Unknown Rewards in Finite-Horizon Domains

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

Abstract

"Human computation" is a recent approach that extracts information from large numbers of Web users. reCAPTCHA is a human computation project that improves the process of digitizing books by getting humans to read words that are difficult for OCR algorithms to read (von Ahn *et al.* 2008). In this paper, we address an interesting strategic control problem inspired by the reCAPTCHA project: given a large set of words to transcribe within a time deadline, how can we choose the difficulty level such that we maximize the probability of successfully transcribing a document on time? Our approach is inspired by previous work on timed, zero-sum games, as we face an analogous timed policy decision on the choice of words to present to users. However, our Web-based word transcribing domain is particularly challenging as the reward of the actions is not known; i.e., there is no knowledge if the spelling provided by a human is actually correct. We contribute an approach to solve this problem by checking a small fraction of the answers at execution time, obtaining an estimate of the cumulative reward. We present experimental results showing how the number of samples and time between samples affects the probability of success. We also investigate the choice of aggressive or conservative actions with regard to the bounds produced by sampling. We successfully apply our algorithm to real data gathered by the reCAPTCHA project.

Introduction

A CAPTCHA is a challenge-response test that can be used to tell humans and computers apart (von Ahn et al. 2003). CAPTCHAs are typically used to prevent automated registrations on Web-based email services and other Web sites. The most common form of CAPTCHA consists of an image that contains distorted words or letters. The user is prompted to type in the letters that appear in the image. In order to correctly distinguish between humans and computers, a CAPTCHA must present the user with a challenge that is relatively easy for humans to solve, but difficult or impossible for computers to solve. Therefore, every time a human solves a CAPTCHA, he/she is performing a task that is known to be difficult for state-of-the-art AI algorithms.

The reCAPTCHA project, `http://recaptcha.net`, aims to make positive use of this precious "human computation" power by using human responses to CAPTCHAs to aid in the digitization of old books, newspapers, and other texts. Typically, such texts are digitized by scanning the entire source document, then running optical character recognition (OCR) software on the resulting images in order to recover the original text in digital form. However, state-of-the-art OCR software cannot achieve perfect transcription accuracy, especially on old books in which the words are often faded and distorted. In these cases, the OCR software will output its best guess for the word, but with a low confidence score. reCAPTCHA takes these low-confidence OCR words and displays them to users in the form of a CAPTCHA challenge. These human answers are then used to improve the accuracy of the digitized text. Figure 1 shows an example of this process. In this example, the circled word is read as "announcacl" by the OCR software, but with low confidence. This *unknown word* is extracted from the source document and shown to users, who type in the correct spelling: "announced." To check whether the user is a human, the CAPTCHA also displays a *known word*, for which the correct spelling is already known—"Eugene" in the example above. To ensure that automated programs cannot read the

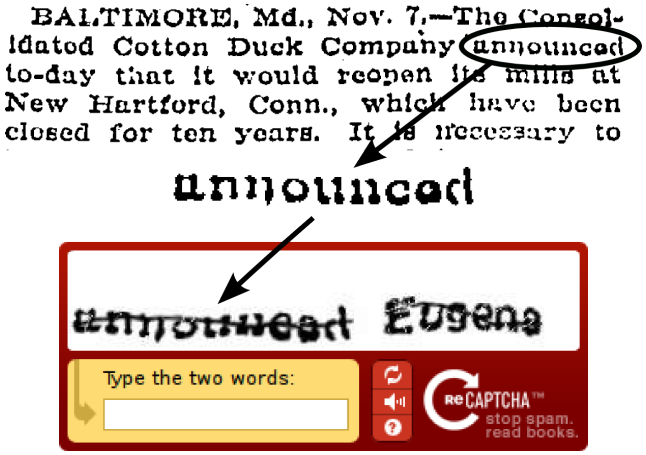

Figure 1: reCAPTCHA digitization example. OCR software reads the circled word as "announcacl," but with low confidence. This word is then extracted and presented as a CAPTCHA challenge to users, who type in the correct spelling: "announced."

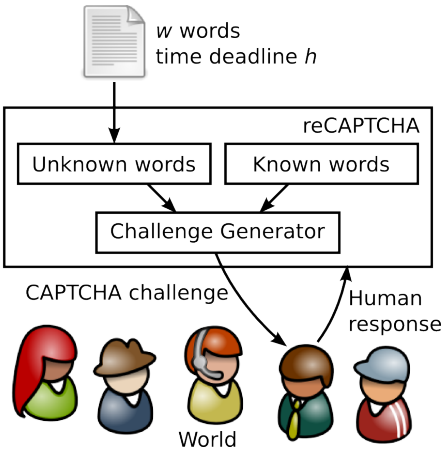

Figure 2: Outline of the reCAPTCHA system. The challenge generator is responsible for choosing words from the pools of known and unknown words and showing them to users such that that the probability of digitizing the entire document by the time deadline h is maximized.

words, both words are distorted, and a black line is drawn through them.

Figure 2 shows a simplified outline of the complete reCAPTCHA system. reCAPTCHA maintains pools of known words and unknown words. When a new document is added to the system, any words that OCR cannot confidently read are added to the pool of unknown words. For some documents, there may also be an associated time deadline; if so, the system only has a limited amount of time available to digitize the document. For the purpose of this work, the most important aspect of the reCAPTCHA system is the "Challenge Generator," which is responsible for choosing words from the word pools and displaying them to users in the form of CAPTCHA challenges. If we have a hard time deadline, the challenge generator is faced with a control problem: how should we choose challenges such that we maximize the probability of digitizing the document before the deadline? This turns out to be a particularly challenging problem because the rewards achieved are unknown; i.e., the system does not know whether a user's response to an unknown word is actually the correct spelling of that word.

During normal reCAPTCHA operation, the challenge generator sends one known word and one unknown word to each user. If the user answers the known word correctly, their response for the unknown word is counted as a "vote" for the correct spelling of that word. If enough users agree on the spelling of the unknown word, it is moved into the pool of known words.[1]

While most users make a good-faith effort to correctly transcribe the words, some users maliciously submit incorrect answers a high fraction of the time, which could potentially result in incorrect transcriptions. If a relatively large proportion of users are malicious at a given time, the challenge generator can limit the damage done by temporarily serving two known words to all users. This means that no new words are transcribed, but the quality of the transcription is not negatively affected by the malicious users. Conversely, if we know that all users are not malicious, the challenge generator could theoretically send two unknown words to each user, doubling the transcription rate but increasing the chance that any malicious user would be able to seed incorrect transcriptions into the system.[2]

In this work, we focus on a control problem inspired by reCAPTCHA: given a document containing w unknown words and a hard time deadline h, how can the challenge generator choose challenges in order to maximize the probability of successfully transcribing the entire document on time? This poses a challenging control problem, because the reCAPTCHA system does not know whether an answer to an unknown word is correct. We are therefore posed with the difficult problem of trying to obtain some given amount of reward (number of words digitized) without actually knowing the amount of reward achieved during execution.

In the following section, we briefly present a background of the thresholded-rewards framework that has inspired our work. We then present our formalization of a reCAPTCHA-inspired domain as a thresholded-rewards MDP. We present an algorithm that aims to address the problem of unknown rewards by taking periodic samples of reward values. We experimentally evaluate the performance of this algorithm against the optimal thresholded-rewards policy and the policy that maximizes expected rewards (without considering time or cumulative reward). We also compare the effects of two different reward estimation functions.

Background

In this paper, we build upon our work on MDPs with thresholded rewards (McMillen and Veloso 2007). A thresholded-rewards MDP (TRMDP) is a tuple (M, f, h), where M is an MDP, f is a threshold function, and h is the time horizon. The goal of a TRMDP is to maximize the value of $f(r)$, where r is the cumulative reward obtained by the end of h time steps. TRMDPs are also similar to the concept of risk-sensitive MDPs (Liu and Koenig 2005; 2006). In previous work, we focused on the problem of timed, zero-sum games such as robot soccer, where the goal is to be ahead at the end of the time horizon. In this work, we address a domain that also has a hard time deadline, but is not zero-sum. Instead, our "score" is the number of words successfully digitized so far. We can model the reCAPTCHA domain as an MDP, and the threshold function f is simple: 1 if we successfully digitize w words before the time deadline, 0 otherwise. This makes TRMDPs a natural choice for finding the optimal policy in this domain.

However, the optimal policy for a TRMDP depends on

[1] More details on the production reCAPTCHA system, which has been online since May 2007, can be found in (von Ahn *et al.* 2008).

[2] The production reCAPTCHA system would never send two unknown words to a user, because this would severely compromise the ability to distinguish humans from computers. For the purposes of this paper, we only address the problem of digitizing documents, ignoring the security implications of such a choice.

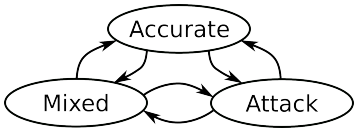

Figure 3: MDP model of the reCAPTCHA domain.

$R(s,a)$	*standard*	*two-unknown*	*two-known*
accurate	1 (p=0.9522) -2 (p=0.0478)	2 (p=0.7067) -1 (p=0.1910) -4 (p=0.1023)	0 (p=1.0)
mixed	1 (p=0.8105) -2 (p=0.1895)	2 (p=0.5569) -1 (p=0.3572) -4 (p=0.0859)	0 (p=1.0)
attack	1 (p=0.4783) -2 (p=0.5217)	2 (p=0.1288) -1 (p=0.5490) -4 (p=0.3222)	0 (p=1.0)

Figure 4: Reward distribution (from reCAPTCHA data).

the state, the time remaining, and the cumulative reward actually obtained during execution time. In the reCAPTCHA domain, this presents a problem—since the system does not know the correct spelling of each word, the system does not know how much cumulative reward has been obtained during execution time. In this paper, we extend the TR-MDP approach by presenting a sampling-based algorithm that chooses actions based on an estimate of the cumulative reward achieved so far.

In addition to the optimal solution algorithm, we have also presented heuristic solution methods for TRMDPs (McMillen and Veloso 2007). One of these heuristic approaches is *uniform-k*, in which the agent only considers changing its policy every k steps to save computation time. In this paper, we show how the *uniform-k* heuristic is also useful for the development of a sampling-based approach.

It is important to note that the problem of unknown rewards in a thresholded-rewards domain is distinct from the problem of an unknown or unmodeled domain, such as is addressed by reinforcement learning (Kaelbling, Littman, and Moore 1996). In reinforcement learning, the agent does not know the transition or reward functions *a priori*, but must learn about the domain using the transitions and rewards experienced during execution. Rather, we assume that the transition and reward functions are known, but the actual reward values received during execution are unknown.

TRMDP Domain Model

Given a source document containing w unknown words and a time deadline h steps in the future, we model the reCAPTCHA domain as a TRMDP (M, f, h), where M is an MDP containing three states (*accurate*, *mixed*, and *attack*) as shown in Figure 3. f is the threshold function:

$$f(r) = \begin{cases} 1 & \text{if } r \geq w \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

$T(s,*,s')$	s' =*accurate*	s' =*mixed*	s' =*attack*
s =*accurate*	0.9	0.09	0.01
s =*mixed*	0.09	0.9	0.01
s =*attack*	0.09	0.01	0.9

Figure 5: Transition probabilities (reCAPTCHA domain).

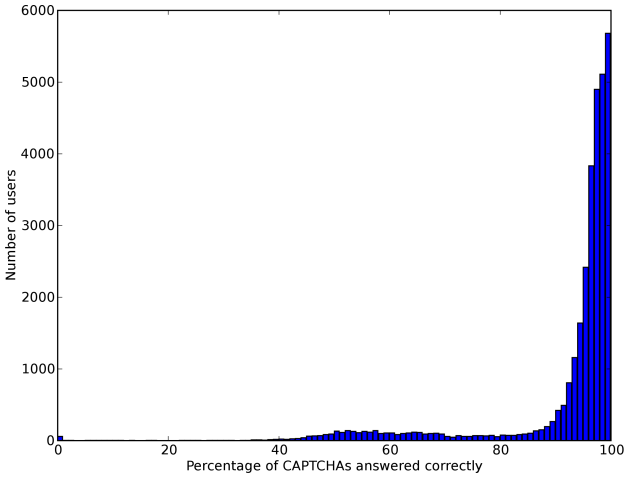

Figure 6: Histogram of per-user solution accuracy for 31,163 of the most active reCAPTCHA users.

Each step in the MDP corresponds to a single user requesting a CAPTCHA challenge. The challenge generator then needs to choose an action—what type of CAPTCHA challenge to send to the user. We consider three types of challenges: *standard*, in which the user is shown one known word and one unknown word; *two-known*, in which the user is shown two known words; and *two-unknown*, in which the user is shown two unknown words.

The reward distribution for the reCAPTCHA domain was derived by analyzing the answers provided by 31,163 of the most active reCAPTCHA users. Over a six-month period, these users submitted over 29 million answers to reCAPTCHA. We measured each user's solution accuracy; Figure 6 shows a histogram of per-user solution accuracies. Based on this histogram, we have partitioned the users into three classes:

1. *Accurate users.* This group consists of the 27,286 users that attain a solution accuracy of 85% or higher. These users submitted a total of 28,921,000 answers; 95.22% of these answers were correct. Accurate users constitute the majority of reCAPTCHA users.
2. *Inaccurate users.* This group consists of the 3,807 users with solution accuracies in the range [10%-85%). These users submitted a total of 873,000 answers; 66.88% of these answers were correct. Inaccurate users are not necessarily trying to seed incorrect answers into the system; many of them are from countries where English is not a commonly-spoken language. This makes it more difficult to pass the reCAPTCHA challenge, since most of the reCAPTCHA challenges consist of English words.

3. *Malicious users.* This group consists of 70 users with solution accuracies lower than 10%. These users submitted a total of 108,000 answers; 0.44% of these answers were correct. Though these users provide a small proportion of total answers, it is important to consider them in our analysis because their traffic can be "bursty"; i.e. multiple malicious users may simultaneously submit many bad answers to reCAPTCHA as part of an organized attack. After time passes, the malicious users generally disappear.

Our MDP model has three states: *accurate*, *mixed*, and *attack*. The *accurate* state corresponds to the most common case, when nearly all the users are "accurate": answering reCAPTCHA challenges with high accuracy. In the *mixed* state, we assume that half the users are "accurate" and half the users are "inaccurate." We therefore expect that a standard reCAPTCHA challenge will get answered correctly approximately 81% of the time. In the *attack* state, we similarly assume that half the users are "accurate" and half the users are "malicious," leading to an overall accuracy rate of approximately 48%.

Figure 4 shows the reward distribution for our model, which is derived from actual reCAPTCHA answer data. We assume that we get 1 point of reward for every unknown word answered correctly by a user, and -2 points of reward for every word answered incorrectly, as it generally takes two correct answers to override each incorrect answer when it comes time to produce the final output. For a *standard* CAPTCHA challenge, the agent receives reward of either 1 or -2. For a *two-unknown* challenge, the agent receives reward of 2 (if both words are answered correctly), -1 (if one word is answered correctly and one word is answered incorrectly), or -4 (if both words are answered incorrectly). For a *two-known* challenge, no rewards are obtained because no unknown words are presented to the user.

Figure 5 shows the transition probabilities for our domain, which model the fact that the most common state is *accurate*, followed by *mixed*, with *malicious* occurring only a small proportion of the time. We assume that our choice of action does not affect the transition probabilities; therefore the transition probabilities only depend on the current state.

Sampling-Based Control Policy

The optimal policy in a thresholded-rewards domain depends on the time remaining and the cumulative reward obtained by the agent. In this section, we present our sampling-based control policy which aims to address the challenge of having unknown rewards at execution time. We assume that our agent occasionally observes the reward received when taking an action; for the reCAPTCHA domain, we assume there is a trusted human who checks the results of every kth CAPTCHA response and verifies whether the solutions to the unknown words (if any) were correct. The agent will only consider changing its policy when it receives a reward sample, which means that the agent's policy will be equivalent to the *uniform-k* policy except that we use an estimate of the cumulative reward rather than the true cumulative reward.

Algorithm 1 shows the sampling-based control policy we have developed for thresholded-rewards domains with unknown rewards. This algorithm takes the same inputs as the TRMDP solution algorithm (McMillen and Veloso 2007): an MDP M, threshold function f, and time horizon h. For the reCAPTCHA domain presented above, f is the threshold function shown in Equation 1: we receive reward 1 if the cumulative reward is greater than or equal to the number of unknown words w in the document; 0 otherwise. Algorithm 1 also takes as input an integer k, the number of time steps elapsed between each reward sample; and a reward estimation function E, which takes in a set of reward samples S and outputs an estimate of how much cumulative reward the agent has received so far. In this paper, we consider two reward estimation functions. The first, MEAN, computes the mean of the samples, then multiplies this value by the total number of steps taken so far:

$$\textsc{Mean}(S) = \frac{\sum_{r \in S} r}{|S|} \times (h - t) \qquad (2)$$

The other reward estimation function we consider is LOW, which estimates the per-step reward as the lower bound of the 95% confidence interval of the samples seen so far:

$$\textsc{Low}(S) = \text{CI-}\textsc{Lower-Bound}(S) \times (h - t) \qquad (3)$$

Algorithm 1 Sampling-based control policy.

```
1: Given: MDP M, threshold function f, time horizon h,
   sampling interval k, reward estimation function E
2: π ← UNIFORM(M, f, h, k)
3: S ← {}      // set of reward samples
4: s ← s_0     // current state
5: for t ← h to 1 do
6:     if |S| = 0 then
7:         r̂ ← 0
8:     else
9:         r̂ ← nearest_integer(E(S))
10:    a ← π(s, t, r̂)
11:    (s, r) ← ACT(M, a)
12:    when r then   // get reward sample (every k steps)
13:        |S| ← |S| ∪ {r}
```

Line 2 of Algorithm 1 first calls the *uniform-k* TRMDP solution algorithm. This returns the best policy π that only considers changing strategies every k time steps. On lines 3–4, we initialize the set of reward samples to the empty set and the current state of the system to be the initial state of MDP M. Line 5 begins the main control loop of the agent: in each iteration through this loop, the agent executes a single action. Lines 6–9 determine the reward estimate $\hat{r}$ that will be used to determine the action. If the agent has not yet received any reward samples, it assumes that the cumulative reward is 0. If the agent does have samples, it calls the reward estimation function to estimate the cumulative reward. The result of the reward estimation function is rounded to the nearest integer, because the policy π returned by the *uniform-k* solution algorithm requires integer reward values. Line 10 determines the optimal action to take

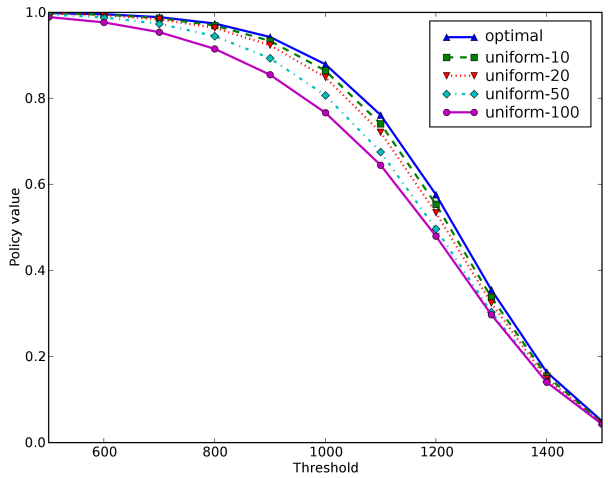

Figure 7: Value of the optimal policy and uniform-k for the reCAPTCHA domain, with the time horizon $h = 1000$, values of k in $\{10, 20, 50, 100\}$, and thresholds from 500–1500.

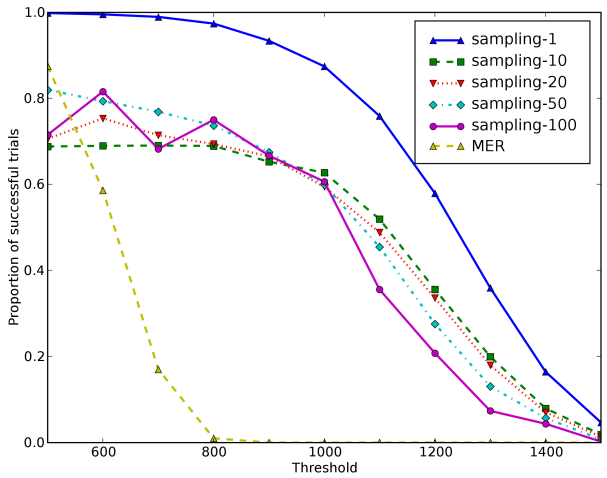

Figure 8: Results when sampling using the MEAN reward estimation function on the reCAPTCHA domain, with time horizon $h = 1000$, values of k in $\{1, 10, 20, 50, 100\}$, and thresholds from 500–1500. The y-axis shows the proportion of 3,600 trials which were successes. The success rate of the maximize-expected-rewards ("MER") policy is also shown.

given the current state, the time remaining, and our estimate of the cumulative reward. On line 11, the agent acts in the world. As a result of this action, the agent receives the new state of the system s, and may also receive knowledge of the reward r it received for taking that action. If a reward sample is received, the agent adds it to the set of reward samples seen. Regardless, the loop then continues at line 5, until h time steps have elapsed and the process completes.

Results

In this section, we present results showing the effectiveness of our algorithm on the example reCAPTCHA domain. First, we consider the question: how effective would an agent be if the agent actually knew the reward it received at execution time? By running the optimal TRMDP solution algorithm on this domain, we can find the value of the optimal policy, which serves as an upper bound on the value of any solution algorithm for this domain. Further, the *uniform-k* algorithm provides the optimal policy that only considers changing policy every k steps. This gives an upper bound on value of any policy that samples every k steps: if the reward estimation function E always returns the actual cumulative reward, the sampling policy will always choose the optimal *uniform-k* action; however, if E estimates incorrectly, the sampling policy might choose a suboptimal action.

Figure 7 shows the results of running the optimal solution algorithm and *uniform-k* heuristic on the reCAPTCHA domain, with the time horizon set to $h = 1000$ steps, threshold values ranging from 500 to 1500, and k set to 10, 20, 50, and 100. The value of each policy is equal to the probability than an agent following that policy will achieve the desired reward threshold. For a threshold of 500 to 800 words, the figure shows that the agent can achieve the threshold with near certainty ($> 95\%$). As the threshold increases, the probability of achieving the reward threshold decreases, but the optimal algorithm can still succeed over 50% of the time when the threshold is set to 1200. The value drops off sharply after that point; with the reward threshold set to 1500, the target is achieved less than 5% of the time. The *uniform-k* policies perform worse as k increases; this is unsurprising since higher values of k correspond to rougher approximations to the optimal policy. However, even *uniform*-100 has performance that is reasonably close to optimal.

Figure 8 shows the results of our sampling algorithm when the MEAN function is used to estimate the reward. Again, we have set the time horizon to $h = 1000$ steps and thresholds from 500–1500. For the sampling algorithm, k determines how often the agent receives a reward sample, and is set to 1, 10, 20, 50, or 100. *Sampling*-1, which receives a reward sample at every time step, is equivalent to the optimal policy. For each combination of k and threshold, we have run 3,600 trials; the graph shows the fraction of these trials which were successes (i.e. in which the reward threshold was met). For comparison with the non-thresholded-rewards solution, we also show the success rate of the policy that simply maximizes expected rewards ("MER"). The MER policy always chooses the *standard* CAPTCHA challenge when the MDP is in the *accurate* or *mixed* states, and the *two-known* challenge when the MDP is in the *attack* state. Since the MER policy does not depend on the reward received so far, it chooses an action at every time step (as does *sampling*-1).

Figure 8 clearly shows that the sampling-based thresholded-rewards policy outperforms the policy which maximizes expected rewards. However, there is a significant gap between the sampling-based policies and the theoretical upper bounds shown in Figure 7. With the MEAN reward estimator, agents' reward estimates are effectively "too optimistic" in a significant number of trials. An agent

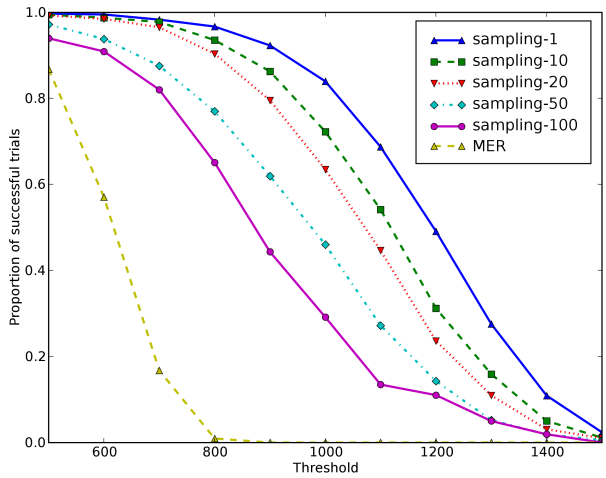

Figure 9: Results when sampling using the LOW reward estimation function on the reCAPTCHA domain, with time horizon $h = 1000$, values of k in $\{1, 10, 20, 50, 100\}$, and thresholds from 500–1500. The y-axis shows the proportion of 3,600 trials which were successes.

which believes it has enough reward to meet the threshold will begin to act conservatively, selecting the *two-known* CAPTCHA because the *two-known* CAPTCHA cannot lead to negative reward. If the reward estimate is correct, this is indeed the best strategy, but if the reward estimate is too high, the agent's "complacent" choice of the *two-known* CAPTCHA prevents it from achieving the remaining reward that is needed. When the desired reward is relatively easy to achieve, the chance of incorrectly falling into this "complacent" strategy is higher, explaining the relatively large gap between the sampling-based policies and the theoretical upper bounds when the threshold value is low.

Figure 9 shows the results of our sampling algorithm when the LOW function is used to estimate the reward. The LOW function is more pessimistic with regard to the estimated reward; it assumes that the average per-step reward is the lower bound of the 95% confidence interval of the reward samples seen so far. Since the LOW function underestimates the reward (compared to MEAN), an agent using the LOW function is unlikely to erroneously fall into the "complacent" strategy. The results indicate that the performance of the LOW function is better than the performance of MEAN for almost every setting of k and the threshold value. The performance of *sampling*-10 nearly matches the theoretical upper-bound of *uniform*-10. As expected, the overall performance degrades as we take samples less often. However, even *sampling*-100 significantly outperforms the maximize-expected-rewards policy, which is quite impressive since the *sampling*-100 agent only receives 10 reward samples over the entire time horizon.

These results show that there is a significant benefit to reasoning about reward and time in thresholded-rewards domains, even if our agent only obtains a small sample of the reward values received during execution.

Conclusion

In this paper, we have presented a sampling-based algorithm that enables agents to reason about cumulative rewards and time deadlines in domains where the exact rewards achieved by the agent are not known to the agent at execution time. Our previous work was directed towards zero-sum games; in this work we address a problem that has a hard time deadline and a notion of "score", but that is not a game and does not have a specific opponent. To this end, we have used over 29 million solutions from over 31,000 reCAPTCHA users to come up with a plausible model of a domain inspired by the reCAPTCHA project. Using this domain, we have shown that reasoning about time and "score" can lead to a significant benefit, even if we only obtain a small sample of reward values during execution time. We have tested the effectiveness of two possible reward estimation functions: MEAN, which estimates cumulative reward by using the mean of all samples seen so far; and LOW, which estimates cumulative reward by using the lower bound of the 95% confidence interval. If the MEAN function happens to overestimate the reward achieved by the agent, this can potentially cause the agent to adopt an overly conservative strategy, which detracts from overall performance. In comparison, the LOW function provides a somewhat pessimistic estimate of the reward obtained by the agent, which prevents the agent from erroneously adopting an overly conservative strategy.

Acknowledgements

We would like to thank Luis von Ahn for providing the first author with the opportunity to work on the reCAPTCHA project and also for providing helpful feedback. This research was sponsored in part by United States Department of the Interior under Grant No. NBCH-1040007, and in part by the Boeing Corporation. The views and conclusions contained in this document are those of the authors only.

References

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*.

Liu, Y., and Koenig, S. 2005. Existence and finiteness conditions for risk-sensitive planning: Results and conjectures. In *Proceedings of Uncertainty in Artificial Intelligence*.

Liu, Y., and Koenig, S. 2006. Functional value iteration for decision-theoretic planning with general utility functions. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*.

McMillen, C., and Veloso, M. 2007. Thresholded rewards: Acting optimally in timed, zero-sum games. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*.

von Ahn, L.; Blum, M.; Hopper, N. J.; and Langford, J. 2003. CAPTCHA: Using hard AI problems for security. In *Eurocrypt 2003*.

von Ahn, L. *et al.* 2008. Manual character recognition using online security measures: An example of crowd computing. *(Under review)*.