Thresholded Rewards: Acting Optimally in Timed, Zero-Sum Games

Colin McMillen and **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.
{mcmillen,veloso}@cs.cmu.edu

Abstract

In timed, zero-sum games, the goal is to maximize the probability of *winning*, which is not necessarily the same as maximizing our expected reward. We consider *cumulative intermediate reward* to be the difference between our score and our opponent's score; the "true" reward of a win, loss, or tie is determined at the end of a game by applying a threshold function to the cumulative intermediate reward. We introduce *thresholded-rewards* problems to capture this dependency of the final reward outcome on the cumulative intermediate reward. Thresholded-rewards problems reflect different real-world stochastic planning domains, especially zero-sum games, in which time and score need to be considered. We investigate the application of thresholded rewards to finite-horizon Markov Decision Processes (MDPs). In general, the optimal policy for a thresholded-rewards MDP will be nonstationary, depending on the number of time steps remaining and the cumulative intermediate reward. We introduce an efficient value iteration algorithm that solves thresholded-rewards MDPs exactly, but with running time quadratic on the number of states in the MDP and the length of the time horizon. We investigate a number of heuristic-based techniques that efficiently find approximate solutions for MDPs with large state spaces or long time horizons.

Introduction

Markov Decision Processes (MDPs) are a powerful tool for planning in the presence of uncertainty. MDPs provide a theoretically sound means of achieving optimal rewards in uncertain domains. The standard MDP problem is to find a policy $\pi : S \rightarrow A$ 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 $t \rightarrow \infty$ (Kaelbling, Littman, & Moore 1996; Mahadevan 1996). 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.

Our work is motivated by zero-sum games with score and limited time; in particular, robot soccer. In timed, zero-sum games, winning against the opponent is 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 allows the opponent to score more easily. McMillen and Veloso discuss how a team of soccer-playing robots can change *plays* (high-level team strategies) based on factors such as the time remaining in a game and the score difference (McMillen & Veloso 2006; Stone 1998). However, this strategy selection was hand-tuned, using simple rules such as, "If our team is losing and there is less than one minute remaining, play aggressively".

In this paper, we consider an alternative objective function for finite-horizon MDPs. Rather than maximizing the cumulative reward over h time steps, we apply a threshold function f to the final cumulative reward and seek to maximize the value of f. We call this the *thresholded rewards* objective function. This objective function allows us to derive optimal strategy selections for timed, zero-sum games, such as robot soccer, in which the goal is *to win*: to be ahead of the opponent after some number of time steps. The optimal policy for such a domain is one that maximizes the probability of being ahead at the end of the game. Such a policy will generally be nonstationary: the optimal action from a given state depends on the number of timesteps remaining and the current score difference. In this paper, we present an exact algorithm for finding optimal policies for thresholded-rewards MDPs. However, the running time of this algorithm has a quadratic dependence on the number of states in the MDP and the length of the time horizon. For MDPs with large state spaces or long time horizons, the exact algorithm may be intractable. We therefore investigate a variety of approximate solution techniques.

Thresholded-Rewards MDPs

We use the standard (S, A, T, R, s_0) notation for representing MDPs; for simplicity, we assume that rewards are found in the states of the MDP (rather than in state/action pairs). The optimal policy π of an MDP can be found exactly using a technique known as *value iteration*, which uses the Bellman equation (Puterman 1994):

$$V^{n+1}(s) = \max_{a \in A} \Big\{ R(s) + \gamma \sum_{s' \in S} T(s, a, s') V^n(s') \Big\},$$

where $V^0(s) = R(s)$ and $\gamma \in (0, 1]$ is a *discount factor*. For an infinite-horizon problem, V^k converges to some V^*

as $k \to \infty$ (for $\gamma < 1$). The optimal policy π^* for an infinite-horizon MDP is *stationary* (does not depend on time). For a finite-horizon problem with k timesteps remaining, V^k allows us to find the optimal next action from any state. This optimal action may depend on the number of time steps remaining; such a policy is said to be *nonstationary*. MDPs can also be solved with *policy iteration*. In this paper, we primarily focus on value iteration techniques; however, the algorithms presented in this paper can be trivially generalized to policy iteration techniques.

Definitions

Let a *thresholded-rewards MDP* (TRMDP) be a tuple (M, f, h), where M is an MDP (S, A, T, R, s_0), f is a threshold function, and h is an integer (the time horizon). Informally, M runs for h time steps while our agent collects cumulative intermediate rewards $r_{intermediate}$; at the end, the agent receives a true reward r_{true} according to $f(r_{intermediate})$. A policy π for a TRMDP is nonstationary: it takes in a state $s \in S$, the time remaining, and the intermediate reward achieved so far. Formally, the dynamics of a TRMDP are as follows:

Algorithm 1 Dynamics of a thresholded-rewards MDP.

```
$s \leftarrow s_0$
$r_{intermediate} \leftarrow 0$
**for** $t \leftarrow h$ to $1$ **do**
    $a \leftarrow \pi(s, t, r_{intermediate})$
    $s \leftarrow s' \sim T(s, a)$
    $r_{intermediate} \leftarrow r_{intermediate} + R(s)$
$r_{true} \leftarrow f(r_{intermediate})$
```

Our main focus in this paper is timed, zero-sum games. In these domains, we consider the intermediate reward to be the difference between our agent's score and our opponent's score. We define the *zero-sum reward threshold function* as:

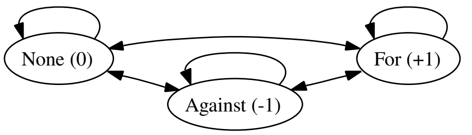
$$r_{true} = \begin{cases} 1 & \text{if } r_{intermediate} > 0 \\ 0 & \text{if } r_{intermediate} = 0 \\ -1 & \text{if } r_{intermediate} < 0. \end{cases}$$

This function assigns true reward of 1 for a win, -1 for a loss, and 0 for a tie. In a thresholded-rewards problem, we wish to find the optimal policy π^* that maximizes the expected value of r_{true}. It is important to note that, in general, 1) π^* will be nonstationary and 2) π^* is not the policy that maximizes expected intermediate reward. Though we focus on the zero-sum reward threshold function in this paper, our results generalize to arbitrary threshold functions.

Example

In a TRMDP, the optimal policy will generally be nonstationary. To illustrate this, we present an example—inspired by robot soccer—that will be used throughout the remainder of this paper. We simplify the robot soccer domain significantly by modeling it as an MDP M with three states, as shown in Figure 1:

1. FOR: our team scores a goal (reward $+1$)



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 1: Example MDP M, inspired by robot soccer.

2. AGAINST: the opponents score a goal (reward -1)
3. NONE: no score occurs (reward 0)

Our agent is a team of robots, and each action choice corresponds to a strategy (play) the team can adopt. In this example, we consider three different plays: *balanced*, *offensive*, and *defensive*. We treat the opponent team as static, using the transition probabilities to model the opponent as part of the environment. The transition probabilities of these actions are shown in Figure 1. When the *balanced* play is chosen, our agent has a 5% chance of scoring and the opponent has a 5% chance of scoring. The *offensive* play is risky: it increases our team's chance of scoring but gives the opponent an even greater chance of scoring. Inversely, the *defensive* play is conservative. For simplicity, these transition probabilities do not depend on the current state.

The expected one-step reward of action a from state s is equal to $\sum_{s'} R(s') \times T(s, a, s')$. Using this, we can compute the expected one-step reward of each action:

- *balanced*: $0 = 0.05 \times 1 + 0.05 \times -1 + 0.9 \times 0$
- *offensive*: $-0.25 = 0.25 \times 1 + 0.5 \times -1 + 0.25 \times 0$
- *defensive*: $-0.01 = 0.01 \times 1 + 0.02 \times -1 + 0.97 \times 0$

In the standard reward-maximization problem, the optimal policy for M is to execute the *balanced* action at every time step. The *balanced* play has the highest expected one-step reward, and (for this MDP) also has the optimal expected long-term reward for any choice of γ in the range $[0, 1]$.

However, our team's goal is *not* to maximize expected rewards, but to maximize the probability that we finish the game with a higher score than the opponent. We view this as a thresholded-rewards problem: we apply the zero-sum threshold function to our agent's cumulative intermediate reward and maximize the expected value of this threshold function. With thresholded rewards, the policy of always choosing *balanced* has an expected true reward of 0, with the probability of winning being equal to the probability of losing. The exact probability of each result depends on the time horizon h and can be determined by the multinomial probability distribution. For $h = 120$, the probability of winning is 44.2%, the probability of losing is 44.2%, and the probability of tying is 11.6%.

However, this policy is suboptimal for the thresholded-rewards problem, as it is possible to achieve a positive true reward in this domain. Later in this paper, we show how to derive the optimal policy for this domain, which has an expected true reward of 0.1457 (for $h = 120$). Qualitatively, the optimal policy is nonstationary, choosing the *offensive* action if our agent is losing near the end of the game and choosing the *defensive* action if our agent is winning near the end of the game. By doing so, the optimal policy increases the chance of getting a "win" or "tie" result, at the expense of maximizing the expected value of the intermediate rewards.

Related Work

Sutton and Barto present a blackjack domain which is similar to a thresholded-rewards problem (Sutton & Barto 1998). In the blackjack domain, each hand proceeds until the end of the game, at which time the agent receives a score of $-1, 0$, or $+1$ based on the current state. Each game is of a finite (though not fixed) length. However, there is no notion of time in this domain—the optimal policy is completely determined by the current state and is stationary with respect to the amount of time that has passed.

Bacchus *et al.* discuss the concept of *non-Markovian rewards* (NMRDPs) as a way to assign rewards to behaviors that extend over time (Bacchus, Boutilier, & Grove 1996). NMRDPs are a generalization of MDPs in which the reward function R takes in *histories*, of the form $\langle s_1, s_2, \ldots s_n \rangle$. Thresholded rewards are closely related to NMRDPs, as the thresholded-rewards objective function is also a form of non-Markovian reward. However, with thresholded rewards, we are not interested in accumulating rewards based on the past history of the system. Instead, we are interested in choosing actions based on the expected reward accumulated at some fixed point in the future. Bacchus *et al.* present an algorithm for converting an NMRDP into a standard MDP by "expanding" the MDP: annotating each state with the additional history information needed to ascribe the rewards. Unfortunately, this transformed MDP can be exponentially larger than the base MDP. We use a similar technique to solve TRMDPs; however, the size of our transformed MDP is only polynomially larger than the original MDP. In this paper, we present a value iteration algorithm that exploits the structure of our transformed MDP to efficiently find the optimal policy.

Thresholded-Rewards MDP Conversion

In order to solve a TRMDP (M, f, h), we create a new MDP M' such that finding the policy that maximizes reward in M' is equivalent to finding the policy that maximizes $f(r_{intermediate})$ in M.[1] In the next section, we present an algorithm that solves the converted MDP M' efficiently.

Algorithm 2 shows our TRMDP conversion algorithm. Each state s' in the converted MDP M' is a tuple (s, t, ir), where s is a base state from M, t is the number of time steps remaining, and ir is the cumulative intermediate reward received by the agent within the first $h - t$ time steps. By design, the optimal action for a state $s' = (s, t, ir)$ in M' is the optimal action for being in some state s in M with t time steps remaining and cumulative intermediate reward of ir.

Algorithm 2 Converts a TRMDP (M, f, h) into an MDP M' suitable for finding the optimal thresholded-rewards policy.

```
 1: Given: MDP $M = (S, A, T, R, s_0)$, threshold function
    $f$, time horizon $h$
 2: $s'_0 \leftarrow (s_0, h, 0)$
 3: $S' \leftarrow \{s'_0\}$
 4: for $i \leftarrow h$ to 1 do
 5:     for all states $s'_1 = (s_1, t, ir) \in S'$ such that $t = i$ do
 6:         for all transitions $T(s_1, a, s_2)$ in $M$ do
 7:             $s'_2 \leftarrow (s_2, t - 1, ir + R(s_2))$
 8:             $S' \leftarrow S' \cup \{s'_2\}$
 9:             $T'(s'_1, a, s'_2) = T(s_1, a, s_2)$
10: for all states $s' = (s, t, ir)$ in $M'$ do
11:     if $t = 0$ then
12:         $R'(s') \leftarrow f(ir)$
13:     else
14:         $R'(s') \leftarrow 0$
15: return $M' = (S', A, T', R', s'_0)$
```

Solving M' allows us to extract the optimal non-stationary policy for the TRMDP (M, f, h).

We now explain Algorithm 2 in detail. On line 2 of the conversion algorithm, the initial state s'_0 of M' is set to $(s_0, h, 0)$, indicating that the agent starts in state s_0, with h time steps remaining, and no cumulative intermediate reward. S' is the set of states in the converted MDP; S' initially contains only the starting state s'_0.

The first loop (lines 4–9) generates all the remaining states in S' and the new transition function T'. This loop iterates from h steps remaining down to 1; at iteration i it generates all states that have $i - 1$ time steps remaining, by finding all the states in S' with i time steps remaining (line 5) and generating all possible successors of these states. For each such state $s'_1 = (s_1, t, ir)$, the algorithm finds all transitions in M from s_1 to some other state s_2 on action a (line 6). A new state s'_2 is created for each such s_2. Each s'_2 has base state s_2, $i - 1$ time steps remaining, and cumulative intermediate reward $ir + R(s_2)$ (line 7). s'_2 is added to S' if it doesn't already exist (line 8). The transition probability $T'(s'_1, a, s'_2)$ is equal to the transition probability $T(s_1, a, s_2)$ of the base MDP M (line 9).

The second loop (lines 10–14) assigns rewards to each state in S'. Each final state is assigned reward based on applying the threshold function f to the cumulative intermediate reward ir (line 12). Non-final states are assigned reward 0 (line 14). The algorithm has now defined S', T', R', and s'_0; the action space A is left unchanged. The converted MDP M' is then (S', A, T', R', s'_0) (line 15). Figure 2 shows the result of applying the conversion algorithm to the example MDP M (with $h = 3$).

Exact Solutions

The optimal policy for a TRMDP (M, f, h) is the solution to M', which is generated by Algorithm 2 and can be solved using any MDP solution technique. The following facts about M' allow for an efficient value iteration algorithm:

[1] Throughout the remainder of this paper, we will use s', $T'(s', a, s'_2)$, and $R'(s')$ to refer to the states, transitions, and rewards of the converted MDP M'. We will use s, $T(s, a, s_2)$, and $R(s)$ to refer to the base MDP M.

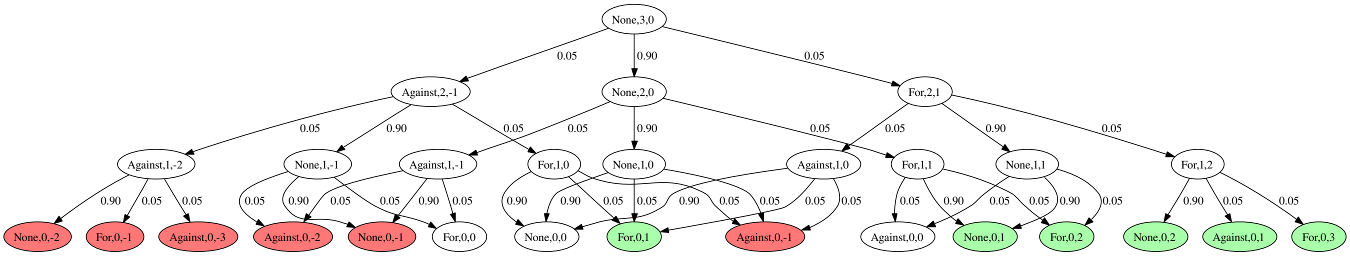


Figure 2: The MDP M' returned by Algorithm 2 given the MDP M and $h = 3$. Lightly-shaded states have reward 1; darkly-shaded states have reward -1; unshaded states have reward 0. Transition probabilities for the *balanced* action are shown.

Fact 1 *M' has a layered, feed-forward structure: every layer contains transitions only into the next layer.*

All MDPs generated by the conversion algorithm will have this structure due to the fact that t must decrease by 1 at every time step. (Figure 2 shows this fact visually.)

Fact 2 *At iteration k of value iteration, the only values that change are those for the states $s' = (s, t, ir)$ such that $t = k$.*

By design, non-zero rewards are found only in the bottom layer of the MDP; with each iteration of value iteration, these rewards propagate up to all the states in the next-higher layer. The value iteration algorithm completes after computing V^h; that is, when all the rewards have percolated up to the initial state.

These facts allow for an efficient implementation of value iteration on M': we start at the $t = 0$ layer and apply Bellman backups until the rewards "bubble up" to the top. At iteration k, we only need to calculate the value of the states in layer k of M' (that is, the states where $t = k$). Also, we do not need to sum over all states in S' when computing each value but only its $O(|S|)$ potential successors in the next lower layer (those states where $t = k - 1$.) Since each state is backed up only once, the running time of value iteration is proportional to $|S'|$, the number of states in M'.

The states of M' are arranged into $h + 1$ layers. At most, each layer k will have one state for every combination of s and ir that is possible to achieve after $h - k$ steps. At the top level, there is only one possible intermediate reward value. If we assume that intermediate rewards are drawn from a set N of small integers (which is typically the case for timed, zero-sum games), then the number of possible intermediate-reward values at each subsequent layer grows by (at most) the magnitude m of the largest element in N. The number of states in layer k is therefore upper bounded by $|S| \times (h - k) \times m$, and the total number of states in the $h + 1$ layers of M' is $O(|S|h^2m)$. Each Bellman update requires a maximization over $|A|$ actions of a sum of $\leq |S|$ possible successor states. Since each state is updated exactly once, the worst-case running time of value iteration on M' is $O(|A||S|^2h^2m)$.

Figure 3 shows the optimal policy for the example MDP M (Figure 1) with time horizon $h = 120$. The y-axis shows the number of time steps remaining; the x-axis shows the cumulative intermediate reward (score difference). The shaded areas show the optimal action for every possible combination of time remaining and intermediate reward. (Since M's transition probabilities are the same from every state, the policy does not depend on the current state.) This policy has an expected reward of 0.1457. By following this policy, our agent will win approximately 50% of the time, lose 35% of the time, and tie 15% of the time.

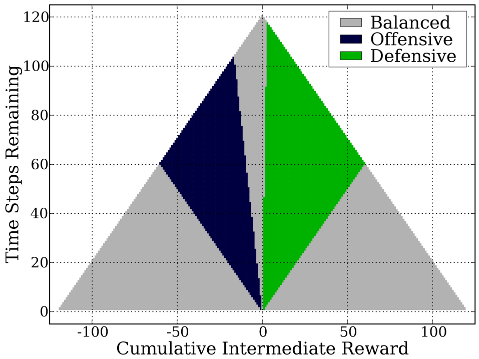


Figure 3: The optimal policy for M (shown in Figure 1), with time horizon $h = 120$ steps.

Figure 3 shows that the optimal policy for M is nonstationary: the policy depends on the number of time steps remaining and the cumulative intermediate reward. Qualitatively, the optimal policy is to choose the *defensive* play when winning by a significant number of points and to choose the *offensive* play when losing by a significant number of points. When the score is close to a tie, the best play is *balanced*. As the time remaining decreases, the point difference needed to choose *offensive* or *defensive* decreases—the agent acts more urgently. The *balanced* regions in the lower-left and lower-right of the figure are states from which the actions of the agent no longer have any effect on the outcome, because the number of steps remaining is greater than the score difference. In these regions, all actions have equal expected reward and the agent chooses *balanced* by default.

In Figure 4, we show the effect of changing the opponent's capabilities. Specifically, we vary the probability $T(*, \textit{balanced}, \textsc{Against})$ of our opponent scoring when we choose the *balanced* action. The y-axis shows the expected true reward of following the optimal maximize-expected-rewards policy (MER) and of following the opti-

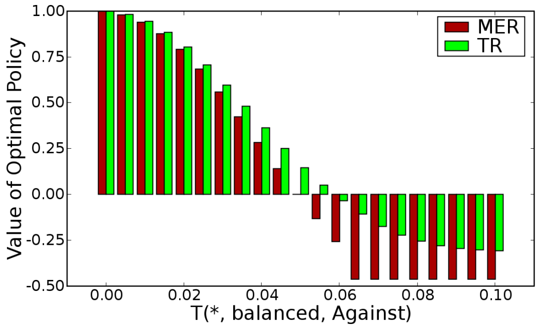


Figure 4: Effect of changing the opponent's capabilities.

mal thresholded-rewards policy (TR). In all cases, TR performs better than MER. It is interesting to note that the difference between the two objective functions is greatest when the capabilities of each team are similar—that is, where $T(*,$ *balanced*, $\textsc{Against})$ is close to 0.5.

We also consider the performance of MER and TR on 5000 randomly generated MDPs. Each of these MDPs has the same structure as M (shown in Figure 1), but the transition probabilities of each state/action pair are chosen as follows: $T(s, a,\ \textsc{Against})$ is chosen uniformly from $[0.0, 0.5)$; $T(s, a,\ \textsc{For})$ is chosen uniformly from $[0.9, 1.0) \times T(s, a,\ \textsc{Against})$; and $T(s, a,\ \textsc{None})$ is set such that the three probabilities sum to 1. Note that our team is less likely to score than the opponents at every timestep, no matter which action is chosen. Therefore, the expected true reward of MER is negative for every MDP. Figure 5 is a histogram depicting the distribution of true rewards for MER and TR on these 5000 MDPs. Each bar shows the number of MDPs that have optimal policies within a given range of true rewards. The mean true reward for MER is -0.0659; the mean true reward for TR is 0.1971. These results show that explicitly reasoning about score and time remaining allows our team to win with high probability, even against an opponent that is otherwise superior.

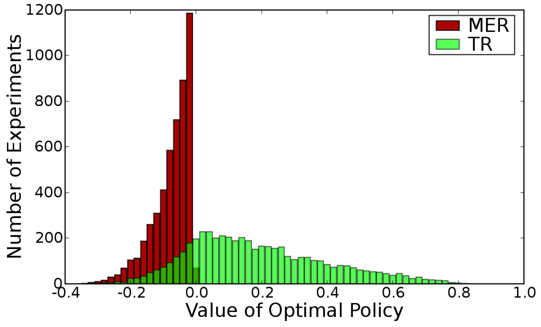


Figure 5: Performance of maximize-expected-rewards and thresholded-rewards on 5000 randomly generated MDPs.

Heuristic Techniques

The efficient value iteration algorithm presented above finds optimal solutions to thresholded-rewards MDPs in $O(|A||S|^2h^2m)$ time. The quadratic dependence on the state space size and time horizon length will be an issue for problems in which the base MDP has a large number of states or in which the time horizon is long. Several general techniques have been proposed to find approximate solutions to large MDPs efficiently, including state aggregation (Li, Walsh, & Littman 2006), factored MDPs (Guestrin *et al.* 2003; Hoey *et al.* 1999), and sparse sampling (Kearns, Mansour, & Ng 2002). In this section, we present three different heuristic techniques, specific to TRMDPs, that allow us to arrive at an approximate solution. Each of these heuristics can be seen as an informed version of state aggregation in which states are aggregated based on the time remaining.

The *uniform-k* heuristic. With this heuristic, our agent adopts a non-stationary policy but only considers changing its policy every k time steps. The net effect of this change is to "compress" the time horizon uniformly by a factor of k. This change directly leads to a decrease in the state space of the expanded MDP M', allowing for a more efficient solution. However, this solution is suboptimal because the agent does not consider switching policies at every time step.

The *lazy-k* heuristic. With this heuristic, our agent ignores the reward threshold until there are k steps remaining. For the first $h-k$ time steps, the agent is completely ignorant of the reward threshold, acting in accordance with the stationary, optimal policy for the base MDP M. Once there are k steps remaining, the agent creates a thresholded-rewards MDP M' with a time horizon k and an initial state chosen to reflect the actual current state of the system (including the cumulative intermediate reward). The agent solves M' and uses the optimal policy to adopt a nonstationary policy for the remaining k time steps. The main idea of this technique is to concentrate computational effort near the end of the run, when the agent's actions may have a greater effect on the overall outcome.

The *logarithmic-k-m* heuristic. With this heuristic, the agent makes a number of decisions that is logarithmic in the time horizon. The *lazy-k* heuristic saves computation by only considering the end of the time horizon; the *uniform-k* heuristic saves computation by compressing the entire time horizon. The *logarithmic* heuristic is a hybrid approach in which the time resolution becomes finer as we approach the time horizon. For instance, we might allow the agent to switch policies at every step during the final 10 steps of the run, every two steps for the next 20 steps of the run, every four steps for the next 40 steps of the run, and so on. The *logarithmic* heuristic depends on two parameters: k, the number of decisions the agent makes before the time resolution is increased, and m, the multiple by which the resolution is increased. For the example given above, $k = 10$ because the agent needs to take 10 actions before each increase, and $m = 2$ because the time resolution doubles on each increase.

We tested the performance of these heuristic techniques, for a variety of parameter settings, on 60 different MDPs. These MDPs were chosen randomly from the 5000 MDPs that were used in the previous section. Figure 6 summarizes the results. Each point on the graph corresponds to a heuristic technique with some parameter setting. The x-axis shows the number of states in the expanded MDP (averaged

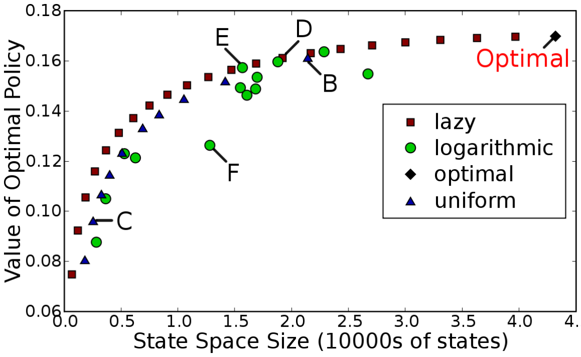


Figure 6: Performance of heuristic techniques on 60 randomly generated MDPs.

over the 60 MDPs); the y-axis shows the mean expected true reward of that heuristic technique. Ideally, we would like a technique that provides a high true reward with a low number of states. Points in the upper-left frontier of the graph represent Pareto-efficient tradeoffs between state space size and expected true reward.

The optimal algorithm, labeled "Optimal" in the figure, has a mean reward of 0.1699 and requires 43,200 states to compute. For small values of k, the *uniform* heuristic closely approximates the optimal solution while significantly reducing the size of the state space. *Uniform*-2, labeled "B", has mean reward 0.1608 and requires 21,420 states. The selection of k is a tradeoff between solution time and quality; *uniform*-15 (labeled "C") uses only 2,544 states, but the reward drops to 0.0957. *Lazy*-80 (labeled "D") has mean reward 0.1612 and uses only 19,200 states; this is fewer states than *uniform*-2 and a higher mean reward. In general, the *lazy* heuristic consistently has a higher reward than *uniform* at a given state space size. *Logarithmic*-8-2, labeled "E", closely matches the performance of *lazy-k*; however, *logarithmic*-2-4 (labeled "F") performs much worse than both *lazy* and *uniform*. In general, the performance of *logarithmic* seems highly parameter-dependent, and in no case does *logarithmic* significantly outperform *lazy* at a given state space size. Of the heuristics considered in this section, *lazy* consistently offers the best tradeoff in terms of solution time and quality, which indicates that acting optimally is most important near the end of the time horizon.

Conclusion

In this paper, we introduced *thresholded-rewards* problems, in which an agent gains *intermediate rewards* during execution in a finite-horizon environment. At the end of the horizon, the agent receives a *true reward*, which is determined by applying a threshold function to the intermediate rewards. Thresholded rewards are particularly applicable to timed, zero-sum games, such as robot soccer. In these domains, thresholded rewards allow us to maximize the probability of winning.

We have presented an algorithm that converts a base MDP into an expanded MDP suitable for solving thresholded-rewards problems. Solving this expanded MDP yields the optimal, nonstationary policy for the original MDP. In this paper, we focus on an efficient value iteration algorithm which finds solutions to thresholded-rewards MDPs in $O(|A||S|^2h^2m)$ time. We also investigated three heuristic techniques that can be used to find approximately optimal policies for thresholded-rewards MDPs. The *lazy-k* heuristic, which concentrates computational effort on the end of the time horizon, consistently has the highest performance at a given state space size. We have showed experimentally that the thresholded-rewards objective function allows our team to win with high probability, even against a wide variety of otherwise superior opponents.

As presented in this paper, thresholded-rewards MDPs assume that the opponent is static and can be treated as part of the environment. If the opponent is unknown, or changes its strategy during the course of the game, we cannot model the opponent as a static part of the environment. Our future work aims to address these challenges.

Acknowledgements

This work was supported by United States Department of the Interior under Grant No. NBCH-1040007. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements of any sponsoring institution.

References

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *Proc. AAAI-96*.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *JAIR*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *JAIR*.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for MDPs. In *Symposium on Artificial Intelligence and Mathematics*.

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

McMillen, C., and Veloso, M. 2006. Distributed, play-based role assignment for robot teams in dynamic environments. In *Proc. Distributed Autonomous Robotic Systems*.

Puterman, R. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Stone, P. 1998. *Layered Learning in Multi-Agent Systems*. Ph.D. Dissertation, Carnegie Mellon University.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. MIT Press.