Towards the Application of Reinforcement Learning to Undirected Developmental Learning

Jonathan Mugan
Computer Science Department
University of Texas at Austin
Austin Texas, 78712 USA
jmugan@cs.utexas.edu

Benjamin Kuipers
Computer Science Department
University of Texas at Austin
Austin Texas, 78712 USA
kuipers@cs.utexas.edu

Abstract

We consider the problem of how a learning agent in a continuous and dynamic world can autonomously learn about itself, its environment, and how to perform simple actions. In previous work we showed how an agent could learn an abstraction consisting of contingencies and distinctions. In this paper we propose a method whereby an agent using this abstraction can create its own reinforcement learning problems. The agent generates an internal signal that motivates it to move into states in which a contingency will hold. The agent then uses reinforcement learning to learn to move to those states effectively. It can then use the knowledge acquired through reinforcement learning as part of simple actions. We evaluate this work using a simulated physical agent that affects its world using two continuous motor variables and experiences its world using a set of fifteen continuous and discrete sensor variables. We show that by using this method the learning agent can autonomously generate reinforcement learning problems allowing it to do simple tasks, and we compare its performance to that of a hand-created reinforcement learner using tile coding.

1. Introduction

A major goal of Epigenetic Robotics is to discover how an agent (robot or human) can learn high-level concepts of objects and actions from low-level sensory and motor experience. In previous work, we have shown how an agent can identify contingencies among events in its world, can learn new distinctions and increasingly accurate rules to predict those contingencies, and can backchain through those rules to accomplish certain goals. Backchaining is adequate for combining actions with few dependencies among them. The contribution of this paper is to extend our method to handle situations where actions have significant interactions. Such situations can often be identified via the preconditions of other desired actions. We use these preconditions to define the state-space for a simple reinforcement learning problem. The result of learning is a policy to navigate through the state space and achieve those conditions.

We apply reinforcement learning in the context of undirected developmental learning by having the agent generate these reinforcement learning problems autonomously. The agent generates a reinforcement learning problem by defining a state space and a goal within that space. The agent uses only a relevant subset of the variables for the state space to keep the reinforcement learning problem small. However even with a small number of variables, reinforcement learning in continuous domains is difficult. To overcome this, our agent learns a discretization for these state variables that conforms to the dynamics of the environment.

The agent generates these reinforcement learning problems using the learned abstraction that consists of events, contingencies, and distinctions. A *contingency* temporally links the occurrence of one event E_1 with the occurrence of another event E_2. For each contingency, the agent can search for new *distinctions* that delimit regions where this contingency is more reliable. Using these distinctions, the agent can then greedily search for context variables whose discrete values accurately predict the reliability of the contingency. The agent can then create a reinforcement learning problem from the contingency and the context whose goal is to reach a state in which the contingency is reliable. The state space for this problem consists of the variables in the context, and the discretization for those variables is determined by the learned distinctions. The action space for all of the generated reinforcement learning problems is determined by the distinctions learned on the motor variables.

We first summarize the abstraction learning algorithm of [Mugan and Kuipers, 2007a] and [Mugan

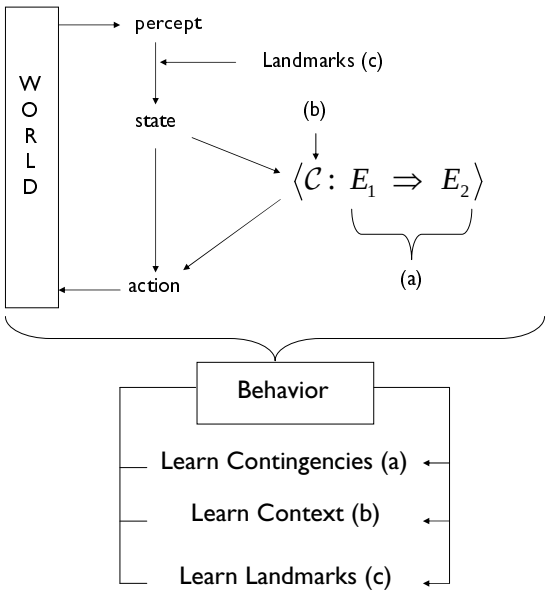

Figure 1: (a) The agent interacts with the world to learn contingencies stating that when one event E_1 occurs, that another event E_2 will soon occur. (b) The agent then makes each contingency into a rule. For each rule the agent learns a context $\mathcal{C}$ that consists of a set of variables upon which the agent can learn a conditional probability table $CPT(r) = Pr(succeeds(r)|\mathcal{C})$ that tells the agent the probability of E_2 following E_1 given the values of the variables in the context. (c) The agent also learns distinctions in the form of landmarks that it uses to discretize its input.

and Kuipers, 2007b]. We then describe how an agent can use this abstraction to generate reinforcement learning problems. We then describe how the agent explores by using a "rich get richer" strategy to continually choose which contingency to bring about. Next, we evaluate our learning algorithm using a simple but realistic physical simulation. We demonstrate that the autonomous agent's performance is comparable to the performance achieved by a hand-created, tile-coding reinforcement learning algorithm trained only on the evaluation task. Finally, we discuss the results and related work, and then conclude.

2. The Learned Abstraction

In this section we summarize the abstraction presented in [Mugan and Kuipers, 2007b] and we discuss the backchaining method presented in [Mugan and Kuipers, 2007a]. A summary of the learning process is shown in Fig. 1. Distinctions are made using *landmarks*, we use landmarks to convert each continuous variable $\tilde{v}$ into a discrete (qualitative) variable v. These landmarks divide the infinite set of values for the continuous variable $\tilde{v}$ into a discrete set of qualitative values $\mathcal{Q}(v)$ called a *quantity space* [Kuipers, 1994]. A quantity with two landmarks might be described by (v_1^*, v_2^*), which implies five distinct qualitative values, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$. For example, if $v_1^* = 100.0$ and $v_2^* = 200.0$ and $\tilde{v}(t) = 287.53$ then $v(t) = (v_2^*, +\infty)$. This allows the agent to work with discrete values.

Once we have qualitative values, we can define *events*. If a is a qualitative value of a discrete variable A, meaning $a \in \mathcal{Q}(A)$, then the *event* $E = A_t{\rightarrow}a$ is defined by $A(t-1) \neq a$ and $A(t) = a$. That is, an event takes place when a discrete variable A changes to value a at time t, from some other value.

Once the agent can recognize events, it can learn *contingencies*. We define a contingency $E_1 \Rightarrow E_2$ to consist of two events, the *antecedent* E_1, and the *consequent* E_2 such that if the antecedent event E_1 occurs then the consequent event E_2 is more likely to occur within a short time period (currently 0.25 seconds) after E_1 than it would have otherwise.

For each contingency $E_1 \Rightarrow E_2$, the agent creates a *predictive rule* $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ to maintain statistics for when the contingency holds. These statistics are over the predicate $succeeds(r)$ that is true when the contingency holds. Using the statistics gathered on $succeeds(r)$, the agent learns a context $\mathcal{C}$ that consists of a set of qualitative variables. The context induces a conditional probability table $CPT(r)$ on the predicate $succeeds(r)$. In Bayesian network terminology, the variables in $\mathcal{C}$ are the parents of $succeeds(r)$. This context allows the agent to predict when the contingency will hold.

The agent also learns new landmarks by using $succeeds(r)$ as a supervisory signal. It does this by storing the real values of all variables for the last 200 activations. This allows the agent to find new landmarks using the standard cutpoint detection algorithm of Fayyad and Irani [2003] . Inserting a new landmark v^* into (v_i^*, v_{i+1}^*) allows that interval to be replaced in $\mathcal{Q}(v)$ by two intervals and the dividing landmark: (v_i^*, v^*), v^*, (v^*, v_{i+1}^*). Adding this new landmark into the quantity space $\mathcal{Q}(v)$ refines existing rules and also allows the agent to learn new contingencies involving the events $v{\rightarrow}(v_i^*, v^*)$, $v{\rightarrow}v^*$, and $v{\rightarrow}(v^*, v_{i+1}^*)$.

Once the predictive rules are learned they can be used for backchaining as described in [Mugan and Kuipers, 2007a]. During backchaining, the agent can immediately set the values of its motor variables. If the agent desires to bring about an event E_2 that is not on a motor variable, then it must look for a rule $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$. If the event E_1 is on a motor variable the agent can immediately set it, otherwise it must look for a rule that predicts E_1, continuing in this fashion until a motor variable is reached. This allows the agent to achieve goals that consist of single events.

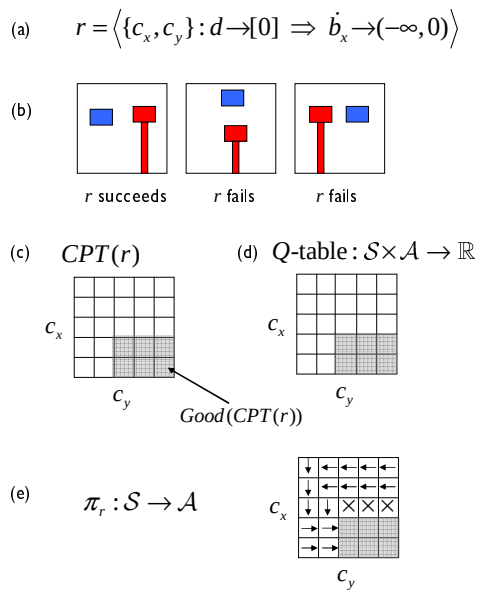

Figure 2: (a) The rule $r = \langle \{c_x, c_y\} : d \rightarrow [0] \Rightarrow \dot{b}_x \rightarrow (-\infty, 0) \rangle$ is an example of a rule learned by the robot. It states that if the distance d between the hand and the block goes to 0, then the event $\dot{b}_x \rightarrow (-\infty, 0)$ of the block moving to the left will occur. The predicted success of this rule depends on the context variables c_x and c_y that give the location of the hand in the frame of reference of the block. (b) The agent gathers experience in the world to learn the context values for which r is successful. The agent learns that the hand must be to the right of and level with the block for r to be successful. (c) Based on $\mathcal{C} = \{c_x, c_y\}$ the agent creates a conditional probability table $CPT(r)$ for r and uses a threshold to determine the set $Good(CPT(r))$ of values of $\mathcal{C}$ for which the rule r is likely to succeed. (d) The agent can then define a simple reinforcement learning problem in which $\mathcal{C}$ defines the state space $\mathcal{S}$, and $Good(CPT(r))$ defines the goal states. The agent is rewarded for reaching a state in which the rule r is likely to succeed. To do this, the agent creates a Q-table that maps $\mathcal{S} \times \mathcal{A}$ to a value, where $\mathcal{A}$ is the set of primitive actions (defined by the qualitative values of the motor variables u_x and u_y). (e) The agent then defines a policy π_r by associating each cell in $\mathcal{S}$ with the primitive action with maximum value.

3. Generating Reinforcement Learning Problems

The abstraction learning method described in the previous section results in a set of landmarks on variables and a set of predictive rules. For each rule $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ the agent has learned a conditional probability table $CPT(r)$ over the variables in the context $\mathcal{C}$ on the success of the contingency $E_1 \Rightarrow E_2$. To use rule r to form a complete action to bring about event E_2, the agent needs to get into the part of the state space that $CPT(r)$ predicts will lead to the successful completion of E_2. We refer to this part of the state space as $Good(CPT(r))$ where

$$Good(CPT(r)) = \{q \in \mathcal{Q}(\mathcal{C}) \quad | Pr(succeeds(r)|q) > \theta_{sr}\} \tag{1}$$

where the constant θ_{sr} was set by hand to be 0.6 based on experiments. The agent learns a policy π_r to get into this good part of the state space $Good(CPT(r))$ using reinforcement learning. The result is that the policy π_r coupled with the ability to bring about E_1 through backchaining serve as a complete action to bring about E_2. An example of this process is shown in Figure 2.

The agent autonomously generates a reinforcement learning problem to learn a policy π_r for each rule $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ that has a context $\mathcal{C}$ that contains more than one variable and $Good(CPT(r))$ is not empty. (Recall that goals over single a variable can be achieved by backchaining.) The state space $\mathcal{S}$ for the problem based on r is determined by the variables $\{v_1, \ldots, v_n\}$ in the context $\mathcal{C}$

$$\mathcal{S} = \mathcal{Q}(v_1) \times \ldots \times \mathcal{Q}(v_n)$$

Continuous action spaces are challenging for reinforcement learning agents, but this agent is able to take advantage of the landmarks learned on its motor variables to create an action space $\mathcal{A}$. To do this, we define a set

$$\mathcal{Q}(U) = \mathcal{Q}(u_1) \times \ldots \times \mathcal{Q}(u_n)$$

where $u_1, \ldots, u_n$ is the set of motor variables. We can then define a primitive action $a \in \mathcal{A}$ as choosing a $u \in \mathcal{Q}(U)$, taking random values from the ranges of the qualitative values in u, and maintaining those values until the state $\mathcal{S}$ changes or the real values underlying the variables that make up $\mathcal{S}$ stop changing. The reward function is a pseudo-reward function that is defined using a goal-reward representation [Koenig and Simmons, 1996] where the set of goal states is $Good(CPT(r))$.

Using this representation, the agent learns a value-action function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. It does this using the Sarsa(λ) algorithm [Sutton and Barto, 1998] where $\lambda = 0.9$, α is 1 divided by the number of times the state-action combination has been taken, and the discount parameter $\gamma = 0.9$.

During learning, the agent uses ϵ-greedy action selection where $\epsilon = 0.25$. A learning episode begins when the rule is invoked by the agent, and it ends when the agent makes it to a goal state or when 20 primitive actions have been taken. Once the value-action function is learned, a policy π_r can be simply to choose the best action for each state $s \in \mathcal{S}$. An example of a policy that is learned by the agent is shown in Figure 3.

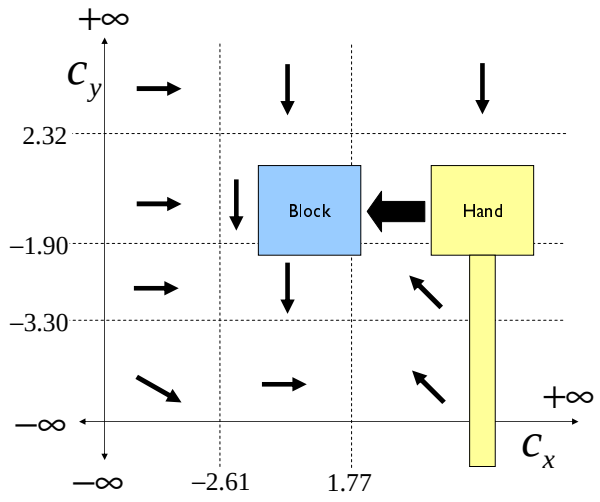

Figure 3: This figure shows the learned policy π_r of a reinforcement learning problem for hitting the block to the left. The state space $\mathcal{S}$ is determined by the landmarks on the variables in the context $\mathcal{C} = \{c_x, c_y\}$ of the rule r. The variables c_x and c_y give the location of the hand relative to the center of the block in the x and y directions respectively, and the dotted lines represent landmarks on c_x and c_y. Because both the hand and the block are squares with a diameter of 2.0, a perfect set of landmarks for both c_x and c_y would be $\{-2.0, +2.0\}$. The small arrows represent the action given by policy π_r for each state. To hit the block to the left, the hand must be on the right side, and so the policy directs the hand under the block to the right-hand side. The large arrow represents the motor command issued by backchaining to complete the action.

4. Exploration

In [Mugan and Kuipers, 2007a] the agent explored by picking goals it could achieve with moderate success and working to achieve them. In this work, the agent is motivated by the drive to improve the reliability of predictive rules. The agent explores by continually activating a predictive rule $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ and then working to bring about the contingency $E_1 \Rightarrow E_2$ (to keep from getting in a rut, the agent may take a random motor babbling action with probability 0.2). Exploration follows a developmental pattern that begins with the contingencies close to the agent's motor variables and progressively moves to external objects. This occurs because the agent learns contingencies outward from its motor variables. The agent learns the contingency $E_1 \Rightarrow E_2$ if event E_2 is more likely to occur if event E_1 has occurred, and the agent already has a rule that reliably predicts the event E_1.

When a rule r is activated, if the current state of the agent is not in the set of states $Good(CPT(r))$, then the agent first works to bring the agent to one of those states by using backchaining if the context consists of a single variable, or by using reinforcement learning if the context consists of more than one variable. The agent then works to bring about the event E_1 by backchaining.

The agent's learning is undirected. The set of possible predictive rules, and thus the set of possible learning targets, is determined by the abstraction learning process. During exploration the agent must continually choose which of the rules to pursue. To do this, the agent uses two criteria. The first criteria is the increase in reliability I_r of rule r. If we use $brel_t$ to denote the current best reliability of r and $brel_{t-\tau}$ to be the best reliability that r had τ activations ago, then

$$I_r = brel_t - brel_{t-\tau} \tag{2}$$

This first criterion is inspired by the calculation of similarity-based learning progress of Oudeyer, Kaplan, and Hafner [2007].

However, this may still leave too many choices for the agent. Given the current context of the agent, we want to focus its attention on contingencies that it can reliably bring about. To do this, the agent uses a second criteria, the predicted reliability P_r of rule r in the current context, to add a "rich get richer" element to its choice. The predicted reliability P_r is defined as

$$P_r = Pr(succeeds(r)|\mathcal{C}, s) \tag{3}$$

where $Pr(succeeds(r)|\mathcal{C}, s)$ means the probability of success of r in the current state s.

The agent then calculates the weight w_r of a rule r as $w_r = max(\epsilon, I_r P_r)$ where $\epsilon = 0.001$ to ensure that no rule is completely ignored. It then chooses rule r with probability p_r where

$$p_r = \frac{w_r}{\sum_i w_{r_i}} \tag{4}$$

This method of calculating the weights so that $w_r = max(\epsilon, I_r P_r)$ was chosen because it worked well in experiments. A more rigorous comparison of different methods of exploration would be an interesting area of future work.

5. Experimental Evaluation

It is difficult to directly evaluate an autonomous learning agent. During training the agent is not given any external task and decides on its own what to learn. We therefore evaluate our agent indirectly by picking a task that seems natural to an outside observer. We then determine how well the agent can use the knowledge it has acquired to perform this task. To evaluate the claim that the agent has learned to perform an action, we compare the performance of our autonomous agent on this task to the

performance of a hand-created reinforcement learner trained only on this task.

We evaluate our algorithm using the simulated agent shown in Figure 4. The evaluation task we have chosen is for the agent to hit the block in a specified direction. The agent learns to hit the block during its undirected exploration by learning contingencies that predict the movement of the block. These contingencies generate reinforcement learning problems that allow the agent to learn to get into the right position to hit the block, and once in that position it can backchain to the motor variables to make the hand movement that strikes the block. We trained ten agents total, five autonomous agents described in this paper, and five hand-created learning agents. Videos of the autonomous agent can be viewed at www.cs.utexas.edu/~jmugan/DevelopmentalAgent.

We trained each agent in the environment shown in Figure 4 for 340,000 timesteps (almost five hours of physical experience). During this time, the hand-created agents continually repeated episodes of the task, and the autonomous agents performed the learning algorithm described in this paper. During training of the autonomous agents, if the block fell off the tray, moved out of reach of the agent, or was not moved for an extended time, the block was moved to a random location within reach of the agent. For all agents we stored the state of each agent's knowledge every 20,000 timesteps during training (corresponding to about sixteen minutes of physical experience). We then ran the evaluation for each agent using their respective stored knowledge bases.

Each evaluation consisted of 100 trials. At the beginning of each trial the block was placed in a random location within reach of the agent and the evaluator picked one of three goals: hitting the block to the left, hitting the block to the right, or hitting the block forward. The agent then had 300 timesteps to use its knowledge to hit the block in the correct direction. A trial was terminated unsuccessfully if the agent hit the block in the wrong direction. The evaluation metric was the success rate for hitting the block during the 100 trials.

We tested both types of agents under two goal selection regimes, *uniform* and *hard*. During both uniform and hard goal selection, the evaluator selects the goal randomly, with a uniform distribution, and filters out goals that are impossible to achieve. (For example, if the block is on the far left, the agent cannot get its hand on the left side of the block to move it to the right.) During *hard* goal selection, *easy* goals are also filtered out, where a goal is easy if it can be achieved with a single straight-line motion. (During the training period for the hand-created agents, a goal selection regime was randomly chosen at the beginning of each episode, and then the goal was chosen based on that regime.)

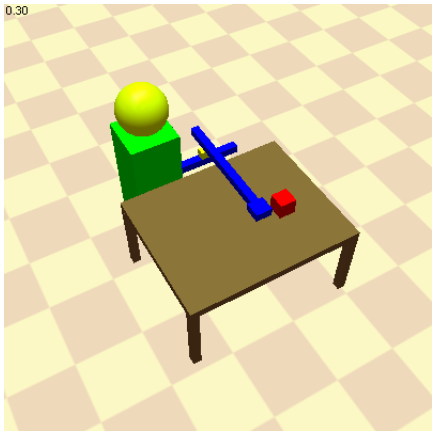

Figure 4: A simulated agent and environment is implemented in Breve [Klein, 2003]. It has a torso with a 2-dof orthogonal arm and is sitting in front of a tray with a block. The robot has two motor variables $\tilde{u}_x$ and $\tilde{u}_y$ that move the hand in the x and y directions, respectively. The perceptual system creates variables for each of the two tracked objects in this environment: the hand and the block. The hand is described by two continuous variables $\tilde{h}_x(t)$, $\tilde{h}_y(t)$ that represent the location of the hand in the x and y directions, respectively. The variables corresponding to the block are $\tilde{b}_x(t)$, $\tilde{b}_y(t)$, and $b_a(t)$. The variables $\tilde{b}_x(t)$, and $\tilde{b}_y(t)$ have the same respective meanings as the variables for the hand, and the Boolean variable $b_a(t)$ represents whether the block is in view. The relationship between the hand and the block is represented by the continuous variables $\tilde{c}_x(t)$, $\tilde{c}_y(t)$, and $\tilde{d}(t)$. The variables $\tilde{c}_x(t)$ and $\tilde{c}_y(t)$ represent the coordinates of the center of the hand in the frame of reference of the center of the block, and the variable $\tilde{d}(t)$ represents the distance between the hand and the block. For each continuous input variable $\tilde{v}(t)$ there is also a direction of change variable $\dot{v}(t)$, giving a total of seventeen variables. The values of all variables are updated by perceptual trackers at each timestep as the objects move.

5.1 The Hand-Built Learner

The hand-built reinforcement learning agents used linear, gradient-descent Sarsa(λ) with binary features [Sutton and Barto, 1998] where the binary features come from tile coding. We chose this method because tile coding is a standard method for coping with continuous variables in reinforcement learning [Santamaria et al., 1997]. Tile coding works by using multiple partitions of the state space such that each partition (tiling) is offset just a little from the others. This allows the agent to generalize more effectively than using a single partition with higher resolution.

We now explain the details of the tile coding implementation. The motor variables u_x and u_y were each divided into 10 equal-width bins, and the direction of change variables were each divided into 3

bins: $(-\infty, -0.05), [-0.05, 0.05], (0.05, \infty)$. The goal was represented with a discrete variable that took on three values, one for each of the three goals. The remaining variables were treated as continuous. There were 16 tilings, the tiling was done using a hashing function with a memory size of 65,536. The parameter values used were $\lambda = 0.9$, $\gamma = 0.9$, and $\alpha = 0.1$. To prevent the task diameter from being too high, during both training and testing the agent chose a new action every 10 timesteps (0.5 seconds). Action selection was ϵ-greedy where $\epsilon = 0.05$.

5.2 Results

The results are shown in Figure 5 and Figure 6. As the agent gains more experience in the world its ability to perform the task improves. We also see that the agent has indeed learned the action as its performance under both the difficult and uniform task selection regime is comparable to that of the hand-created learner.

The hand-created learner enjoys the advantage of only being trained on the evaluation task. But the hand-created learner is at an important disadvantage, it does not know which variables are important for the task. Our agent learns which variables are important autonomously by adding them to the contexts of predictive rules, and this allows it to perform comparably even though it learns more than how to perform the evaluation task. For example, our agent learns when the block will disappear off the tray. It does this by learning landmarks on b_x and b_y delimiting the edges of the tray suggested by the event of the block disappearing $b_a \rightarrow \mathsf{false}$. It then learns a contingency when says when b_x reaches the edge that the block will disappear. It also learns the limits of its movement by learning landmarks on the maximum values h_x and h_y. When these variables are added to the contexts of the rules that predict how the hand moves, they specify that the hand can move in a direction if it is not at the limit. The agent also learns to move away from the block instead of towards it.

The agent can use the knowledge learned during one task to learn another. Of particular importance is that the agent learns a discretization of the action space. Each motor variable is initially given a landmark at 0, but it takes a force of 300 in the simulator to move the arm in any direction. Our agent finds those important landmarks and can then use that knowledge when learning a new task. In contrast, the hand-created learner would need to be trained from scratch for each new task and would not be able to easily use what it learned in previous tasks for future tasks.

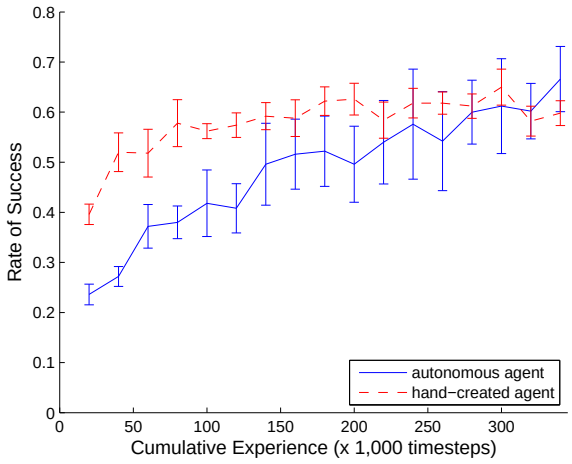

Figure 5: Results under the hard goal selection regime. Both the autonomous agent and the hand-created agent improve with experience. The hand-created agent training only on the task initially has better performance, but the autonomous agent later matches it. All error bars are standard error.

6. Discussion

During a typical run the agent creates an average of 30.80 ($s = 3.77$) reinforcement learning problems. These reinforcement learning problems include those that allow the hand to move into position to hit the block, as is shown in Figure 3, those that move the hand into position to change the relationship between the hand and the block, and some that do not appear useful to an outside observer. Each of these reinforcement learning problems is relatively small because the learned abstraction provides a relevant set of state variables and landmarks. This allows for rapid learning.

One concern with learning algorithms that construct knowledge is computational efficiency. To learn the contingencies the agent must maintain statistics on every pair of events. This can be expensive, but it is not prohibitive because with the assumption of a maximum number of landmarks per variable, the number of pairs of events is $\mathcal{O}(n^2)$ where n is the number of variables. In this case, the agent handles the learning of contingencies with no difficulty, but a focus of attention would be necessary with a large number of variables. The agent must also maintain statistics on each rule learned. In a typical run the agent learns about 250 to 300 rules. In this case we drop rules if their reliability falls too low, but more sophisticated methods could be employed.

One important feature that our environment does not have is variables whose behavior is highly dependent upon the values of other variables. One example of this is a serial arm, this is currently under inves-

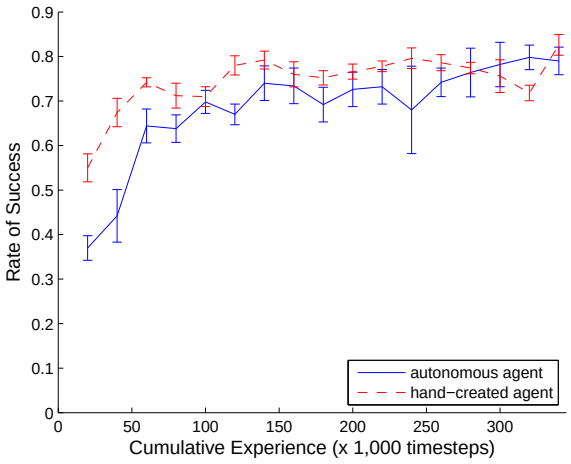

Figure 6: Results under the uniform goal selection regime. As with the hard goal selection regime, both the autonomous agent and the hand-created agent improve with experience, and the autonomous later matches the performance of the hand-created agent. All error bars are standard error.

tigation.

7. Related Work

In this work we have shown how an agent can autonomously learn a model of its environment and generate its own reinforcement learning problems. The model learning portion of [Mugan and Kuipers, 2007a] and [Mugan and Kuipers, 2007b] is inspired by Drescher [1991], although we move beyond that work by learning in a continuous environment. A related body of work [Zettlemoyer et al., 2005, Pasula et al., 2007] presents a method that allows an agent to learn probabilistic planning rules. Using first-order logic they learn rules that given a context and an action provide a distribution over results. These rules then allow them to do planning. Our work can be considered complementary to this work because our algorithm allows the agent to learn a discrete state space and a set of basic actions, both of which are assumed by their algorithm.

The generation of reinforcement learning problems is closely related to learning options. Options [Sutton et al., 1999] allow an agent to break up a large reinforcement learning problem into a set of temporally extended actions. McGovern and Barto [2001] proposed a method whereby an agent autonomously finds subgoals based on bottleneck states that are visited often during successful trials. Subgoals have also been found by searching for "access states" [Simsek and Barto, 2004, Simsek et al., 2005] that allow the agent to go from one part of the state space to another. We define the subgoals for reinforcement learning to the regions defined by landmarks in which a predictive rule is reliable. Our focus on learning distinctions also distinguishes our work from the hierarchical reinforcement work of Bakker and Schmidhuber [2004], which clusters low-level observations.

The agent can also define options to achieve salient states [Barto et al., 2004,Singh et al., 2005]. Bonarini et. al. [2006] define options for states that are rarely reached or are easily left once reached. Both of these bodies of work define an intrinsic reward signal based on prediction error, motivating the agent to explore parts of the space for which it currently does not have a good model. This form of intrinsic motivation is also used in [Huang and Weng, 2002, Marshall et al., 2004]. However, focusing attention to states where the model has poor prediction ability can cause the agent to explore spaces where learning is too difficult. Our agent's intrinsic motivation is inspired by the work on intelligent adaptive curiosity [Oudeyer et al., 2007] that motivates the agent to explore parts of the space where it is making progress in learning the results of actions. We add an additional factor to motivate the agent to not only explore where it is making progress, but also where it already has some competency. This helps to further focus the agent's attention, which may become increasingly important as developmental agents explore more complex environments.

8. Conclusion and Future Work

The contribution of this paper is a method for enabling an undirected learning agent to apply reinforcement learning, allowing it to handle situations where actions have significant interactions. The agent discretizes its continuous environment while learning contingencies. The subspaces where these contingencies are reliable then serve as goal states for the agent. This allows the agent to create its own reinforcement learning problems to reach those states. These reinforcement learning problems are made simple because the state space has been discretized in a way that aligns with the natural environment, and because the relevant variables have been identified.

Currently, hierarchical reinforcement learning [Barto and Mahadevan, 2003] is an active area of research. Of particular interest is how an agent can autonomously build its own hierarchy. The work presented here provides a step towards that goal. Central to our method are contingencies linking an antecedent E_1 event with a consequent event E_2, wrapped in a rule that has a context that says when the contingency will be reliable. Both this context and the antecedent provide natural subtasks. In future work we will explore how these subtasks can be put together into a coherent whole.

Acknowledgements

This work has taken place in the Intelligent Robotics Lab at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Intelligent Robotics lab is supported in part by grants from the Texas Advanced Research Program (3658-0170-2007), from the National Science Foundation (IIS-0413257, IIS-0713150, and IIS-0750011), and from the National Institutes of Health (EY016089).

References

Bakker, B. and Schmidhuber, J. (2004). Hierarchical Reinforcement Learning Based on Subgoal Discovery and Subpolicy Specialization. *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pages 438–445.

Barto, A. and Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

Barto, A., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *Proc. of the 3rd Int. Conf. on Developmental Learning*.

Bonarini, A., Lazaric, A., and Restelli, M. (2006). Incremental Skill Acquisition for Self-Motivated Learning Animats. *Lecture Notes in Computer Science*, 4095:357.

Drescher, G. L. (1991). *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, Cambridge, MA.

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuousvalued attributes for classification learning. In *Proc. Int. Joint Conf. on Articial Intelligence*, volume 2, pages 1022–1027.

Huang, X. and Weng, J. (2002). Novelty and Reinforcement Learning in the Value System of Developmental Robots. *Proc. 2nd Inter. Workshop on Epigenetic Robotics*.

Klein, J. (2003). Breve: a 3d environment for the simulation of decentralized systems and artificial life. In *Proc. of the Int. Conf. on Artificial Life*, pages 329–334.

Koenig, S. and Simmons, R. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1):227–250.

Kuipers, B. (1994). *Qualitative Reasoning*. The MIT Press, Cambridge, Massachusetts.

Marshall, J., Blank, D., and Meeden, L. (2004). An emergent framework for self-motivation in developmental robotics. *Proc. of the 3rd Int. Conf. on Development and Learning (ICDL 2004), Salk Institute, San Diego*.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. Int. Conf. on Machine Learning*, pages 361–368.

Mugan, J. and Kuipers, B. (2007a). Learning distinctions and rules in a continuous world through active exploration. In *Proc. of the Int. Conf. on Epigenetic Robotics*.

Mugan, J. and Kuipers, B. (2007b). Learning to predict the effects of actions: Synergy between rules and landmarks. In *Proc. of the Int. Conf. on Development and Learning*.

Oudeyer, P., Kaplan, F., and Hafner, V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *Evolutionary Computation, IEEE Transactions on*, 11(2):265–286.

Pasula, H., Zettlemoyer, L., and Kaelbling, L. (2007). Learning Symbolic Models of Stochastic Domains. *Journal of Artificial Intelligence Research*, 29:309–352.

Santamaria, J., Sutton, R., and Ram, A. (1997). Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2):163.

Simsek, O. and Barto, A. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proc. of the Twenty-First Int. Conf. on Machine Learning*, pages 751–758.

Simsek, O., Wolfe, A., and Barto, A. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proc. of the Twenty-Second Int. Conf. on Machine Learning*, pages 816–823.

Singh, S., Barto, A., and Chentanez, N. (2005). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems*, 17:1281–1288.

Sutton, R., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

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

Zettlemoyer, L. S., Pasula, H., and Kaelbling, L. P. (2005). Learning planning rules in noisy stochastic worlds. In *AAAI*, pages 911–918.