Macro-Actions in Reinforcement Learning: An Empirical Analysis

Amy McGovern and Richard S. Sutton
$\{amy|rich@cs.umass.edu\}$
Computer Science Department
University of Massachusetts, Amherst
Amherst, MA 01003

August 19, 1998

Abstract

Several researchers have proposed reinforcement learning methods that obtain advantages in learning by using temporally extended actions, or *macro-actions*, but none has carefully analyzed what these advantages are. In this paper, we separate and analyze two advantages of using macro-actions in reinforcement learning: the effect on exploratory behavior, independent of learning, and the effect on the speed with which the learning process propagates accurate value information. We empirically measure the separate contributions of these two effects in gridworld and simulated robotic environments. In these environments, both effects were significant, but the effect of value propagation was larger. We also compare the accelerations of value propagation due to macro-actions and eligibility traces in the gridworld environment. Although eligibility traces increased the rate of convergence to the optimal value function compared to learning with macro-actions but without eligibility traces, eligibility traces did not permit the optimal policy to be learned as quickly as it was using macro-actions.

1 Introduction

Many problems in artificial intelligence are too large to be solved practically at the level of the most primitive actions. One strategy for overcoming this difficulty is to combine smaller actions into larger, temporally-extended actions, thus reducing the effective length of the solutions. For example, Korf (1985), Laird et al. (1986), and Iba (1989) have studied the use of *macro-operators*, or fixed sequences of actions treated as single larger actions. They and others have shown that searching with macro-operators can yield solutions much more quickly than when search is restricted to primitive actions.

The work described in this paper is part of an ongoing effort to understand how we can achieve something similar in the realm of reinforcement learning and Markov decision processes. This framework is appealing because the temporally extended actions are not limited to open-loop sequences, but can be closed-loop subpolicies that are conditional on environmental events.

Many researchers have explored issues related to temporally extended actions, modularity and hierarchy in reinforcement learning (e.g., Lin, 1993; Kaelbling, 1993; Dayan & Hinton, 1993; Singh, 1992). Recently, several researchers have focused on a representation of temporally extended actions as the combination of a policy and a termination condition (e.g., McGovern, Sutton, & Fagg 1997; Parr & Russell, 1997; Precup, Sutton, & Singh, 1998; Dietterich, 1998; Hauskrecht et al., 1998; Huber & Grupen, 1997). Some of this research has extended the theory of reinforcement learning with temporally extended actions and some has proposed new methods for learning and planning with such actions. In this paper, we use the term *macro-actions* to refer to temporally extended actions, whereas Sutton, Precup, & Singh (1998) use the term *options*. Options may be either multiple step policies or primitive actions while macro-actions are restricted to temporally extended actions. This paper focuses on analyzing the effects of macro-actions in accelerating (or decelerating) learning.

2 Reinforcement Learning and Macro-actions

Reinforcement learning is a collection of methods for approximating optimal solutions to stochastic sequential decision problems (Sutton & Barto, 1998). A reinforcement learning system does not not require a teacher to specify correct actions. Instead, the learning agent tries different actions and observes the consequences to determine which are best. More specifically, in the reinforcement learning framework, a learning *agent* interacts with an *environment* at some discrete time scale $t = 0, 1, 2, 3, \ldots$. At each time step t, the environment is in some *state*, s_t. The agent chooses an action, a_t, which causes the environment to transition to state s_{t+1} and to emit a reward, r_{t+1}. The next state and reward depend only on the preceding state and action, but they may depend on it in a stochastic fashion. The objective is to learn a (possibly stochastic) mapping from states to actions called a *policy*, π, which maximizes the cumulative discounted reward received by the agent. More precisely, the objective is to choose action a_t, for all $t \geq 0$, so as to maximize the expected *return*, $\mathrm{E}\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}\right\}$, where $\gamma \in [0, 1)$ is a discount-rate parameter.

A common solution strategy is to approximate the *optimal action-value function*, Q^*, which maps state-action pairs (s, a) to the maximal expected return that can be obtained starting in state s and taking action a:

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

In this paper we use a method to approximate Q^* known as *one-step tabular Q-learning* (Watkins, 1989). In this method, the approximation to Q^* is represented by a table with an entry, $Q(s, a)$, for each state-action pair. After each transition from state s_t to state s_{t+1}, under action a_t and with reward r_{t+1}, the estimated action value $Q(s_t, a_t)$ is updated by:

$$Q(s_t, a_t) \quad \leftarrow \quad Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)\right], \tag{1}$$

where α is a positive step-size parameter.

Macro-actions are policies with termination conditions. On each time step, the agent can choose either a macro-action or a primitive action, unless it is already executing a macro-action. Once the agent has chosen a macro-action, it selects the primitive actions in accordance with the macro-action's policy until the macro-action's termination condition is satisfied. For example, walking from the lab to the cafeteria could be a macro-action. This macro-action enables the walker to skip thinking or planning at the level of muscle movements or even at the level of gross body movements. If a pile of snow is encountered along the way, the walker can safely change the primitive actions of walking to keep from falling while still executing the macro-action of going to the cafeteria.

To provide for learning when to select macro-actions, we extend the notion of the optimal action-value function, Q^*, to include macro-actions. That is, for each state s and macro-action m we define a *macro value* $Q^*(s,m)$, as the maximal expected return given that we start in state s and take macro-action m. This definition naturally leads to an update rule: Upon each termination of a macro-action, its value is updated using the cumulative discounted reward received while executing the macro-action and the maximum value at the resulting state. More precisely, after a multi-step transition from state s_t to state s_{t+n} using macro-action m, the approximate action value $Q(s_t,m)$ is updated by:

$$Q(s_t,m) \quad \leftarrow \quad Q(s_t,m) + \alpha \left[r + \gamma^n \max_a Q(s_{t+n},a) - Q(s_t,m) \right], \tag{2}$$

where the max is taken over both actions and macro-actions, and

$$r \quad = \quad r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n}.$$

This is a discrete-time version of the *Semi-Markov Decision Process Q-learning* method studied by Bradtke & Duff (1996) and proven to converge by Parr (1998). The algorithm that we focus on in this paper performs update (2) as well as the conventional Q-learning update for each primitive action given by (1). We call the resulting algorithm *Macro Q-learning* (McGovern, Sutton, & Fagg, 1997).

3 Illustrative Example

As an illustration of the effects of macro-actions on learning, consider the two gridworld environments shown inset in Figure 1. The task in each case is to travel from the start state labeled "S" to the goal state labeled "G" as quickly as possible. Each gridworld is 11 states long, 11 states high, and is surrounded by four walls. There are four primitive actions, `up`, `down`, `right`, and `left`, which have stochastic effects: 75% of the time each action causes motion in the named direction, and 25% of the time each action causes a motion in one of the three other directions. In any event, if the movement would take the agent into a wall, then the agent remains in the same state. There are also four macro-actions: `macro-up`, `macro-down`, `macro-right`, `macro-left`. Each macro-action takes the corresponding primitive-action as many steps as needed (possibly zero) until the agent

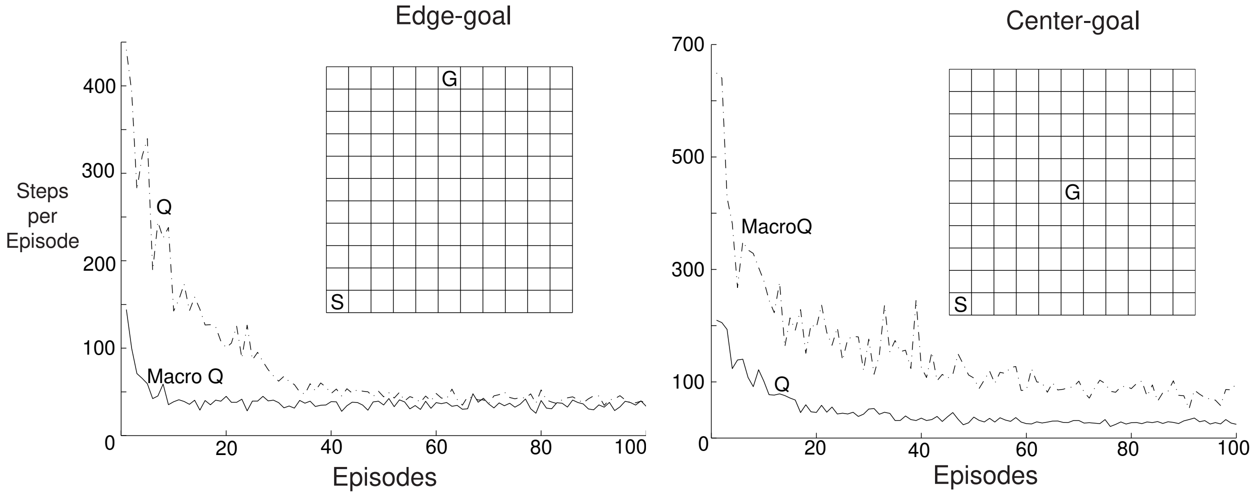

Figure 1: *Comparison of Q-learning and Macro Q-learning on two gridworld navigation tasks. Each line is averaged over 30 runs.*

reaches a wall. The macro-action terminates just before hitting the wall. Note that one gridworld has the goal at the edge of the grid whereas the other has the goal in the center.

We applied Q-learning and Macro Q-learning to both environments. In both cases, actions were selected according to the ϵ-greedy method (Sutton & Barto, 1998). The learning parameters were $\alpha = 0.05$, $\epsilon = 0.05$, and $\gamma = 0.9$. Figure 1 shows the number of primitive steps used to transition from the start state to the goal state for 100 episodes of each algorithm. An episode consists of one trajectory from the start state to the goal state. Each data point is an average over 30 runs, where a run is a fixed number of episodes starting with a different random seed. In the edge-goal environment, Macro Q-learning converged to the optimal policy much faster (approximately five episodes) than Q-learning (approximately 50 episodes). However, in the center-goal environment, Q-learning converged much more quickly (approximately 40 episodes) than Macro Q-learning (greater than 100 episodes).

This experiment demonstrates the intuitive idea that macro-actions will sometimes help and sometimes hinder learning, depending on their appropriateness to the task. In the next two sections we isolate and evaluate two different hypotheses about how macro-actions affect the rate of learning. The first hypothesis is that macro-actions influence the exploratory behavior of the agent such that more relevant states are visited more often. The second hy-

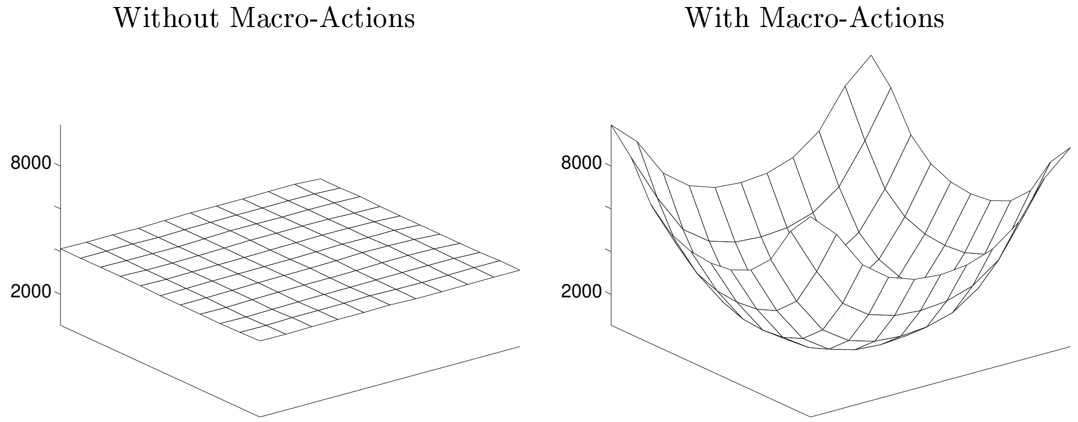

Figure 2: *State-visitation histograms for the gridworld environment when randomly selecting from the primitive actions (left) or both primitive and macro-actions (right).*

pothesis is that the macro-action backup propagates correct value information more widely and more rapidly. We analyze these two effects first in these gridworld environments and then in a larger simulated robot task.

4 Hypothesis 1: Effect on Exploration

We first consider the hypothesis that macro-actions bias the behavior of the agent to spend more time in relevant states, i.e., states that are closer to the goal. In the case of the gridworlds described above, we hypothesize that the macro-actions cause the learner to spend the majority of its time near the edges of the grid.

To examine this effect independently of learning effects, we measured how often each state was visited when primitive actions and macro-actions were selected at random. We used the same gridworlds as described above (Section 3) but with no goal state. Each agent started in the lower left hand corner state, chose an action at random, and transitioned to a neighboring state. This continued for 500,000 steps. Figure 2 shows two histograms indicating how often each state was visited on average when actions were selected randomly from the primitives (left panel) and from both the primitives and the macro-actions (right panel) for the 500,000 steps. These histograms average over 30 runs. In the case with only primitive actions, all states were visited equally often, whereas with the macro-actions the edge states were visited much more often than the other states.

This difference in exploratory behavior, independent of learning, explains part of the performance differences between Q-learning and Macro Q-learning observed in the experiment described in Section 3. When macro-actions were taken, the goal state in the edge-goal gridworld was visited on average about 5556 times out of 500,000 steps. With only primitive actions, this state was visited only about 4097 times. However, this difference does not seem large enough to fully explain the dramatic performance differences shown in Figure 1. Another possibility is that the Macro Q-learning algorithm may be more efficient at learning the value function than conventional Q-learning.

5 Hypothesis 2: Effect on Value Propagation

Our second hypothesis about the effect of macro-actions on learning is that they affect the rate at which correct action-values propagate through the state space. In one-step Q-learning, values propagate backwards one time step per backup. However, when backing up macro values, value information can propagate over several time steps. When a macro-action takes the agent to a good (or bad) state, the macro value for the state in which the macro-action was chosen is updated immediately with useful information even though that state may be many primitive actions away from the good state.

To examine the effect that macro-actions have on the rate of value propagation independent of behavior, we compared the propagation rate for Q-learning and Macro Q-learning when operating on exactly the same experience. This experience was generated by the random selection among both primitive and macro-actions as described in the experiment in Section 4. Each algorithm was applied separately to this experience. By applying each algorithm to the same behavior, we eliminate any effect due to state visitation differences, and the effect of macro-actions on value propagation can be seen more clearly.

Figure 3 shows two snapshots of the average value propagation for the two algorithms. The first row shows the values after the goal has been reached twice, and the second row shows the values after the goal has been reached 13 times. Each graph is the average over 30 runs. After the goal state had been reached only twice, Macro Q-learning, on average, had already learned non-zero values for states all the way back to the start state. This is

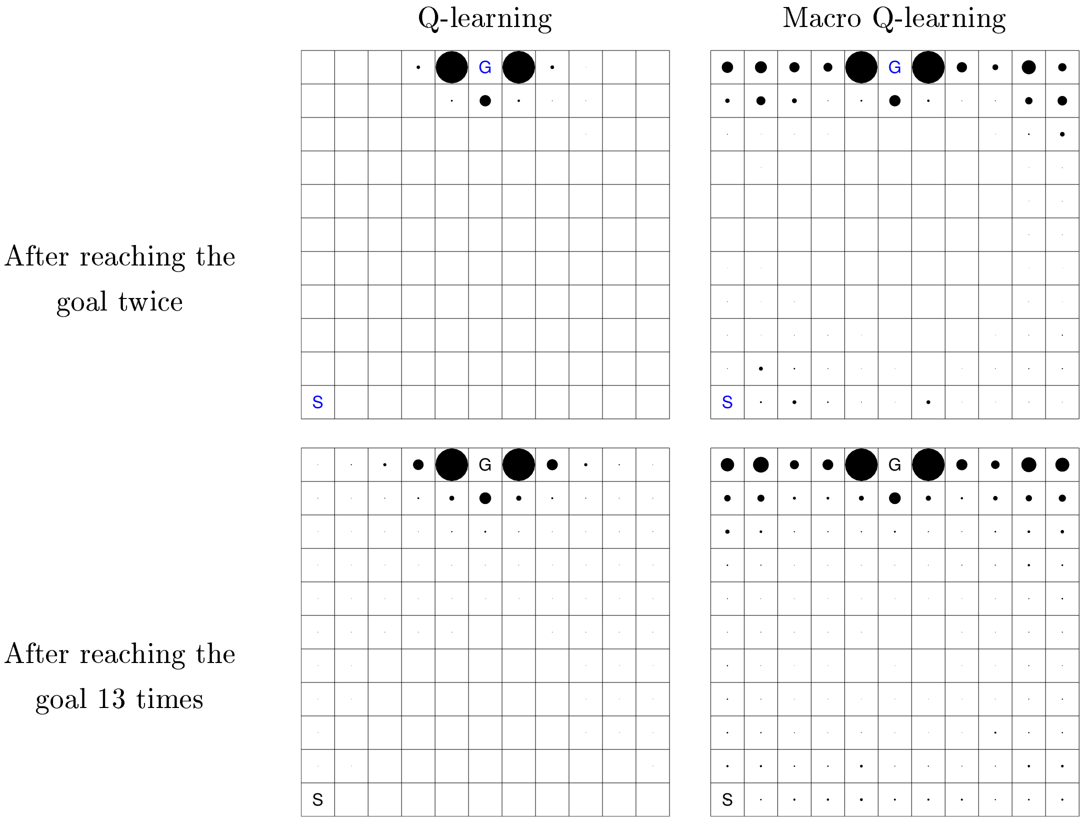

Figure 3: *Comparison of the propagation of state values by Q-learning and Macro Q-learning for the same behavior early and late during learning. Circle area is proportional to the maximum of the action values (Q-learning) or action and macro values (Macro Q-learning) in that state.*

shown in the first row of Figure 3. In fact, the greedy policy formed by evaluating Macro Q-learning's value function at this point was already effective in bringing it to the goal. In contrast, Q-learning had only learned a few action values, and its greedy policy was not effective in bringing it from the start state to the goal state. After reaching the goal 13 times, Macro Q-learning had good value information on all but 3 states in the average case, while Q-learning was missing values for 39 states. Also, the values were propagated differently by the two algorithms. While Q-learning spread the values backwards from the goal almost uniformly on average, Macro Q-learning first spread the information around the edges of the grid, and then into the center.

Clearly, adding macro values to the backup equation affects the propagation rate of value information. This can lead to faster convergence to the optimal policy. In the next section we discuss how this effect compares to the effect of eligibility traces.

6 Comparison with Eligibility Traces

Eligibility traces are a well-known mechanism for speeding value propagation in reinforcement learning. Each state-action pair is marked as eligible for backup with a trace indicating how recently it has been experienced. Then, on each step, the values of *all* state-action pairs are updated in proportion to their eligibility traces at the time. Because many recent state-action pairs may have non-zero traces, value information is propagated backwards many steps and may become accurate more quickly. In these experiments we use standard *replacing* eligibility traces (Singh & Sutton, 1996).

Figure 4 compares the performance of Q-learning and Macro Q-learning with the eligibility trace method known as Watkins's $Q(\lambda)$ (Watkins, 1989; Sutton & Barto, 1998) on the edge-goal gridworld, when processing the same random experience as used in the experiment described in Section 5. Again, by using a fixed behavior, we isolate the effect of the algorithm on value propagation from any effects of experience. For $Q(\lambda)$ we used a variety of values for the trace parameter, $\lambda \in [0, 1]$, which determines the duration of the traces, that is, how quickly they fade away, or how far back values propagate during a single episode. We used λ values to produce half-lives of 2, 4, 8, 16, 32, and 64 time steps, and also $\lambda = 1$, corresponding to infinite-length traces.

We evaluated $Q(\lambda)$ Macro Q-learning, and Q-learning in two ways. To evaluate the convergence to the optimal action-value function, we calculated the optimal action-value function, Q^*, using Dynamic Programming and summed the absolute differences between the optimal action-value function and the current learned value function after each time the goal was reached. To evaluate the optimal policy, we froze the value function and evaluated it greedily. This was done after each time the agent reached the goal state. Figure 4 shows the results of these experiments.

$Q(\lambda)$ with $\lambda > 0$ converged to the optimal policy faster than one-step Q-learning, but

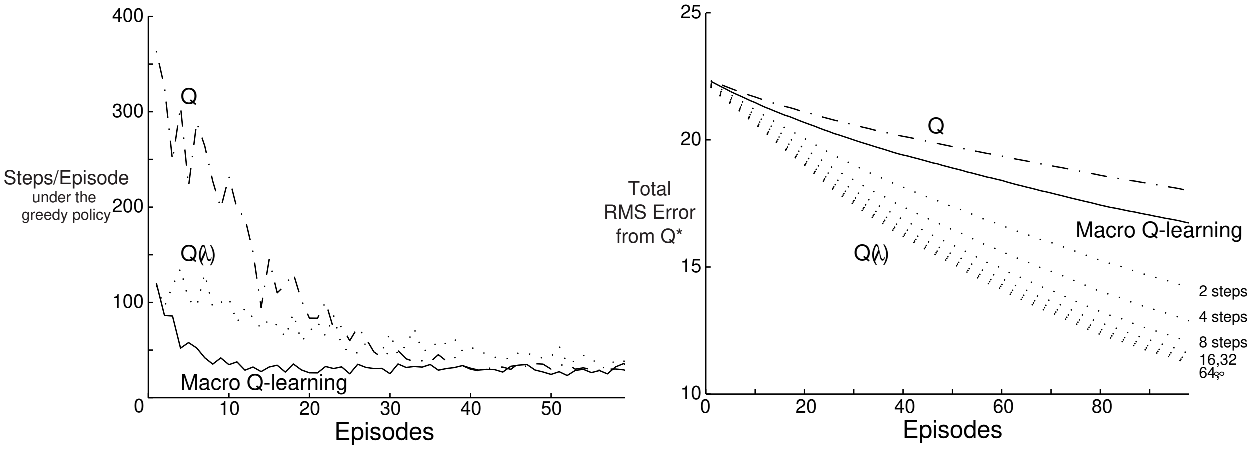

Figure 4: *Comparison with $Q(\lambda)$ on the edge-goal task, with experience held constant (random selection among both primitive and macro-actions). Averages over 30 runs.*

Macro Q-learning was faster still (Figure 4, left panel). However, $Q(\lambda)$ with $\lambda > 0$ converged to the optimal action values more quickly than either Q-learning or Macro Q-learning (Figure 4, right panel), and Macro Q-learning converged more quickly than one-step Q-learning. $Q(\lambda)$ learned the optimal value function more quickly because it disseminated the value information at an even more rapid rate than Macro Q-learning. However, it accomplished this at the cost of the policy. Macro Q-learning learned values for the edge states first (as shown in Section 5). Because these states are on the path from the start state to the goal state, Macro Q-learning learned correct action-values for the relevant states faster than $Q(\lambda)$ and was able to learn the optimal policy more quickly.

With appropriate macro-actions, convergence to the optimal policy and the optimal action-value function can be faster than learning without macro-actions. Combining eligibility traces with macro-actions may produce an even more efficient algorithm.

7 A Larger Illustration

As a larger illustration of these effects we used a continuous two-dimensional (x-y plane) simulated robot foraging task. The circular robot inhabits a world with two rooms, one door, and one food object as shown in Figure 5. Each room is 10 feet by 10 feet with a 3 foot wide doorway. The robot is able to discern which room it is in. The robot has simulated sonars to sense the distance to the nearest wall in each of five fixed directions, three forward and two back. The forward sonars are fixed at $0^o, 30^o$, and -30^o from the heading of the robot. The back sonars are at -135^o and 135^o from the robot's heading. The robot can also sense the direction and distance to the doorway from any point in the room, and to the food object if it is within 10 feet of the robot. When food comes within a smaller distance of the robot (2 feet), it is consumed and the agent receives a reward of $+1$ (otherwise the reward is zero). After the food is consumed, it re-appears in the middle of the room that the robot is *not* in. All experiments in this world use a starting position $(x, y) = (5, 1)$. The first piece of food starts in the same room as the robot.

The robot uses simple inertial dynamics, with friction and inelastic collisions with the walls. There are 13 possible primitive actions. On each step, the robot can either linearly accelerate in the direction in which it is oriented (to one of 3 positive and 3 negative degrees), apply an angular acceleration (to one of 3 positive and 3 negative degrees), or apply no acceleration at all. The available discrete linear and rotational accelerations are $[-0.03, -0.02, -0.01, 0.01, 0.02, 0.03]$. Two macro-actions are also available: `orient-to-door` and `forward-until-wall`. The former activates a PD (position and derivative) controller to turn the robot to face the doorway. The latter activates a PD controller to go forward unless the sonars indicate a wall nearby. The PD controllers for the macro-actions can only select from the set of available primitive actions. To do this, they round the continuous suggested move to the nearest available primitive action. Both PD controllers were critically damped. The `orient-to-door` controller used a position gain of 0.04 and a velocity gain of 0.4 while the `forward-until-wall` controller used a position gain of 0.0225 and a velocity gain of 0.3.

The robot's equations of the motion are as follows:

$$
\begin{aligned}
x_{t+1} &= x_t + v_t \cos\theta_t \\
y_{t+1} &= y_t + v_t \sin\theta_t \\
v_{t+1} &= (1-\mu)v_t + a_t \\
\theta_{t+1} &= \theta_t + \dot{\theta}_t \\
\dot{\theta}_{t+1} &= (1-\mu)\dot{\theta}_t + \ddot{\theta}_t
\end{aligned}
$$

Simulated Environment

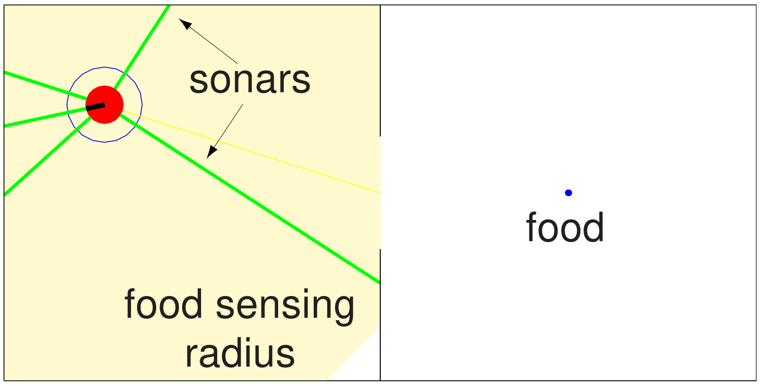

Figure 5: *The simulated robotic foraging task.*

where (x_t, y_t) are the global coordinates of the robot at time t, a_t is the linear acceleration, $\ddot{\theta}_t$ is the rotational acceleration, $\mu = 0.1$ is the coefficient of friction, $\theta_t \in [0, 2\pi]$ is the robot's absolute angle of orientation, v_t is the robot's linear speed, and $\dot{\theta}_t$ is the robot's directional velocity. If the robot chooses a rotational acceleration, a_t is set to zero. Likewise, if the robot accelerates linearly, $\ddot{\theta}_t$ is set to zero. The robot has a maximum linear speed and rotational velocity (0.5 and $\frac{\pi}{8}$ respectively) past which positive accelerations have no effect. The linear speed v is further constrained to be non-negative. The rotational velocity may be either positive (clockwise) or negative (counter-clockwise). The robot's position is constrained only by the walls of the world. Collisions with the walls are inelastic, which means that if a robot's new position (x_{t+1}, y_{t+1}) at time $t + 1$ intersects a wall, the robot remains at position (x_t, y_t) and its linear speed and rotational velocity are set to zero.

Although the simulator had complete and perfect knowledge of the robot's state, the robot could only see egocentric information: the sonar readings, the food sensors, the doorway sensors, its linear speed, rotational velocity, and the room number. Because the state space is continuous, we used a tile-coding function approximator, also known as a CMAC (Albus, 1981 and Sutton & Barto, 1998). The robot had 10 tilings over different subsets of the available information as summarized in Table 1.

We structured the experiments in this robotic domain to be similar to those presented

# of Tilings	Variables	Size
1	room color, door distance, door angle, forward sonar distances, linear speed, rotational velocity, food eaten in current room	$172,800$
1	all 5 sonar distances, room number, linear speed, rotational velocity, food eaten in current room	$180,000$
8	activation in each $\frac{\pi}{4}$ slice of the robot's food sensors, linear speed, rotational velocity, all 5 sonar distances	$81,000$ each

Table 1: *Tile coding for the simulated robot experiments*

earlier with the gridworlds. The first set of experiments compared the online performance of Macro Q-learning to Q-learning for one million steps. The parameters were $\alpha = 0.05$, $\epsilon = 0.15$, and $\gamma = 0.9$. Figure 6 (left panel) shows the cumulative reward received by each method. Both curves are averages over 30 runs. Figure 6 (right panel) also shows the cumulative reward received using random behavior for one million steps with and without macro-actions. Although both learning methods outperformed their respective random behaviors, Macro Q-learning converged to a solution for finding food much more quickly than did Q-learning. This is in accord with what we found in the gridworld domain where Macro Q-learning with appropriate macro-actions vastly outperformed Q-learning. The robot's two macro-actions cut the learning time in half.

The second set of experiments examined the behavior for one million steps when the actions were selected randomly from the primitive actions and from both the primitive and macro-actions. The two right panels in Figure 7 show a projection of the first 100,000 steps of one such trajectory onto the two spatial dimensions. We cannot simply present a histogram of state visitation in this task, as we did for the gridworlds, because the state space here is of higher dimension. Nevertheless, it is clear that here, as in the gridworlds, the macro-actions have a large influence on the initial exploratory behavior.

With macro-actions, the robot more often crosses between the two rooms and travels with a higher speed. This is shown graphically in Figure 8. The left panel shows the average number of time steps that the robot spent in the first room for each 100,000 steps

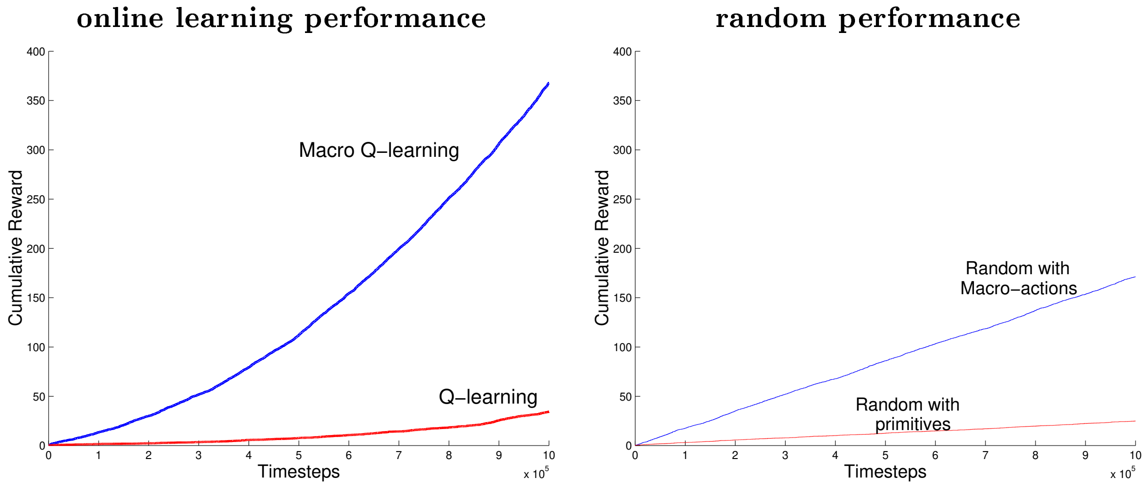

Figure 6: *The left panel shows cumulative reward for online Macro Q-learning and Q-learning. The right panel shows the cumulative reward for random behavior with and without macro-actions on the same scale as the left-hand graph.*

of the million-step random walk taken with and without macro-actions. These numbers are all averages over 30 runs. Although the overall means do not differ by much (44% for macro-actions and 48% without macro-actions), the visitations over time are different for each random walk. For example, without macro-actions the robot remained in the room in which it started for 66% of the first 100,000 steps of its random trajectory, whereas with macro-actions it spent only about 45% of those steps in the initial room. The right hand panel of Figure 8 shows the total number of steps that the robot spent in the first room for each of the 30 runs. The random walks with only primitive actions have more variance and tend to spend more time in the first room than when macro-actions are used. When macro-actions are added to the set of available actions, there is less variation across runs.

The results of the experiments with random behavior show that the use of macro-actions affected the exploratory behavior of the agent in both the gridworld and in the more complicated continuous domain. In the gridworld, the macro-actions caused a non-uniform visitation of the states, whereas in the robotic domain, they caused a more uniform initial visitation of the world.

The final experiments examined the effect of macro-actions on learning independent of

Without Macro-actions

With Macro-actions

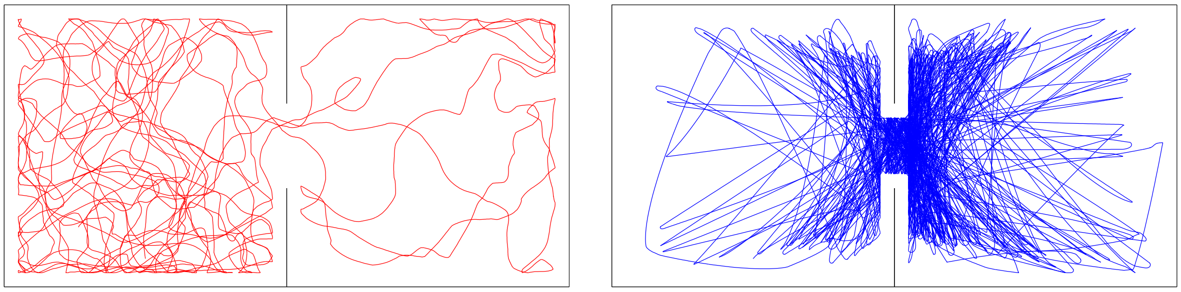

Figure 7: *The graphs on the right hand side represent the position of the robot during a random walk.*

behavior. To do this, both Macro Q-learning and Q-learning were applied to the fixed set of experiences generated from the million-step random walks using both macro-actions and primitive actions. The parameters of this experiment were the same as in the previous set of experiments in this world. In the similar gridworld experiment, we were able to examine the value function for each state directly and measure how quickly values were backed up. Because the state space in the robotic domain is continuous and multi-dimensional, we could not examine it in the same way. Instead, we examined the value function indirectly by freezing the values and evaluating the behavior. Every 50,000 steps, the value-function was frozen and the resulting ϵ-greedy ($\epsilon = 0.01$) policy was executed for 5,000 steps. Figure 9 shows the cumulative reward achieved by Macro Q-learning and Q-learning over the 20 evaluations. It is clear that backing up values for both macro-actions and primitive actions leads to faster convergence to a good foraging policy. This agrees with the results found in the gridworld experiments (Section 5) where learning with macro-actions caused the value information to propagate more rapidly through the state space than did learning with primitive actions only.

8 Conclusions

Our experiments in the simulated robot task are broadly consistent with those obtained in the gridworld tasks, and in our earlier work (McGovern, Sutton, & Fagg, 1997). We have verified and demonstrated the hypothesis that macro-actions may either speed or

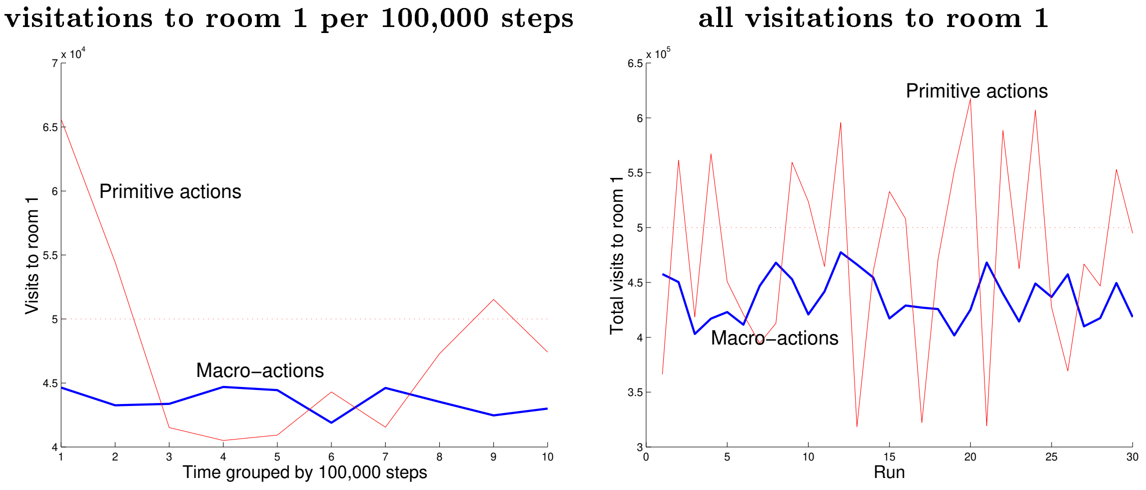

Figure 8: *The left panel shows the average number of time steps that the random walks spent in the first room over each 100,000 time slice. The dashed line in the middle represents the 50% line. The right panel shows the total number of steps over all one million steps that the robot spent in the first room for both types of random walks.*

slow learning depending on their appropriateness to the task. More importantly, we have separated the effect of macro-actions into components and measured them independently. In particular, we have analyzed the contributions to performance of macro-actions' effects on exploratory behavior and on value propagation, both of which can be substantial. Value propagation can also be accelerated through the use of eligibility traces; we have assessed this effect and compared it to the effect of macro-actions. An obvious extension would be to combine macro-action methods with eligibility traces to obtain the advantages of both. Although all of our results are empirical, we believe this is not inappropriate. Today's understanding of temporally abstract actions is limited; we need more empirical experience before we can answer, or even ask, the most important questions.

Acknowledgments

The authors wish to thank Andrew Barto, Andrew Fagg, Doina Precup, Paul Utgoff, and all the members of the ANW group for comments and discussions. This work was sup-

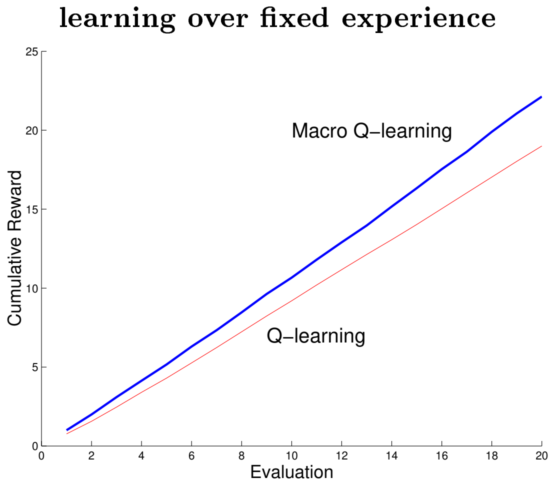

Figure 9: *Cumulative reward for both algorithms when learning over fixed experience.*

ported in part by the National Physical Science Consortium, Lockheed Martin, Advanced Technology Labs, NSF grant ECS-9511805, AFOSR grant AFOSR-F49620-96-1-0254, and NSF grant IRI-9503687.

References

[Albus, 1981] Albus, J. S. (1981). *Brain, Behavior, and Robotics.* Byte Books.

[Bradtke and Duff, 1995] Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems 7*. MIT Press.

[Dayan and Hinton, 1993] Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, pages 271–278. Morgan Kaufmann.

[Dietterich, 1998] Dietterich, T. G. (1998). Hierarchical reinforcement learning with the MAXQ value function decomposition. In *Proceedings of the 15th International Conference on Machine Learning ICML'98*, San Mateo, CA. Morgan Kaufmann.

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

[Huber and Grupen, 1997] Huber, M. and Grupen, R. A. (1997). Learning to coordinate controllers - reinforcement learning on a control basis. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Iba, 1989] Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317.

[Kaelbling, 1993] Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning ICML'93*, pages 167–173, San Mateo, CA. Morgan Kaufmann.

[Korf, 1985] Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77.

[Laird et al., 1986] Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46.

[Lin, 1993] Lin, L. J. (1993). *Reinforcement Learning for Robots using Neural Networks*. PhD thesis, Carnegie Mellon University, School of Computer Science.

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

[Parr, 1998] Parr, R. (1998). *Hierarchical Control and learning for Markov decision processes*. PhD thesis, University of California at Berkeley.

[Parr and Russell, 1997] Parr, R. and Russell, S. (1997). Reinforcement learning with hierarchies of machines. In *Proceedings of Advances in Neural Information Processing Systems 10*. MIT Press.

[Precup et al., 1998] Precup, D., Sutton, R. S., and Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract behaviors. In *Proceedings of the Tenth European Conference on Machine Learning, ECML'98*. Springer-Verlag.

[Singh, 1992] Singh, S. P. (1992). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 202–207, Menlo Park, CA. AAAI Press/MIT Press.

[Singh and Sutton, 1996] Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

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

[Sutton et al., 1998] Sutton, R. S., Precup, D., and 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.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, Cambridge University, Cambridge, England.