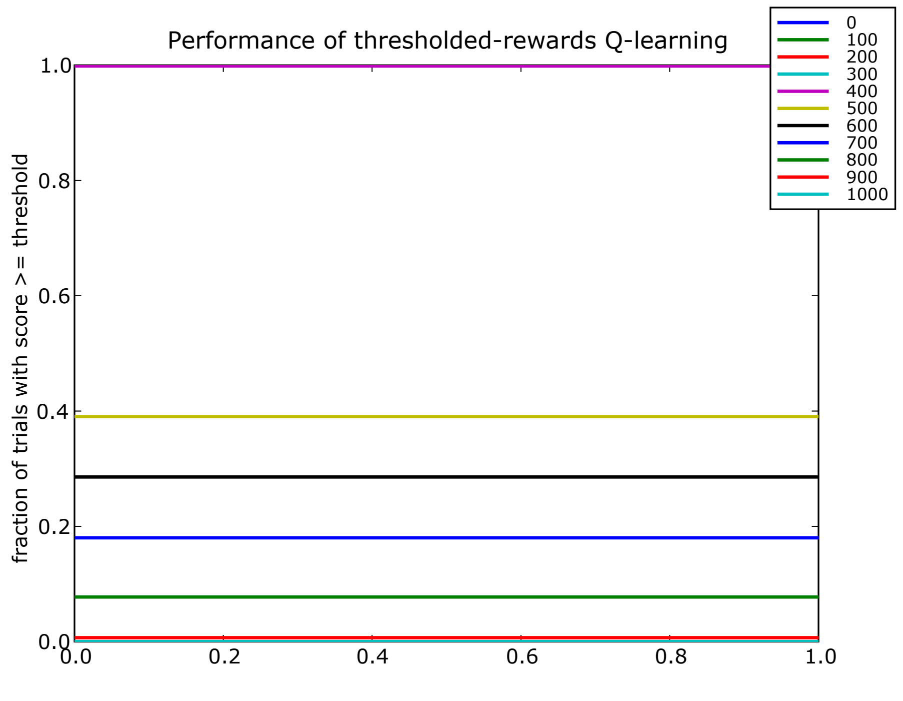

Figure 10: Performance of thresholded-rewards agents in the 20×1 world for various values of the reward threshold r.

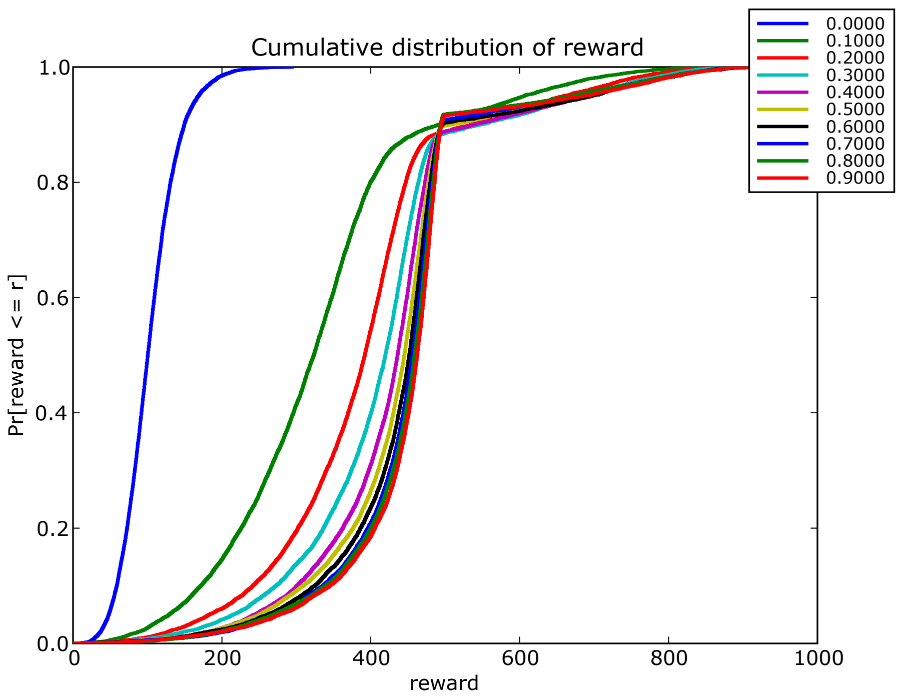

Figure 11: CDF of rewards for ϵ-greedy agents in the 20×5 world. Each line represents a single setting of the δ parameter.

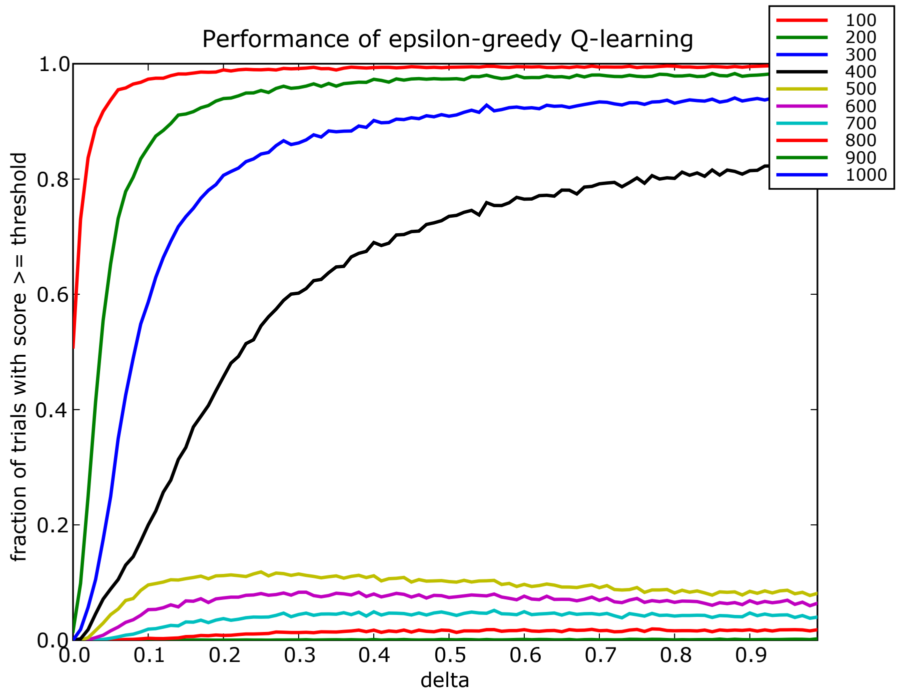

Figure 12: Performance of ϵ-greedy agents in the 20×5 world for various values of the reward threshold r.

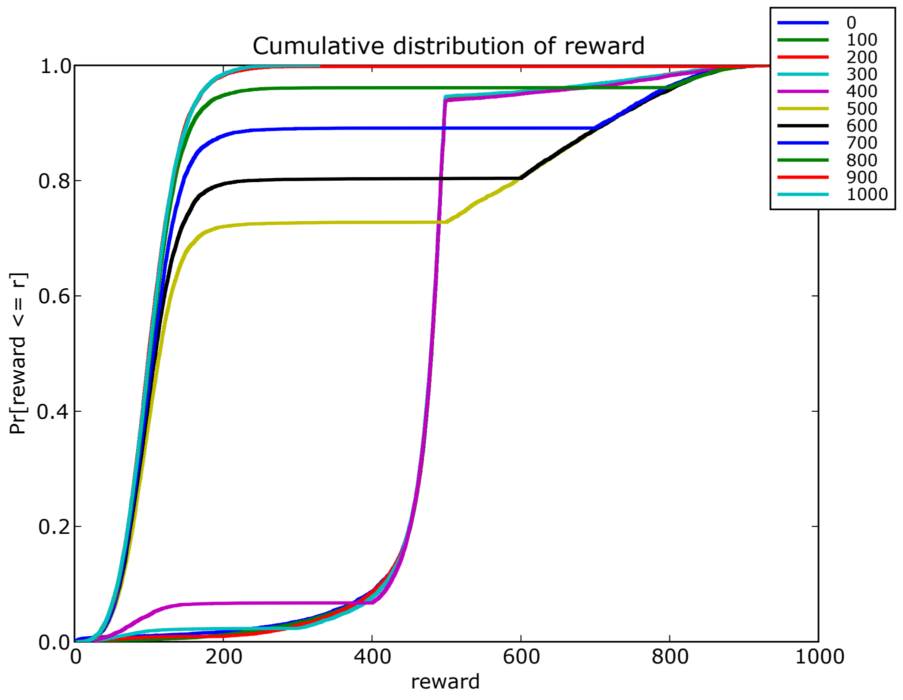

Figure 13: CDF of rewards for thresholded-rewards agents in the 20×5 world. Each line represents a single setting of the "desired reward" parameter.

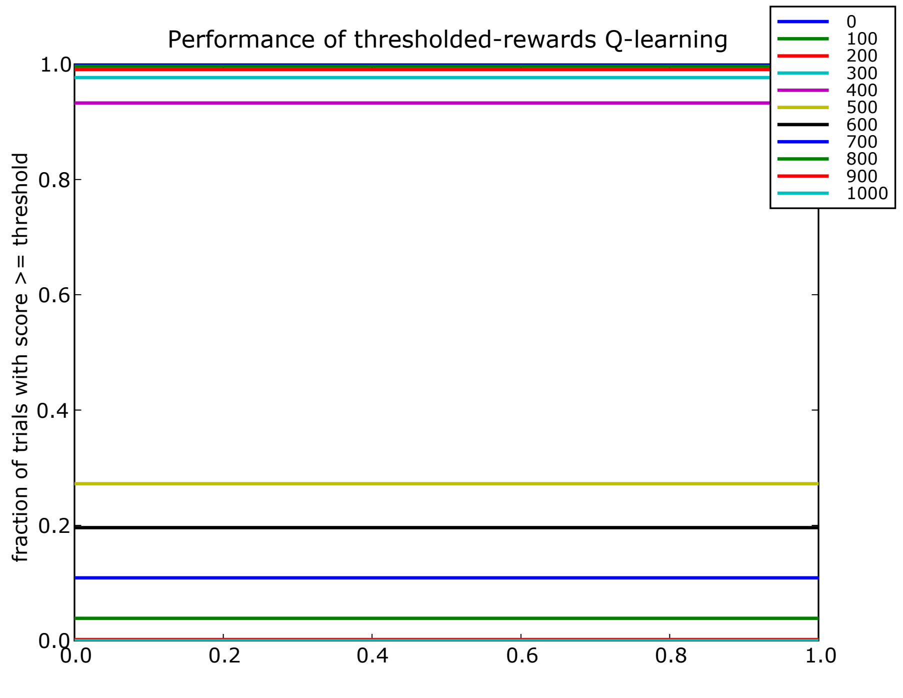

Figure 14: Performance of thresholded-rewards agents in the 20×5 world for various values of the reward threshold r.