Accelerating Reinforcement Learning through the Discovery of Useful Subgoals

Amy McGovern *(amy@cs.umass.edu)* and Andrew G. Barto *(barto@cs.umass.edu)*
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

Keywords Abstraction, Reinforcement Learning (RL), Subgoals, Mobile robots

Abstract

An ability to adjust to changing environments and unforeseen circumstances is likely to be an important component of a successful autonomous space robot. This paper shows how to augment reinforcement learning algorithms with a method for automatically discovering certain types of subgoals online. By creating useful new subgoals while learning, the agent is able to accelerate learning on a current task and to transfer its expertise to related tasks through the reuse of its ability to attain subgoals. Subgoals are created based on commonalities across multiple paths to a solution. We cast the task of finding these commonalities as a multiple-instance learning problem and use the concept of diverse density to find solutions. We introduced this approach in [10] and here we present additional results for a simulated mobile robot task.

1 Introduction

Because robots can better withstand the hardships of radiation and vacuum than can humans, and are able to work almost continuously, they are ideal candidates for helping to create and maintain human habitats in space. Such robots will be most effective if they learn and adapt autonomously. The research presented in this paper is a step towards creating such adaptable and autonomous robots. The ability to decompose a learning problem into a set of simpler learning problems can greatly expand the range of applications in which a learning system can be successful. Within the reinforcement learning (RL) paradigm, one way to do this is to introduce subgoals with their own reward functions, learn policies for achieving these subgoals, and then use these policies as temporally-extended actions, or options [13] for solving the overall problem. In addition to accelerating learning on the current task, this strategy can facilitate skill transfer to other tasks in which the same subgoals are useful.

This paper describes a method by which an RL agent can discover useful subgoals automatically. It is based on the idea of "mining" an ensemble of behavioral trajectories accumulated by the agent as it interacts with its environment. This ensemble can be processed in various ways to form concepts useful to the learning agent. In this paper, the focus is on discovering subgoals of achievement by searching online for "bottlenecks" in observation space. Informally, a bottleneck is a region in the agent's observation space that the agent tends to visit frequently on successful paths to a goal but not on unsuccessful paths (for some suitable definition of success). The bottlenecks of interest are those that appear early and persist throughout learning. If the agent can discover these bottleneck regions and learn policies to reach them during the initial stages of learning, it can use these policies for more effective exploration as well as to more quickly refine its overall policy. These subgoal policies can then be used to facilitate learning in similar tasks.

We treat the problem of finding bottleneck regions as a *multiple-instance learning problem* as defined by [4]. In this type of problem, a system attempts to identify a target concept on the basis of "bags" of instances: positive bags have at least one positive instance, while negative bags consist of all negative instances. A successful trajectory corresponds to a positive bag, where the instances are the agent's observations along that trajectory. A negative bag consists of observations made over an unsuccessful trajectory. We argue that the problem of finding bottleneck regions is well fit by this paradigm, and we use the concept of *diverse density* [8] to detect the bottleneck regions.

Methods for automatically introducing subgoals have been studied in the context of adaptive production systems, where subgoals are created based on examinations of problem-solving protocols (e.g., [1, 2]). Iba's [6] macro-growing heuristic is reflected in several parts of our algorithm and discussed in more detail below. For RL systems, several researchers have proposed methods by which policies learned for a set of related tasks are examined for commonalities [14] or are probabilistically combined to form new policies [3]. However, neither of these RL methods introduce subgoals. The most closely related research is that of [5]. In his system, states that are visited frequently or states where the reward gradient is high are chosen as subgoals. Some of this material was also presented in [10].

2 Reinforcement Learning

In the RL framework, a learning *agent* interacts with an *environment* over a series of time steps $t = 0,1,2,3,\ldots$. At each time t, the agent observes the environment *state*, s_t, and chooses an action, a_t, which causes the environment to transition to state s_{t+1} and to emit a reward, r_{t+1}. In a Markovian system, the next state and reward depend only on the preceding state and action, but they may depend on these in a stochastic manner. The objective of the agent is to learn to maximize the expected value of reward received over time. It does this by learning a (possibly stochastic) mapping from states to actions called a *policy*. More precisely, the objective is to choose each action a_t 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. Other return formulations are also possible. A common solution strategy is to approximate the optimal action-value function, or Q-function, which maps each state and action to the maximum expected return starting from the given state and action and thereafter always taking the best actions. See [12] for details.

We use the options framework (Sutton et al., 1999; Precup, 2000) to define subgoals. An option is a temporally-extended action which, when selected by the agent, executes until a termination condition is satisfied. While an option is executing, actions are chosen according to the option's own policy. An option is like a traditional macro except that instead of generating a fixed sequence of actions, it follows a closed-loop policy so that it can react to the environment. By augmenting the agent's set of base actions—its *primitive actions*—by a set of options, the agent's performance can be enhanced. More specifically, a (Markov) option is a triple $\langle I,\pi,\beta\rangle$, where I is the option's input set, i.e., the set of states in which the option can be initiated; π is the option's policy defined over all states in which the option can execute; and β is the termination condition, i.e., the option terminates with probability $\beta(s)$ for each state s. Each option that we use in this paper bases its policy on its own internal value function, which can be modified over time in response to the environment.

We also define an additional term. A *state trajectory* of length n starting at time step t is a sequence of states $s_t, s_{t+1},\ldots,s_{t+n}$. In episodic tasks, each trajectory corresponds to a single episode. For continuing tasks, one can use a variety of methods to segment experience into finite-length trajectories. For example, a trajectory could end when a reward peak is reached as suggested by Iba's "peak-to-peak" heuristic [6].

3 Autonomous Subgoal Discovery

The simplest approach to creating useful options is to search by generating many new options, randomly or based on simple heuristics, and letting the agent test them by adding them to its set of actions. Although some of these options may be useful, others can degrade the agent's performance, e.g., by adversely affecting its mode of exploration [9]. Performance can also deteriorate due to the agent having too many actions from which to select. Instead, the agent needs a more focused method for creating new options. In the approach described in this paper, the focus is on discovering useful subgoals that can be defined in the agent's observation space. New options are then created to accomplish those subgoals.

To discover useful new subgoals, the agent searches for bottleneck regions in its observation space. The idea of looking for bottleneck regions was motivated by studying room-to-room navigation tasks where the agent should quickly discover the utility of doorways as subgoals. If the agent can recognize that a doorway is a kind of bottleneck by detecting that the sensation of being in the doorway always occurred somewhere on successful trajectories but not always on unsuccessful ones, then it can create an option to reach the doorway. This option can accelerate learning on the current task—if created early enough—as well as enable the agent to learn more rapidly on related tasks in the same or similar environments.

One motivation for using bottlenecks as subgoals is the effect of the subgoal options on the agent's exploration. If the agent uses some form of randomness to select exploratory primitive actions, it is likely to remain within the more strongly connected regions of the state space. An option for achieving a bottleneck region, on the other hand, will tend to connect separate strongly connected areas. For example, in a room-to-room navigation task, navigation using primitive movement commands produces relatively strongly connected dynamics within each room but not between rooms. A doorway links two strongly connected regions. By adding an option to reach a doorway subgoal, the rooms become more closely connected. This allows the agent to more uniformly explore its environment. We have shown in previous work [9] that the effect on exploration is one of two main reasons that options are sometimes able to dramatically affect learning.

The idea of using bottlenecks as subgoals is not confined to gridworlds or navigation tasks. We expect that other tasks with similar dynamics would also benefit from subgoal discovery as described in this paper. For example, consider a game in which the agent must find a key to open a door before it can proceed. If it can discover that having a key is a useful subgoal, then it will more quickly be able to learn how to advance from level to level. Clearly this approach will not work for every task since bottlenecks or subgoals of achievement do not always make sense for particular environments.

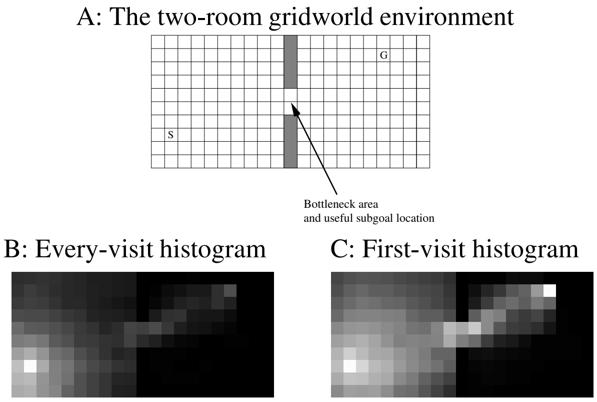

Figure 1: State visitation histograms for the middle stages of learning in a two-room gridworld

Automatically finding bottleneck regions in observation space is a difficult task. It is even more challenging if it has to be done online. An offline method could examine optimal trajectories of several related tasks for commonalities which may reveal bottlenecks. An online method would need to inspect trajectories as they are gathered while learning. Consider the case where each discrete state is completely observed. A first approach might be to simply look for states that are more frequently visited. However, if we examine every-visit frequencies, where each state is counted each time that it is visited, the resulting histogram is not generally helpful for bottleneck detection. In a room-to-room navigation task, for example, the agent spends most of its time within a room and very little time moving through a doorway. If one counts only the first visit to each state within a trajectory, on the other hand, bottlenecks are more readily visible. Examples of these two types of state-visitation frequencies for a two-room example are shown as histograms in Figure 1. The histograms in Panels B and C are shaded by visitation frequency counts collected over 10 trajectories, with higher counts shaded more lightly than lower counts. Each histogram shows data collected over 30 runs of the 30-40th trials of an agent using Q-learning in the 21x10 gridworld shown in Panel A of the figure. The agent used ϵ-greedy exploration with $\epsilon = 0.1$ and a fixed step-size parameter of 0.05. All rewards were zero until the agent reached the goal state where it received a reward of 1. The discount factor was 0.9, which meant that the agent was learning how to find the goal as quickly as possible.

The histograms in Figure 1 indicate that first-visit frequencies (Panel C) are better able to highlight the bottleneck states (states in the doorway) than are the every-visit frequencies (Panel B). However, the idea of just using visitation frequencies to detect bottleneck states has several drawbacks. The first is that it is clearly a noisy process, as the graph illustrates. The second is that it is not immediately obvious how to do this in problems with continuous or very large state spaces. Last, visitation frequencies do not incorporate negative evidence for bottleneck states in a principled manner. All of these problems motivate our use of the multiple-instance learning paradigm and the concept of diverse density to precisely define and detect bottlenecks.

4 Multiple-Instance Learning and Diverse Density

Multiple-instance learning problems as described by [4] are supervised learning problems in which each object to be classified is represented by a set of feature vectors, only one of which may be responsible for its observed classification. An example from explanation-based learning that these authors give is when many explanations for an observed result can be obtained from a domain theory, but only one explanation can account for all observed results. More specifically, there are multiple positive and negative bags of instances [8]. Each positive bag must contain at least one positive instance from the target concept but may contain many negative instances. Each negative bag must contain all negative instances. The individual instances within each bag are not labeled. The goal is to learn the concept from the evidence presented by the different bags.

The problem of mining collections of trajectories for bottlenecks, or other concepts useful for defining subgoals, can be formulated as a multiple-instance learning problem. Each trajectory can be viewed as a bag, with the agent's individual observation vectors being the instances within the bag. Positive bags are successful trajectories; negative bags are unsuccessful trajectories. What constitutes a successful or unsuccessful trajectory can be defined in a problem-dependent way. For example, successful trajectories might be all those trajectories in which the agent reached a goal state no matter how many steps it took. Or success might depend on reaching a goal within a certain number of steps. A bottleneck region of observation space as described above corresponds to a target concept in this multiple-instance learning problem: the agent experiences this region *somewhere* on every successful trajectory and not at all on unsuccessful trajectories.

Maron and Lozano-Pérez [8] devised the concept of diverse density to solve multiple-instance learning problems. The most diversely dense region in feature space is the region with instances from the most positive bags and the least negative bags. This differs from the concept of simple density by including the idea of using many different bags rather than one large set of instances. The region of maximum diverse density can be detected using either exhaustive search or gradient descent. Maron defines the diverse density of a target concept c_t to be $DD(t) = Pr(t|B_1^+, \ldots, B_n^+, B_1^-, \ldots, B_m^-)$, where $Pr(t)$ is the probability that the t^{th} concept from the concepts $\{c_t\}$ is the correct concept, B_i^+ is the i^{th} positive bag, and B_i^- is the i^{th} negative bag. The concept with the maximum DD value is the output of a DD search.

To perform this search, we must more precisely instantiate the definition of DD given above. Using Bayes' rule several times and assuming a uniform prior, we look for the concept with the highest value as defined by:

$$DD(t) = \prod_{1 \le i \le n} Pr(t|B_i^+) \prod_{1 \le i \le m} Pr(t|B_i^-).$$

Maron generally uses a noisy-or model to model the probability of any event in a bag causing the t^{th} concept to be correct, which yields:

$$\begin{aligned} Pr(t|B_i^+) &= 1 - \prod_{1 \le j \le p} (1 - Pr(B_{ij}^+ \in c_t)) \\ Pr(t|B_i^-) &= \prod_{1 \le j \le p} (1 - Pr(B_{ij}^- \in c_t)), \end{aligned}$$

where B_{ij} is the j^{th} instance of the i^{th} bag. Lastly, the probability of a particular instance being in the target concept, $Pr(B_{ij} \in c_t)$, is defined to be a Gaussian based on the distance from the particular instance to the target concept. In this paper, we use exhaustive search to find the concept with the highest DD value since it is feasible for our tasks. Many other search methods can be used successfully on larger problems.

The concept of diverse density corresponds exactly to our concept of a bottleneck region. The region with maximum diverse density will be a bottleneck region which the agent passes through on multiple successful trajectories and not on unsuccessful ones. Abstract types of concept spaces can be used within this framework. However, in this paper we restrict attention to the simplest concept space consisting of individual states. In this case, a subgoal is always to reach a given state, and we use bags that are simply state trajectories.

To take full advantage of diverse density calculations in the problem of defining subgoals, we do not add all states to a bag. For example, if the agent always starts or ends each trajectory in the same set of states, then those states will have a high diverse density since each will appear in all of the positive bags. We exclude states surrounding the starting and ending states from any bag. This follows Iba's [6] use of static filters for a macro-growing mechanism. A static filter can be defined on a per-task basis and used to filter out any undesired subgoals before they can be discovered by the agent.

5 Forming New Options

Once the agent has created a set of bags from its saved trajectories, it searches for the maximum diverse density regions. If the agent has only created a small number of bags because it is still within the initial stages of learning, the diverse density peaks will be noisy. The bottlenecks of interest are those that appear early within the maximum diverse density regions and persist throughout learning. An easy way to detect which regions appear early and persist as peaks is to use a running average of how often each state appears as a peak. The average for each state is initialized to zero. At the end of each trajectory, the agent creates a new bag and searches for concepts with high diverse density. The average, $\bar{c}$, for each concept c found is updated by: $\bar{c} = \lambda(\bar{c} + 1)$ where $\lambda \in [0, 1)$. If a concept is an early and persistent maximum, then its average will rise quickly and converge to $\frac{\lambda}{1-\lambda}$. The agent examines the concepts whose averages rise above a specified threshold and uses those to create new options.

To create a new option for a subgoal extracted in this way, the option's input set, I, can be initialized in several ways. The method used in this paper is to search the agent's saved state trajectories for occurrences of the subgoal state(s). When a subgoal has been found in a trajectory at time step t, the agent adds to I the set of states visited by the agent from time $t - n$ to t, where n is a positive integer specified as a parameter. The input set of the subgoal option is therefore the union of all such states over all of the agent's saved trajectories. Other methods based on examination of the optimal value function could be used if the subgoals were not created until the learning was completed. The termination condition, β, is set to 1 when the subgoal is reached or when the agent is no longer in the input set, and is set to 0 otherwise. The option's policy, π, is initialized by creating a new value function that uses the same state space as the overall problem. The reward function used for the option is to give a reward of -1 on each step and 0 when the option terminates. The agent can also be rewarded negatively for leaving the input set. The option's value function is learned using experience replay [7] with the saved trajectories.

The algorithm for discovering subgoals and creating new options is summarized in pseudocode in Table 1. Although saving the full history seems expensive in terms of space, it is linear in the size of the trajectories and can quite reasonably be saved. Also, if memory is at a pre-

Table 1: Pseudocode of the subgoal-discovery algorithm presented in this paper

Init full trajectory database to $\emptyset$
For each trial
    Interact with environment/Learn using RL
    Add observed full trajectory to database
    Create positive or negative bag from state trajectory
    Search for diverse density peaks
    For each peak concept c found
        Update the running average by $\bar{c} = \lambda(\bar{c} + 1)$
        If $\bar{c}$ is above threshold
            If c passes the static filter
                Create a new option $o = <I, \pi, \beta>$
                    of reaching concept c
                Init I by examining trajectory database
                Set $\beta(c) = 1$, $\beta(S - I) = 1$, $\beta(\cdot) = 0$ else
                Init policy π using experience replay

mium, the agent could save only the last n trajectories. Memory space has not been a problem in practice over a number of different tasks. The next section illustrates the utility of this algorithm on several illustrative problems.

6 Experimental Results

6.1 Two-Room Gridworld Illustration

We illustrate diverse density and subgoal discovery on two simple gridworld problems. The first is the two-room environment shown in Figure 1A. For these experiments, the goal state was placed in the lower right-hand corner and each trial started from a randomly chosen state in the left-hand room. The primitive actions are the usual four primitive actions of `up`, `down`, `right`, and `left`. Each action succeeds in moving the agent in the chosen direction with probability 0.9 and in a uniform random direction with probability 0.1. The reward, learning parameters, and learning algorithm are the same as described in Section 3. Once the agent created an option, it switched its learning algorithm from Q-learning to Macro Q-learning [11]. The agent was limited to creating only one option per run.

The agent created a positive bag for each trajectory in which it successfully reached the goal state from the start state. It did not take into account the number of steps within a trajectory. No negative bags were created. Because this gridworld has a small number of states, we were able to calculate the diverse density exactly for each state. Figure 2A shows the average log likelihood of the diverse density for each state after 25 trials. The results in this graph are averaged over 30 trials. Although this average graph is slightly smoother than the individual graphs for each run, the peak in diverse density near the doorway remains the same. Figure 2B shows the locations of the subgoals created over the 30 runs. As expected, locations near the door were the most frequently detected subgoal locations.

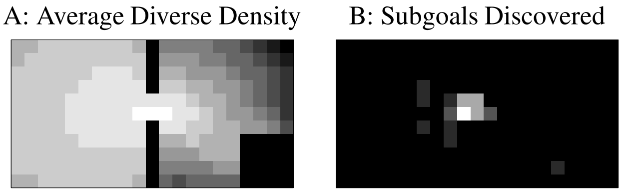

Figure 2: A: Average DD values where the states with higher DD values are shaded more lightly. The static filter excluded the goal region while the walls were never reached and therefore have very low DD values. B: Locations of the new subgoals formed by the learning agent. Each state is shaded by the number of times that it was selected as a subgoal location where the lighter the square, the more often it was selected as a subgoal.

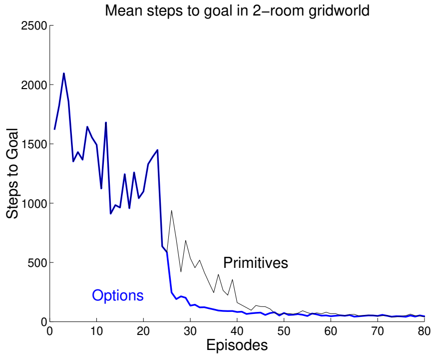

Figure 3: Average steps to goal with and without automatic subgoal detection two-room gridworld.

One reason that it is important for the learning agent to be able to detect these bottleneck states is the effect on the rate of convergence to a solution. If the subgoals are useful, then learning should be accelerated. To ascertain that these subgoals were helping the agent to improve its policy more quickly, the average number of steps that the agent took to reach the goal when using subgoal discovery was compared to learning using only primitive actions. The average results of this comparison over 30 different runs are shown in Figure 3. It is clear that learning with automatic subgoal discovery has considerably accelerated learning compared to learning with primitive actions alone. The initial trials were the same because the options were not added until approximately trial 20. The automatically created options were useful for accelerating learning on this simple task. Although the variances are not shown on the graph, the difference between the two learning curves is greater than one standard deviation.

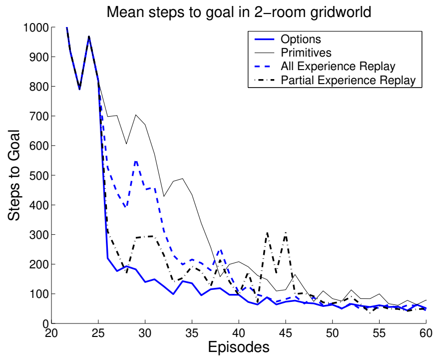

Figure 4: Comparison of the effect of experience replay over different parts of the state space to option-growing

These results indicate that automatically discovered options can be useful for accelerating learning within a given task. To more fully understand why the options are useful, we performed two additional experiments in the two-room gridworld. First, it is possible that the time spent in experience replay during initialization of the option's input set could have been better spent by performing experience replay over the entire state space. To test this, we allowed the agent to use an equal amount of experience replay over the entire state space starting at the same trial where the agent would have created its new options. However, no new options were created. Because the option-discovery agent only performed experience replay using the states in the option's input set, it performed a much smaller number of backups than the total experience available per trajectory. Counting the number of backups that the option-discovery agent would have performed for each trajectory and probabilistically choosing experiences from the entire trajectory for replay approximately equalized the amount of computational work performed by the two agents. Figure 4 shows the comparison of this approach with the option-discovery agent and learning with primitive actions only. The latter results are the same as in Figure 3 but the axes have been changed to highlight the meaningful regions. These results indicate that while experience replay can help to accelerate learning on the current task, option discovery is more advantageous.

We also examined the effect of the option's initial policy. In this case, the agent performed experience replay over the states in the option's input set except that the backups occurred in the main value function instead of in the option's value function. No options were created. The results of this experiment are also shown in Figure 4. The performance of the agent using partial experience replay was almost indistinguishable from performance of the option-discovery agent. The effect of this partial experience replay is to give the agent a useful policy for navigating to the doorway. Since this appears to accelerate learning almost as quickly option-discovery, it is clear that a primary reason that the newly created options are beneficial to the agent is their initial policy.

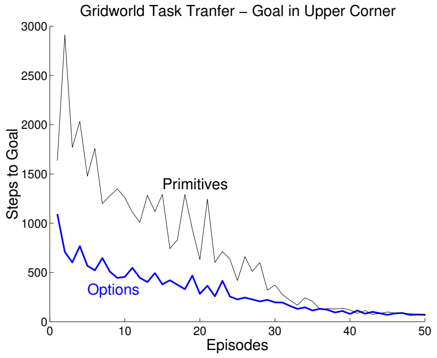

Figure 5: Learning curves in the new two-room task comparing use of the previously learned options to primitive actions.

Another reason that automatic discovery of subgoals is useful is that the agent can use the subgoals and their corresponding options to facilitate learning on similar tasks. To illustrate how learned subgoals can be useful for task transfer, the gridworld task was changed by moving the goal to the upper right-hand corner and decreasing the success probability from 0.9 to 0.8. The agent using options was initialized with the options discovered in the corresponding run of the previous task. Figure 5 compares the average number of steps to the goal for the agent using only primitive actions with the average for the agent using the previously learned subgoal options in addition to the primitive actions. Clearly, the availability of these options considerably accelerated learning. This was consistently observed across many different transfer tasks using this environment.

To ensure that it is the options themselves which are useful and not just the availability of an appropriate multi-step policy in a new environment, we compared the same task transfer results with two other experiments. In the first experiment, the agent started the new task with the learned primitive value function from the prior task. In the second experiment, the agent started with values from the states in the option's input set but did not use the options themselves. We then compared the learning of these two agents to the results discussed above. This comparison is shown in Figure 6. It is clear that simply reusing the value function in whole or in part is drasti-

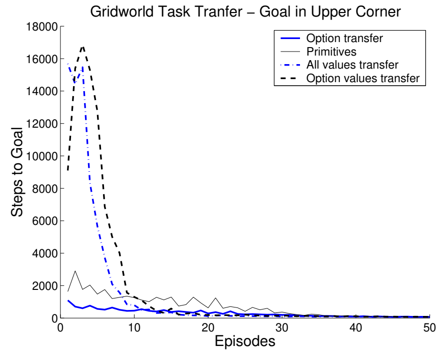

Figure 6: Comparison of knowledge transfer using the value function to using the newly discovered options.

cally worse. Instead, the advantage of the new options for task transfer is highlighted even more distinctly.

7 Simulated Robot Task

We also tested this method using a simulated Pioneer mobile robot (Figure 7). The robot has 7 sonars, a simple vision system with a 1-D "retina" having a 140 degree field of view, and a gripper with a sensor that signals when a ball is being gripped. The primitive action set consists of 7 primitive actions (3 distances forward, 2 left and 2 right rotations all implemented by 2 PD controllers). The task is to find and pick up a ball and deposit it close to a wall without running into a wall at any time. The robot received a reward of $+10$ for putting the ball in the desired location, -1 for touching a wall, and -0.1 per time step until correctly depositing the ball. This latter negative reward encourages the robot to accomplish the task in minimum time. The robot learned through the use of Q-learning with a CMAC tile encoding to represent the state space (see [12]).

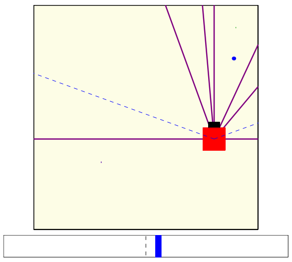

Figure 7: Simulated Pioneer mobile robot. The lines emanating from the robot represent the sonar beams and the two dashed lines show the vision cone. The rectangle on the bottom shows the ball's appearance on the retina.

The robot was able to create one new option per run. We restricted the subgoal to be expressed as subset of the projection of the sensory space onto the subspace spanned by the width and displacement of the ball's appearance on the retina. A subgoal that is useful in any task involving the robot moving the ball is for the robot to have the ball in its gripper. To do this, the ball should be looming directly in front of the robot with a large image width on the retina. Figure 8 shows the option subgoals discovered by the robot for 30 runs as projections onto the two relevant sensory variables. As the figure shows, the robot discovered the subgoal of getting close to the ball and oriented directly toward it.

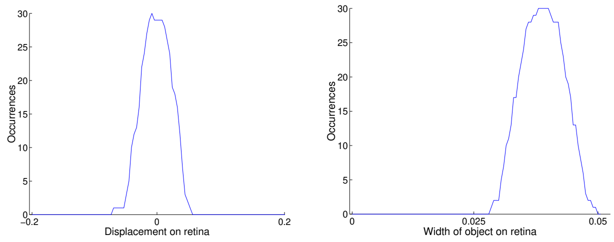

Figure 8: Frequencies of each sensory variable's occurrence in a subgoal over 30 runs. To achieve the subgoal, the robot must use both displacement and width of the ball's appearance on the retina.

This result shows that the robot was able to extract a useful subgoal in a relatively difficult task. This subgoal will then be beneficial to the agent in new tasks or as the environment changes. Previous experiments have been in discrete state spaces, but the use of the algorithm in a continuous space such as this helps to demonstrate its applicability.

8 Discussion and Conclusions

We introduced a method for automatically creating subgoal options online by searching for bottlenecks in observation space. We formulated the problem as a multiple-instance learning problem and used the concept of diverse density to solve it. We illustrated this approach in several simulated tasks and showed that it can both accelerate learning on the current task and facilitate transfer to related tasks. Although these were not large-scale tasks, we believe that our results suggest that this is a promising approach to one aspect of the challenge of automatic abstraction.

This research is part of our broader focus on mining the agent's experience to create many different types of useful new options. It is clear that the subgoals of achievement like the type discovered in this paper are not useful for every environment. However, the broader idea of mining the agent's past experience can yield useful options across many environments. This research is one part of that approach.

Our current research extends this approach by considering more abstract concept spaces in which to compute diverse density. For example, concepts in the form of linearly discriminable surfaces might allow a robotic agent to detect concepts not expressible as a single point in feature space. We are also developing an online method for filtering and deleting options that prove less useful, which can further accelerate learning. This can also help if an option is less useful on a new task than expected and in cases where an initially useful subgoal is keeping the agent from discovering an even shorter path to the goal.

One potential drawback to this method is the requirement that negative bags cannot contain any positive instances. In the case of more complicated environments, it can be tricky to define what constitutes the positive and negative bags. It is desirable to allow the agent to occasionally visit useful subgoals on unsuccessful trajectories while not affecting the results of the diverse density calculations. We are currently investigating how to best utilize such noisy bags. One such method is to decrease the influence of each negative instance through the width of the Gaussians. It may also be possible to give each bag a relative measure of success.

Another drawback to this method of subgoal discovery is that the agent must first be able to reach the overall goal using only the given primitive actions (or any options that are pre-defined). This limits the problems to which it can be applied. Current research addresses ways to extend the approach so that it can be applied when goal states cannot easily be reached using only primitive actions.

Acknowledgments

The authors are grateful for comments and helpful discussions from Andrew Fagg and Balaraman Ravindran. This work was supported by the National Physical Science Consortium, Lockheed Martin, Advanced Technology Labs, and the National Science Foundation under grant ECS-9980062 and EIA 9703217.

References

[1] S. Amarel. On representations of problems of reasoning about actions. In D. Michie, editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier, North Holland, 1968.

[2] Yuichiro Anzai and Herbert A. Simon. The theory of learning by doing. *Psychological Review*, 86(2):124–140, 1979.

[3] Daniel S Bernstein. Reusing old policies to accelerate learning on new MDPs. Technical Report UM-CS-1999-026, Dept. of CS, U. of Massachusetts, Amherst, MA, 1999.

[4] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[5] Bruce Digney. Learning hierarchical control structure for multiple tasks and changing environments. In *From animals to animats 5: SAB 98*, 1998.

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

[7] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[8] Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *NIPS 10*, pages 570–576, Cambridge, MA, 1998. MIT Press.

[9] Amy McGovern. Roles of macro-actions in accelerating reinforcement learning. Master's thesis, U. of Massachusetts, Amherst, 1998. Also Technical Report 98-70.

[10] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *To appear in Proceedings of the 18th International Conference on Machine Learning ICML 2001*, 2001.

[11] Amy McGovern, Richard S. Sutton, and Andrew H. Fagg. Roles of macro-actions in accelerating reinforcement learning. In *Proc. of the 1997 Grace Hopper Celebration of Women in Computing*, pages 13–18, 1997.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning. An Introduction*. MIT Press, Cambridge, MA, 1998.

[13] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[14] S. B. Thrun and A. Schwartz. Finding structure in reinforcement learning. In T. Leen G. Tesauro, D. Touretzky, editor, *NIPS 7*, pages 385–392, San Mateo, CA, 1995. Morgan Kaufmann.