Reusing Old Policies to Accelerate Learning on New MDPs

Daniel S. Bernstein
{*bern@cs.umass.edu*}
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

April 7, 1999

Abstract

We consider the reuse of policies for previous MDPs in learning on a new MDP, under the assumption that the vector of parameters of each MDP is drawn from a fixed probability distribution. We use the options framework, in which an option consists of a set of initiation states, a policy, and a termination condition. We use an option called a *reuse option*, for which the set of initiation states is the set of all states, the policy is a combination of policies from the old MDPs, and the termination condition is based on the number of time steps since the option was initiated. Given policies for m of the MDPs from the distribution, we construct reuse options from the policies and compare performance on an $m+1$st MDP both with and without various reuse options. We find that reuse options can speed initial learning of the $m+1$st task. We also present a distribution of MDPs for which reuse options can slow initial learning. We discuss reasons for this and suggest other ways to design reuse options.

1 Introduction

Recently, researchers in the field of artificial intelligence have begun to stress the importance of *life-long learning* (Nilsson, 1995; Thrun, 1996). Much of machine learning up until now has been geared towards learning solutions for single problems. However, most researchers would agree that it would be more useful to have systems that can solve several problems over time,

using the knowledge obtained from previous problem instances to guide in learning on new problems.

As a concrete example, consider an automated house. This is a house that automatically performs functions for the family residing in it. These functions could include preparing meals, turning lights on and off at appropriate times, taking out the garbage, or vacuuming the carpets. The house may have a learning component that allows it to tailor itself to the particular family inhabiting it. It can be viewed as a learning agent that adjusts its behavior based on feedback from the family members. Since different families have different needs, every time a new family moves into the house, the learning agent faces a new environment, i.e., a new task. The agent could start the learning process anew every time a new family moves into the house. However, this would be inefficient because some behaviors will be useful for almost any family. For instance, it is usually good for the lights of a given room to be turned off when no one is in the room. What is required is a way for the agent to *learn how to learn*, by generalizing across tasks, while still being able to exploit the specifics of each task. Besides the automated house example, other possible applications of knowledge transfer across learning tasks include a robot factotum that performs various tasks in an office building (this is Nilsson's, 1996, challenge), a space exploration robot that goes on missions to different planets, or a chess player that faces different opponents.

We will discuss the issue of performing multiple tasks in the realm of reinforcement learning (RL). An RL task is typically modeled as a Markov decision process (MDP), and most RL algorithms are aimed at finding a good partial policy for the accessible states of an MDP. Given a set of MDPs, the already established techniques can be used to learn a policy for each MDP independently. However, if there is some overlap between good policies for the MDPs, we would like our agent to discover this "structure" and use it to accelerate future learning.

In this paper, we consider fixed *distributions* of MDPs. Policies for some of the MDPs in the distribution are combined and can be used to accelerate learning on a new MDP drawn from the distribution. The combination of policies is used as the policy for an *option* (Sutton, Precup & Singh, 1998). We will refer to our constructed options as *reuse options* because they reuse old policies. One benefit of our approach is that it requires only policy information from the old MDPs and nothing else. We do not care how the policies were obtained in the first place. They could have been obtained using supervised learning or even hand-coded by the system designer. Furthermore, we put no special restrictions on the distribution

from which the MDPs are drawn. Another advantage is that the reuse option does not put any constraints on what can be learned in the new MDP. The option can be used often to direct exploration during the initial phases of learning and can then be used less frequently when the agent has discovered structure specific to the task at hand.

The issue of solution reuse across multiple tasks has been considered in the supervised learning literature (Thrun, 1996; Baxter, 1997), and is starting to be considered in RL (Thrun & Schwartz, 1995; Boyan, 1998; Hauskrecht et al., 1998; Parr, 1998; Ryan & Pendrith, 1998; Sutton, Precup & Singh, 1998; Van Roy, 1998; Perkins & Precup, in prep.). However, this work is still in its early stages, and few experiments designed explicitly to test across-task transfer have been run.

The structure of the paper is as follows. First we discuss RL, options, and an algorithm for learning with options. In Section 3 we present reuse options. In Section 4 we describe the domains for our experiments. The two types of experiments along with their results are presented in Section 5. In Section 6 we present an example in which reuse options actually retard learning. Finally, in Section 7 we discuss directions for future research, including ways to adapt our techniques to more difficult problems.

2 Reinforcement Learning

2.1 Markov Decision Processes (MDPs)

In the RL framework (Sutton & Barto, 1998), a learning *agent* interacts with its *environment* at some discrete time scale $t = 0, 1, 2, 3, \ldots$. At each time step, the agent perceives the state of the environment, $s_t \in \mathcal{S}$, and chooses a primitive action, $a_t \in \mathcal{A}_{s_t}$. One time step later, the environment produces a numerical reward, $r_{t+1} \in \mathbb{R}$, and a next state, s_{t+1}. The union of the action sets is denoted by $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$. If $\mathcal{S}$ and $\mathcal{A}$ are finite, then the environment's transition dynamics are modeled by state-transition probabilities and expected rewards:

$$\begin{aligned} p^a_{ss'} &= \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \text{and} \\ r^a_{ss'} &= E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}, \end{aligned}$$

for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}_s$. The agent learns a *policy*, $\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$, which is a mapping from states to probabilites of taking each action. The agent's objective is to maximize the expected discounted future reward. This expected reward starting from a state s and thereafter following policy π is

called the *value* of state s and is expressed formally as:

$$V^{\pi}(s) = E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; s_t = s, \pi\right\},$$

where $\gamma \in [0, 1)$ is a *discount-rate* parameter. We also denote the *action-value* function, which computes the expected discounted future reward starting from a state s, taking an action a, and thereafter following π:

$$Q^{\pi}(s, a) = E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; s_t = s, a_t = a, \pi\right\}.$$

For this paper, it will be useful to have a notion of a *most stochastic* policy that maximizes exploration while still choosing actions that are greedy with respect to a value function. Under a most stochastic policy, if there are ever two actions that are both optimal with respect to the value function, they will each have equal probabilities of being taken. Formally, most stochastic policies always satisfy the following equation:

$$\pi(s, a) = \begin{cases} \frac{1}{n_s} & \text{if } a = \arg\max_{a \in \mathcal{A}_s} Q^{\pi}(s, a) \\ 0 & \text{otherwise,} \end{cases}$$

where $n_s = \left|\{a \in A_s | a = \arg\max_{a \in \mathcal{A}_s} Q^{\pi}(s, a)\}\right|$. Note that every action-value function has associated with it exactly one most stochastic policy. Since each MDP has exactly one optimal action-value function, each MDP has exactly one most stochastic optimal policy.

2.2 Distributions of MDPs

Here we define a distribution of MDPs over a finite state set S and finite action sets $\mathcal{A}_s, s \in \mathcal{S}$. Each MDP can be described by its state-transition probabilities $p^a_{ss'}$ and expected rewards $r^a_{ss'}$. (We chose to ignore other information about the distribution of rewards.) Thus, an MDP can be viewed as a vector with $2 \cdot \sum_{s \in \mathcal{S}} |\mathcal{A}_s||\mathcal{S}|$ real-valued components. A distribution of MDPs is defined as a probability distribution over the space of these vectors.

2.3 Q-learning

One strategy for direct learning on finite MDPs is to approximate the optimal action-value function:

$$Q^*(s, a) = \max_{\pi} E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; s_t = s, a_t = a, \pi\right\}.$$

An algorithm called *one-step Q-learning* is used to approximate Q^* (Watkins, 1989). Using this method, after action a_t is taken at state s_t, yielding reward r_{t+1} and next state s_{t+1}, $Q(s_t, a_t)$ is updated by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t) \right].$$

fix this where α is a positive step-size parameter.

2.4 Options and Macro Q-Learning

The term *option* is used for the generalization of primitive actions to include temporally extended courses of action (Sutton, Precup & Singh, 1998). In this paper, we will be dealing with a special case of *semi-Markov options* in which the option "times out", i.e., terminates after some fixed period of time has elapsed. These options consist of three components: a policy $\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$, a time limit $\beta \in \mathbb{N}^+$, and an initiation set $\mathcal{I} \subseteq \mathcal{S}$. An option $\langle \mathcal{I}, \pi, \beta \rangle$ is available in state s if and only if $s \in \mathcal{I}$. If the option is used, the actions are selected according to π for β time steps. The agent can select another option when a given option terminates.

We note that a primitive action a is a special case of an option as defined above. Each action corresponds to an option that is available exactly when a is available ($\mathcal{I} = \{s : a \in \mathcal{A}_s\}$), that always lasts exactly 1 step ($\beta = 1$), and that always selects a ($\pi(s, a) = 1, \forall s \in \mathcal{I}$). Therefore, we can view the agent as choosing entirely among options.

The concept of an action-value function generalizes naturally to an *option*-value function. We define $Q^*(s, o)$ to be the maximal expected return given that the agent starts in state s and takes option o. As in the case of primitive actions, this leads to a Q-learning-type update rule. When an option terminates, its value is updated with the cumulative discounted reward obtained during its execution and the maximum option-value at the resulting state. This is expressed formally in the following equation:

$$Q(s_t, o_t) \leftarrow Q(s_t, o_t) + \alpha \left[r + \gamma^k \max_{o \in \mathcal{O}_{s_{t+k}}} Q(s_{t+k}, o) - Q(s_t, o_t) \right],$$

where $r = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k}$ and $k = \beta$ for option o. When these backups are performed in addition to those performed in standard one-step Q-learning, the resulting algorithm is called *Macro Q-learning* (McGovern, Sutton & Fagg, 1997).

3 Reuse Options

In this section, we describe reuse options and how they are used. The intuition behind reuse options comes from the simple idea that an agent's probability of behaving a certain way should be proportional to how often that behavior has been successful in the past. When a reuse option is being executed, the probability of an action being chosen at a given state is related to the number of policies in which that action is one of those taken at that state. We now formalize this notion. Given a set of policies $\pi_1, \pi_2, \ldots, \pi_m$, we let π_M be the policy formed by averaging all the policies:

$$\pi_M = \frac{1}{m}(\pi_1 + \pi_2 + \cdots + \pi_m).$$

We refer to π_M as the *mixed policy*.

To illustrate how the mixed policy extracts structure from a set of policies, consider the following example. We are given two policies for a domain in which $|\mathcal{S}| = 2$ and $|\mathcal{A}| = 3$:

$\pi_1 =$

	a_1	a_2	a_3
s_1	1	0	0
s_2	$\frac{1}{2}$	$\frac{1}{2}$	0

and $\pi_2 =$

	a_1	a_2	a_3
s_1	0	1	0
s_2	0	$\frac{1}{2}$	$\frac{1}{2}$

.

The resultant mixed policy is shown below:

$\pi_M =$

	a_1	a_2	a_3
s_1	$\frac{1}{2}$	$\frac{1}{2}$	0
s_2	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$

.

From state s_1, action a_3 is never chosen by either of the policies. Consequently, it is never taken from s_1 while the mixed policy is being executed. From state s_2, all actions have a non-zero probability of being chosen in at least one of the policies. However, a_2 is optimal in both policies, while the other two actions are not. This is taken into account in the mixed policy, in which all actions have a non-zero probability of being chosen, but a_2 is given a higher probability than the other actions.

A *reuse option*, $\langle S, \pi_M, n \rangle$, is formed from the mixed policy, where $n \in \mathbb{N}^+$ is the option time-limit. Reuse options execute π_M for exactly n time steps, and can be initiated from any state. Given a reuse option, Macro Q-learning can be used, with an option set consisting of all the primitive actions plus the reuse option.

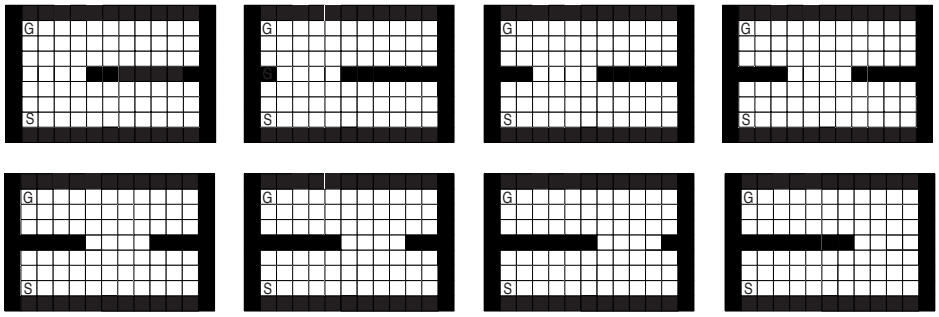

Figure 1: The eight gridworlds contained in the finite distribution.

4 Domains

4.1 Gridworld

In order to clearly demonstrate our techniques, we first consider a simple domain in which an agent must find the shortest path from a single starting position to a single goal on a grid. The states are the cells on the grid, and there are four primitive actions: **up**, **down**, **left**, and **right**, which have a stochastic effect. With probability 0.9, the agent moves in the intended direction, and with probability 0.1, it is equally likely to move in each of the other three directions. If an action takes the agent into a wall, then the agent does not move. A reward of 0 is obtained at every time step, except when the action taken in that time step leads to the goal, in which case the reward is +1. In order to ensure that the aim is to reach the goal as quickly as possible, there is a discounting factor ($\gamma = 0.9$). This task is episodic, with a new episode starting whenever the goal state is reached. The start state is in the lower-left corner of the grid, the goal state is in the upper-left corner, and a wall separates the upper and lower halves of the grid. There is a single opening in the wall, and it is the position of this opening that determines the MDP that the agent faces. Thus, we have a finite distribution consisting of 8 MDPs. The gridworlds in the distribution are shown in Figure 1. Note that there is significant overlap in optimal policies for the MDPs in the distribution. For instance, the optimal actions for the entire top part of the grid are the same for each task. Also, the action **down** is not optimal from any of the states that are above or below the wall in any of the gridworlds.

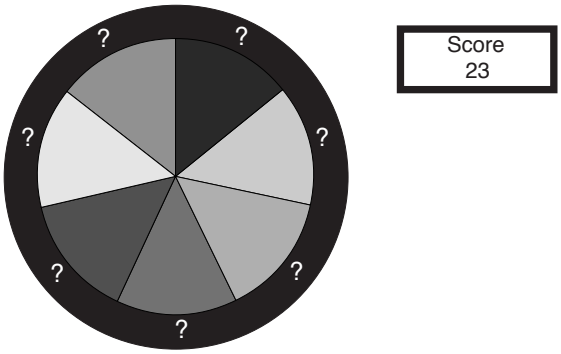

Figure 2: RL Darts.

4.2 RL Darts

The second domain we used for experiments was the RL Darts game. In this version of darts, a player starts with a score of zero and throws darts at the board, one at a time. The board is divided into seven wedges, with distinct point values ranging from -3 to 3 (see Figure 2). Unlike in regular darts, the point values are not displayed next to the wedges. Each time a dart is thrown, the point value corresponding to the wedge that is hit is added to the player's score (for simplicity, we do not allow the score to drop below zero). The object of the game is to reach a target score, T, exactly in as few throws as possible. Again, unlike in regular darts, the target score is not known to the player. If a throw ever causes the score to go above T, the score is reset to its previous value.

RL Darts can be formulated as a finite MDP in the following way: The states are the scores $0, 1, \ldots, T$. There are seven primitive actions, corresponding to the wedge that the player chooses to aim at. These actions have a stochastic effect. With probability 0.75, the player hits the wedge that she is aiming at, and with probability 0.25, the player is equally likely to hit any of the other wedges. There is a reward of 0 for every time step, except for the one during which the target score is reached, in which case the reward is 1. There is a discounting factor $\gamma = 0.9$. The task is episodic, with an episode corresponding to a complete game. If we vary the target score, T, from 1 to 60, we form a finite distribution consisting of 60 MDPs. Here the policy overlap may not be as obvious as in the gridworld domain. In particular, there is no state-action pair for which $\pi^c_M(s, a) = 1$. However, there is still invariance across the tasks. For instance, the optimal action from state 0 is to throw at the $+3$ wedge in all but two of the games in the

distribution (the games for which $T = 1$ or $T = 2$). Also, it is never optimal to throw at the 0, -1, -2, or -3 wedges from a reachable state. There are, however, unreachable states in the MDPs (e.g. a score of 31 when $T = 30$), at which all actions are considered optimal.

5 Experiments

In order to perform experiments, we had to choose the policies to use in the construction of the reuse options. For our initial tests, we tried to construct what would intuitively be considered a "best possible" reuse option. For a finite distribution of MDPs (which is all we consider in our experiments), this is called a *complete reuse option*. Suppose the distribution contains m MDPs. (When we say that a finite distribution "contains" m MDPs, we mean that each of the m MDPs is equally likely to be drawn from the distribution.) Then the complete mixed policy, π^c_M, is the average of the most stochastic optimal policies of the m MDPs, and the n-step complete reuse option is $\langle S, \pi^c_M, n\rangle$.

Of course, it is not realistic to assume that the agent has a policy for *every* MDP in the distribution. Thus, we define an *incomplete reuse option* to be an option whose policy is the average of the most stochastic optimal policies of some k MDPs drawn from the distribution. We note that it is still not reasonable to assume that the agent has access to an *optimal* policy of any of the MDPs, and future work will include tests in which the mixed policy is formed *on-line* using policies generated by an algorithm such as Q-learning.

5.1 Experiment 1

The first experiment was designed to test the effect of reuse options on initial exploration. Our hypothesis was that for some distributions of MDPs, exploration with a reuse option would be biased to get the agent more reward than if it did not have a reuse option. In this experiment, there was no learning. We drew MDPs from the distribution and measured the performance of the agent as it chose each of the options in its set with equal probability at each choice point. The following four sets of options were compared:

1. Primitive actions
2. Primitive actions with 1-step complete reuse option, $\langle \mathcal{S}, \pi^c_M, 1\rangle$

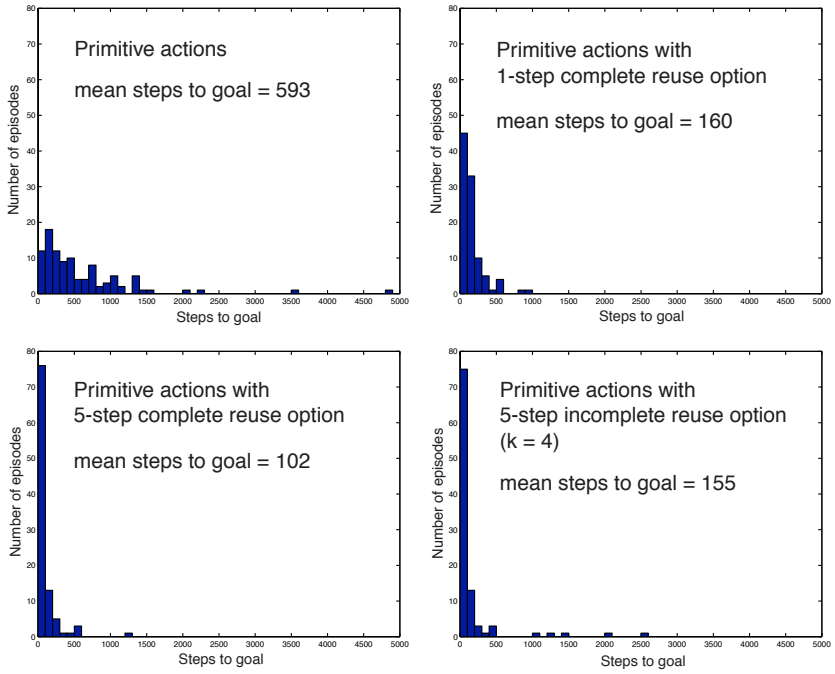

Figure 3: Results of Experiment 1 in the gridworld domain.

3. Primitive actions with 5-step complete reuse option, $\langle \mathcal{S}, \pi_M^c, 5 \rangle$

4. Primitive actions with 5-step incomplete reuse option, $\langle \mathcal{S}, \pi_M, 5 \rangle$ (with $k = 4$)

5.1.1 Results of Experiment 1 for Gridworld

In this experiment, we drew gridworlds randomly from the distribution and recorded the number of steps for the agent to get from the start state to the goal state. As described above, four different option sets were compared. The test was repeated for 100 randomly drawn gridworlds. Due to the nature of the domain, our distributions of total steps were skewed. These distributions, along with their means, are shown as histograms in Figure 3. We see that for all of the reuse options, the addition of the option leads to improved performance. Furthermore, the results suggest that increasing the time limit of a reuse option can increase its effectiveness.

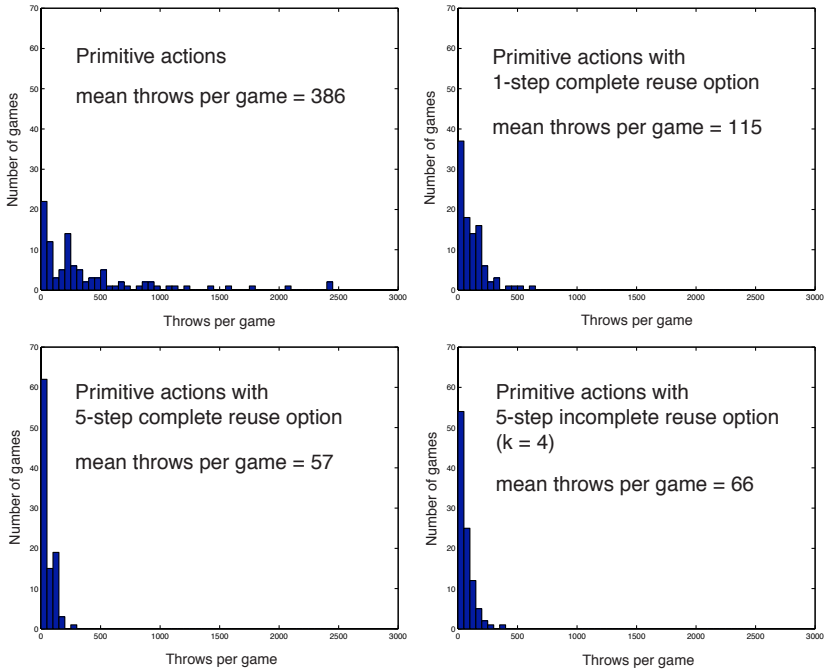

Figure 4: Results of Experiment 1 in the RL darts domain.

5.1.2 Results of Experiment 1 for RL Darts

We drew games from the distribution and recorded the number of turns to reach the target score for an agent choosing randomly among the options in its set. We compared the four different option sets, and repeated the test for 100 randomly drawn target scores. Again, our distributions of total throws were skewed. These distributions, along with their means, are shown as histograms in Figure 4. The results are essentially the same as in the corresponding gridworld experiment.

5.2 Experiment 2

In this experiment, we examined the effect of the reuse option on time to convergence. The results of Experiment 1 told us what to expect in a first episode, and this experiment helped us to see what happens afterwards, as the agent learns from experience. We drew MDPs from the distribution and compared the following algorithms for solving them:

1. Q-learning

2. Macro Q-learning with the 1-step complete reuse option, $\langle \mathcal{S}, \pi_M^c, 1 \rangle$

3. Macro Q-learning with the 5-step complete reuse option, $\langle \mathcal{S}, \pi_M^c, 5 \rangle$

4. Macro Q-learning with the 5-step incomplete reuse option, $\langle \mathcal{S}, \pi_M, 5 \rangle$ (with $k = 4$)

To ensure that any difference in performance depended on having the *reuse* option and not just any option, we introduced *random* options. During the execution of the random option, all primitive actions are taken with equal probability. Formally, the n-step random option is $\langle \mathcal{S}, \pi_D, n \rangle$, where

$$\forall s \in \mathcal{S}, \pi_D(s,a) = \begin{cases} \frac{1}{|\mathcal{A}_s|} & \text{if } a \in \mathcal{A}_s \\ 0 & \text{otherwise.} \end{cases}$$

We tested the following algorithm:

5. Macro Q-learning with the 5-step random option, $\langle S, \pi_D, 5 \rangle$

Finally, for comparison, we also tested the following, non-learning, algorithm:

6. Using the complete mixed policy as a fixed policy, with no learning

Whenever we used either Q-learning or Macro Q-learning, we adopted ϵ-greedy action selection. That is, given the current estimates $Q(s,o)$, let $o^* = \arg\max_{o \in \mathcal{O}_s} Q(s,o)$ denote the best-valued option (with ties broken randomly). Then the policy used to select options was

$$\mu(s,o) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{O}_s|} & \text{if } o = o^* \\ \frac{\epsilon}{|\mathcal{O}_s|} & \text{otherwise,} \end{cases}$$

for all $s \in \mathcal{S}$ and $o \in \mathcal{O}_s$. We set $\epsilon = 0.1$ in all cases. For each experiment, we used $\alpha = 0.1$. Action-values were always set to zero at the start of a new task. This was done to ensure a large amount of exploration. In both of our domains, the agent does not receive any reward until it reaches a goal state. Thus, it takes at least a few episodes for values to propagate back from the goal state. Since it takes some time for values to change, we force a significant amount of exploration at the beginning of learning. This seems artificial, but recall that we are testing how the reuse options bias exploration, not how to obtain adequate exploration.

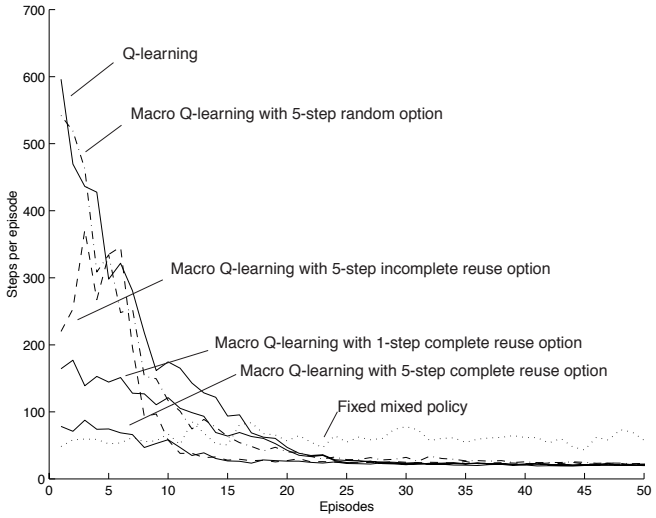

Figure 5: Results of Experiment 2 in the gridworld domain. Each line is averaged over 100 runs.

5.2.1 Results of Experiment 2 for Gridworld

The aim of this experiment was to test the performance of the agent using the six different learning algorithms described above on gridworlds drawn from the distribution. In this case, good performance meant a short walk from the start state to the goal state. In Figure 5, we plot time to goal versus episode number. The data are averaged over runs on 100 randomly drawn gridworlds. We see that one-step Q-learning is the slowest to converge. Macro Q-learning with just a 1-step complete reuse option converged much faster than without a reuse option. And, Macro Q-learning with a 5-step complete reuse option was even faster. The incomplete reuse option gave improved convergence as well. As expected, Macro Q-learning with the 5-step random option did not converge much faster than regular one-step Q-learning. Using the complete mixed policy as a fixed policy gave good initial performance, but it was eventually overtaken by all of the learning algorithms, which could exploit structure specific to the particular task being solved.

5.2.2 Results of Experiment 2 for RL Darts

Figure 6 shows a graph of the number of throws to reach the target score versus game number for a randomly drawn target score. The data are an

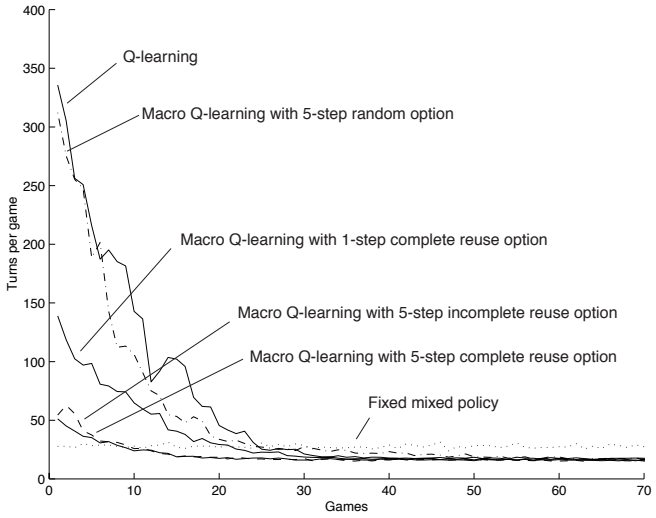

Figure 6: Results of Experiment 2 in the RL darts domain. Each line is averaged over 100 runs.

average over 100 runs with randomly chosen target scores. Again, the results obtained in this domain were similar to those obtained in the gridworld domain.

6 A Counterexample: Moving Target

In this section we describe a distribution of MDPs for which the addition of the reuse option to the set of options actually biases exploration in a negative way. In this domain, the agent operates a stationary gun, which it must fire at a target that repeatedly moves in a line from one point to another (see Figure 7). Furthermore, the agent knows only the absolute coordinates of the target at any given time. This can be modeled as a simple MDP. The state is the absolute position of the target, and the actions are whether or not to fire the gun. If the agent fires the gun at the same time that the target is directly in front of it, it receives a reward of $+1$. If the gun is fired at any other time, the agent receives a reward of -0.01. This is a continuing task with a discounting factor $\gamma = 0.9$. By varying the position of the gun, we can form a distribution of MDPs.

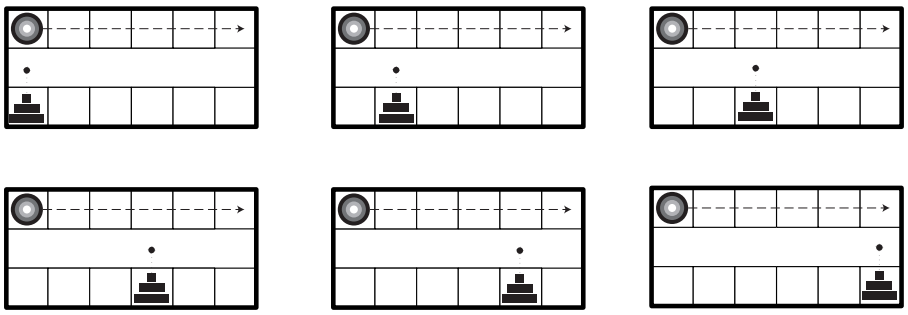

Figure 7: The moving target task distribution.

6.1 Experiment

We tested to see the effect of the optimal reuse option on exploration in this domain. As in Experiment 1 above, we compared (1) primitive actions, (2) primitive actions with the 1-step complete reuse option, and (3) primitive actions with the 5-step complete reuse option. For simplicity, we did not test any incomplete reuse options. A task was drawn from the distribution, and for 1000 time steps the agent chose randomly among the options in its option set. This was repeated for 100 randomly drawn tasks and the means and standard deviations of the total reward obtained are shown in Table 1. In this domain, the option set that included the 5-step complete reuse option yielded the least reward, and the set containing the 1-step complete reuse option followed. In this case, it was better not to have either of the complete reuse options. The distributions of total rewards for the three option sets were approximately normal, so we applied a one-tailed ttest to determine the significance of the results. Let μ_i be the true mean of the probability distribution of total rewards for the ith option set. We found that $\mu_1 > \mu_2$ with $p < 0.001$ and $\mu_2 > \mu_3$ with $p < 0.001$.

To understand these results, we must analyze the complete mixed policy for the distribution. For any given state, the most stochastic optimal policy for exactly one of the MDPs is to fire with probability 1, while the most stochastic optimal policy for the rest of the MDPs is to abstain from firing with probability 1. Thus, at every state, the complete mixed policy assigns a probability of $\frac{1}{6}$ to **fire** and a probability of $\frac{5}{6}$ to **don't fire**. Therefore, during the execution of the reuse option, the agent is less likely to fire than if it was simply choosing each of the two primitive actions with equal probability. This is not a good strategy. At the start of learning, the agent is better off firing frequently because the benefit of quickly finding the target

Options available	**Mean Reward Per Run**
Primitive actions	13.1 ± 2.6
Primitive actions with 1-step complete reuse option	9.6 ± 2.7
Primitive actions with 5-step complete reuse option	4.8 ± 2.9

Table 1: Results of the experiment in the moving target domain. Means and standard deviations computed for 100 runs.

outweighs the small penalties incurred for firing and missing the target. The complete mixed policy does not coincide with this reasoning because it does not take value information into account. In the optimal policies of the individual tasks, shooting to get a reward of $+1$ is equally as important as not shooting to avoid a penalty of -0.01. If value information is taken into account in the construction of the mixed policy, this problem can be avoided. This suggests an extension to the reuse option in which old value information (when available) is considered along with old policy information.

7 Discussion

In our experiments, we only considered MDPs with few states. This is acceptable as a first step, but RL algorithms must be able to "scale up" to address real-world problems, which can have very large or even infinite state spaces. With state abstraction, we have an implicitly represented policy, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. It is possible to simulate the average of a set of implicitly represented policies by simply choosing the output of each with equal probability. Another type of abstraction in RL is temporal abstraction. With temporal abstraction, the agent can have multiple temporally-extended options available to it. In this case, the reuse option can simply be one option among those available.

We note that the extent to which the reuse option can bias initial exploration depends partly on the total number of options available to the agent. As the total number of options increases, the rate at which the reuse option is chosen decreases. It may be useful to extend our algorithm to allow the frequency that the reuse option is chosen in the initial part of learning to be

independent of the total number of options available. On the other hand, we may not want to treat the reuse option as special. Regardless, the more general question of how the number of options available affects learning is an important one.

Also, it is likely that the optimal time limit for the reuse option is domain-dependent, which makes β another parameter that must be chosen by the system designer. For future research, we would like to automate the assignment of a value to β, or change the reuse option altogether, so that termination of the option is state-dependent.

One way to extend the reuse option is to allow its mixed policy to be a *weighted* average of old policies. The weights could be proportional to the "confidence" that the agent has in each policy. For instance, a policy that results from 100,000 time steps of Q-learning will usually be more useful than a policy that results from 1,000 steps of Q-learning and should thus be given more weight. Also, if we assume that the distribution of tasks is not fixed, but instead varies with time, we may want to give more weight to policies that were derived more recently.

Finally, we assume in this paper that the agent is told every time its environment changes. In some cases (e.g. those mentioned in the Introduction), this assumption is realistic. However, sometimes the environment simply changes without notice. In future work, we would like to find ways for our learning agents to detect changes in the environment and find the invariance across these changes. Ideally, we would like autonomous agents that can perform multiple tasks in the world and can go long periods of time without requiring more input from the system designer.

Acknowledgements

The author wishes to thank Andrew Barto, Anders Jonsson, Amy McGovern, Doina Precup, and all the members of the ANW group for their help. This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation. This research was also supported by a grant from the Air Force Office of Scientific Research, Bolling AFB (AFOSR F49620-96-1-0254).

References

Baxter, J. (1997). Theoretical models of learning to learn. In S. Thrun & L. Pratt (Eds.), *Learning to Learn*. Kluwer.

Boyan, J. A. (1998). *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T. & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*.

McGovern, A., Sutton, R. S. & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In *Proceedings of the 1997 Grace Hopper Celebration of Women in Computing* (pp. 13–18).

Nilsson, N. J. (1995). Eye on the prize. *AI Magazine*, *16(2)*, 9–17.

Parr, R. (1998). Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence*.

Perkins, T. J. & Precup, D. (in preparation). Using options for knowledge transfer in reinforcement learning.

Ryan, M. R. K. & Pendrith, M. D. (1998). RL-TOPs: An architecture for modularity and re-use in reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 481–487). Morgan Kauffman.

Selman, B., Brooks, R. A., Dean, T., Horvitz, E., Mitchell, T. M. & Nilsson, N. J. (1996). Challenge problems for artificial intelligence. In *Proceedings of AAAI-96, National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D. & Singh, S. (1998). Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. Technical Report 98-74, University of Massachusetts, Amherst.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Proceedings of Advances in Neural Information Processing Systems 8* (pp. 640–646). Cambridge, MA: MIT Press.

Thrun, S. & Schwartz, A. (1995). Finding structure in reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems 7* (pp. 385–392). Cambridge, MA: MIT Press.

Van Roy, B. (1998). *Learning and Value Function Approximation in Complex Decision Processes.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Watkins, C. (1989). *Learning from Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, England.