Autonomously Learning an Action Hierarchy Using a Learned Qualitative State Representation

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

Benjamin Kuipers
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109 USA
kuipers@umich.edu

Abstract

There has been intense interest in hierarchical reinforcement learning as a way to make Markov decision process planning more tractable, but there has been relatively little work on autonomously learning the hierarchy, especially in continuous domains. In this paper we present a method for learning a hierarchy of actions in a continuous environment. Our approach is to learn a qualitative representation of the continuous environment and then to define actions to reach qualitative states. Our method learns one or more options to perform each action. Each option is learned by first learning a dynamic Bayesian network (DBN). We approach this problem from a developmental robotics perspective. The agent receives no extrinsic reward and has no external direction for what to learn. We evaluate our work using a simulation with realistic physics that consists of a robot playing with blocks at a table.

1 Introduction

Reinforcement learning (RL) is a popular method for enabling agents to learn in unknown environments [Sutton and Barto, 1998]. Much work in RL focuses on learning to maximize a reward given a set of states S and a set of actions A. In this paper we focus on continuous environments and we present a method for *learning* a qualitative state representation S^* and a hierarchical set of qualitative actions A^* using only intrinsic reward. We call the actions that the agent learns *qualitative actions* because they allow the agent to reach qualitative states. We call our algorithm the Qualitative Learner of Actions and Perception, QLAP.

QLAP makes use of the options framework [Sutton *et al.*, 1999] and learns one or more options to perform each qualitative action, where each option serves as a different way to perform the action. Boutilier [1995] proposed making MDP planning more tractable by exploiting structure in the problem using dynamic Bayesian networks (DBNs). In QLAP there is a one-to-one correspondence between DBNs and options, and each option is treated as a small MDP problem. Each option is learned by first learning a small dynamic Bayesian network (DBN). The variables in the DBN determine the initial state space for the option. This leads to a state abstraction in the option, because the variables in the DBN are a subset of the available variables. The conditional probability table of the DBN is used to specify the transition function for the option. Since the option's state space is small, the policy can then be learned using dynamic programming.

In QLAP, the agent begins with a very coarse discretization that indicates if the value of each variable is increasing, decreasing, or remaining steady. Using this discretization, the agent first motor babbles and then explores by repeatedly choosing a qualitative action and an option to achieve that action, and then following the policy of that option. While it is exploring it is learning DBNs. Once a DBN is sufficiently deterministic an option is created based on it. The agent also learns new distinctions to improve previously learned DBNs. These new distinctions update the agent's qualitative state representation.

We consider the options that QLAP learns to be *hierarchical options* because (1) they invoke qualitative actions instead of primitive actions, (2) they use a state abstraction that is specific to the option, and (3) they use a pseudo-reward to learn the policy independent of the calling context. The disadvantage of this hierarchical approach is that the agent may not always find the optimal solution. One important advantage is that the temporal abstraction afforded by the options decreases the *task diameter* (the number of actions needed to achieve the goal) and therefore reduces the amount of time that the agent spends randomly exploring. Another important advantage is that the agent is able to ignore variables that are not necessary to complete the task. QLAP options are each created to achieve a goal, and since they use a state abstraction and a pseudo-reward, they share similarities with MAXQ subtasks [Dietterich, 1998].

In [Mugan and Kuipers, 2007] a method was proposed for learning both a discretization and small models to describe the dynamics of the environment. In [Mugan and Kuipers, 2008] a method was proposed for enabling an agent to use those models to autonomously formulate reinforcement learning problems. The contributions of this paper are to put the learned small models into the DBN framework and to organize the formulated reinforcement learning problems into a hierarchy using options.

We first discuss how QLAP learns the action hierarchy. We then discuss how the agent executes QLAP. We then evaluate

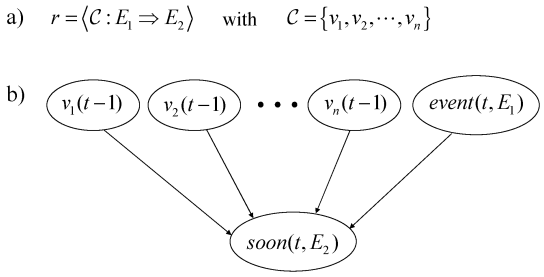


Figure 1: Correspondence between QLAP DBN notation and traditional graphical DBN notation. (a) QLAP notation of a DBN. Context $\mathcal{C}$ consists of a set of qualitative variables. Event E_1 is an *antecedent event* and event E_2 is a *consequent event*. (b) Traditional graphical notation. Boolean variable $event(t, E_1)$ is true if event E_1 occurs at time t. Boolean variable $soon(t, E_2)$ is true if $event(t, E_1)$ is true and event E_2 occurs within k timesteps of t. The conditional probability table (CPT) gives the probability of $soon(t, E_2)$ for each value of its parents. For all elements of the CPT where $event(t, E_1)$ is false, the CPT gives a probability of 0. The remaining probabilities are learned through experience.

QLAP using a simulated robot with realistic physics, and we show that using QLAP the agent can learn an action hierarchy that allows it to perform both temporal abstraction and state abstraction to complete the task of hitting a block. We then discuss related work and conclude.

2 Learning the Action Hierarchy

2.1 Qualitative Representation

QLAP uses a qualitative representation to abstract the continuous world. A qualitative representation allows an agent to cope with incomplete information and to focus on important distinctions while ignoring unimportant ones [Kuipers, 1994]. A qualitative state representation includes both the continuous state variables, called *magnitude* variables, as well as *change* variables that encode the direction of change (increasing, decreasing, or steady) of each magnitude variable. The values of the variables are represented qualitatively using *landmarks*, and the value of a qualitative variable can be either at a landmark or between two landmarks. In this way, landmarks can convert a continuous variable $\tilde{v}$ with an infinite number of values into a qualitative variable v with a finite set of qualitative values $\mathcal{Q}(v)$ called a *quantity space* [Kuipers, 1994]. A quantity space $\mathcal{Q}(v) = L(v) \cup I(v)$, where $L(v) = \{v_1^*, \cdots, v_n^*\}$ is a totally ordered set of landmark values, and $I(v) = \{(-\infty, v_1^*), (v_1^*, v_2^*), \cdots, (v_n^*, +\infty)\}$ is the set of mutually disjoint open intervals that $L(v)$ defines in the real number line. A quantity space with two landmarks might be described by (v_1^*, v_2^*), which implies five distinct qualitative values, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$.

The magnitude variables initially have no landmarks, and the agent must learn the important distinctions. Each direction of change variable $\dot{v}$ has a single intrinsic landmark at 0, so its quantity space is $\mathcal{Q}(\dot{v}) = \{(-\infty, 0), 0, (0, +\infty)\}$. *Motor* variables are a third type of qualitative variable. They are also given an initial landmark at 0, but the agent must learn further distinctions. For example, in the evaluation the agent learns that it takes a force of at least 300.0 to move the hand.

Using a qualitative representation allows the agent to focus its attention on *events* and ignore unimportant sensory input. We define the event $X_t \rightarrow x$ to be when $X(t-1) \neq x$ and $X(t) = x$, where $x \in \mathcal{Q}(X)$ is a qualitative value of a qualitative variable X.

2.2 Qualitative Actions

The qualitative representation defines a set of qualitative actions. QLAP creates a qualitative action $qa(v, q)$ to achieve each qualitative value $q \in \mathcal{Q}(v)$ for each qualitative variable v. There are three types of qualitative actions corresponding to the three types of qualitative variables: motor, magnitude, and change. A motor action $qa(v, q)$ sets $\tilde{v}$ to a random continuous value within the range covered by q.

Magnitude and change actions are high-level actions. When a magnitude or change action is executed it chooses an option and executes the policy for that option. Each option is associated with only one qualitative action, and the actions that an option can invoke are qualitative actions.

A magnitude action $qa(v, q)$ has one option to achieve $v = q$ if the value of v is currently less than q, and another option to achieve $v = q$ if the value of v is currently greater than q. A change action $qa(v, q)$ may have multiple options to choose from to achieve $v = q$. To learn each option, QLAP first learns a DBN. There is a one-to-one correspondence between DBNs and options.

2.3 DBN Representation

In Mugan and Kuipers [2007] a method was presented for learning small models to predict qualitative values of variables. In this paper we put those models into the dynamic Bayesian network framework. We refer to these models as dynamic Bayesian networks and not simply Bayesian networks because we are using them to model a dynamic system. The notation we use for these DBNs is $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ where $\mathcal{C}$ is a set of qualitative variables that serves as a context, event $E_1 = X \rightarrow x$ is the *antecedent event*, and event $E_2 = Y \rightarrow y$ is the *consequent event* (see Figure 1). This DBN r can be written as

$$r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle \quad (1)$$

DBN r gives the probability that event $Y{\rightarrow}y$ will soon follow event $X{\rightarrow}x$ for each qualitative value in $\mathcal{C}$. Focusing only on timesteps in which event $X{\rightarrow}x$ occurs helps to focus the agent's attention to make learning more tractable. And using a time window for event $Y \rightarrow y$ allows the DBN to account for influences that may take more than one timestep to manifest. Notice that these DBNs differ from the typical DBN formulation, e.g. [Boutilier *et al.*, 1995], in that there is no action associated with the DBN. This is because QLAP does not begin with a set of primitive actions, it only assumes that the agent has motor variables. The DBNs in QLAP are tied to the agent's motors because the antecedent event of the DBN can be on a motor variable.

To learn each DBN, QLAP finds a pair of events E_1 and E_2 such that E_2 is more likely to occur soon given that E_1

has occurred than otherwise. It then creates a DBN with an empty context. QLAP then iteratively adds context variables that improve the predictive ability of the DBN (cf. marginal attribution [Drescher, 1991]). Additionally, predicting when the Boolean child variable will be true is a supervised learning problem. This formulation allows the agent to learn new landmarks (distinctions) that improve the predictive ability of the DBN (see [Mugan and Kuipers, 2007; 2008] for details.)

The DBNs we have just discussed predict events on change variables and are called *change* DBNs. QLAP also uses *magnitude* DBNs to predict events on magnitude variables. For each magnitude variable v and each qualitative value $q \in \mathcal{Q}(v)$, QLAP creates two DBNs, one that corresponds to approaching the value $v = q$ from below on the number line, and another that corresponds to approaching $v = q$ from above. Magnitude DBNs are similar to change DBNs. For example, if $v < q$ and the robot successfully performs the action to achieve $\dot{v} = (0, +\infty)$, then the DBN gives the probability, for each value of the variables in the context, of event $v{\rightarrow}q$ occurring before $\dot{v} \neq (0, +\infty)$. Magnitude DBNs have no concept of "soon," as long as $v < q$ and $\dot{v} = (0, +\infty)$, the agent will wait for $v{\rightarrow}q$.

For a DBN r, we denote the probability of the child variable being true in state s by $CPT_r(s)$ (we only consider the cases where the non-context parent variable is true). We call the highest probability $CPT_r(s^*)$ for any state s^* the *best reliability* of DBN r.

2.4 Options

An option [Sutton *et al.*, 1999] is like a subroutine that can be called to perform a task. An option o_i is typically expressed as the triple $o_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$ where $\mathcal{I}_i$ is a set of initiation states, π_i is the policy, and β_i is a set of termination states or a termination function. Options in QLAP follow this pattern except that π_i is a policy over qualitative actions instead of being over primitive actions or options. Additionally, since each option in QLAP learns its policy using its own state abstraction, associated with option o_i is a state space $\mathcal{S}_i$, a set of qualitative actions $\mathcal{A}_i^s$ for each state $s \in \mathcal{S}_i$, and a transition function $T_i : \mathcal{S}_i \times \mathcal{A}_i^s \rightarrow \mathcal{S}_i$.

QLAP creates a *magnitude option* for each magnitude DBN. QLAP creates a *change option* for each change DBN where (1) the best reliability is estimated to be greater than $\theta_r = 0.75$ and (2) the antecedent event can be achieved with an estimated probability greater than $\theta_r = 0.75$ and (3) the option would not create a cycle of change options. (We also limit the number of change options to 3 for any qualitative action). The goal of the option is to make the child variable of the DBN true. If this occurs, then the option *succeeds*. If the option succeeds, then the qualitative action that invoked it also *succeeds*.

Creating an Option from a DBN

When an option o_r is created for a DBN r, the set of initiation states $\mathcal{I}_r$ is the set of all states, and the termination function β_r terminates o_r when it *succeeds* (the child variable becomes true) or when it becomes stuck for 10 timesteps or exceeds resource constraints (300 timesteps, or 5 suboption calls). To learn the policy π_r, QLAP uses the DBN to create a transition function $T_r : \mathcal{S}_r \times \mathcal{A}_r^s \rightarrow \mathcal{S}_r$ and then learns the Q-table using dynamic programming with value iteration [Sutton and Barto, 1998]. The pseudo-reward is 10.0 for reaching the goal and a transition cost of 0.50 is imposed for each transition.

We now describe how QLAP constructs the state space $\mathcal{S}_r$, the set of available actions $\mathcal{A}_r^s$, and the transition function T_r for option o_r from a change DBN $r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$. Recall that $\mathcal{Q}(v)$ is the set of qualitative values for qualitative variable v. If we define the set $Z_r = \mathcal{C} \cup X \cup Y$, then the state space $\mathcal{S}_r$ for option o_r is $\mathcal{S}_r = \prod_{v \in Z_r} \mathcal{Q}(v)$. (We will see in Section 2.5 how more variables can be added to state spaces.)

Recall that the notation $qa(v, q)$ means the qualitative action to bring about $v = q$. The set of qualitative actions $\mathcal{A}_r^s$ available in state $s \in \mathcal{S}_r$ is

$$\mathcal{A}_r^s = \mathcal{A}_\mathcal{C} \bigcup \{qa(X, x)\} - \{qa(v, q) | s(v) = q\} \quad (2)$$

The definition of $\mathcal{A}_r^s$ consists of three parts. (1) The qualitative actions $\mathcal{A}_\mathcal{C}$ allow the agent to move within the context

$$\mathcal{A}_\mathcal{C} = \{qa(v, q) | v \in \mathcal{C} \text{ and } q \in \mathcal{Q}(v)\} \quad (3)$$

(although any action on the corresponding magnitude variable of Y is excluded from $\mathcal{A}_\mathcal{C}$ to prevent infinite regress). (2) The qualitative action $qa(X, x)$ brings about the antecedent event of r. (3) QLAP subtracts those actions whose goal is already achieved in state s.

To construct $T_r : \mathcal{S}_r \times \mathcal{A}_r^s \rightarrow \mathcal{S}_r$, QLAP must compute a set of possible next states for each $s \in \mathcal{S}_r$ and $a \in \mathcal{A}_r^s$. It must then compute the distribution $P(s'|s, a)$. To compute $P(s'|s, a)$, QLAP uses the statistics gathered on DBN r and the statistics gathered to estimate the probability $Pr(a)$ of success for qualitative action a. When calculating the possible next states after a qualitative action, QLAP limits its scope to those next states that are most important for learning the Q-table. For a qualitative action $a = qa(v, q)$ with $v \in \mathcal{C}$ to change the value of a context variable, QLAP considers two possible next states. State s'_1 where the action is successful and the only change is that $s'_1(v) = q$, and state s'_2 where the action fails and $s'_2 = s$. The probability distribution over s' then is $Pr(s'_1|s, a) = Pr(a)$ and $Pr(s'_2|s, a) = 1 - Pr(a)$. For the qualitative action $a = qa(X, x)$ to bring about the antecedent of r, QLAP also considers two possible next states. State s'_1 is where the antecedent event occurs and the consequent event follows, so that s'_1 is the same as s except that $s'_1(X) = x$ and $s'_1(Y) = y$. State s'_2 is where the antecedent event occurs but the consequent event does not follow, so that s'_2 is the same as s except that $s'_2(X) = x$. The probability distribution over s' is $Pr(s'_1|s, a) = CPT_r(s)$ and $Pr(s'_2|s, a) = 1 - CPT_r(s)$.

For a magnitude option the state space $\mathcal{S}_r$, the set of available actions $\mathcal{A}_r^s$, and the transition function T_r are computed similarly as they are for change options. One difference is that magnitude options have a special action called $wait$. For the option to reach $v = q$ from below on the number line, the action $wait$ can be taken if the value of variable v is less than q and is moving towards q.

2.5 Second-Order DBNs

Once a change or magnitude option is created, a *second-order* DBN is created to track its progress. The statistics stored

for second-order DBNs help the agent choose which option it should invoke to perform a qualitative action. Second-order DBNs also allow the agent to determine if there are additional variables necessary for the success of an option. A second-order DBN r_o^2 created for option o has the form

$$r_o^2 = \langle \mathcal{C}_o : invoke(t,o) \Rightarrow succeeds(t,o) \rangle \quad (4)$$

The child variable of second-order DBN r_o^2 is $succeeds(t,o)$, which is true if option o succeeds after being invoked at time t and is false otherwise. The parent variables of r_o^2 are $invoke(t,o)$ and and the context variables in $\mathcal{C}_o$. The Boolean variable $invoke(t,o)$ is true when the option is invoked at time t and is false otherwise. When created, DBN r_o^2 initially has an empty context, and context variables are added in as they are for magnitude and change DBNs.

Second-order DBNs allow the qualitative action to choose an option that has a high probability of success in the current state. The option is chosen randomly based on the weight w_o. Weight w_o is calculated using the current state, the original change or magnitude DBN r, and the second-order DBN r_o^2 where

$$w_o(s) = CPT_r(s) \times CPT_{r_o^2}(s) \quad (5)$$

(To prevent loops in the call stack, an option whose DBN has its antecedent event already in the call stack is not considered a valid choice.)

Second-order DBNs allow the agent to identify other variables necessary for the success of an option o because those variables will be added to its context. Each variable that is added to r_o^2 is also added the state space $\mathcal{S}_r$ of its associated change or magnitude DBN r. For example, if DBN $r = \langle \mathcal{C} : X \rightarrow x \Rightarrow Y \rightarrow y \rangle$, the state space $\mathcal{S}_r$ is updated so that $\mathcal{S}_r = \prod_{v \in Z'_r} \mathcal{Q}(v)$ where $Z'_r = \mathcal{C}_o \cup \mathcal{C} \cup X \cup Y$. For both magnitude and change options, a qualitative action $qa(v,q)$ where $v \in \mathcal{Q}(\mathcal{C}_o)$ is treated the same way as those where $v \in \mathcal{Q}(\mathcal{C})$.

3 Execution

The agent motor babbles for the first 20,000 timesteps (1000 seconds of physical experience) by picking random motor values and maintaining those values for random numbers of timesteps ($\leq$ 40). During this time the agent learns landmarks on motor variables and its first DBNs, options, and qualitative actions. Exploration then follows a developmental progression as the agent randomly chooses qualitative actions for execution among those actions that have at least one option.

To perform an action the agent chooses one of the action's options and then follows the policy. When following a policy QLAP uses ϵ-greedy action selection and updates the Q-tables using Sarsa(λ). The reward for reaching the goal is 10.0, the step cost is 0.01 for each timestep, and it uses the parameter values $\lambda = 0.9$, $\epsilon = 0.05$, $\gamma = 1.0$, and $\alpha = 0.2$.

Every 2000 timesteps the agent learns new DBNs, augments the contexts of existing DBNs, and learns new landmarks (see [Mugan and Kuipers, 2007]). QLAP also creates new options corresponding to the new DBNs and it does a Dyna-inspired [Sutton and Barto, 1998] update of the existing Q-tables by recalculating the transition function and then doing a one-step update of each Q-table.

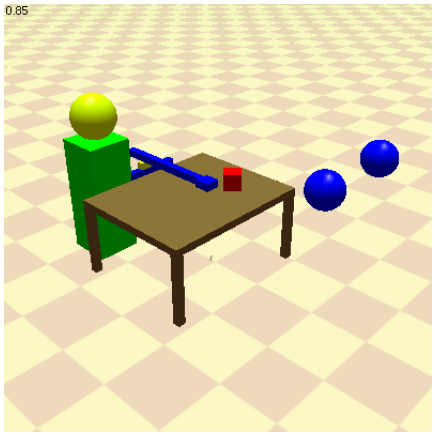


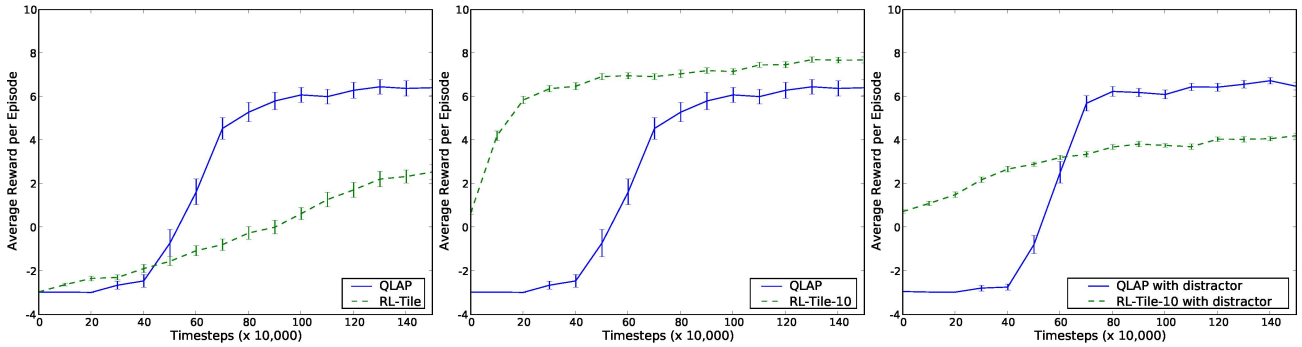
Figure 2: The simulated agent and environment implemented in Breve. It has a torso with a 2-dof orthogonal arm and is sitting in front of a table with a block. The robot has two motor variables $\tilde{u}_x$ and $\tilde{u}_y$ that move the hand in the x and y directions, respectively. The hand is described 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 $\tilde{b}_x(t)$, and $\tilde{b}_y(t)$ give the location of the block in the x and y direction. The relationship between the hand and the block is represented by the continuous variables $\tilde{x}_{rl}$, $\tilde{x}_{lr}$, and $\tilde{y}_{tb}$, where $\tilde{x}_{rl}$ is the distance from the right side of the hand to the left side of the block in the x direction, $\tilde{x}_{lr}$ is the distance from the left side of the hand to the right side of the block in the x direction, and $\tilde{y}_{tb}$ is the distance from the top of the hand to the bottom of the block in the y direction. There are also two distractor floating objects f^1 and f^2. The variables for f^1 are $\tilde{f}_x^1$,$\tilde{f}_y^1$, and $\tilde{f}_z^1$. (The variables for f^2 are analogous.)

4 Evaluation

4.1 The Environment and Task

We evaluate QLAP using the environment shown in Figure 2. The environment is implemented in Breve [Klein, 2003] and has realistic physics. The simulation consists of a robot at a table with a block, and in some experiments there are floating objects that the robot can perceive but cannot interact with. As the agent explores, each time the block is knocked out of reach it is replaced with a different block. Each block has the same mass, but the block size varies randomly in length from 1.0 to 3.0.

QLAP autonomously learns without being specified a task. To evaluate QLAP, we choose the task of having the agent hit the block in a specified direction, either left, right, or forward. During learning, the agent does not know that it will be evaluated on this task. We know of no other RL algorithm that learns in an unsupervised way that would be appropriate for this task, so we compare our method to reinforcement learning using tile coding, which we call RL-Tile. RL-Tile was trained only on the evaluation task. This puts QLAP at a disadvantage on the evaluation task because QLAP learns more than the evaluation task. For example, QLAP learns to move its hand away from the block as well as towards it. It is be-



(a) Base Comparison (b) RL-Tile Extended Actions (c) With Distractor Objects

Figure 3: Comparison of QLAP and RL-Tile. (All error bars are standard error.) (a) QLAP compared to RL-Tile in the environment with no distractor objects. The high task diameter causes RL-Tile to learn more slowly than QLAP. QLAP learns in its developmental progression to hit the block at about 80,000 timesteps. (b) QLAP compared to RL-Tile-10 in the environment with no distractor objects, with the change that actions of RL-Tile-10 last for 10 timesteps. The temporally extended actions allow RL-Tile-10 to perform well immediately. (c) QLAP compared to RL-Tile-10 in the environment with distractor objects. QLAP outperforms RL-Tile-10 because QLAP uses state abstraction to ignore irrelevant variables.

cause of the simplicity of the environment that we can count on QLAP learning the task that was assigned to RL-Tile.

RL-Tile was trained using linear, gradient-descent Sarsa(λ) with binary features [Sutton and Barto, 1998] where the binary features came from tile coding. There were 16 tilings and a memory size of 65,536. The motor variables u_x and u_y were each divided into 10 equal-width bins, so that there were 20 actions with each action either setting u_x or u_y to a nonzero value. The 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 (normalized to the range $[0, 1]$) with a generalization of 0.25. The parameter values used were $\lambda = 0.9$, $\gamma = 1.0$, and $\alpha = 0.2$. Action selection was ϵ-greedy where $\epsilon = 0.05$. The code for the implementation came from PLASTK [Provost, 2008].

For each experiment we trained 30 QLAP agents and 30 RL-Tile agents, and we trained each for 150,000 timesteps (about two hours of physical experience). The QLAP agents autonomously explored the environment, and the RL-Tile agents continually repeated the task. We compare QLAP and RL-Tile by storing the learned state of each every 10,000 timesteps (about every 8 minutes of physical experience). We then test how well each can do that task starting from this stored learned state. Each evaluation consisted of 100 episodes. Each episode lasted for 300 timesteps or until the block was moved. The agent received a penalty of -0.01 for each timestep, and it received a reward of 10.0 if it hit the block in the specified direction.

4.2 How QLAP Achieves the Task

The QLAP agent learns a qualitative action to hit the block in each specified direction. For example, to hit the block to the right, QLAP learned the qualitative action $qa(\dot{b}_x, (0, +\infty))$. QLAP is able to perform this action because it learned an option o_r to bring about $\dot{b}_x = (0, +\infty)$. QLAP learned this option by learning the DBN $r = \langle y_{tb} : x_{rl} \rightarrow [0] \Rightarrow \dot{b}_x \rightarrow (0, +\infty) \rangle$, which predicts that if the distance between the right side of the hand and the left side of the block goes to 0, then the block will soon move to the right. The CPT of this DBN says that in order for this to occur reliably the top of the hand must be above the bottom of the block. QLAP learned the landmarks at 0 on both y_{tb} and x_{rl}. These landmarks allow the agent to make the important distinctions of the hand being above the bottom of the block and being to the left of the block. The second order DBN $r_o^2 = \langle x_{rl} : invoke(t, o_r) \Rightarrow succeeds(t, o_r) \rangle$ predicted that for o_r to work the hand must be to the left of the block. This allowed the agent to add moving the hand to the left of the block as an action in o_r.

4.3 Results

In the first experiment we compare QLAP with RL-Tile in the environment without the distractor objects. The results are shown in Figure 3(a). QLAP learns the necessary actions at around 80,000 timesteps. Although it appears that RL-Tile could eventually overtake QLAP, RL-Tile learns much more slowly. This is because the task diameter is so high that the RL-Tile agent initially does a lot of flailing around before it reaches the goal.

In the second experiment we make the task easier for RL-Tile by reducing the task diameter by making its actions last for 10 timesteps (we call this RL-Tile-10). The results are shown in Figure 3(b). RL-Tile-10 learns the task quickly and then improves only gradually because it cannot take actions that last for less than 10 timesteps.

To demonstrate that QLAP ignores irrelevant variables, the third experiment was conducted using distractor objects. The results are shown in Figure 3(c). When the distractor objects are added, RL-Tile's reward per episode degrades significantly, but QLAP is unaffected.

5 Related work

Given a DBN model of the environment, the VISA algorithm [Jonsson and Barto, 2006] learns a hierarchical decomposition of a factored Markov decision process. VISA learns options and finds a state abstraction for each option. However, VISA requires a discretized state and action space. QLAP learns actions from the agent's continuous motors and learns a discretized state representation.

Options can be learned by first identifying a subgoal and then learning an option to achieve that subgoal. McGovern and Barto [2001] proposed a method whereby an agent autonomously finds subgoals based on bottleneck states that are visited often during successful trials and rarely during unsuccessful ones. Subgoals have also been found by constructing a transition graph based on recent experience and then searching for "access states" [Simsek *et al.*, 2005] that allow the agent to go from one partition of the graph to another. In Barto *et al.* [2004] options are learned to achieve salient events. However, these salient events are determined outside the algorithm, and all of this work takes place in discrete environments.

Once an option is identified, the agent must learn how to achieve it. One way to do this is by learning a model. In environments with large state spaces, the model in the form of a transition function cannot be represented explicitly and the agent must learn a structured representation. Degris *et al.* [2006] proposed a method called SDYNA that learns a structured representation and then uses that structure to compute a value function. Similarly, Strehl *et al.* [2007] learn a DBN to predict each component of a factored state MDP. Both of these methods are evaluated in discrete environments where transitions occur over one-timestep intervals. Another method is learning probabilistic planning rules [Pasula *et al.*, 2007]. In the domain of first-order logic they learn rules that given a context and an action provide a distribution over results. This algorithm also assumes a discrete state space and that the agent already has basic actions such as *pick up*.

6 Conclusion

We present QLAP, a method for enabling an agent to learn a hierarchy of actions in a continuous environment. QLAP assumes that the agent is able to change the values of individual variables. Future work will focus on determining the importance of this assumption and how QLAP can overcome it. QLAP is designed for continuous environments where the agent has very little prior knowledge. QLAP bridges an important gap between continuous sensory input and motor output and a discrete state and action representation.

Acknowledgments

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), and from the National Science Foundation (IIS-0413257, IIS-0713150, and IIS-0750011). The authors would also like to thank Matt MacMahon and Joseph Modayil, as well as the anonymous reviewers for helpful comments and suggestions.

References

[Barto *et al.*, 2004] A.G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. *ICDL*, 2004.

[Boutilier *et al.*, 1995] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1113, 1995.

[Degris *et al.*, 2006] T. Degris, O. Sigaud, and P.H. Wuillemin. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *ICML*, pages 257–264, 2006.

[Dietterich, 1998] T.G. Dietterich. The MAXQ method for hierarchical reinforcement learning. *ICML*, 1998.

[Drescher, 1991] Gary L. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA, 1991.

[Jonsson and Barto, 2006] A. Jonsson and A. Barto. Causal graph based decomposition of factored MDPs. *The Journal of Machine Learning Research*, 7:2259–2301, 2006.

[Klein, 2003] Jon Klein. Breve: a 3d environment for the simulation of decentralized systems and artificial life. In *Proc. of the Int. Conf. on Artificial Life*, 2003.

[Kuipers, 1994] Benjamin Kuipers. *Qualitative Reasoning*. The MIT Press, Cambridge, Massachusetts, 1994.

[McGovern and Barto, 2001] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*, pages 361–368, 2001.

[Mugan and Kuipers, 2007] J. Mugan and B. Kuipers. Learning to predict the effects of actions: Synergy between rules and landmarks. In *ICDL*, 2007.

[Mugan and Kuipers, 2008] J. Mugan and B. Kuipers. Towards the application of reinforcement learning to undirected developmental learning. In *Proc. of the Int. Conf. on Epigenetic Robotics*, 2008.

[Pasula *et al.*, 2007] H.M. Pasula, L.S. Zettlemoyer, and L.P. Kaelbling. Learning symbolic models of stochastic domains. *JAIR*, 29:309–352, 2007.

[Provost, 2008] J. Provost. sourceforge.net, 2008.

[Simsek *et al.*, 2005] O. Simsek, A. Wolfe, and A. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. *ICML*, pages 816–823, 2005.

[Strehl *et al.*, 2007] A.L. Strehl, C. Diuk, and M.L. Littman. Efficient structure learning in factored-state MDPs. In *AAAI*, volume 22, page 645, 2007.

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

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