Roles of Macro-Actions in Accelerating Reinforcement Learning

Amy McGovern, Richard S. Sutton, Andrew H. Fagg
Computer Science Department, University of Massachusetts, Amherst, MA 01003
{amy|rich|fagg}@cs.umass.edu

Abstract

We analyze the use of built-in policies, or *macro-actions*, as a form of domain knowledge that can improve the speed and scaling of reinforcement learning algorithms. Such macro-actions are often used in robotics, and macro-operators are also well-known as an aid to state-space search in AI systems. The macro-actions we consider are closed-loop policies with termination conditions. The macro-actions can be chosen at the same level as primitive actions. Macro-actions commit the learning agent to act in a particular, purposeful way for a sustained period of time. Overall, macro-actions may either accelerate or retard learning, depending on the appropriateness of the macro-actions to the particular task. We analyze their effect in a simple example, breaking the acceleration effect into two parts: 1) the effect of the macro-action in changing exploratory behavior, independent of learning, and 2) the effect of the macro-action on learning, independent of its effect on behavior. In our example, both effects are significant, but the latter appears to be larger. Finally, we provide a more complex gridworld illustration of how appropriately chosen macro-actions can accelerate overall learning.

1 Problem and Approach

Many problems in artificial intelligence (AI) are too large to be solved practically by searching the state-space using available primitive operators. By searching for the goal using only primitive operators, the AI system is bounded by both the depth and the breadth of the search tree.

One way to overcome this difficulty is through macro-actions (or *macros*). By chunking together primitive actions into macro-actions, the effective length of the solution is shortened. Both [Korf, 1985] and [Iba, 1989] have demonstrated that using macro-actions to search for a solution has resulted in solutions in cases where the system was unable to find answers by searching in primitive state-space, and in finding faster solutions in cases where both systems could solve the problem.

Reinforcement learning (RL) is a collection of methods for discovering near-optimal solutions to stochastic sequential decision problems [Watkins, 1989]. An RL system interacts with the environment by executing actions and receiving rewards from the environment. Unlike supervised learning, RL does not rely on an outside teacher to specify the correct action for a given state. Instead, an RL system tries different actions and uses the feedback from the environment to determine a closed loop policy which maximizes reward.

In this work, we treat macro-actions as closed-loop policies with termination conditions. Prior work that has included closed-loop macro-action policies using RL includes [Singh, 1994, Dayan and Hinton, 1993, Precup and Sutton, 1997, Mahadevan and Connell, 1992].

In this paper, we introduce an extension of current RL algorithms that allows the learner to backup both primitive and macro action values. We demonstrate two specific tasks, which illustrate that, depending on the situation, the use of macro operators can either significantly improve or significantly hinder the rate of learning. We then explore in depth some possible reasons for these effects and finally demonstrate how appropriately chosen macros can accelerate learning in more complex environments.

2 Reinforcement Learning

In an RL system, a learning *agent* interacts with an *environment* at some discrete time scale $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent is in some *state* s_t. The agent chooses an action a_t which causes a state transition at the next time step to state s_{t+1}. The environment emits a reward r_{t+1} for the transition from state s_t to s_{t+1} using action a_t. The goal of the agent is to learn a mapping from states to actions, called a *policy*, which maximizes the agent's discounted reward over time from the environment. This discounted reward is represented by the sum $\sum_{i=0}^{\infty} \gamma^i r_{t+i}$ where γ is the discount parameter.

In general, the states can be multi-dimensional, real-valued, and incompletely known. In this paper we examine the simple discrete case described above for conceptual clarity. The ideas and results described here should generalize to larger and more complex cases.

In order to maximize the reward from the environment, the RL agent learns the estimated reward from each state s for each action a. These values are known as action values. In 1-step Q-learning [Watkins, 1989], the action values are stored in an array called Q where $Q(s_t, a_t)$ is updated at each time step t as follows:

$$Q(s_t, a_t) \quad \leftarrow \quad Q(s_t, a_t) + \alpha \times \tag{1}$$

$$(\gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) + r_{t+1})$$

where α is the learning rate parameter, and γ is the discount parameter.

3 Macro-actions and RL

A macro-action can be any closed loop policy with a termination condition. The agent can choose either a macro or a primitive action from any state unless the agent is already executing a macro-action. Once the agent has chosen a macro-action, it must follow the actions defined by the macro's policy until the termination condition is satisfied or the goal is reached.

In the gridworld examples shown in this paper, the primitive actions allow the agent to move one square at a time either up, down, right, or left. The macro-actions take the agent from the current state to a wall or obstacle in the same four directions.

As described above, in order to maximize the reward received from the environment, the agent learns action values for each state. In the corresponding macro algorithm to Q-learning, called Macro-Q, the action values for the primitive actions are updated using equation 1. The set of all actions available from a state is augmented to include both primitive and macro-actions. The macro action-values for $Q(s_t, m_t)$ are updated according to:

$$\begin{aligned} Q(s_t, m_t) \quad \leftarrow \quad & Q(s_t, m_t) + \alpha \times \\ & (\gamma^n \max_{a'} Q(s_{t+n}, a') - Q(s_t, m_t) + \\ & r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n}) \end{aligned} \tag{2}$$

where s_t is the state in which the agent chose a macro action m_t, n is the number of time steps that the macro took to execute, s_{t+n} is the new state reached after the macro terminates, a' is index of the maximal valued action from s_{t+n}, and r_{t+i} is the reward received at time $t+i$. Macro versions of other RL methods can be constructed analogously.

To illustrate how the behavior of an RL agent using Q-learning compares to the behavior of an agent using Macro-Q, two experiments were performed. In both cases, an 11x11 empty gridworld was used, in which the agent received a reward of 1 for reaching the goal and a zero otherwise. The primitive actions moved the agent one square at a time either up, down, right, or left while the macro-actions moved the agent to a wall or obstacle in the same four directions. Both environments are shown in Figure 1. In the first experiment (Figure 1A), the goal was located in the top row, while in the second experiment (Figure 1B), the goal was located in the center.

In the first experiment, 1-step Macro-Q performed significantly better than Q-learning. Not only did Macro-Q converge faster, but it used fewer total steps to find the optimal path. However, Macro-Q did significantly worse than Q-learning in the second experiment. Figure 1 shows the learning curve of the agent for both experiments.

It is clear from these experiments that macros can help or hinder the behavior of a system. When appropriate macros are used for a given task, the agent performs very well. However, the agent's performance suffers with inappropriate macros. In the next two sections, we isolate and evaluate two different hypotheses about why the use of macros may improve performance.

4 Hypothesis 1: Effect of Macros on Exploration

One possible explanation for Macro-Q's performance differences is that the chosen macros bias the behavior of the agent such that it spends a majority of its time near the edges of the world.

To examine the issue of how the macros bias the agent's behavior independent of how they affect the backup of the action values, we examine what states were visited in two different random walks where all actions were chosen with an equal probability. In both cases the agent took 500,000 random actions on an 11x11 gridworld without a goal state. In the first case, the agent used primitive actions, while in the second case the agent used both primitive and macro-actions. The macro-actions were the same as in the previous experiments.

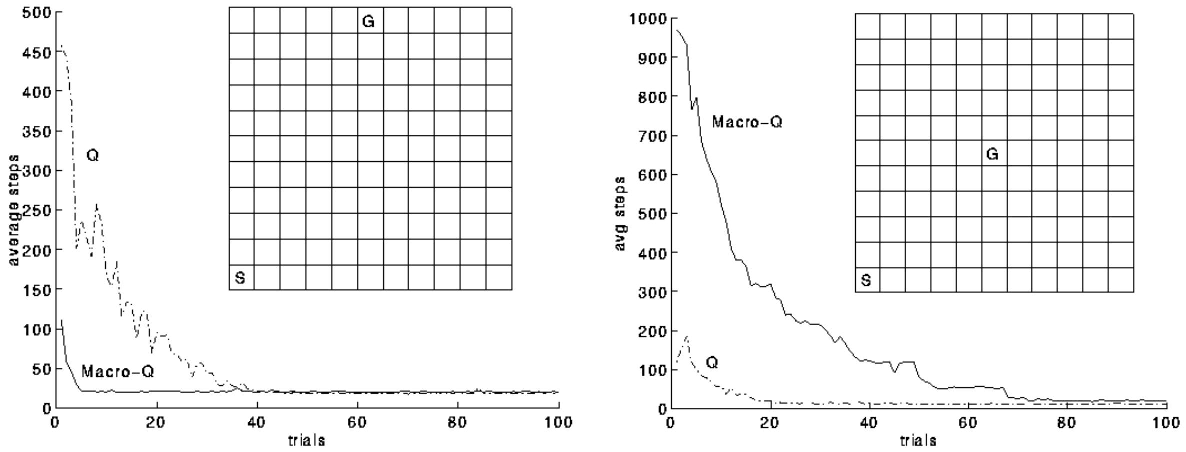
Figure 2 shows the results of these experiments. As expected, with only primitive actions, the agent visited all states equally often (Figure 2A). When the agent had macro-actions available to it, it spent the majority of its time along the edges of the environment (Figure 2B).

Clearly, when the agent spends the majority of its time near the edges and the goal is near the edge, the agent has a higher probability of encountering the goal. This is supported by experiments where the goal state was in the top row. The agent with macro actions available passed the goal state (in the top row) 6630 times out of 500,000 random steps. Without macro actions, the agent passed the goal only 4321 times out of 500,000 random steps. This difference is statistically significant but not large enough to explain the dramatic performance difference in learning with macros shown in Figure 1.

The same quality that makes certain macros perform well in one task hinders their usefulness when the macros do not bring the learning agent to a relevant state. In the second world, macros necessarily move the agent away from the goal (Figure 1B). Thus, it is more difficult for Macro-Q to discover an appropriate path to the goal.

5 Hypothesis 2: Effect of Macros on Propagation of Action-Values

A second reason that macros can help an RL agent learn more quickly is that the macro action values affect the rate of action-value backup. In the case of Q-learning, values propagate backwards one step at a time. However, when macro values are used in the action-value update, value information can propagate



A B

Figure 1: Learning curves of 1-step Macro-Q as compared with 1-step Q-learning for two different tasks. When the goal is located on the edge of the world (A), Macro-Q converges to an optimal policy much quicker than Q-learning. However, when the goal is located in the center of the world (B), Q-learning converges more quickly than Macro-Q.

over several time steps. When a macro action takes the agent to a good state, the corresponding action value is updated immediately with useful information, despite the fact that the original state is several primitive actions away from the good state. On subsequent passes through the original state, the new information is available for actions leading to this state, whether they are primitives or macros.

In order to asses this effect, we compare the propagation of action-values during a random walk, using Macro-Q and Q-learning. The random walk was the same as that described in the previous section (the random walk that used both primitive and macro actions). By comparing the two update algorithms using a fixed behavior, we eliminate any effects that may be due to differences in state visitation probabilities. When the agent reached the goal state, it updated its action-values as if the trial had ended, and then exited from the goal state without updating the action-value for the exit move. If the agent was executing a macro-action when it reached the goal, it updated the macro's action-value when it reached the goal, but continued to follow that macro's policy until termination of the macro. The agent learned the path to the goal using both Macro-Q and Q-learning.

Figure 3 shows the results of these experiments. The relative action values for each state are represented by circles in that state. The radius of the circle is proportional to the maximum action value in that state. Figure 3B shows the action values after the agent had reached the goal three times using Macro-Q. At this point, the agent had learned how to reach the goal from the start state. The maximum action in the start state was a macro that moved it to the upper left corner. From there, the maximum action was a macro-action which took the agent directly to the goal.

In contrast, Figure 3A shows a snapshot of the action values after reaching the goal three times using the same random walk but while updating only primitive action values. At this point, the agent had acquired very little information about the appropriate policy, as indicated by the fact that only three of the states had non-zero action values.

The lower row of Figure 3 (C and D) shows the same two agents after each had reached the goal 20 times. In the case where macro values were being backed up (Figure 3D), the agent had backed up information about much of its world. After passing the goal only 20 times, the agent could get to the goal from 53 of the 121 states. In the case where the agent learned only primitive values, the policy at the start state was still undefined. The agent had only backed up action values for 24 of the 121 states. These illustrations show that backing up action-values with macro values can led to faster convergence.

All of these results have been replicated on larger gridworlds (20x20). The results for the smaller case are easier to graph and see.

6 A Larger Illustration

In the experiments described so far, the environments were very simple. Here, we examine the utility of Macro-Q in more complicated situations, in which 30 10x10 gridworlds contain 25 randomly placed obstacles. The goal was always in the upper right quadrant of the environment and the start state was always in the lower left hand corner. The primitive actions remained the same as before (up, down, right, left) and the macros moved in the same direction five steps or until a wall or obstacle was reached. The agent was rewarded with a value of 1 when it reached the goal and zero otherwise.

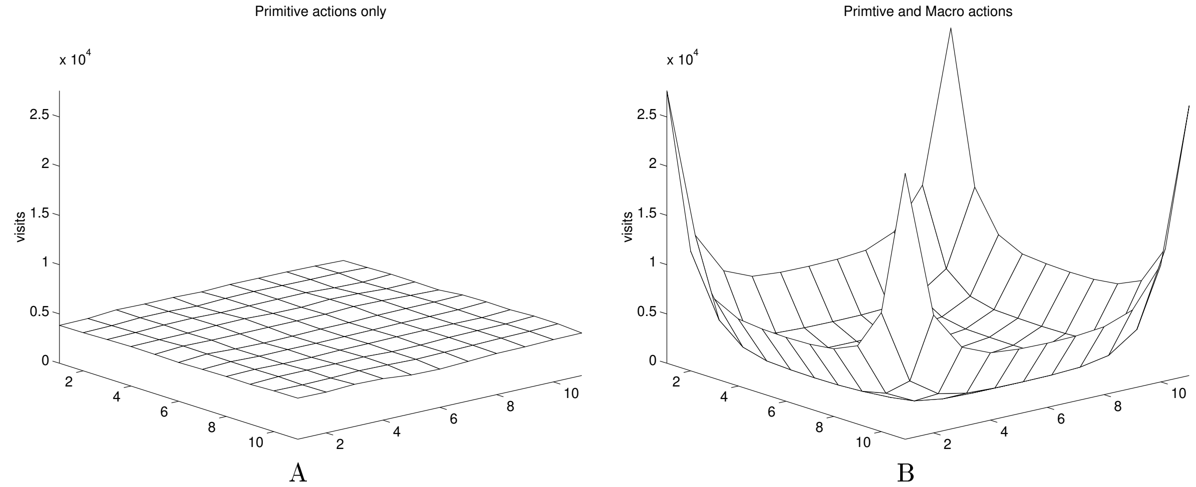


Figure 2: Histograms for random walks using primitive actions (A) and a combination of macro and primitive actions (B). The x and y axes represent the rows and columns of the gridworld, and each square on the graph is one state in the environment. The z axis represents the number of times each state was visited.

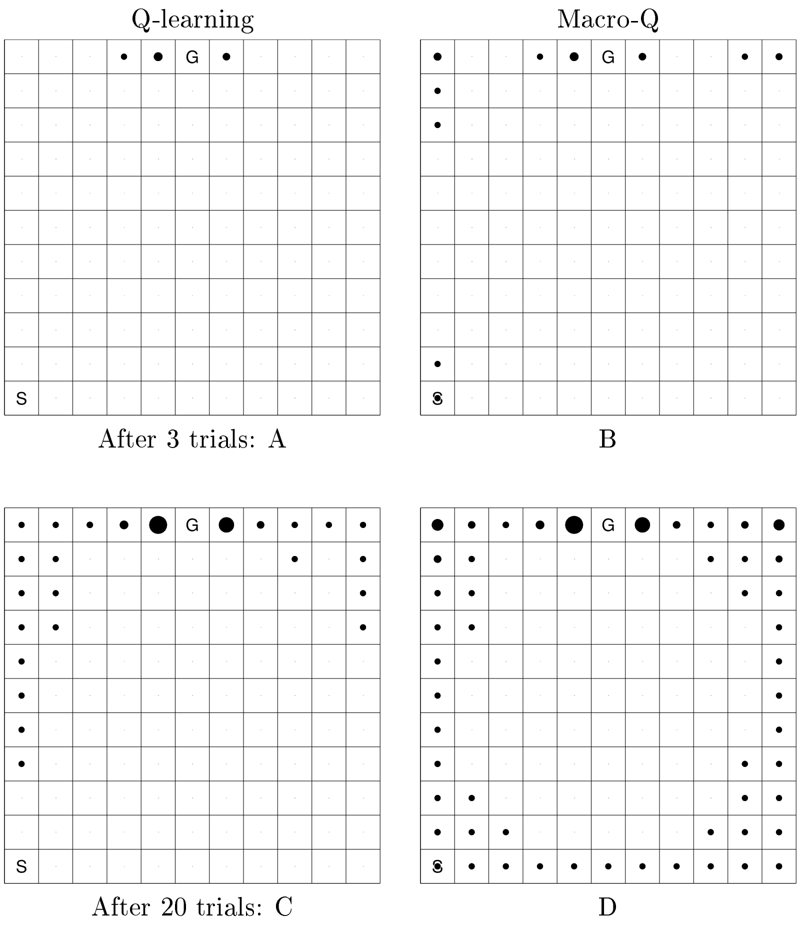


Figure 3: State values for Q-learning and Macro-Q learning under a fixed random walk. Circle size is proportional to the maximum action-value in that state and no circle indicates a maximum action-value of zero.

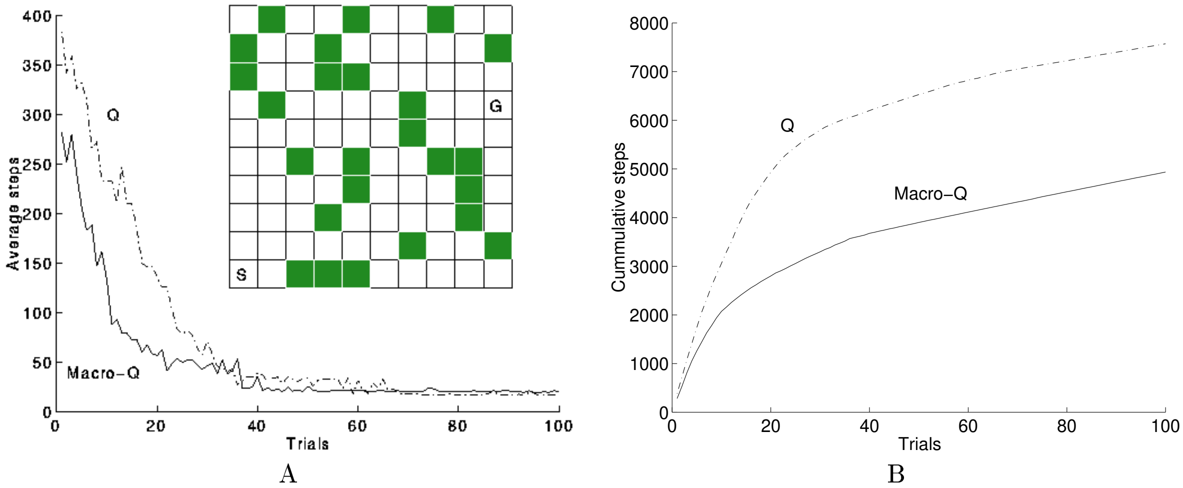


Figure 4: These graphs show Macro-Q's performance against Q learning using only primitive actions averaged over 30 random environments. The graph on the left (A) shows the average number of steps per trial while the agent was learning. The graph on the right (B) shows the cumulative number of steps during learning.

Figure 4 shows the average number of steps and cumulative number of steps per trial averaged over all 30 environments. One of the 30 environments is shown in Figure 4A. In these worlds, Macro-Q converged on average more quickly than Q-learning (Figure 4A). Furthermore, Macro-Q required on average far less experience for convergence, as illustrated in Figure 4B.

Out of the 30 randomly generated environments, only two were troublesome for the chosen macros. Both of these cases contained a single narrow path through a set of obstacles between the start and the goal. The paths were such that the macros alone could not bring the agent close to the goal.

We have shown in these experiments that macro-actions may either speed-up or retard learning, depending on the appropriateness of the macro-actions to the particular task. We also analyzed their effect on the behavior of the learning agent and on action-value propagation. Both effects were shown to be significant but the propagation of action-values seems to be larger. Future work will investigate the automatic acquisition of macro operators as the agent learns to behave in the environment.

7 Acknowledgments

The authors wish to thank Andrew Barto and Doina Precup for comments on earlier versions of the paper. This work is supported in part by the National Physical Science Consortium, Lockheed Martin, Advanced Technology Labs, NSF grant ECS-9214866, and NSF grant ECS-9511805.

References

[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.

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

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

[Mahadevan and Connell, 1992] Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365.

[Precup and Sutton, 1997] Precup, D. and Sutton, R. S. (1997). Multi-time models for temporally abstract planning. In *Proceedings of Advances in Neural Information Processing Systems 10.* MIT Press.

[Singh, 1994] Singh, S. P. (1994). *Learning to Solve Markovian Decision Processes.* PhD thesis, University of Massachusetts, Amherst.

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