In *Proceedings of the 19th International FLAIRS Conference (FLAIRS-2006)*. pp. 580-585, Melbourne Beach, FL, May 2006.

Using Active Relocation to Aid Reinforcement Learning

Lilyana Mihalkova and **Raymond Mooney**
University of Texas, Department of Computer Sciences,
1 University Station, C0500,
Austin, TX 78712
{lilyanam,mooney}@cs.utexas.edu

Abstract

We propose a new framework for aiding a reinforcement learner by allowing it to *relocate*, or move, to a state it selects so as to decrease the number of steps it needs to take in order to develop an effective policy. The framework requires a minimal amount of human involvement or expertise and assumes a cost for each relocation. Several methods for taking advantage of the ability to relocate are proposed, and their effectiveness is tested in two commonly-used domains.

Introduction

Reinforcement learning (RL) comprises a set of machine learning methods designed to address a particular learning task in which an *agent* is placed in an unknown *environment* and is allowed to take *actions* that bring it delayed numeric *rewards* and can change its *state* in the environment. The agent's goal is to learn a policy that tells it what action to take in each state so that it maximizes the amount of reward it obtains from its interaction with the environment.

Traditional RL methods such as Q-learning (Watkins 1989), require no human intervention. The agent is simply placed in the unknown environment and allowed to explore it independently in order to determine an effective policy. One potential problem with this approach is that a large amount of environment interaction may be necessary to learn a good policy. To alleviate this problem, several approaches for guiding the agent have been developed. For example, guidance may be provided in the form of rules suggesting actions to take in particular situations (Maclin & Shavlik 1996) or as trajectories of successful runs through the environment (Driessens & Džeroski 2004).

Naturally, guidance allows the agent to find good policies faster. Sometimes, however, the required amount of human interaction may not be available. We introduce a framework that addresses this case by providing the agent with a much weaker form of assistance that does not require a large degree of human involvement or expertise. More specifically, at any time during training the agent can request to be placed in any state of the environment. We will use the term *relocation* to refer to such change of state. Relocation is particularly useful in cases in which changing the agent's state is easy and inexpensive such as when a simulator of the environment is available. Our framework assumes a cost for each relocation and seeks to limit the number of relocations. This contrasts with work such as (Kearns, Mansour, & Ng 2000) and (Lagoudakis & Parr 2003) that exploits the assumption that an available *generative model*, or simulator, of the environment can be freely used.

As an intuitive example, consider a person learning to play a strategy game. Some stages of the game are easy and the player needs little experience to become proficient at them. However, the player may not have developed a reliable strategy for overcoming the preceding hurdles. Thus, rather than continuing in an easy part of the game where she is already confident, the player prefers to save time by skipping back and practicing the challenging situations. In addition, if, while trying new approaches, the player makes a silly mistake that would inevitably end the game, she can save time by *relocating* back to a more favorable position.

This example suggests two ways in which an RL agent can take advantage of an ability to relocate—by skipping parts of the environment it is already familiar with, and by extricating itself from states it entered by mistake. In the remainder of the paper we contribute methods that exploit this capability and allow the agent to develop effective policies by taking a smaller number of steps in the environment. Empirical results in two domains—Maze and Bicycle—demonstrate the utility of the methods.

Background

An RL task is defined by a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of actions and two functions—$T : \mathcal{S} \times \mathcal{A} \rightarrow \Pr(\mathcal{S})$, which returns the probability that each state in $\mathcal{S}$ results from taking a given action a from a given state s, and $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which gives the reward obtained by taking a from s. T and R are not given but their effects can be observed through interactions with the environment at discrete time-steps. The agent looks for a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that determines what action to take in each state so that at each time-step t the expected discounted return given by the quantity $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is maximized (Sutton & Barto 1998). Here, $0 \leq \gamma \leq 1$ is the discount factor and r_k is the reward obtained at step k.

Q-learning (Watkins 1989) is a popular RL algorithm that uses the function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which computes for each (s, a) pair the expected return obtained by taking action a

in state s and following the optimal policy thereafter. Supposing that the learner transitioned to state s' and obtained reward r after taking action a from state s, Q-learning performs the following update (also called a back-up):

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma Q(s',a^*) - Q(s,a)\right)$$

where $a^* = \arg\max_a\{Q(s',a)\}$ and α is the learning rate. We will call each $Q(s,a)$ a Q-value. The Q-function determines the agent's policy. To ensure that the agent *exploits* its current knowledge but also *explores* its environment, an ϵ-*greedy* policy is frequently followed with which the agent deviates from the greedy policy with probability ϵ.

If all state/action pairs continue to be visited, Q-learning is guaranteed to converge to the Q-function of an optimal policy (Sutton & Barto 1998). When this happens, the agent can select the best action from each state. In terms of optimal behavior, however, the agent will be just as effective if it evolves the Q-values only to the extent that the best action from each state has the highest value. In this case, the correct Q-values or the relative ordering of the rest of the actions do not matter. We will call this the *best-only* characteristic.

Relocation

Q-learning assumes that the agent receives no assistance as it wanders through the environment. In this work, we slightly ease the learner's task by assuming the existence of an assistant who can at any time *relocate* the agent to any state desired by the agent. The learner can only relocate to states it has visited at least once. It is worth noting that the common case when the agent is trained in simulation to perform an episodic task already assumes the ability to relocate because at the end of each episode the simulation moves, or relocates, the agent back to a start state. We propose to make fuller use of this assumption by allowing the agent to control when and where it relocates so that it finds a good policy with less environment interaction.

To account for the additional work involved in moving the agent, a cost c will be charged for each relocation and for each restart of an episode, which is also a form of relocation. More formally, the cost is defined in terms of time-steps. Each transition that results from taking an action counts as 1 time-step, and each relocation counts as c time-steps where $c \geq 0$. The addition of the cost does not modify the learned Q-function or the reward function of the environment.

As an important aside, the ability to relocate at any time does not violate the convergence condition of Q-learning that each (s,a) pair continues to be visited. Thus, if the base exploration method does not completely ignore certain pairs, they will also not be ignored after we add relocation because the methods we propose always leave a non-zero probability of *not* relocating from a state.

In order to take advantage of the ability to relocate, the agent needs to decide *when* it should interrupt its current line of action and *where*, or to which state, it should relocate. Next, we present methods that address these questions.

When to Relocate

At every step the agent decides to relocate with probability ρ, calculated differently by the two methods discussed next.

Agent is "In Trouble." This approach is based on the intuition that if the Q-values of the best actions decrease as the agent transitions from state to state, then it is probably moving away from the useful regions of the environment. It targets the situations when the learner takes exploratory actions that turn out to be a poor choice. In such cases, even though the agent learns from the negative experience by updating its Q-values accordingly, continuing down the same path would cause it to waste time-steps in a part of the state space that is unlikely to be visited during optimal behavior.

As an illustration, consider the task of balancing a bicycle. If the agent abruptly shifts its center of mass and turns the handlebars, it will lose balance, and in a few more steps will crash, regardless of the subsequent actions it attempts. After experiencing this several times, based on its Q-values, the learner starts to recognize when it is in a helpless state and, rather than waiting for the inevitable crash, prefers to relocate in order to focus on identifying where its policy fails.

Two parameters, μ and ν, are used. Initially and after relocating ρ is set to μ, which is usually small in order to discourage repeated relocations. Otherwise, after each step ρ is modified as follows:

$$\rho = \rho + \nu(Q(s,a) - Q(s',a^*)).$$

In the above formula ν controls the rate with which ρ is modified. $Q(s,a)$ is the value of the current state/action pair, and $Q(s',a^*)$ is the greedy action value in the state that resulted from taking action a in s. This approach increases the probability with which the learner decides to relocate as the Q-values of subsequent state/action pairs decrease and vice versa. The value of ρ is not allowed to exceed 0.9. In our experience, ρ never became so large.

Agent is "Bored." The idea behind this approach is that if the algorithm updates the Q-values of the state/action pairs it visits by tiny amounts, the agent is not learning anything new in the current part of the environment.

To reflect the above intuition, the function computing ρ is designed so that the learner relocates with greatest probability when updating a particular Q-value does not change it. The only parameter used by this method is ϕ. The variable δ gives the difference between the Q-values of the current state/action pair following and preceding the update. ΔQ_{min} and ΔQ_{max} are the largest decrease and increase, respectively, in any particular Q-value observed throughout the operation of the learner. ρ is calculated as follows:

$$\rho = \begin{cases} \frac{1}{2}e^{\frac{\delta\cdot\phi}{\Delta Q_{min}}} & \text{if} \quad \delta \in [-\Delta Q_{min}, 0); \\ \frac{1}{2}e^{\frac{-\delta\cdot\phi}{\Delta Q_{max}}} & \text{if} \quad \delta \in [0, \Delta Q_{max}]. \end{cases}$$

ρ is largest (equal to 0.5) when the update to the current Q-value is 0. It drops exponentially as the amount of increase or decrease to the current Q-value grows.

Where to Relocate

Ideally, the state to which the agent chooses to relocate should fulfill two requirements: it should be likely to be encountered while following an optimal policy, and it should be one in which the agent is uncertain about the best action.

The first requirement ensures that the agent does not waste time in states that are unlikely to be visited during optimal behavior. The difficulty is that since the optimal policy is unknown, the agent cannot be sure whether a state is relevant. However, as the agent accumulates more experience, its policy is likely to come closer to optimal. Therefore, states encountered while following a later policy are more likely to be relevant. This is why the agent considers for relocation only states visited during the current episode.

The second requirement ensures that the state to which the agent relocates is one in which additional experience is necessary. Our uncertainty measure is informed by the *best-only* characteristic and depends not on how confident the agent is about the individual Q-values but on how confident it is that the action with the highest Q-value is indeed the best action. The agent's uncertainty is measured as the width of the confidence interval of the difference between the means of the two best Q-values in a given state s. A wider confidence interval indicates greater uncertainty. The width of the confidence interval is equal to (Devore 1991):

$$uncert(s) = 2 \cdot t_{\alpha/2,m+n-2} \cdot v_p \sqrt{\frac{1}{m} + \frac{1}{n}}$$

In this formula m and n are the number of times respectively each of the two best actions has been attempted. $t_{\alpha/2,m+n-2}$ is the critical value for the t distribution at confidence level α with $m + n - 2$ degrees of freedom. v_p is given by:

$$v_p^2 = \frac{(m-1) \cdot v_1^2 + (n-1) \cdot v_2^2}{m+n-2}$$

where v_1^2 and v_2^2 are the sample variances given by:

$$v_1^2 = \frac{\sum_i q_i^2 - (\sum_i q_i)^2/m}{m-1}$$

For a particular (s, a_1) pair, the q_i's are obtained by "sampling" $Q(s, a_1)$ each time the action a_1 is taken in state s. To compute v_2, we replace m by n and a_1 by a_2, where a_1 and a_2 are the actions that appear best and second-best.

Visiting states in which the learner is uncertain is related to *selective sampling* in active classifier learning (Cohn, Atlas, & Ladner 1994) where the algorithm requests the label of an example about whose correct class it is uncertain.

Putting it all together

At the beginning of an episode, the agent chooses the ϵ-greedy action from its current state, increments the relocation counter (used to calculate the cost), and initializes ρ. "In Trouble" initializes ρ to μ, and "Bored" initializes it to 0. The agent behaves as described in Algorithm 1. In this algorithm, $\mathcal{S}_{curr}$ is the set of states visited during the current episode for which the current greedy action is different from the action actually taken during the last visit to this state. The relocation cost is taken into consideration by dividing ρ by $1 + c$. Thus, the agent relocates more conservatively when relocation is expensive. Despite its simplicity, this approach to dealing with the relocation cost was preferred because it outperformed several more complex techniques we tested that were based on the differences in improvement during episodes when the agent relocated and when it did not. Immediately after relocating, the agent always takes the action with the highest Q-value. This is done because gaining more experience with the best action available from a state increases the agent's confidence in that state.

Experiments

Domains

We tested the above techniques in two common domains—Maze and Bicycle.

Maze is a standard undiscounted ($\gamma = 1$) 50×50 grid world in which the task is to reach the goal in the smallest number of steps. There are four actions—up, down, left, and right—and the selected action succeeds with probability p (we used $p = 0.9$); otherwise, a random move is executed. The agent remains in its current state if the move is impossible. The reward is -1 at each step before reaching the goal and 0 upon entering the terminal state.

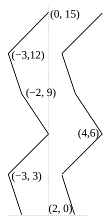

Figure 1: Bicycle Lane

In the Bicycle domain, adapted from (Randløv & Alstrøm 1998), the agent learns to balance a bicycle moving at a constant speed within a narrow lane. The state is described by the angle θ of the handlebars with respect to straight ahead, the angular velocity of θ, the angle ω of the bicycle with respect to vertical, and the angular velocity and acceleration of ω. These variables were discretized as in (Randløv & Alstrøm 1998). Two additional variables determine the position of the bicycle within the lane. D takes on the values $near$, if continuing in the same direction would take the bicycle off the lane within 5 steps; $nearing$, if this would happen within 30 steps; and far otherwise. W tells the agent whether the closer wall is on its left or right. The agent provides the torque applied to the handlebars and the amount of displacement of the center of mass, each of which has three possible values, thus adding up to a total of nine actions. A small amount of noise is added to the displacement. The agent receives a reward of -1000 and the episode ends if the bicycle falls over or drives off the lane. At all other times the reward is 0. The discount factor γ was 0.999. Figure 1 shows the lane we used. The agent starts from location $(0, 0)$. The Bicycle domain is well-suited to relocation because a simulator for the environment is available, and thus the relocation cost is tiny. Moreover, although most people can ride a bicycle, it is hard to provide detailed guidance in terms of the parameters describing the state, thus making it difficult to apply the guidance techniques mentioned in the Introduction.

Methodology

The performance of Q-learning with relocation when "Bored" and when "In Trouble" was compared to that of the baseline, Q-learning with no relocation. As an off-policy method, Q-learning is well-suited to relocation because the Q-values begin to approximate the state/action values of the optimal policy regardless of the policy followed during training. Thus the policy learned does not rely on relocation

Algorithm 1 Relocation algorithm

Suppose the agent is in state s', reached after taking action a in state s.
Update $Q(s, a)$
$attemptRelocate = true$ with probability $\rho/(c+1)$
if $\mathcal{S}_{curr} \neq \emptyset$ AND $attemptRelocate$ **then**
  $s_r = \arg\max_{s \in \mathcal{S}_{curr}} uncert(s)$
  Relocate to s_r
  Increment relocation counter
  Select greedy action from s_r
if the agent did not relocate **then**
  Select ϵ-greedy action from s'
Update ρ according to the method used

and the learning task is not altered by the ability to relocate.

A learning curve was produced by interrupting learning every 500 steps and testing on 30 episodes with a maximum length of $2,000$ steps. Each curve is the pointwise average over 140 random runs. It was smoothed using a sliding window of width 100. During training, the baseline agent followed the ϵ-greedy policy ($\epsilon = 0.1$). To ensure fairness and decrease the variance of the results, during testing the agent followed the greedy policy and was *not allowed* to relocate or to change its Q-values. In the Maze problem, performance was measured as the negative of the number of steps needed to reach the goal, and in the Bicycle task—as the number of successful steps. The learning rate α was set to the values at which the baseline achieved the best performance during preliminary experiments: 0.7 and 0.2 in the Maze and Bicycle domains respectively. The Q-function was initialized (optimistically) to 0 in all cases.

To set the parameters ν and μ used by "In Trouble" and ϕ used by "Bored," we ran preliminary experiments that also demonstrate the robustness of the proposed methods to the parameter settings. All settings in these preliminary experiments were as described above except that to save time each curve was the pointwise average of 60 instead of 140 runs. The cost c was 0. We ran "In Trouble" with all possible combinations of settings from the sets $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for ν and $\{0.1, 0.2, 0.3, 0.4\}$ for μ. Generally, larger values of these parameters cause more relocations. Figure 3 shows the best ($\nu = 0.5, \mu = 0.4$) and worst ($\nu = 0.1, \mu = 0.1$) performance.[1] The best and worst performing settings were reversed in the Bicycle task, demonstrating that more relocation does not always improve performance.

Similarly, we ran "Bored" with ϕ set to the values $10, 15, 20, 25$. The difference between the best ($\phi = 10$) and the worst ($\phi = 25$) is shown in Figure 2. Similar results are obtained in the Bicycle domain. Generally, the value of ϕ had a larger effect on the amount of relocation than on the performance. Roughly 40 relocations more per 500 training steps were performed with $\phi = 10$ than with $\phi = 25$. Given these results, we chose the most conservative setting

[1]We omit the first $500,000$ training steps in the Maze domain when the agent performs its allotment of $2,000$ steps per test episode without reaching the goal.

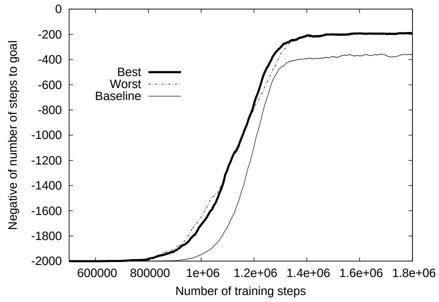

Figure 2: Parameter sensitivity of "Bored" in Maze

in terms of relocations: $\nu = 0.1, \mu = 0.1, \phi = 25$.

In the main set of experiments, we measured the relative performance of the methods at three values of the cost c: 0, 1, and 5. When $c = 5$, one relocation is 5 times more expensive than taking an ordinary action. A relocation cost of 1 is a safe overestimate in the cases when training occurs in simulation because resetting the simulator to a different state is likely to be faster than calculating the new state based on the action taken. Relocation was not found to be useful for $c \geq 10$, since in this case, not enough relocation is performed to produce a significant effect.

The significance of the results was tested at the 95% level using a z-test, which is appropriate because of the large sample size (140). One complication was that because there were slight differences in the number of relocations performed during different random runs of the same method, after factoring in the relocation cost, the number of training steps at which performance was measured did not line up. This difficulty was overcome by interpolating at 500-step intervals based on the actual measurements.

Results

Figures 4-6 show the performance for different relocation costs. In each graph, the top horizontal line is formed by plotting a dot for each point at which the improvement of "In Trouble" over the baseline is statistically significant. The bottom horizontal line is formed similarly for "Bored." Note that because *no* relocation is allowed during testing, the superior performance of the relocating agent is due entirely to the better use of training time. Although a cost of 0 is not realistic, the results in Figure 4 are interesting because they provide insights into the effects of relocation. In this case, both relocation methods give a considerable advantage over the baseline. Interestingly, "Bored" allows for faster convergence in the Maze problem but slows down learning in the Bicycle domain. This is most likely due to the fact that getting out of states where little or no updates are made focuses the agent on areas in which the Q-values are updated by large amounts. In the Maze task, this forces faster propagation once the goal is discovered. In the Bicycle domain, however, because non-zero rewards are received only upon failure, during the early stages, the agent focuses on the bad states that are unlikely to be visited frequently under a good policy. Learning how to behave in these states pays off in the long run, as indicated by the superior performance of "Bored" in the later stages of learning. On the other hand,

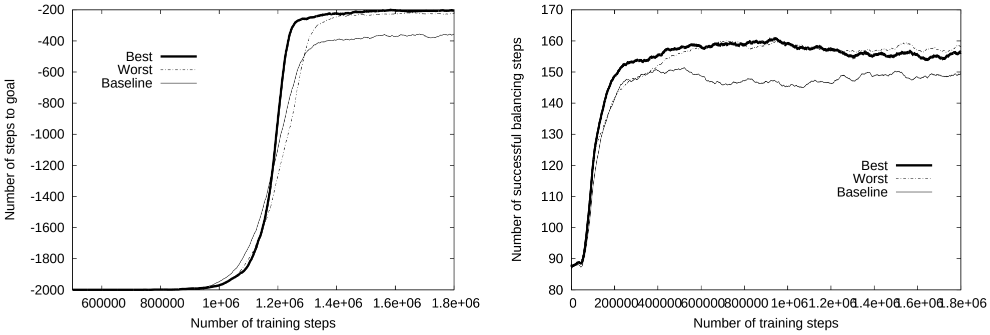

Figure 3: Parameter sensitivity of “In Trouble” in Maze (left) and Bicycle (right)

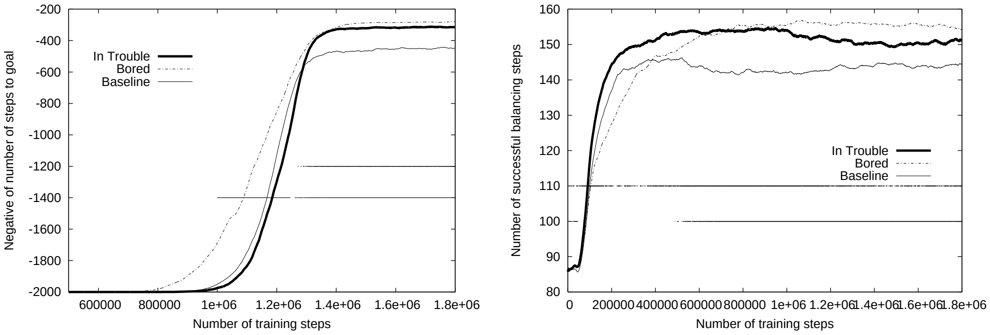

Figure 4: Performance in Maze (left) and Bicycle (right) with c = 0

“In Trouble” is very effective early on in the Bicycle task because it helps remove the agent from the bad states during training and it learns how to avoid them faster.

As the cost is increased, the relocating agent seems to be slower at discovering the goal in the Maze problem. This is particularly pronounced with “Bored” and happens because before the goal state is found, even a small amount of relocation, for which the agent pays in terms of additional time-steps, has a negative effect that cannot be immediately recouped by improved performance. In the Bicycle task, “In Trouble” never performs worse than the baseline regardless of the cost. However, at larger costs the advantages are smaller because fewer relocations take place. In summary, “Bored” is appropriate when the early cost penalty is acceptable in the interests of obtaining a superior policy later. On the other hand, “In Trouble” is not as effective when the cost of relocation is large and the task involves discovering a goal state, as in the Maze problem, but gives excellent early performance in domains like Bicycle.

Related Work

Prioritized Sweeping (PS), introduced simultaneously by Moore & Atkeson and Peng & Williams, is a method related to relocation(1993). In PS, the agent learns both a policy and a world model. As with relocation, the PS agent visits states that are not directly reachable from its current position. However, these visits occur “in its mind” based on the learned model and are performed to single states regularly after each actual step, as opposed to relocation, in which the agent permanently interrupts its line of action and decides when to relocates. The two methods also differ in how they select the states to visit. In PS, the learner prefers state/action pairs whose Q-values would undergo the largest update, whereas the relocating agent prefers uncertain states.

Driessens and Džeroski (2004) describe Active Guidance in which the learner asks for guided traces originating in a state it picks. As in relocation, the agent can control in what part of the environment it gains more experience. Unlike relocation, the starting state is one from which the learner previously failed, and restarts cannot occur at any time.

In our experiments, we have assumed the existence of a simulator for the environment. In this respect, the current project is related to work that exploits the assumption that a *generative model* of the environment is provided. For example, in (Kearns, Mansour, & Ng 2000) a simulator is used to generate reusable trajectories in the environment, and in (Lagoudakis & Parr 2003) approximate policy iteration is performed based on the generative model. However, our methods limit the number of relocations and assume that the agent relocates only rarely while following its normal course of action most of the time. This makes our framework easier to apply in real domains, particularly in ones where the cost of physically moving the learner is small.

Finally, relocation is a method for efficient exploration. So, it is related to the large body of research that addresses exploration in RL. Some methods include Interval Estimation (Kaelbling 1993), the E^3 algorithm (Kearns & Singh 1998), and model-based exploration (Wiering & Schmidhuber 1998). These methods differ from our approach mainly in that they do not assume an ability to relocate.

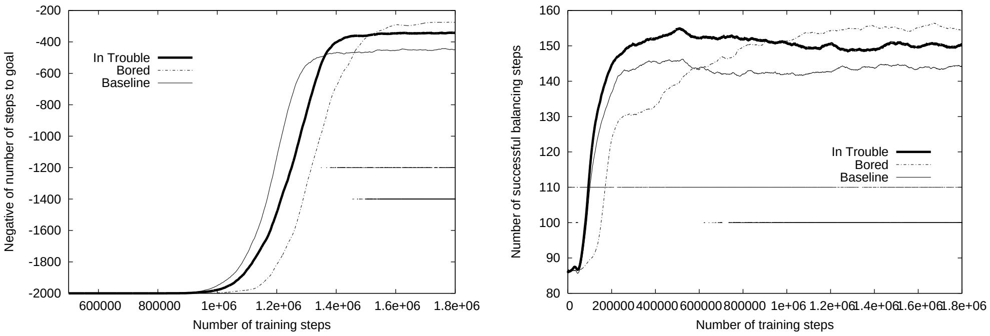

Figure 5: Performance in Maze (left) and Bicycle (right) with c = 1

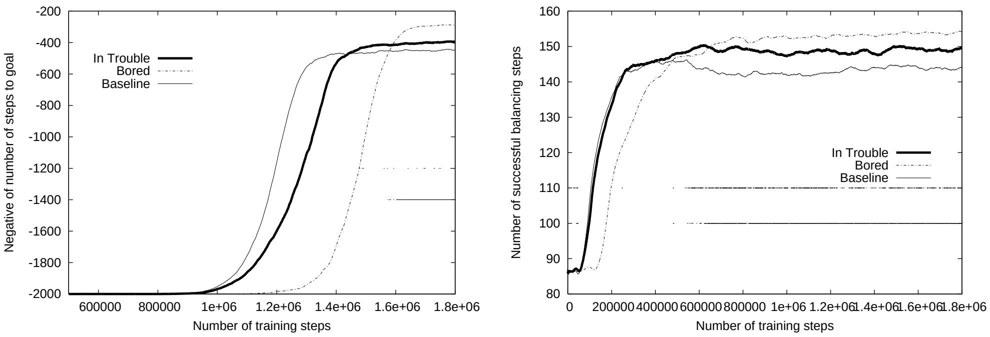

Figure 6: Performance in Maze (left) and Bicycle (right) with c = 5

Conclusions and Future Work

We proposed a framework for aiding an RL agent by allowing it to relocate, introduced methods for taking advantage of relocation, and demonstrated their effectiveness in two domains. Our approaches require minimal human expertise or involvement and consider the cost of relocation.

An important direction for future work is to extend the method for computing certainty to continuous state spaces. If tile coding (Sutton & Barto 1998) is used, one possibility is to compute confidence intervals from every tile and return the sum of the widths of the intervals corresponding to active tiles as the uncertainty of a state. A second research direction is to combine the currently used measure of uncertainty with the method PS uses to determine what state to visit. Thus the agent would relocate to states in which it is both uncertain and would update the corresponding Q-value by a large amount. Finally, it would be useful to explore ways in which the agent can adaptively determine whether "In Trouble" or "Bored" is better suited to the current domain.

Acknowledgments

We thank Nicholas Jong, Alexander Sherstov, and Peter Stone for their insightful comments on early versions of this paper. This research was partially supported by DARPA grant HR0011-04-1-0007. Lilyana Mihalkova received the MCD fellowship from the University of Texas at Austin. A large portion of the experiments were run on the Mastodon cluster provided by NSF grant EIA-0303609.

References

Cohn, D.; Atlas, L.; and Ladner, R. 1994. Improving generalization with active learning. *Machine Learning* 15(2):201–221.

Devore, J. L. 1991. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, third edition.

Driessens, K., and Džeroski, S. 2004. Integrating guidance into relational reinforcement learning. *Machine Learning* 57:271–304.

Kaelbling, L. 1993. *Learning in Embedded Systems*. MIT Press.

Kearns, M. J., and Singh, S. 1998. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2000. Approximate planning in large POMDPs via reusable trajectories. In *Advances in Neural Information Processing Systems 12*.

Lagoudakis, M. G., and Parr, R. 2003. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of 20th International Conference on Machine Learning (ICML-2003)*.

Maclin, R., and Shavlik, J. W. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22(1-3):251–281.

Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

Peng, J., and Williams, R. J. 1993. Efficient learning and planning within the dyna framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*.

Randløv, J., and Alstrøm, P. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*.

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

Watkins, C. J. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, Cambridge University.

Wiering, M., and Schmidhuber, J. 1998. Efficient model-based exploration. In Pfeiffer, R.; Blumberg, B.; Meyer, J. A.; and Wilson, S. W., eds., *Proceedings of the Fifth International Conference on Simlation of Adaptive Behavior*.