Autonomous Qualitative Learning of Distinctions and Actions in a Developing Agent

Dissertation

Jonathan Mugan

How can an agent bootstrap up from a pixel-level representation to autonomously learn high-level states and actions using only domain general knowledge? This thesis attacks a piece of this problem and assumes that an agent has a set of continuous variables describing the environment and a set of continuous motor primitives, and poses a solution for the problem of how an agent can learn a set of useful states and effective higher-level actions through autonomous experience with the environment. There exist methods for learning models of the environment, and there also exist methods for planning. However, for autonomous learning, these methods have been used almost exclusively in discrete environments.

This thesis proposes attacking the problem of learning high-level states and actions in continuous environments by using a qualitative representation to bridge the gap between continuous and discrete variable representations. In this approach, the agent begins with a broad discretization and initially can only tell if the value of each variable is increasing, decreasing, or remaining steady. The agent then simultaneously learns a qualitative representation (discretization) and a set of predictive models of the environment. The agent then converts these models into plans to form actions. The agent then uses those learned actions to explore the environment.

The method is evaluated using a simulated robot with realistic physics. The robot is sitting at a table that contains one or two blocks, as well as other distractor objects that are out of reach. The agent autonomously explores the environment without being given a task. After learning, the agent is given various tasks to determine if it learned the necessary states and actions to complete them. The results show that the agent was able to use this method to autonomously learn to perform the tasks.

Chapter 1

Introduction

We would like to build intelligent agents that can autonomously learn to predict and control the environment using only domain-general knowledge. Such agents could simply be placed in an environment, and they would learn it. After they had learned the environment, the agents could be directed to achieve the goals specified by the engineers. The intelligence of the agents would free engineers from having to design new agents for each environment. These agents would be flexible and robust because they would be able to adapt to aspects of the environment not anticipated by engineers.

Designing such agents is a difficult problem because the environment can be almost infinitely complex. This complexity means that an agent with limited resources cannot represent and reason about the environment without describing it in a simpler form. And possibly more importantly, the complexity of the environment means that it is a challenge to generalize from experience since each experience will be in some respect different.

A solution to the difficulty of learning in a complex environment is for the agent to autonomously learn useful and appropriate abstractions. Pierce and Kuipers [1997] created a method that searched through abstractions to find those best suited for representing the environment. Modayil and Kuipers [2007] built on this work to enable a robot to identify moving objects. The work presented in this thesis is a continuation of this work.

Given the groundwork laid by [Pierce and Kuipers, 1997] and [Modayil and Kuipers, 2007], an agent can represent the world with a set of continuous variables and affect the world using a set of continuous effectors. The specific problem addressed by this thesis is: *given an environment consisting of a set of time-varying continuous variables, how can an agent learn models to predict the environment and learn high-level actions to control it.* The thesis of this dissertation is that a learning algorithm can be constructed that builds many small models, and then bootstraps to more complex and more reliable models, and then turns those models into plans for actions.

1.1 Autonomous Learning from the Environment

This thesis addresses the problem of how an agent can autonomously learn from the environment. By learning, we mean that an agent should learn a predictive model and a set of actions using autonomous exploration.

Predictive models allow the agent to simulate the environment, which allows it to construct plans. These plans can be converted to actions that allow the agent to alter the environment. We desire that the agent learn through autonomous exploration for two reasons. The first is that we want to free the engineer from having to specify how the environment should be explored. Second, autonomous exploration provides added flexibility because the environment could change in a way

that the engineer did not anticipate.

We focus on environments that are continuous, dynamic, and have state. Continuous environments require the agent to form an abstracted representation. Dynamic environments change and so cannot be memorized. They require an agent to build a model. Environments that have state require the agents to make decisions over time and to make different decisions in different situations.

1.2 Approaches to the problem

We are interested in autonomous learning in continuous, dynamic environments. There have been many approaches to this problem. The method presented in this thesis uses some of these methods and is related to others.

1.2.1 Learning Rules

One way to learn from the environment is to learn STRIPS-like [Nilsson, 1980] rules of the form "in context x, if the agent does y, then z will occur." In the classic AI tradition, these rules can be chained together to form plans to achieve results. Our method learns models that are similar to these rules, and the structure of these rules helps our agent to plan.

1.2.2 Reinforcement Learning

The classic AI planning model assumes deterministic actions [Boutilier *et al.*, 1999]. As research has moved towards less deterministic environments, Markov Decision Process (MDP) planning [Puterman, 1994] has become popular. In MDP planning, an agent learns a policy that specifies the best action for each state. One way to learn this policy is reinforcement learning [Sutton and Barto, 1998]. Our method uses MDP planning in addition to classic AI planning. Our method learns the state space for the MDP and uses reinforcement learning to learn an action policy.

1.2.3 Forward and Backward Models

Forward models predict how the environment will change based on the motor values, and backward models determine what motor values are needed to give a particular environment change. These models are generally fine-grained, predicting the real values of variables. Work has been done in the area of learning forward and backward models of the dynamics of the environment either using regression [Atkeson *et al.*, 1997a; 1997b], or by neural nets [Jordan and Rumelhart, 1992].

These methods require that a plan be specified in order for an agent to perform actions (e.g. a plan for how to go around objects). Additionally, these methods do not generally model interactions between objects. However, these methods have been successful at allowing robots to learn complex tasks [Vijayakumar *et al.*, 2005]. This is an important approach that could be complementary to the discretization-based approach of this thesis. Chapter 10 describes how our method might be extended to take advantage of this kind of fine-grained control.

1.2.4 Evolutionary Approach

Artificial life/evolutionary algorithms/genetic algorithms perform a search over a space to find a policy that maximizes a fitness function [Bedau, 2003]. There have been some impressive results on a small scale [Nolfi and Floreano, 2002], but making the search more tractable is an open problem. For example, [Whiteson *et al.*, 2005] found that task decomposition is essential for evolutionary

methods to work on the task of soccer keepaway. Our approach has an evolutionary flavor because it does a search of a large number of predictive models and uses their predictive accuracy as a fitness function.

1.2.5 Explicitly Build in Domain Knowledge

Our approach is to begin with no domain knowledge (a "bottom-up" approach) and to build up competency. It is worth noting that there are other approaches that seek to build flexible robots by engineering in domain knowledge and using learning to generalize when necessary (a "top-down" approach). The subsumption architecture [Brooks, 1986] has as its philosophy that research in robotics should start with small, insect-like behaviors, and that more complex behaviors should be built (by hand if necessary) on top of simpler ones. Behavior-based robotics [Arkin, 1998] is a related approach. It posits that robots should have a set of behaviors, and that each behavior should be learned or coded separately. Using the behavior-based approach, Stoytchev [2005a, 2005b] focuses on tool use and advocates a *behavior grounded* approach to learning in which an agent is endowed with a set of behaviors, and through experimentation the agent learns which behaviors, tools, and objects go together to achieve a desired result. Using this approach, in [Sinapov and Stoytchev, 2007] the agent was able to learn how a puck would move for different combinations of tools and actions.

Other notable work includes [Fitzpatrick *et al.*, 2003]. Given a set of initial hand positions, they learn the direction an object moved when the robot reached for it. And there has been work on learning how to grasp novel objects [Saxena *et al.*, 2007]. Related to this approach of learning particular behaviors is demonstration learning. In this approach, the robot learns from a human teacher [Price and Boutilier, 2003; Knox and Stone, 2010].

Both bottom-up and top-down approaches appear to be needed. A top-down approach might be the fastest way to obtain narrowly-useful robots, but bottom-up approaches may eventually enable a robot to have the human-like flexibility and intelligence needed to display common sense.

1.3 Principles of Our Approach

Our approach is based on three broad principles. (1) The learning agent should break up the environment. (2) The learning agent should proceed developmentally in a bootstrapping fashion. (3) The agent should base its models on learned contingencies.

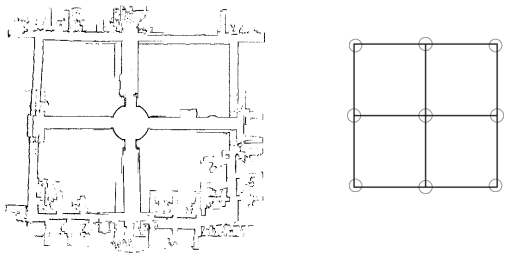


Figure 1.1: The left-hand side shows a metrical map. The metrical map has high resolution. The right-hand side shows a topological map. The topological map has less resolution, but it explicitly represents the important points and the paths between them.

1.3.1 Breaking up the Environment

Our method breaks up the environment in two ways. It discretizes the input, and it fragments the environment by representing the environment and plans for actions using many small models. This philosophy is analogous to learning a topological map instead of a metrical map [Kuipers, 2000] as shown in Figure 10.1. Fragmenting the environment into many small models makes learning easier because to learn each part, the agent need only focus on a small amount of the input.

Discretizing the input makes learning easier because the agent can generalize. In a continuous environment, each learning instance is unique. But by mapping similar instances together, the agent can use simple statistical learning. Discretizing the input also allows the agent to autonomously define goals. The discretization in our method is represented using a qualitative representation. A qualitative representation encodes the values of continuous variables relative to known landmark values [Kuipers, 1994].

1.3.2 Developmental Learning

Children learn developmentally. In Piaget's [1952] theory of cognitive development, children's cognitive development progresses in stages. More recently, Cohen [2002] proposed an information processing theory of cognitive development in which children are endowed with a domain-general information processing system that they use to bootstrap knowledge. Gibson (1988) proposed that human children are endowed with systems to allow them to explore and learn about the world. She emphasized that it was this exploration that enabled cognitive development.

In our method, the agent bootstraps new skills on top of existing skills. The driving force behind the developmental learning is the desire to predict the environment. What the agent strives to predict is based on the models and the discretizations it has previously learned, and new discretizations are learned to make those models more reliable. Additionally, the skills that the agent has already learned determine how it can explore the environment.

1.3.3 Using Contingencies to Represent Knowledge

Our method uses contingencies to represent knowledge. The contingencies are of the form: if event a occurs, then event b will soon occur. It has been proposed that humans have an innate contingency detection module [Gergely and Watson, 1999], and it has been shown that human infants can detect contingencies in their environment shortly after birth [DeCasper and Carstens, 1981]. Contingencies have the advantage of being easy to learn because the agent need only compare each pair of events to see if they form a contingency. Additionally, contingencies form a natural representation for planning, since they can represent sequences of events.

However, as Watson [2001] has pointed out, a prerequisite for learning contingencies is determining when an event has occurred. Fortunately, our method learns a discretization, and so the learning of contingencies can be tied to the definition of events through discretization.

1.4 The Qualitative Learner of Action and Perception, QLAP

In this thesis, we present the Qualitative Learner of Actions and Perception, QLAP. QLAP is an algorithm that allows an agent to use low-level sensors and effectors to learn high-level representations and actions in continuous, dynamic environments. QLAP consists of four steps:

1. Begin with a very broad discretization of the environment.

2. Simultaneously learn a discretization and a set of predictive models of the environment.

3. Convert the models into plans and form the plans into a set of hierarchical actions.

4. Use learned actions to explore the environment

Begin with a broad discretization. The discretization comes from the set of learned landmarks. Initially, the agent can tell if the value of each variable is increasing, decreasing, or remaining steady. Similarly, the agent can tell if a motor value is positive, negative, or zero.

Simultaneously learn a discretization and a set of predictive models of the environment. (Shown in Figure 1.2.) QLAP represents models using dynamic Bayesian networks (DBNs) [Dean and Kanazawa, 1989] because they are simple and probabilistic. Given the current discretization, QLAP defines predicates for each possible antecedent event and consequent event. To determine which models to learn, QLAP tracks statistics on all possible pairs of events. QLAP creates a DBN for each pair of events that form a contingency where the consequent event soon follows the antecedent event with sufficient reliability.

As QLAP gathers experience in the world, it tracks statistics on the learned DBNs and iteratively adds context variables to make the contingency more reliable. To find context variables that predict when the consequent event will soon follow the antecedent event, QLAP may have to introduce new landmarks. To find these landmarks, QLAP stores the real values of each variable each time the antecedent event of some model is observed. It then looks to see if the consequent event is soon observed. QLAP can then use an information-theoretic method to determine if there is a landmark on some continuous variable that, if defined, would allow the model to more reliably predict when the consequent event will follow the antecedent event. An important aspect of QLAP is that it considers new distinctions to be broadly useful. Each new landmark that is learned modifies the current discretization and enables new models to be learned.

Convert the models into plans and form the plans into a set of hierarchical actions. (Shown in Figure 1.3.) Each model predicts when a consequent event will occur. QLAP converts a model that reliably predicts when the consequent event will occur into a plan to bring about that event. Plans in QLAP are represented in the options framework [Sutton *et al.*, 1999]. An option is like a subroutine, and QLAP uses the variables in the model to determine the state space of the option, and uses the consequent event as the goal state of the option.

A given plan possibly does not reference any motor variables. To overcome this difficulty, QLAP uses a hierarchical representation. For each discrete (qualitative) value of each variable, QLAP creates an *action* to bring that variable to that value. This means that in a plan, to bring about the antecedent event, or to change the value of any context variable, QLAP simply has to call the action to bring about that desired value.

Each plan that QLAP learns is associated with the action that matches the consequent event of the contingency that led to the plan. A given consequent event may have more than one antecedent event that predicts it. This is good because different plans might be best in different situations. This means that each QLAP action can have multiple possible plans. Each QLAP action keeps statistics on how reliable each of its plans is in each situation. This allows it to pick the best plan for the current state.

Explore the environment using learned actions. QLAP explores the environment by choosing learned actions to "practice." So that it can explore the space effectively, QLAP uses Intelligent Adaptive Curiosity [Oudeyer *et al.*, 2007] to choose actions that are neither too hard nor too easy.

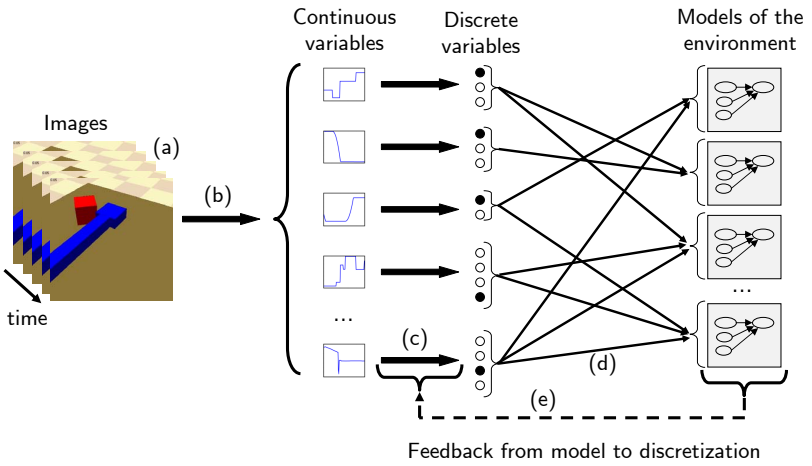


Figure 1.2: Perception in QLAP.

1.5 Contributions

QLAP is the only algorithm that we are aware of that learns states and hierarchical actions in continuous, dynamic environments with continuous motors through autonomous exploration. In addition, QLAP contributes to two subfields. QLAP contributes to the field of autonomous mental development and the field of reinforcement learning.

1.5.1 Contribution to Autonomous Mental Development

QLAP provides a method for an agent to learn through a developmental progression. Specifically, QLAP provides a method for a developing agent:

1. to learn its first temporally-extended actions.
2. to learn more complex actions on top of previously-learned actions.

1.5.2 Contributions to Reinforcement Learning

Reinforcement learning is about enabling an agent to learn from experience to maximize a reward signal. QLAP addresses three challenges in reinforcement learning: (1) continuous states and actions, (2) automatic hierarchy construction, and (3) automatic generation of reinforcement learning problems.

1. Continuous states and actions are a challenge because it is hard to know how to generalize from experience since no two states are exactly alike. QLAP provides a method for discretizing the state and action space so that the discretization corresponds to the "natural joints" in the environment.
2. Learning of hierarchies can enable an agent to explore the space more effectively because it can aggregate smaller actions into larger ones. QLAP creates a hierarchical set of actions from continuous motor variables.
3. Currently, most reinforcement learning problems have to be designed by the experimenter. QLAP autonomously creates reinforcement learning problems as part of its developmental progression.

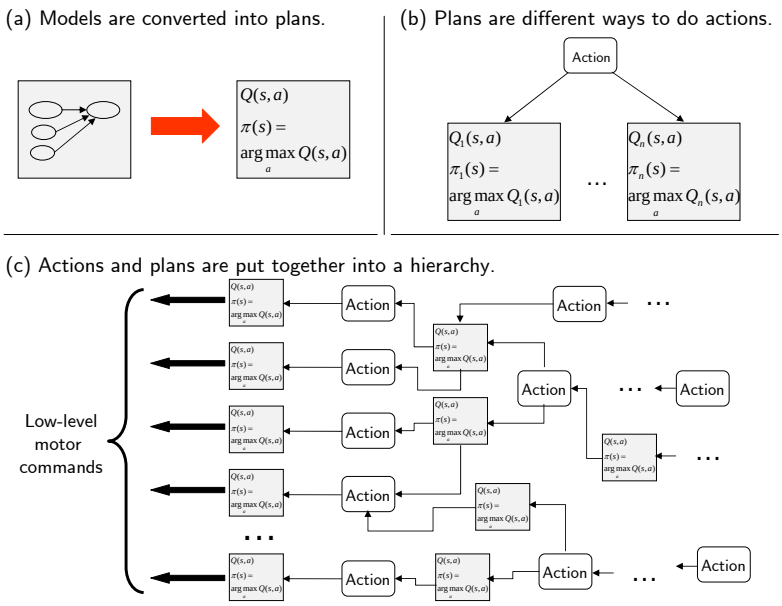


Figure 1.3: Actions in QLAP.

1.6 Assumptions of QLAP

As discussed at the beginning of this chapter, QLAP builds on the work by [Pierce and Kuipers, 1997] and [Modayil and Kuipers, 2007]. This means that the QLAP agent interacts with the world by using a set of continuous variables and acts in the world using a set of continuous effectors. This assumption is represented in Figure 1.4.

1. The environment produces a large, undifferentiated stream of information.
2. The data factoring process ((a) in Figure 1.4) takes this stream of information and converts it into a time-varying set of continuous variables ((c) in Figure 1.4).
3. QLAP receives these variable values and sets the values for the primitive motor variables.
4. The primitive motor variables are converted into raw motor variables by the motor conversion process ((b) in Figure 1.4).

1.6.1 Data Factoring Process

The data factoring process converts the sensory stream of the environment into a set of continuous variables. The data factoring may come directly from the sensors in the environment (for example, a network of temperature readings) or it may be done by humans.

A data factoring process is almost always assumed, e.g. [Pasula *et al.*, 2007; Stoytchev, 2005a; Degris *et al.*, 2006; Vigorito and Barto, 2008; Strehl *et al.*, 2007]. For the experiments in this dissertation, we rely upon the results of [Modayil and Kuipers, 2007]. In their work, they identified and tracked moving objects. From this, it is relatively straight-forward to get variables for object locations and distances between objects that are used for experiments.

QLAP makes some assumptions about how the environment is factored. QLAP assumes that any goal that an outside observer would want the agent to accomplish is represented with an input

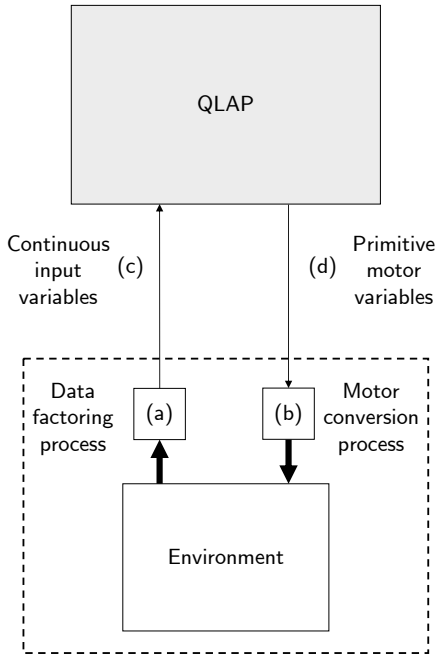


Figure 1.4: Assumptions of QLAP.

variable. QLAP also assumes that meaningful landmarks can be found on single variables. One could envision a situation where there is only a sensible landmark on some combination of variable values, for example their product.

1.6.2 Motor Conversion Process

QLAP assumes a set of continuous primitive motor variables that correspond to orthogonal directions of movement. The motor conversion process converts these primitive motor variables assumed by QLAP into the raw motor variables of the robot. QLAP builds on the work of Pierce and Kuipers [1997]. In their work, an agent was able to use principal components analysis (PCA) [Duda *et al.*, 2000] to learn a set of primitive actions corresponding to turn and travel for a robot that had motors to turn each of two wheels independently.

1.6.3 Related Assumptions

QLAP makes assumptions about the dynamics of the environment. QLAP uses statistical learning, this has the effect that if the dynamics are such that important contingencies occur too rarely, then QLAP will not be able to learn them. QLAP searches for context variables that predict when a learned contingency will be reliable. These context variables are added one-by-one by hill climbing on the reliability of the contingency. Thus, QLAP assumes that each variable will help individually. Relatedly, contingencies cannot require too many context variables to be reliable. This means that QLAP assumes that the environment is decomposable into relatively small fragments. And QLAP is not designed for learning fine-grained movement within fragments. Chapter 10 will present ideas for future work to extend QLAP to learn such fine-grained models.

Chapter 2

Supporting Work

QLAP builds on a large body of supporting work. QLAP represents the dynamics of the environment using learned graphical models. QLAP uses information theory for measuring how deterministic these models are and for evaluating possible model improvements. QLAP uses Markov decision processes (MDPs) and reinforcement learning for making decisions over time and learning plans. And QLAP uses the options framework for temporal abstraction. This chapter provides a brief introduction to each of these topics and describes how each is used in QLAP.

2.1 Predictive Models

As discussed in Section 6 of Chapter 1, QLAP interacts with the world using a set of continuous variables. QLAP discretizes these variables using learned landmarks. This discretization results in a factored state representation where the state space consists of the Cartesian product of the qualitative values of all of the variables. In a factored representation, the size of the state space grows exponentially with the number of variables. This is called the curse of dimensionality [Duda *et al.*, 2000].

The curse of dimensionality makes it difficult to represent the state space and how it changes over time. One solution is to try to predict each variable independently and to use only the variables that determine the value of that variable in the prediction. This is the approach we will take using graphical models. This section introduces two types of graphical models: Bayesian networks and dynamic Bayesian networks. QLAP uses dynamic Bayesian networks, but we first introduce Bayesian networks because it simplifies the explanation of dynamic Bayesian networks.

2.1.1 Bayesian Networks

We could, in principle, represent the environment using a multidimensional probability distribution. This distribution would give more weight to likely states and less weight to unlikely states. But because of the curse of dimensionality, this is difficult to do. However, if each variable is not dependent on all of the others, then Bayesian networks allow for the compact representation of a probability distribution [Duda *et al.*, 2000]. This compactness comes from conditional independence. If the probability of a value of variable A conditioned on some other variables B and C is independent of the value of a variable D, then $P(A|B,C) = P(A|B,C,D)$.[1] This conditional

[1]We note that if A is conditionally independent of C given B (denoted by $A \perp C \mid B$) then $P(A,C|B) = P(A|B)P(C|B)$. And we can show that this conditional independence implies $P(A|B) = P(A|B,C)$ by

$$P(A|B) = \frac{P(A,C|B)}{P(C|B)} = \frac{P(A,C|B)P(B)}{P(C|B)P(B)} = \frac{P(A,C,B)}{P(C,B)} = P(A|B,C) \tag{2.1}$$

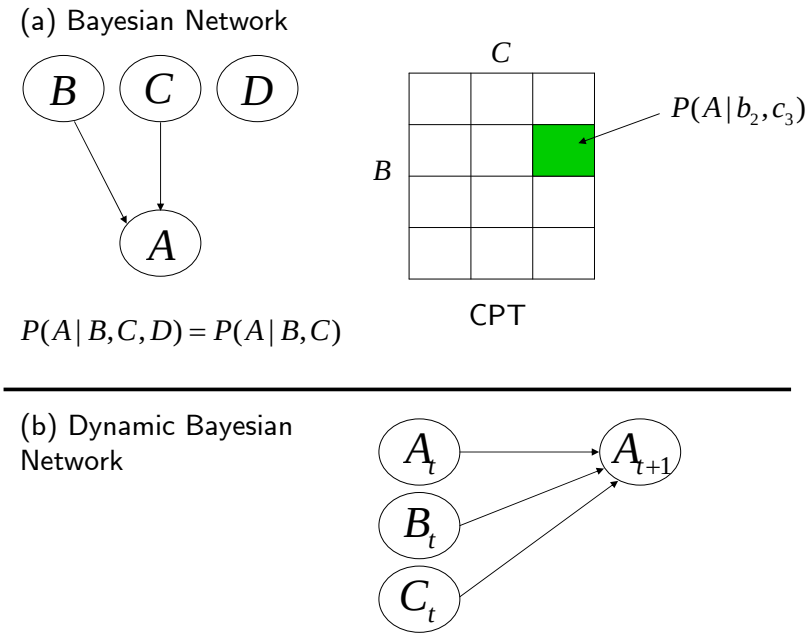


Figure 2.1: (a) Bayesian Network. The value of A depends on the values of variables B and C but not D. The conditional probability table (CPT) gives the probability distribution of A for each value of B and C. (b) Dynamic Bayesian Network. A Bayesian network that models variable values over time.

independence allows variable D to be ignored.

An example of a Bayesian network is shown in Figure 2.1 (a). In this figure, the variables B and C are the parents of child variable A. And in the discrete case, there is a conditional probability table (CPT) that gives the distribution over the values of C for each value of its parent variables.

Bayesian networks give a static description of state. We desire models that represent state changes over time. This can be accomplished with dynamic Bayesian networks.

2.1.2 Dynamic Bayesian Networks

Dynamic Bayesian networks (DBNs) have been created to compactly represent how states change over time [Dean and Kanazawa, 1989]. An example is shown in Figure 2.1 (b). We see that the value of variable A at time $t+1$ is partially determined by its value at time t. QLAP learns DBNs to model the dynamics of the environment.

2.2 Information Theory

QLAP uses information theory to measure how deterministic a model is and to evaluate possible model improvements. Information is anything that reduces uncertainty among a set of alternatives [Shannon, 1948; Ayres, 1994]. We can measure the current uncertainty over a set of alternatives

Showing $A \perp D \mid B, C \;\Rightarrow\; P(A|B,C) = P(A|B,C,D)$ is analogous.

using entropy [Duda *et al.*, 2000]. The *entropy* $H(Y)$ of a random variable is given by

$$H(Y) = \sum_j P(Y = y_j) \log_2 \frac{1}{P(Y = y_j)}$$

The conditional entropy $H(Y|X)$ of a random variable Y given X is

$$H(Y|X) = \sum_i H(Y|X = x_i)P(X = x_i)$$

and is the weighted average of the entropy of Y given $X = x_i$, weighted by the probability $P(X = x_i)$. QLAP uses the conditional entropy of the child variable given its parents as a measure of how deterministic a DBN is.

Information gain is defined as a reduction in entropy (uncertainty) from some piece of information. If we know $H(Y|X)$, then the information gain of Y given knowledge of $H(Y|X)$ is

$$I(Y; X) = H(Y) - H(Y|X) \tag{2.2}$$

QLAP uses information gain as a measure of the value of adding a new discretization or context variable.

2.3 Structuring Decision Problems: MDPs

An agent can make models of the world using dynamic Bayesian networks, but how can the agent make decisions over time? One approach is an MDP. A Markov Decision Process (MDP) is a framework for understanding decision making over time [Puterman, 1994]. An MDP is a four-tuple[2] of the form $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $T(s, a, s') = P(s'|s, a)$ is the transition function over states and actions, and $R(s, a, s')$ is a reward function.

MDPs are often used as the structure for decision making for learning agents [Jonsson and Barto, 2007; Sutton *et al.*, 1999; Brafman and Tennenholtz, 2003]. An MDP breaks the world up into states; it represents what is in the world, and it represents everything that can be perceived about the world. Additionally, an MDP represents how the world can change, including how the agent's actions affect the world. And it also includes what is "good" for the agent in the reward function.

The philosophy of MDP use in QLAP is different, and is closer to the idea of an image schema in developmental psychology. Within developmental psychology, Jean Mandler [2004a] proposed a theory of *perceptual meaning analysis*, which is an experimentally grounded theory that explains how infants can learn concepts and how these concepts can be represented. She describes perceptual meaning analysis as "the central attentive process that redescribes attended perceptual information into a simpler and conceptual (accessible) form" [Mandler, 2004b]. In Mandler's theory, concepts are represented with *image schemas*, and in [Mandler, 2004a] she says that "the image-schemas that perceptual meaning analysis creates are analog representations that summarize spatial relations and movements in space" (p. 79). The notion of image schemas and the idea of using them as a foundation for understanding was advanced by [Lakoff and Johnson, 1980]. Johnson [1987] writes that an image schema consists of set of components that are related by definite structure.

To our knowledge, there have been no convincing computational implementations of image schemas. MDPs are not analog representations and are missing the "image" part of the image

[2]Some authors also include the discount rate γ as well as a distribution τ over initial states.

schemas. But MDPs do encapsulate knowledge about the world in a principled framework, and QLAP takes advantage of MDPs to represent how the agent can interact with the world. As a consequence of using MDPs in this way, QLAP assumes that the world is too large to be represented with a single MDP and therefore QLAP does not assume a single, underlying MDP. Instead, QLAP creates many small MDPs, where each MDP represents some small aspect of the environment. These MDPs are not assumed to objectively exist in the world, but are created by the agent, even up to the discretization.

Once these MDPs are created by QLAP, QLAP can take advantage of the large body of research on MDPs. Within each MDP, QLAP learns a policy π. A policy is a way to represent a plan within an MDP, and specifies which action the agent should choose for each state $s \in \mathcal{S}$. This policy can be learned using reinforcement learning as discussed in the next section.

2.4 Reinforcement Learning

Reinforcement learning is a way to learn a policy for an MDP [Sutton and Barto, 1998]. Its roots trace back to reinforcement learning in psychology where an agent is not told what to do, but rewarded for doing the right thing [Hill, 1990]. This method works well with animals, and the idea is that intelligent agents might be able to learn in the same way.

Within the framework of an MDP, reinforcement learning (RL) is a method to learn a policy π that maximizes the total reward r earned over time. A policy $\pi(s)$ gives a prescribed action (or distribution of actions) for each state s.

2.4.1 Representing what is good: the value function

Integral to the MDP representation is the Markov assumption. The Markov assumption is that the current state has all the information necessary to make decisions. An important consequence of the Markov assumption is the value function. A value function gives a measure of how "good" it is to be in a particular state. More formally, a *state-value function*

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, \pi\right] \tag{2.3}$$

gives the cumulative expected reward, discounted by γ, of being in state s and following policy π thereafter. A value function can also be in the form $Q(s, a)$, called a *state-action-value function*,

$$Q^{\pi}(s, a) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, a_t = a, \pi\right] \tag{2.4}$$

which gives the cumulative expected discounted reward of being in state s and taking action a and following policy π thereafter.

The agent can easily convert the value function to a policy. In the current state s, the agent can simply choose the action a with the highest $Q(s, a)$ value. To make sure that it explores, the agent can choose the best action with probability $1 - \epsilon$, and a random action with probability ϵ. This action selection method is called ϵ-greedy.

2.4.2 Learning the Value Function with a Model

If the agent has a model, it can learn by simulating experience. The agent can imagine all possible trajectories to learn a policy of what to do in each state. More formally, if the transition function T and the reward function R are known, then the policy π can be computed using dynamic

programming with the Bellman equation. The Bellman equation for Q^* for the optimal policy π^* is

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right] \tag{2.5}$$

where $R(s,a,s')$ is the reward that results from taking action a in state s and reaching state s'. The agent can simultaneously learn the optimal policy π^* and the associated optimal state-action value function Q^* using value iteration by turning the Bellman equation into an update rule.

2.4.3 Learning Through Experience

If the agent does not have a model, or the model is incomplete, it can learn through experience. If the transition function T and the reward function R are not known, then there are two approaches to learning the value function (and thus the policy).

1. Learn a model T and a reward function R through experience, and then use dynamic programming.

2. Use a model-free approach to learn $Q(s,a)$ through experience.

One model-free approach to learning $Q(s,a)$ is temporal difference learning. Temporal-difference learning updates the estimate for $Q(s,a)$ after each timestep by updating the current estimated value of $Q(s,a)$ to account for the observed reward and the new state s'.

One model-free method of temporal difference learning is Sarsa, an acronym that comes from $\langle s,a,r',s',a' \rangle$ because the agent takes an action a in a state s and it then arrives in state s' and receives reward r' and chooses next action a'. Using Sarsa, the update equation is

$$Q'(s,a) = Q(s,a) + \alpha[r' + \gamma Q(s',a') - Q(s,a)] \tag{2.6}$$

To speed up learning, the agent can remember the states and actions it has taken so that when it gets a reward it can immediately propagate that reward back to past states and actions. This is done using an *eligibility trace* $e(s,a)$ that gives a measure of how recently action a was taken in state s. After every timestep the eligibility trace is updated so that

$$\forall s,a : e(s,a) \leftarrow \lambda\gamma e(s,a) \tag{2.7}$$

where λ is a decay parameter $0 \leq \lambda \leq 1$. The eligibility trace is then updated for the most recently taken action a in the state s so that

$$e(s,a) \leftarrow 1 \tag{2.8}$$

Using eligibility traces, Equation 2.6 is modified so that each value of Q is updated after each timestep according to the equation

$$\forall s,a : Q'(s,a) = Q(s,a) + e(s,a)\alpha[r' + \gamma Q(s',a') - Q(s,a)] \tag{2.9}$$

2.4.4 Combining Models and Experience: Dyna

Dyna [Sutton and Barto, 1998] provides a framework for using experience to update both the model and the value function policy directly. In Dyna-Q, as the agent explores the world it updates both the model and the Q table, and after each timestep it uses the updated model to update the Q table.

2.5 Hierarchy

Hierarchy provides the advantage of temporal abstraction because decisions are not required at each timestep, but rather only when temporally extended actions terminate [Barto and Mahadevan, 2003]. QLAP learns a hierarchy of actions. These actions are hierarchical because these actions have plans that call other actions. And once an action is called, the calling action does not concern itself with how the called action is carried out.

A common framework for hierarchy within reinforcement learning is 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. Chapter 5 will discuss how QLAP uses options.

Chapter 3

Qualitative Representation

A qualitative representation allows an agent to bridge the gap between continuous and discrete values. It does this by encoding the values of continuous variables relative to known landmarks [Kuipers, 1994]. A qualitative representation breaks the number line up into an ordered set of qualitative values. All of the continuous values between any two landmarks have the same qualitative value. By going "up" or "down" along the number line, a qualitative variable can change from one qualitative value to the next. A qualitative representation allows the agent to generalize and to focus on important events, where an important event occurs when the qualitative value of a variable changes.

A qualitative representation is different from a simple discretization because the landmarks themselves are important. A variable value can be at a landmark, it can be moving towards the landmark, or it can be moving away from it. This chapter discusses the qualitative representation and how it allows the agent to define events. It then poses some questions that will be answered in later chapters.

3.1 Qualitative Representation

QLAP converts continuous variables to qualitative variables using *landmarks*. A *landmark* is a symbolic name for a point on a number line. Using landmarks we 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, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$. This is shown in Figure 3.1.

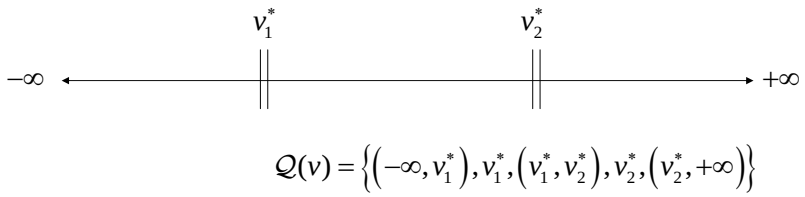


Figure 3.1: Landmarks divide the number line into a discrete set of qualitative values.

QLAP receives a set of continuous input variables from the world and uses a set of continuous motor variables as output.[1] Two qualitative variables are created for each continuous input variable $\tilde{v}$, a discrete variable $v(t)$ that represents the qualitative magnitude of $\tilde{v}(t)$, and a discrete variable $\dot{v}(t)$ that represents the qualitative direction of change of $\tilde{v}(t)$. Also, a qualitative variable $u(t)$ is created for each continuous motor variable $\tilde{u}$.[2] The result of these transformations is three types of qualitative variables that the agent can use to affect and reason about the world: *motor variables*, *magnitude variables*, and *direction of change variables*. The properties of these variables are shown in Table 3.1.

Table 3.1: Types of Qualitative Variables

Type of Variable	Initial Landmarks	Learn Landmarks?
motor	$\{0\}$	yes
magnitude	$\{\}$	yes
direction of change	$\{0\}$	no

Each direction of change variable $\dot{v}$ has a single intrinsic landmark at 0, so its quantity space is $\mathcal{Q}(\dot{v}) = \{(-\infty, 0), 0, (0, +\infty)\}$, which can be abbreviated as $\mathcal{Q}(\dot{v}) = \{[-], [0], [+]\}$. Motor variables are also given an initial landmark at 0. Magnitude variables initially have no landmarks because zero is just another point on the number line. Initially, when the agent knows of no meaningful qualitative distinctions among values for $\tilde{v}(t)$, we describe the quantity space with the empty list of landmarks, $\{\}$, as $\mathcal{Q}(\dot{v}) = \{(-\infty, +\infty)\}$. However, the agent can learn new landmarks for magnitude and motor variables. Each additional landmark allows the agent to perceive or affect the world at a finer level of granularity.

3.2 Events

If a is a qualitative value of a qualitative variable A, meaning $a \in \mathcal{Q}(A)$, then the *event* $A_t{\rightarrow}a$ 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. We will often drop the t and describe this simply as $A{\rightarrow}a$. We will also refer to an event as E when the variable and qualitative value involved are not important, and we use the notation $E(t)$ to indicate that event E occurs at time t.

For magnitude variables, $A_t{\rightarrow}a$ is really two possible events, depending on the direction that the value is coming from. If at time $t-1$, $A(t) < a$, then we describe this event as $\uparrow A_t{\rightarrow}a$. Likewise, if at time $t-1$, $A(t) > a$, then we describe this event as $\downarrow A_t{\rightarrow}a$. However, for ease of notation, we generally refer to the event as $A_t{\rightarrow}a$. We also say that event $A_t{\rightarrow}a$ is *satisfied* if $A_t = a$.

3.3 Conclusion

To use a qualitative representation, the agent must have a set of landmarks that correspond to the dynamics of the environment. How can these landmarks be learned? Additionally, a landmark value can be reached if the agent can set the direction of change of the variable. How can the agent predict these changes and control them? These questions will be answered in the next two chapters.

[1] QLAP can also handle discrete (nominal) input variables. See Appendix A for details.

[2] Note that when the distinction between motor variables and non-motor variables is unimportant, we will refer to the variable as v.

Chapter 4

Learning Concise, Reliable Predictive Models

There are many methods for learning predictive models in continuous environments. Such models have been learned, for example, using regression [Atkeson *et al.*, 1997a; 1997b; Vijayakumar and Schaal, 2000; Vijayakumar *et al.*, 2005] neural networks [Jordan and Rumelhart, 1992], or Gaussian processes [Rasmussen, 2006]. But as described in Chapter 1, we want to break up the environment and represent it using a qualitative representation.

In a discretized environment, dynamic Bayesian networks (DBNs) are a convenient way to encode predictive models. Most work on learning DBNs learn a network to predict each variable at the next timestep for each primitive action, e.g. [Degris *et al.*, 2006; Jonsson and Barto, 2007; Strehl *et al.*, 2007]. However, QLAP does not assume a set of primitive actions, and QLAP works in environments where events may take more than one timestep.

QLAP learns two different types of DBN models. The first type of DBN models are those that predict events on change variables (change DBNs). The second type of DBN models are those for reaching magnitude values (magnitude DBNs). To learn change DBNs, QLAP uses a novel DBN learning algorithm. Given the current discretization, QLAP tracks statistics on all pairs of events to search for contingencies where an antecedent event leads to a consequent event. When such a contingency is found, QLAP converts it to a DBN with the antecedent event as the parent variable and the consequent event as the child variable. QLAP then adds context variables to the DBN one at a time as they make the DBN more reliable. For each DBN, QLAP also searches for a new discretization that will make the DBN more reliable. This new discretization then creates new possible events and allows new DBNs to be learned. This method is outlined in Figure 4.1.

This chapter first describes the novel algorithm for learning change DBNs. It then describes the how magnitude DBNs are learned.

4.1 Searching for Contingencies

The search for change DBNs begins with a search for contingencies. A contingency represents the knowledge that if the antecedent event occurs, then the consequent event will soon occur. An example would be if you flip a light switch, then the light will go off. QLAP searches for contingencies by tracking statistics on pairs of events E_1, E_2 and extracting those pairs into a contingency where the occurrence of event E_1 indicates that event E_2 is more likely to soon occur than it would otherwise. In this section we define contingencies in QLAP, and then we describe how QLAP identifies them.

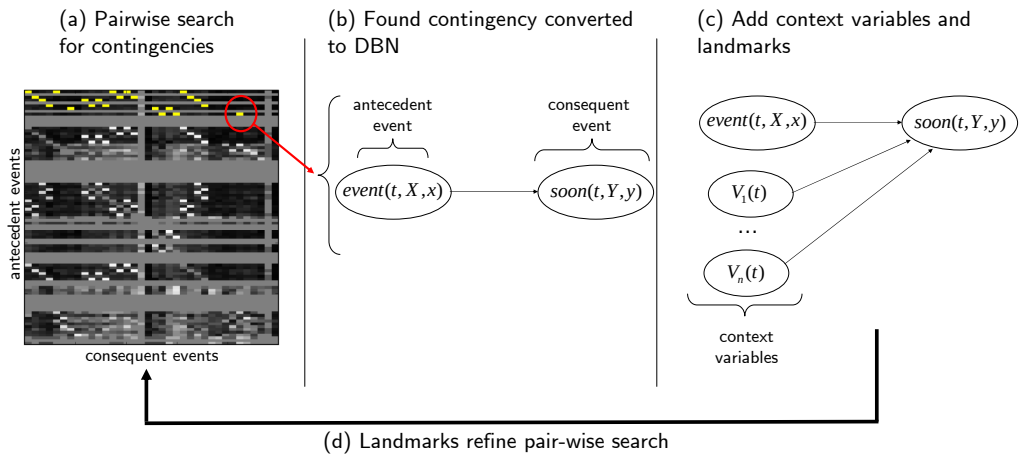


Figure 4.1: (a) Do a pairwise search for contingencies that use one event to predict another. The antecedent events are along the y-axis, and the consequent events are along the x-axis, The color indicates the probability that the consequent event will soon follow the antecedent event (lighter corresponds to higher probability). When the probability of the consequent event is sufficiently high, it is converted into a contingency (yellow). (b) When a contingency is found, it is used to create a DBN. (c) Once a DBN is created, context variables are added to make it more reliable. (d) The DBN creates a self-supervised learning problem to predict when the consequent event will follow the antecedent event. This allows new landmarks to be found. Those landmarks create new events for the pairwise search.

4.1.1 Contingency Definition

To define contingencies in a continuous environment, we have to discretize both the variable values and time. To discretize variable values, we create a special Boolean variable $event(t, X{\rightarrow}x)$ that is true if event $X_t{\rightarrow}x$ occurs

$$event(t, X{\rightarrow}x) = \begin{cases} \mathsf{true}, & X_t{\rightarrow}x \\ \mathsf{false}, & \text{otherwise} \end{cases} \tag{4.1}$$

To discretize time, we use a time window. We define the Boolean variable $soon(t, Y{\rightarrow}y)$ that is true if event $Y_t{\rightarrow}y$ occurs within a time window of length k

$$soon(t, Y{\rightarrow}y) = \begin{cases} \mathsf{true}, & \exists t' \; [t \leq t' < t + k \; \wedge \; event(t', Y{\rightarrow}y)] \\ \mathsf{false}, & \text{otherwise} \end{cases} \tag{4.2}$$

(Appendix C discusses how the agent learns k, which specifies the length of the time window.) With these variables, we define a contingency as

$$event(t, X{\rightarrow}x) \Rightarrow soon(t, Y{\rightarrow}y) \tag{4.3}$$

which represents the proposition that if the *antecedent event* $X_t{\rightarrow}x$ occurs, then the *consequent event* $Y{\rightarrow}y$ will occur within k timesteps.

4.1.2 The Pairwise Search

QLAP looks for contingencies using a pairwise search by tracking statistics on pairs of events $X{\rightarrow}x$ and $Y{\rightarrow}y$ to determine if the pair is a contingency of the form

$$event(t, X{\rightarrow}x) \;\Rightarrow\; soon(t, Y{\rightarrow}y) \tag{4.4}$$

QLAP learns a contingency $E_1 \;\Rightarrow\; E_2$ if when the event E_1 occurs, then the event E_2 is more likely to soon occur than it would have been otherwise

$$Pr(soon(t, E_2)|E_1(t)) > Pr(soon(t, E_2)) \tag{4.5}$$

where $Pr(soon(t, E_2))$ is the probability of event E_2 occurring within a random window of k timesteps. Specifically, the contingency is learned when

$$Pr(soon(t, E_2)|E_1(t)) - Pr(soon(t, E_2)) > \theta_{pen} = 0.05 \tag{4.6}$$

(This determination is made probabilistically using the beta distribution as described in Appendix B.)

QLAP performs this search considering all pairs of events, excluding those where

1. The consequent event is a magnitude variable (since this are handled by the models on magnitude variables introduced in Chapter 3 and discussed later in this chapter).
2. The consequent event is on a direction of change variable to the landmark value [0] (since we want to predict changes that result in moving towards or away from landmarks).
3. The antecedent event and the consequent event are on the same variable (since we want to learn how the values of variables are affected by other variables).

4.2 Converting Contingencies to DBNs

In this section we describe how QLAP converts a contingency of the form

$$event(t, X{\rightarrow}x) \Rightarrow soon(t, Y{\rightarrow}y) \tag{4.7}$$

into a dynamic Bayesian network. As described in Chapter 2, a dynamic Bayesian network (DBN) is a compact way to describe a probability distribution over time-series data. Dynamic Bayesian networks allow QLAP to identify situations when the contingency will be reliable.

4.2.1 Adding a Context

The consequent event may only follow the antecedent event in certain contexts, so we also want to learn a set of qualitative context variables $\mathcal{C}$ that predict when event $Y{\rightarrow}y$ will soon follow $X{\rightarrow}x$. This can be represented as a DBN r of the form

$$r = \langle \mathcal{C} : event(t, X{\rightarrow}x) \Rightarrow soon(t, Y{\rightarrow}y) \rangle \tag{4.8}$$

which we abbreviate to

$$r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle \tag{4.9}$$

In this notation, event $E_1 = X{\rightarrow}x$ is the antecedent event, and event $E_2 = Y{\rightarrow}y$ is the consequent event. We can further abbreviate a QLAP DBN r as

$$r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle \tag{4.10}$$

Figure 4.3 shows the correspondence between this notation and standard DBN notation. QLAP DBNs are only applicable in cases where the antecedent event occurs. The conditional probability table (CPT) of DBN r gives the probability that event $Y{\rightarrow}y$ will soon follow event $X{\rightarrow}x$ for each qualitative value in context $\mathcal{C}$. If the antecedent event does not occur, then the CPT does not define the probability for the consequent event occurring. If the antecedent event occurs, and the consequent event does follow soon after, we say that the DBN *succeeds*. Likewise, if the antecedent event occurs, and the consequent event does not follow soon after, we say that the DBN *fails*. These models are referred to as dynamic Bayesian networks and not simply Bayesian networks because we are using them to model a dynamic system. An example of a DBN learned by QLAP is shown in Figure 4.2.

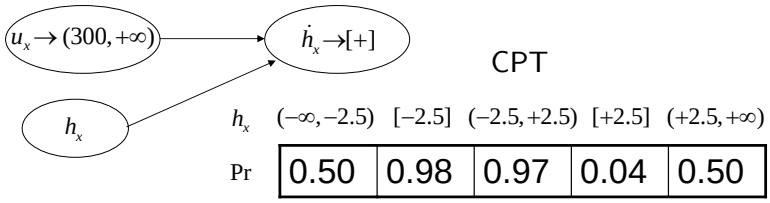


Figure 4.2: An example DBN. This DBN says that if the motor value of u_x becomes greater than 300, and the location of the hand, h_x, is in the range $-2.5 \leq h_x < 2.5$, then the variable $\dot{h}_x$ will most likely soon become [+] (the hand will move to the right). (The limits of movement of h_x are -2.5 and $+2.5$, and so the prior of 0.5 dominates outside of that range.)

The set $\mathcal{C} = \{v_1, \ldots, v_n\}$ consists of the variables in the conditional probability table (CPT) of the DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$. The CPT is defined over the product space

$$\mathcal{Q}(\mathcal{C}) = \mathcal{Q}(v_1) \times \mathcal{Q}(v_2) \times \cdots \times \mathcal{Q}(v_n) \tag{4.11}$$

Since $\mathcal{C}$ is a subset of the variables available to the agent, $\mathcal{Q}(\mathcal{C})$ is an abstraction of the overall state space $\mathcal{S}$

$$\mathcal{Q}(\mathcal{C}) \subseteq \mathcal{Q}(v_1) \times \mathcal{Q}(v_2) \times \cdots \times \mathcal{Q}(v_m) = \mathcal{S} \tag{4.12}$$

where $m \geq n$.[1]

4.2.2 Notation of DBNs

We define the reliability for $q \in \mathcal{Q}(\mathcal{C})$ for DBN r as

$$rel(r, q) = Pr(soon(t, E_2) | E_1(t), q) \tag{4.13}$$

which is the probability of success for the DBN for the value $q \in \mathcal{Q}(\mathcal{C})$. (Note that we may also say $rel(r, s)$ is the reliability of DBN r in state s.) These probabilities come from the CPT and are calculated using observed counts.

Best Reliability

The best reliability of a DBN gives the highest probability of success in any context state. We define the best reliability $brel(r)$ of a DBN r as

$$brel(r) = \max_{q \in \mathcal{Q}(\mathcal{C})} rel(r, q) \tag{4.14}$$

[1] In our experiments, we limit n to be 2.

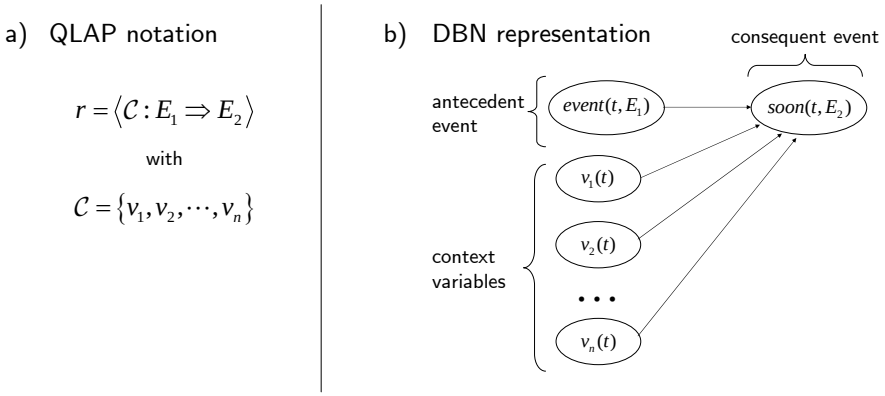


Figure 4.3: Correspondence between QLAP DBN notation and traditional graphical DBN notation. (a) QLAP notation of a DBN. Context $\mathcal{C}$ consists of a set of qualitative variables. Event E_1 is an *antecedent event* and event E_2 is a *consequent event*. (b) Traditional graphical notation. Boolean parent variable $event(t, E_1)$ is true if event E_1 occurs at time t. Boolean child variable $soon(t, E_2)$ is true event E_2 occurs within k timesteps of t. The other parent variables are the context variables in $\mathcal{C}$. 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 probability is undefined. The remaining probabilities are learned through experience.

(We require 5 actual successes for $q \in \mathcal{Q}(\mathcal{C})$ before it can be considered for best reliability. See Appendix B for a further explanation of how statistics are calculated.)

By increasing the best reliability $brel(r)$ we increase the reliability of DBN r. And we say that a DBN r is *sufficiently reliable* if at any time $brel(r) > \theta_{SR} = 0.75$.

Entropy

The entropy of a DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ is a measure of how well the context $\mathcal{C}$ predicts that event E_2 will soon follow event E_1. The *entropy* $H(Y)$ of a random variable Y is given by

$$H(Y) \quad = \quad -\sum_j Pr(Y = y_j) \log_2 Pr(Y = y_j)$$

The conditional entropy $H(Y|X)$ of a random variable Y given X is given by

$$H(Y|X) = \sum_i H(Y|X = x_i) Pr(X = x_i)$$

and is the weighted average of the entropy of Y given $X = x_i$, weighted by the probabilities $Pr(X = x_i)$. Since we only consider the timesteps where event E_1 occurs, we define the entropy $H(r)$ of a DBN r as

$$H(r) = \sum_{q \in \mathcal{Q}(\mathcal{C})} H(soon(t, E_2)|q, E_1(t)) Pr(q|E_1(t)) \tag{4.15}$$

By decreasing the entropy $H(r)$ of DBN r, we increase the determinism of DBN r.

4.3 Adding Context Variables

QLAP hillclimbs by iteratively adding context variables to DBNs to make them more reliable and deterministic. This hillclimbing process of adding one context variable at a time is inspired by

Drescher's marginal attribution [1991]. By only considering the next variable to improve the DBN, marginal attribution decreases the search space. Drescher spun off an entirely new model each time a context variable was added, and this resulted in a proliferation of models. To eliminate this proliferation of models, we instead modify the model by changing the context.

4.3.1 The Hillclimbing Measure

To improve a DBN r, QLAP hillclimbs on best reliability $brel(r)$ until r is sufficiently reliable, at which point QLAP hillclimbs on entropy $H(r)$. QLAP initially hillclimbs on best reliability because these DBN models will eventually be used for planning. In our experiments, we have found this to be necessary to make good plans because we want to find some context state in which the model is reliable. This allows the model to be used for planning, because the agent can first get to that reliable context state. However, we also want the predictive model to be deterministic, so after a model is sufficiently reliable, QLAP hillclimbs on entropy reduction.

To quantify the hillclimbing procedure, we say that a DBN r' is a *sufficient improvement* over DBN r if the Boolean function $isModelImprovement(r, r')$ returns true. This function is given in algorithmic form in Algorithm 1. Essentially, if r is sufficiently reliable, then r' must provide a reduction in entropy. If r is not sufficiently reliable, then r' must provide an increase in best reliability. The required amount of reduction in entropy or increase in best reliability is determined by the relative sizes of the contexts for r and r'.

Algorithm 1 $isModelImprovement$

Require: A DBN r and a DBN r'
let $|r|$ be the number of context variables in r
let $|r'|$ be the number of context variables in r'
let $\Delta_{|r|} = |r'| - |r|$
let $\theta_{pen} = 0.05$ be the needed improvement
if r is sufficiently reliable **then**
  if r' is not sufficiently reliable **then**
    return false
  end if
  if $|r| \leq |r'|$ **then**
    return $H(r) - H(r') > (1 + \Delta_{|r|}) \cdot \theta_{pen}$
  else
    return not $isModelImprovement(r', r)$
  end if
else
  if $|r| \leq |r'|$ **then**
    return $brel(r') - brel(r) > (1 + \Delta_{|r|}) \cdot \theta_{pen}$
  else
    return not $isModelImprovement(r', r)$
  end if
end if

4.3.2 The Hillclimbing Procedure

The hillclimbing procedure is not completely greedy. QLAP also considers the possibility that the DBN needs fewer context variables. So the hillclimbing algorithm for adding context variables

implemented in QLAP first sets aside the current context $\mathcal{C}$. Then, QLAP creates an entirely new context $\mathcal{C}'$ by hillclimbing by adding variables to improve the DBN. QLAP then uses the function $isModelImprovement$ to compare the new context $\mathcal{C}'$ with the original context $\mathcal{C}$ to see if r' with $\mathcal{C}'$ is an improvement over r with the original context $\mathcal{C}$. The pseudocode and additional details for this process are shown in Algorithm 1 in Appendix D.1.

4.4 Learning New Landmarks

Learning new landmarks allows the agent to see the world at a higher resolution. This increase in resolution allows existing models to be made more reliable, and it allows new models to be learned. QLAP has two mechanisms for learning landmarks. The first is to learn a new landmark to make an existing DBN more reliable. The second is to learn a new landmark that predicts the occurrence of an event.

4.4.1 New Landmarks on Existing DBNs

QLAP learns new distinctions based on previously-learned models (DBNs). For any particular DBN, predicting when the consequent event will follow the antecedent event is a supervised learning problem. This is because once the antecedent event occurs, the environment will determine whether the consequent event will occur. QLAP takes advantage of this supervisory signal to learn new landmarks that improve the predictive ability of DBNs.

We first describe how landmarks can make DBNs more reliable. We then discuss how QLAP searches for a landmark to improve a DBN, and we finally discuss the batch process for learning landmarks.

How Landmarks Change DBNs

Inserting a new landmark v^* into the open interval (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 a new landmark v^* into the quantity space $\mathcal{Q}(v)$ allows a new distinction to be made that may transform a DBN r into a new DBN r' in one of three ways.

1. The event E_1 is of the form $v{\rightarrow}(v_i^*, v_{i+1}^*)$, and E_1 becomes $v{\rightarrow}(v_i^*, v^*)$ or $v{\rightarrow}(v^*, v_{i+1}^*)$.
2. $v \in \mathcal{C}$ and $\mathcal{Q}(\mathcal{C})$ is updated.
3. $v \notin \mathcal{C}$, but once v^* is added to $\mathcal{Q}(v)$, then v is added to $\mathcal{C}$.

QLAP can then determine if r' is sufficient improvement over r using $isModelImprovement$.

The Search for a Landmark

For each DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$, QLAP searches for a landmark on each magnitude and motor variable v in each open interval $q \in \mathcal{Q}(v)$ that will sufficiently improve r. This search consists of four steps.

1. *Find the best cutpoint.* This is done using the method of Fayyad and Irani [1992]. To do this, each time the event E_1 occurs QLAP stores the value of variable $\tilde{v}$ and also if r was ultimately successful. (The growth in storage is not unlimited because only the last 200 activations are stored.) These values of $\tilde{v}$ are then sorted, and QLAP considers a cutpoint $c = (lb, ub)$ between each pair of values lb, and ub, that are different ($ub - lb > 0.001$). This

cutpoint c divides the set of success or failure labels S into those S^- whose associated value of $\tilde{v}$ is below c, and those S^+ whose associated value of $\tilde{v}$ is above c. These sets are used to calculate the information gain G_c of putting a cutpoint at that location, where

$$G_c = H(S) - \frac{|S^-|}{|S|}H(S^-) - \frac{|S^+|}{|S|}H(S^+) \tag{4.16}$$

and $|S|$ indicates the number of labels in set S. QLAP performs this calculation for all cutpoints and chooses the cutpoint c whose information gain G_c^* is maximum.[2]

2. *Determine if the cutpoint is good enough.* QLAP then determines if the cutpoint c is good enough to be a candidate landmark. The method of Fayyad and Irani [1993] uses a criterion based on the minimum description length principle to determine if a cutpoint should be accepted. In our experiments, we have found that this creates too many landmarks. Instead, we add potential landmarks based on both the information gain and the desirability of an additional landmark for an interval. To determine the desirability for a landmark for an interval, we use the probability of the world state being in this interval $Pr(v = q)$. And we call the product $G_c^* \cdot Pr(v = q)$ the *weighted gain*. We then consider a cutpoint c to be a candidate landmark if

$$G_c^* > \theta_{IG} \quad \text{and} \quad G_c^* \cdot Pr(v = q) > \theta_{IG}/2 \tag{4.17}$$

where $\theta_{IG} = 0.30$ in our experiments. As the agent learns more landmarks for a variable v, then $Pr(v = q)$ for any q will go down. This helps to keep QLAP from learning too many landmarks.

3. *Create the candidate landmark and potential new DBN.* This is done to determine if the cutpoint should be added as a new landmark. A chosen cutpoint $c = (lb, ub)$ is converted into a landmark v^* with range $[ub, \bar{\tilde{v}}]$ if ub is sufficiently close (within one bin as described in Section 4.4.2) to the maximum observed value $\bar{\tilde{v}}$ of $\tilde{v}$, range $[\underline{\tilde{v}}, lb]$ if lb is sufficiently close to the minimum observed value $\underline{\tilde{v}}$ of $\tilde{v}$, and range $[lb, ub]$ otherwise.[3]

4. *Adopt the landmark candidate if it sufficiently improves the DBN.* A landmark candidate v^* is adopted if it makes a sufficient improvement in r. This test is performed by updating r to r' and checking if r' is a sufficient improvement over r using the saved 200 activations. Note that if the landmark is on the antecedent variable, then we create two DBNs r'_1 and r'_2 corresponding to being below and above the landmark respectively, and let r' be the one with the higher number of successes.

This process for finding landmarks is done in a semi-batch fashion every 2000 timesteps. The pseudocode and additional details for this process are shown in Algorithm 2 in Appendix D.2.

[2]If DBN r currently has a single context variable v_1, then QLAP also looks for cutpoints within groups of values of $\tilde{v}$ partitioned by the different qualitative values of v_1. This partitioning is based on the qualitative value that v_1 had on the timesteps when event E_1 occurred and the value of $\tilde{v}$ was stored. Our experiments have shown that this finds some landmarks that the overall process misses.

[3]Since mathematically, $[lb, ub]$ is a closed set, the range should be $[lb + \epsilon, ub - \epsilon]$ because the values lb and ub were taken from data that falls outside of this set. However, in the implementation, we just make the range of the landmark $[lb, ub]$. Also note that for direction of change variables a range of $[-0.1, 0.1]$ is used for the given landmarks at 0.

4.4.2 New Landmarks to Predict Events

QLAP also learns new landmarks to predict events. QLAP needs this second landmark learning process because some events may not be preceded by another known event. If a landmark v^* is found that co-occurs with some event E, then the agent can predict the occurrence of event E by learning a DBN of the form $\langle \mathcal{C} : v \rightarrow v^* \Rightarrow E \rangle$. QLAP searches for such a landmark preceding event E by looking for a variable $\tilde{v}$ such that the distribution of $\tilde{v}$ is significantly different just before the event E than otherwise.

Finding Differences in Distributions using Bins

To find these landmarks, QLAP needs to compare the distribution of a variable $\tilde{v}$ just before an event E with the overall distribution of $\tilde{v}$. Since we are looking at the continuous values of variables, we use bins to estimate the distributions. We let the bin size for each bin $b_{\tilde{v}}$ for variable $\tilde{v}$ be two times the average change value (excluding the first timestep $t = 0$). A change is determined to occur if $t > 1$ and $|\tilde{v}(t) - \tilde{v}(t-1)| > 0.001$. QLAP then creates a landmark v^* for a bin $b_{\tilde{v}}$ when

$$Pr(\tilde{v}_{t-1} \in b_{\tilde{v}} | E(t)) - Pr(\tilde{v}_{t-1} \in b_{\tilde{v}}) > \theta_E \tag{4.18}$$

where $\theta_E = 0.30$, and $b_{\tilde{v}}$ corresponds to the bin that has the highest value of $Pr(\tilde{v}_{t-1} \in b_{\tilde{v}} | E(t)) - Pr(\tilde{v}_{t-1} \in b_{\tilde{v}})$. The range $[lb, ub]$ of the landmark corresponding to $b_{\tilde{v}}$ is the size of the bucket $b_{\tilde{v}}$.

These probabilities are computed under two different normalization conditions. The first is that QLAP normalizes over three buckets in each direction of $b_{\tilde{v}}$. This allows QLAP to find local spikes in differences of the probability distributions. The second normalization condition is to normalize the probability over all of the buckets. QLAP first looks for a landmark using the first normalization condition. If none is found, QLAP looks for a landmark using the second normalization condition. The pseudocode and additional details for this process are shown in Algorithm 3 in Appendix D.3.

4.5 Magnitude DBN Models

A magnitude value can be less than, greater than, or equal to a qualitative value. We want to have models for a variable $\tilde{v}$ reaching a qualitative value v^*. Intuitively, if we want $v = v^*$ and currently $v(t) < v^*$, then we need to set $\dot{v} = [+]$ as is shown in Figure 4.4. This section describes how this process is modeled.

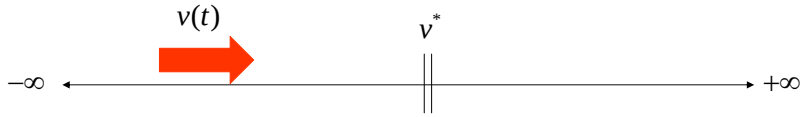


Figure 4.4: Moving to a landmark on a number line.

For each magnitude variable v and each qualitative value $q \in \mathcal{Q}(v)$, QLAP creates two models, 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. For each magnitude variable Y and each value $y \in \mathcal{Q}(Y)$, these models can be written as

$$\dot{Y} \rightarrow [+] \;\Rightarrow\; Y \rightarrow y \tag{4.19}$$

$$\dot{Y} \rightarrow [-] \;\Rightarrow\; Y \rightarrow y \tag{4.20}$$

The first one means that if $Y_t < y$ and $\dot{Y} = [+]$, then eventually event $Y{\rightarrow}y$ will occur (the second model is analogous in this discussion). As the notation suggests, we can treat $\dot{Y}{\rightarrow}[+] \Rightarrow Y{\rightarrow}y$ similarly to how we treat a contingency, and we can learn context variables for when this model will be reliable. These models are based on the test-operate-test-exit (TOTE) models of Miller *et al.* [1960].

Magnitude DBNs do not use the "soon" predicate because how long it takes to reach a qualitative value is determined by how far away the variable is from that value. Instead, statistics are gathered on magnitude DBNs when the agent sets $\dot{Y} = [+]$ to bring about $Y{\rightarrow}y$. The first model is successful if $Y{\rightarrow}y$ occurs while $\dot{Y} = [+]$, and it fails if the agent is unable to maintain $\dot{Y} = [+]$ long enough to bring about event $Y{\rightarrow}y$.

We can use this supervisory signal to add a context just like with change DBNs. When we do this, we get magnitude DBN models r^+ and r^- of the form

$$r^+ = \langle \mathcal{C} : do(t, \dot{Y}{\rightarrow}[+]) \Rightarrow reach(t, Y{\rightarrow}y) \rangle \quad (4.21)$$

$$r^- = \langle \mathcal{C} : do(t, \dot{Y}{\rightarrow}[-]) \Rightarrow reach(t, Y{\rightarrow}y) \rangle \quad (4.22)$$

where the parent variable $do(t, E)$ is a predicate analogous to $event(t, E)$ that is true if the agent is working to bring about E and the value is moving towards E. And the child variable $reach(t, E)$ is analogous to $soon(t, E)$ and is true if the value reaches E. These DBNs can be abbreviated to

$$r^+ = \langle \mathcal{C} : \dot{Y}{\rightarrow}[+] \Rightarrow Y{\rightarrow}y \rangle \quad (4.23)$$

$$r^- = \langle \mathcal{C} : \dot{Y}{\rightarrow}[-] \Rightarrow Y{\rightarrow}y \rangle \quad (4.24)$$

Two such magnitude DBNs are created for each qualitative value on each magnitude variable.[4] Like change DBNs, magnitude DBNs will be used in planning as described in Chapter 5.

[4]While context variables are learned on magnitude DBNs, experiments showed that landmarks learned on these models were not useful to the agent, so these models are not used for learning landmarks in QLAP.

Chapter 5

From Models to Actions and Plans

QLAP uses the learned models to create plans for actions. There are two broad planning frameworks within AI: STRIPS-based goal regression [Nilsson, 1980], and Markov Decision Process (MDP) planning [Puterman, 1994]. Goal regression has the advantage of working well when only some of the variables are relevant, and MDP planning has the advantage of providing a principled framework for probabilistic actions [Boutilier *et al.*, 1999]. Planning in QLAP was designed to exploit the best of both frameworks. As described in Chapter 1, a broad principle of QLAP is that the agent should fragment the environment to make learning and planning more tractable. QLAP uses MDP planning to plan within each learned fragment, and QLAP uses goal regression to stitch the fragments together.

QLAP creates an *action* to achieve each qualitative value of each variable. QLAP creates *plans* from the learned models, and each plan is a different way to perform an action. An action can have zero, one, or many plans. If an action has no plans it cannot be performed. If an action has multiple plans, then the action can be performed in more than one way. The actions that can be called by each plan are QLAP actions to bring about qualitative events. This stitches the plans together and leads to a hierarchy because plans call other QLAP actions as if they were primitive actions. This hierarchical action network encodes all of the learned skills of the agent. This process is shown in Figure 5.1.

Each plan is represented as an MDP, and the policy for each plan is learned using both model-based and model-free methods. QLAP uses MDP planning instead of only goal regression because the transitions are probabilistic. And since the variables in the state space of the plan only come from the model, we minimize the problem of state explosion common to MDP planning. Additionally, we use MDP planning instead of a more specific planning algorithm such as RRT [Kuffner Jr and Lavalle, 2000] because the actions taken by the plan may be arbitrary, such as "hit the block off the table." And most such planning algorithms are designed specifically for moving in space.

In this chapter, we first define actions and plans in QLAP. We then discuss how change and magnitude DBNs are converted into plans. We then discuss how QLAP can learn when variables need to be added to the state space of a plan, and we conclude with a description of how actions are performed in QLAP.

5.1 Actions and Plans in QLAP

Actions are how the QLAP agent brings about changes in the world. An action $a(v, q)$ is created for each combination of qualitative variable v and qualitative value $q \in \mathcal{Q}(v)$. An action $a(v, q)$ is *called* by the agent and is said to be *successful* if $v = q$ when it terminates. Action $a(v, q)$ *fails* if it terminates with $v \neq q$. Statistics are tracked on the reliability of actions. The reliability of an

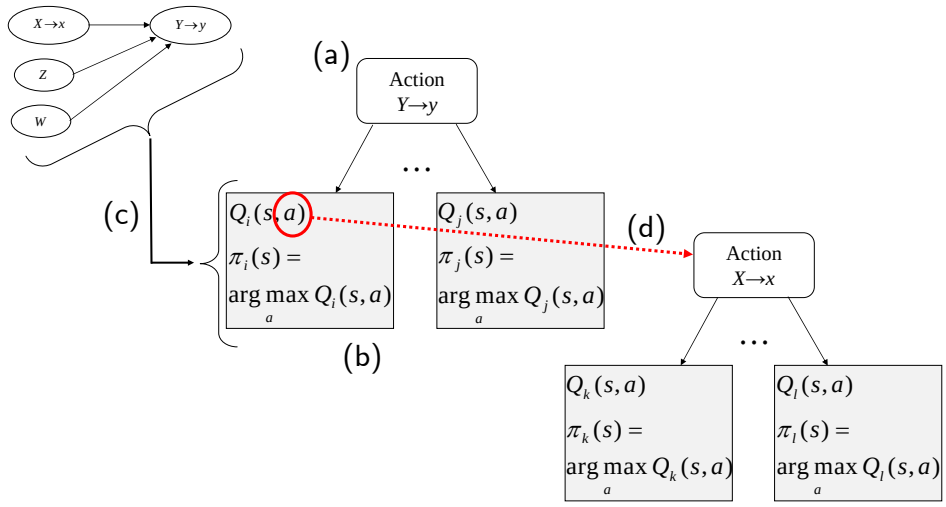


Figure 5.1: Planning in QLAP. (a) QLAP creates an action for each qualitative value of each variable. This action is to bring variable Y to value y. (b) Each action can have multiple plans. Each plan is a different way to perform the action. The MDP plan is represented as an option o_i with policy π_i. (c) Plans are created from models. The state space for an MDP is the cross product of the values of X, Y, Z, and W from the model (although more can be added if needed). (d) The actions for each plan are QLAP actions to move to different locations in the state space of the MDP. This is reminiscent of goal-regression. In this figure, we see that one of the actions for plan o_i is to call the QLAP action to bring about $X{\rightarrow}x$. This link results from event $X{\rightarrow}x$ being the antecedent event of the model to bring about event $Y{\rightarrow}y$.

action a is denoted by $rel(a)$, which gives the probability of succeeding if it is called.

When an action is called, the action chooses a plan to carry it out. Each plan is associated with only one action, and an action can have multiple different plans where each plan is a different way to perform the action. This gives QLAP the advantage of being able to use different plans in different situations instead of having one big plan that must cover all situations. As with actions, we say that a plan associated with action $a(v, q)$ is successful if it terminates with $v = q$ and fails if it terminates with $v \neq q$.

Each plan is represented as a policy π_i over an MDP $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, T_i, R_i \rangle$. As described in Chapter 2, a Markov Decision Process (MDP) is a framework for temporal decision making. QLAP learns multiple MDPs, and each MDP represents a small part of the world. These MDPs come from the models learned by QLAP. The actions available for each MDP are a subset of all the QLAP actions. In this way, the actions and plans of QLAP are tied together and planning takes the flavor of goal regression.

We can think of this policy π_i as being part of an option $o_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$. As described in Chapter 2, 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 QLAP actions instead of being over primitive actions or options.

We use the terminology of a plan being an option because options are common in the literature, and because QLAP takes advantage of the non-Markov termination function β_i that can terminate after a fixed number of timesteps. However, plans in QLAP differ from options philosophically

because options are usually used with the assumption that there is some underlying large MDP. QLAP assumes no large, underlying MDP, but rather creates many little, independent MDPs that are connected by actions. Each small MDP $\mathcal{M}_i$ created by QLAP has one policy π_i.

See Table 5.1 for a summary of the major aspects of models, actions, and plans.

Table 5.1: Objects in QLAP

Object	Properties	Description
Model	$r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$	model of the environment
–	$rel(r)$	overall reliability of r
–	$rel(r, s)$	reliability of r in state s
–	$rel(r, q)$	reliability of r in context value $q \in \mathcal{Q}(\mathcal{C})$
–	$brel(r)$	best reliability of r
–	*sufficiently reliable*	if $brel(r) > \theta_{SR} = 0.75$
–	$H(r)$	conditional entropy of r
Action	$a(v, q)$	action to set qualitative variable v to $q \in \mathcal{Q}(v)$
–	$rel(a)$	reliability of action a
–	*called*	when action is started by agent
–	*processed*	action has already been called, but has not terminated
–	*success*	if terminates with $v = q$
–	*fail*	if terminates with $v \neq q$
Plan	$o = \langle \mathcal{I}, \pi, \beta \rangle$	a way to perform an action $a(v, q)$
–	$rel(o)$	overall reliability of option o
–	$rel(o, s)$	reliability of option o in state s
–	$brel(o)$	best reliability of option o
–	*called*	when plan is started by agent
–	*processed*	plan has already been called, but has not terminated
–	*success*	if terminates with $v = q$
–	*fail*	if terminates with $v \neq q$

5.2 Converting Change DBNs to Plans

When a DBN of the form $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ becomes sufficiently reliable[1] it is converted into a plan to bring about $Y{\rightarrow}y$. This plan can then be called by the action $a(Y, y)$.

This plan is in the form of an MDP $\mathcal{M}_i$. In this section, we will first describe how QLAP creates MDP $\mathcal{M}_i$ from a DBN r_i. We will then describe how QLAP learns a policy for this MDP. And finally, we will describe how this policy is mapped to an option.

5.2.1 Creating the MDP from the DBN

QLAP converts DBNs of the form $r_i = \langle \mathcal{C} : X \rightarrow x \Rightarrow Y \rightarrow y \rangle$ to an MDP of the form $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, T_i, R_i \rangle$. The state space $\mathcal{S}_i$ comes from the variables in DBN r_i. The set of actions $\mathcal{A}_i$ are QLAP actions to bring the agent to the different states of $\mathcal{S}_i$. The transition function T_i comes from the CPT of r_i and the reliability $rel(a)$ of different actions $a \in \mathcal{A}_i$, and the reward function comes from achieving the goal state $Y = y$. The details are provided in the rest of this section.

[1] There are other restrictions that will be discussed in Chapter 6.

Defining the State Space

The state space $\mathcal{S}_i$ for $\mathcal{M}_i$ consists of the Cartesian product of the values of X, Y and $\mathcal{C}$. Recall that $\mathcal{Q}(v)$ is the set of qualitative values for qualitative variable v. The state space $\mathcal{S}_i$ for MDP $\mathcal{M}_i$ is

$$\mathcal{S}_i = \mathcal{Q}(\mathcal{C}) \times \mathcal{Q}(X) \times \mathcal{Q}(Y) \tag{5.1}$$

(we will see in Section 5.4 how more variables can be added to state spaces).

Defining the Action Space

The qualitative representation defines a set of actions $\mathcal{A}$. Recall that QLAP creates an action $a(v, q)$ to achieve each qualitative value $q \in \mathcal{Q}(v)$ for each qualitative variable v. The action space $\mathcal{A}_i$ for DBN $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ is the set of actions to reach each state of the context $\mathcal{C}$ plus one or more actions related to setting the antecedent event. More formally, the set of actions $\mathcal{A}_i \subseteq \mathcal{A}$ consists of

1. The set of actions $\mathcal{A}_\mathcal{C}$ that allows the agent to move within the context

$$\mathcal{A}_\mathcal{C} = \{a(v, q) | v \in \mathcal{C} \text{ and } q \in \mathcal{Q}(v)\} \tag{5.2}$$

2. The action $a(X, x)$ that brings about the antecedent event of r_i. And if X is a magnitude variable, the actions to each value $x \in \mathcal{Q}(X)$,

Thus, the set of actions $\mathcal{A}_i$ is

$$\mathcal{A}_i = \begin{cases} \mathcal{A}_\mathcal{C} \cup \{a(X, q) | q \in \mathcal{Q}(X)\}, & \text{if } X \text{ is a magnitude variable} \\ \mathcal{A}_\mathcal{C} \cup \{a(X, x)\}, & \text{otherwise} \end{cases} \tag{5.3}$$

Not all actions of $\mathcal{A}_i$ are applicable in all states. We denote $\mathcal{A}_i^s \subseteq \mathcal{A}_i$ as the set of actions applicable in state s. To create $\mathcal{A}_i^s$ from $\mathcal{A}_i$, QLAP subtracts each action $a(v, q)$ from $\mathcal{A}_i$ that meets any of the following criteria in state s:

1. v is a magnitude variable and $v = q$
2. $a(v, q)$ would cause infinite regress by causing action $a(Y, y)$ to be called again. For example, consider if r_i were of the form

$$r = \langle \{Y\} : X{\rightarrow}x \Rightarrow \dot{Y}{\rightarrow}[+] \rangle \tag{5.4}$$

   If $Y = (-\infty, 4)$ in state s, then the action $a(Y, [4])$ would need to make $\dot{Y}{\rightarrow}[+]$ and this would lead to a loop. Therefore, action $a(Y, [4])$ is not applicable in state s. (This is related to choosing plans and infinite regress as discussed in Chapter 6.)
3. v is a magnitude variable on the antecedent event (meaning v is variable X), and the action $a(X, q)$ does not make the antecedent event $X{\rightarrow}x$ more easily achievable.

   Recall from Chapter 3 that events on magnitude variables have a direction. In the DBN $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$, if X is a magnitude variable, then the antecedent event is either $\uparrow X{\rightarrow}x$ or $\downarrow X{\rightarrow}x$. The event $\uparrow X_t{\rightarrow}x$ means that $X_{t-1} < x$ and $X_t = x$, and the event $\downarrow X_t{\rightarrow}x$ means that $X_{t-1} > x$ and $X_t = x$.

   An action makes the event $\uparrow X{\rightarrow}x$ more easily achievable if currently $X > x$ and the action is of the form $a(X, q)$ with $q < x$. Likewise, an action makes the event $\downarrow X{\rightarrow}x$ more easily achievable if currently $X < x$ and the action is of the form $a(X, q)$ with $q > x$.

Because change plans are based on contingencies, the action to bring about the antecedent event is special. When this action terminates successfully, the agent waits k timesteps (the length of the learned time window) to see if the consequent event will occur. If the consequent event occurs within this time window, then the plan terminates successfully because $v = q$. If it does not, then the plan continues by calling another action according to the learned policy.

Defining the Transition Function

To construct the transition function $T_i : \mathcal{S}_i \times \mathcal{A}_i^s \rightarrow \mathcal{S}_i$, QLAP must compute a set of possible next states for each $s \in \mathcal{S}_i$ and $a \in \mathcal{A}_i^s$. It must then compute the distribution $P(s'|s, a)$. To compute $P(s'|s, a)$, QLAP uses the statistics gathered on DBN r_i and the statistics gathered to estimate the probability $rel(a)$ of success for action a. To keep planning tractable, QLAP limits the number of next states with positive probability to two. We organize the discussion of the transition probabilities based on the type of action:

1. *Moving to a context value.* For an action $a(v, q)$ (which we abbreviate with a) with $v \in \mathcal{C}$ to change the value of a context variable, QLAP considers two possible next states.

   (a) State s'_1 where the action is successful and the only change is that $v = q$.
   (b) State s'_2 where the action fails and $s'_2 = s$.

   The probability distribution over s' then is $Pr(s'_1|s, a) = rel(a)$ and $Pr(s'_2|s, a) = 1 - rel(a)$.

2. *Achieving the antecedent event.* For the action $a(X, x)$ to bring about the antecedent of r_i, QLAP also considers two possible next states.

   (a) State s'_1 is where the antecedent event occurs[2] and the consequent event follows, so that s'_1 is the same as s except that $X = x$ and $Y = y$.
   (b) 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 $X = x$.

   The probability distribution over s' is $Pr(s'_1|s, a) = rel(r_i, s)$ and $Pr(s'_2|s, a) = 1 - rel(r_i, s)$.

   An exception to this is if the antecedent event is on a magnitude variable, and the state is on the "wrong side." If X is a magnitude variable, then the antecedent event is either $\uparrow X \rightarrow x$ or $\downarrow X \rightarrow x$. For the event $\uparrow X_t \rightarrow x$, if state s has $X > x$, then the state is on the wrong side to make the event occur (analogously for $\downarrow X_t \rightarrow x$). In this case, $Pr(s'_1|s, a) = 0$ and $Pr(s'_2|s, a) = 1$.

Defining the Reward Function

The reward function penalizes each action with a cost of 2, but gives a reward of 10 for reaching the goal of $Y = y$. Formally, this is written as

$$R(s, a, s') = \begin{cases} 10 - 2, & \text{if } Y = y \text{ in state } s' \\ -2, & \text{otherwise} \end{cases} \tag{5.5}$$

[2]Or if it is already satisfied in the case of a direction of change variable (same for (b)).

5.2.2 Learning a Policy for the MDP

QLAP uses three different methods to learn a policy for an MDP. (1) QLAP uses the transition model described above to do dynamic programming to learn the policy. (2) As the agent further experiences the world, this policy is updated using the temporal difference learning method Sarsa. (3) And as the model improves, the policy is updated using Dyna.

Planning Based on a Transition Model: Dynamic Programming

Since the transition function T_i and the reward function R_i have been defined, QLAP can initially learn the policy π_i by learning a Q-table using dynamic programming with value iteration [Sutton and Barto, 1998]. As described in Chapter 2, this equation is

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right] \tag{5.6}$$

with $\gamma = 0.9$.

Learning from Experience: Sarsa

Dynamic programming uses the statistics gathered on individual actions to estimate transition probabilities. But the agent will gather experience in the world, so we do not have to rely on these estimates. We can use that experience to update the value function. We do this using the temporal difference learning method Sarsa as described in Chapter 2.

As the option executes, each time an action terminates, the agent performs a Sarsa update [Sutton and Barto, 1998]. As described in Chapter 2, the Sarsa update equation with eligibility traces is

$$\forall s,a : Q'(s,a) = Q(s,a) + e(s,a)\alpha[r' + \gamma Q(s',a') - Q(s,a)] \tag{5.7}$$

where $r' = R(s) - .01t$ where t is the number of timesteps for the option and

$$R(s) = \begin{cases} 10, & \text{if } Y = y \text{ in state } s \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

And the (replacing) eligibility trace is

$$e(s,a) \leftarrow \lambda\gamma e(s,a) \tag{5.9}$$

with $\lambda = 0.9$ and $\gamma = 0.9$.

QLAP does not discount exponentially based on the lower-level timesteps but instead treats MDP transitions as a single timestep. We do this because we are using a qualitative representation, and how long it takes to achieve an event may be a function of how far away it is from the landmark. However, QLAP does reward faster transitions as can be noted from the Sarsa reward $r' = R(s) - .01t$.

Updating the Plan with a Better Transition Model: Dyna

As the agent gathers more statistics, its transition model may be improved. QLAP uses the Dyna framework to incorporate these possible improvements in the transition model. As described in Chapter 2, Dyna is a framework for incorporating knowledge gained from experience with knowledge

gained from planning. Using the current model, QLAP updates each state and action value of the Q table once using the equation

$$\forall s, a : Q'(s,a) = (1-\alpha)Q(s,a) + \alpha \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') \right] \tag{5.10}$$

The learning rate $\alpha = 0.2$.

5.2.3 Mapping the Policy to an Option

An option has the form $o_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$. We have described how the policy π_i is learned. When an option o_i is created for a DBN $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$, the set of initiation states $\mathcal{I}_i$ is the set of all states.

The termination function β_i terminates option o_i when it *succeeds* (the consequent event occurs) or when it exceeds resource constraints (300 timesteps, or 5 action calls) or when the agent gets stuck. The agent is considered stuck if none of the self variables (see Chapter 5 for a discussion of how the agent learns which variables are part of "self") or variables in $\mathcal{S}_i$ change in 10 timesteps.

5.3 Converting Magnitude DBNs into Plans

As discussed in Chapter 4, each qualitative value $y \in \mathcal{Q}(Y)$ on each magnitude variable Y has two models

$$r^+ = \langle \mathcal{C} : \dot{Y}{\rightarrow}[+] \Rightarrow Y{\rightarrow}y \rangle \tag{5.11}$$

$$r^- = \langle \mathcal{C} : \dot{Y}{\rightarrow}[-] \Rightarrow Y{\rightarrow}y \rangle \tag{5.12}$$

that correspond to achieving the event $Y{\rightarrow}y$ from below and above the value $Y = y$, respectively. Both of these models are converted into a plan to achieve $Y{\rightarrow}y$. The result of this is that each action $a(v,q)$ on a magnitude variable has two plans. One plan to perform the action when $v < q$, and another plan to perform the action when $v > q$. Each magnitude DBN r_i is converted into a plan in the form of an MDP $\mathcal{M}_i$. For MDP $\mathcal{M}_i$, the state space $\mathcal{S}_i$, the set of available actions $\mathcal{A}_i^s$, the transition function T_i, and the reward function R_i are computed similarly as they are for change plans. The state space $\mathcal{S}_i$ and reward function R_i are created in exactly the same way. The action space $\mathcal{A}_i^s$ and the transition function T_i have some differences, as will be explained in the following two subsections.

5.3.1 Defining the Action Space

The action space is computed as described in Section 5.2.1 with the following modifications:

1. 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. Similarly for reaching $v = q$ from above the number line. The *wait* action is added to $\mathcal{A}_i$ but it is only applicable in states $s \in \mathcal{S}_i$ where the antecedent event is satisfied. For example, for r^+, it must be that $\dot{Y} = [+]$.
2. If the antecedent event is satisfied, then the action to bring about the antecedent event is not applicable.
3. There are no applicable actions on the "wrong side" of a magnitude DBN. For example, if the model is $r^+ = \langle \mathcal{C} : \dot{Y}{\rightarrow}[+] \Rightarrow Y{\rightarrow}[2.0] \rangle$ and $Y = (2.0, +\infty)$ in state s, then there are no applicable actions in state s.

The *wait* action keeps the agent waiting (and maintaining its current motor value) as long as the antecedent event is satisfied. If the consequent event occurs, it terminates successfully. If not, it calls another action according to the learned policy.

5.3.2 Defining the Transition Function

Actions to move to a context value have the same transition probabilities as they do for change plans. For magnitude plans, the transition to achieve the antecedent event is calculated in the same way as actions to achieve a context value. For the *wait* action, the probability of success comes from $rel(r, s)$ of model r, just like with direction of change DBNs.

5.4 Improving the State Space of Plans

The state space of a plan consists of the Cartesian product of the values of the variables in the model from which it was created. But what if there are variables that were not part of the model, but that are nonetheless necessary to successfully carry out the plan? To learn when new variables should be added to plans, QLAP keeps statistics on the reliability of each plan and uses those statistics to determine when a variable should be added.

5.4.1 Tracking Statistics on Plans

QLAP tracks statistics on plans the same way it does when learning models. For change DBN models, QLAP tracks statistics on the reliability of the contingency. For magnitude models, QLAP tracks statistics on the ability of a variable to reach a qualitative value if moving in that direction. For plans, QLAP tracks statistics on the agent's ability to successfully complete the plan when called.

To track these statistics on the probability of a plan o being successful, QLAP creates a *second-order* model

$$r_o^2 = \langle \mathcal{C}_o : call(t, o) \Rightarrow succeeds(t, o) \rangle \tag{5.13}$$

The child variable of second-order DBN r_o^2 is $succeeds(t, o)$, which is true if option o succeeds after being called at time t and is false otherwise. The parent variables of r_o^2 are $call(t, o)$ and the context variables in $\mathcal{C}_o$. The Boolean variable $call(t, o)$ is true when the option is called at time t and is false otherwise. When created, model r_o^2 initially has an empty context, and context variables are added in as they are for magnitude and change models. The notation for these models is the same as for magnitude and change models: QLAP computes $rel(o)$, $rel(o, s)$ and $brel(o)$. Therefore a plan can also be sufficiently reliable if at any time $brel(o) > \theta_{SR} = 0.75$.

5.4.2 Adding New Variables to the State Space

Second-order models 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 to the state space $\mathcal{S}_i$ of its associated MDP $\mathcal{M}_i$. For example, for a plan created from model $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$, the state space $\mathcal{S}_i$ is updated so that

$$\mathcal{S}_i = \mathcal{Q}(\mathcal{C}_o) \times \mathcal{Q}(\mathcal{C}) \times \mathcal{Q}(X) \times \mathcal{Q}(Y) \tag{5.14}$$

(variables in more than one of $\mathcal{C}_o$, $\mathcal{C}$, $\{X\}$, or $\{Y\}$ are only represented once in $\mathcal{S}_i$). For both magnitude and change options, an action $a(v, q)$ where $v \in \mathcal{Q}(\mathcal{C}_o)$ is treated the same way as those where $v \in \mathcal{Q}(\mathcal{C})$.

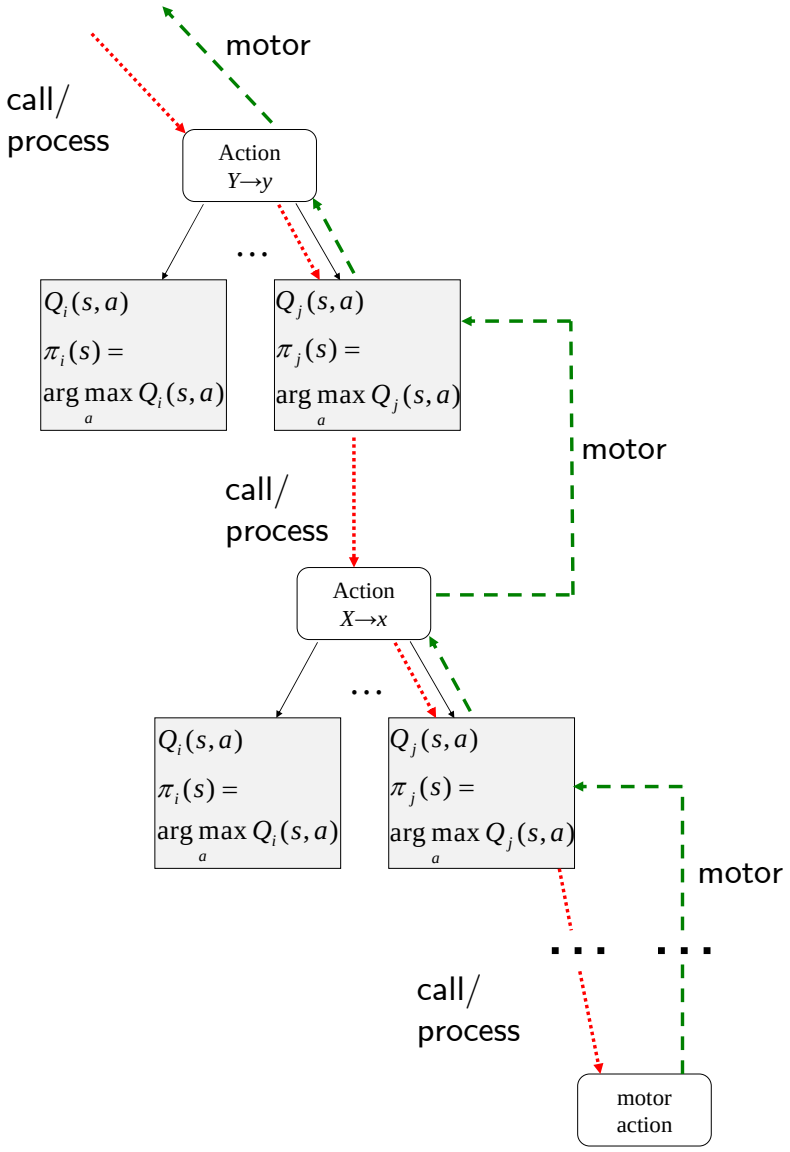


Figure 5.2: When an action is called, it chooses a plan. Plans, in turn, choose actions based on the plan's policy. This process continues until a motor action is reached, at which point the motor value is passed back up. The action is then processed until it terminates. While it is being processed, the plans below it will call actions according to their policies. So there is always a path from the called action to a motor action, but that path changes as the action is processed.

5.5 Performing Actions

QLAP actions are performed using plans, and these plans call other QLAP actions. This leads to a hierarchy of plans and actions. If an action $a(u, q)$ is called on a motor variable u, then QLAP returns a random motor value within the range covered by the qualitative value $u = q$. In the remaining part of this section, we explain how non-motor actions are performed by the QLAP agent.

5.5.1 Calling and Processing Actions

When an action is *called*, it chooses a plan and then starts executing the policy of that chosen plan. Executing that policy results in more QLAP actions being called, and this process continues until a motor action is reached.[3] At which point, the motor value is passed back up. This hierarchical calling of actions results in a *call list*.[4] See Figure 5.2.

This hierarchical structure of actions and plans means that multiple actions will be performed simultaneously. Each plan only keeps track of what action it is currently performing. And when that action terminates, the next action is called according to the policy of the plan. So as the initial action called by the agent is being processed, the path between that initial action and a motor actions continually changes.[5] See Appendix F for further details.

5.5.2 Terminating Actions

An action $a(v, q)$ terminates if $v = q$, in which case it *succeeds*. It also terminates if it *fails*. An action fails if

1. it has no plans, or
2. it has no plan whose action corresponding to the antecedent event of the model leading to the plan is not already in the call list, or
3. its chosen plan fails.

Similar to an action, a plan to bring about $v = q$ terminates if $v = q$, in which case it *succeeds*. It also terminates if it *fails*. A plan to bring about $v = q$ fails if

1. the termination function β is triggered by resource constraints, or
2. there is no applicable action in the current state, or
3. the action chosen by the policy is already in the call list, or
4. the action chosen by the policy immediately fails when it is called.

5.6 Conclusion

This chapter described how QLAP creates actions and plans and how they are performed. An action on a change variable can have zero, one, or multiple plans. An action on a magnitude variable has two plans for reaching that value from above and below on the number line.

[3] The issue of infinite regress is discussed in Chapter 6.

[4] Note that we use a list instead of a stack. This means that when the top level action is completed, all actions below are terminated.

[5] Note that if a direction of change action is called as an action from some plan and it is already achieved, then its plan treats it as not being achieved. This is necessary because it may need to get the motor command that achieves it to support some higher action.

Chapter 6

Exploration and Development

The QLAP agent explores and learns autonomously without being given a task. This autonomous exploration and learning raises many issues. For example, how can the agent decide what is worth exploring? And, as the agent explores, it learns new representations. How can it keep from learning unnecessary representations and getting bogged down? And should the agent use the same criteria for learning all representations? Or should it treat some representations as especially important? And finally, can the agent learn that some parts of the environment can be controlled with high reliability and low latency so that they can be considered part of "self"?

Previous chapters have explained how QLAP learns representations that take the form of landmarks, DBNs, plans, and actions. This chapter explains how learning in QLAP unfolds over time. We first discuss how the agent explores the environment. We then discuss developmental restrictions that determine what representations the agent learns and the order in which it learns them. We then discuss how QLAP pays special attention to goals that are hard to achieve. And finally, we discuss how the agent learns what is part of "self."

6.1 Exploration

The QLAP agent explores the environment autonomously without being given a task. Instead of trying to learn to do a particular task, the agent tries to learn to predict and control all of the variables in its environment. However, this raises difficulties because there might be many variables in the environment, and some may be difficult or impossible to predict or control. This section explains how the agent determines what should be explored and the best way to go about that exploration.

Initially, the agent motor babbles for 20,000 timesteps (see Appendix E for a description of how motor babbling is done). After that point, QLAP begins to practice its learned actions. An outline of the execution of QLAP is shown in Algorithm 2. The agent continually makes three types of choices during its exploration. These choices vary in time scale from coarse to fine:

1. The agent chooses a learned action $a(v, q)$ to practice.
2. The agent chooses the best plan o_i for performing the action $a(v, q)$.
3. Within plan o_i, the agent chooses the action based policy π_i.

6.1.1 Choosing a Learned Action to Practice

One method for choosing where to explore is to measure prediction error and than to motivate the agent to explore parts of the space for which it currently does not have a good model. This

form of intrinsic motivation is used in [Huang and Weng, 2002; Marshall *et al.*, 2004]. However, focusing attention on states where the model has poor prediction ability can cause the agent to explore spaces where learning is too difficult.

Schmidhuber [1991] proposed a method whereby an agent learns to predict the decrease in the error of the model that results from taking each action. The agent can then choose the action that will cause the biggest decrease in prediction error. Oudeyer, Kaplan, and Hafner [2007] apply this approach with a developing agent and have the agent explore regions of the sensory motor space that are expected to produce the largest decrease in predictive error. Their method is called Intelligent Adaptive Curiosity (IAC).

QLAP uses IAC to determine which action to practice. After the motor babbling period of 20,000 timesteps, QLAP chooses a motor babbling action with probability 0.1, otherwise it uses IAC to choose a learned action to practice. Choosing a learned action to practice consists of two steps: (1) determine the set of applicable actions that could be practiced in the current state s, and (2) choose an action from that set.

Determining the Set of Applicable Actions

The set of applicable actions to practice consists of the set of actions that are not currently accomplished, but could be performed. For a change action, this means that the action must have at least one plan. For a magnitude action $a(v, q)$, this means that if $v_t < q$ then $a(\dot{v}, [+])$ must have at least one plan (and similarly for $v_t > q$).

Choosing the Action from the Set of Applicable Actions

QLAP chooses an action to practice by assigning a weight w_a to each action a in the set of applicable actions. The action is then chosen randomly based on this weight w_a. The weights are assigned using Intelligent Adaptive Curiosity (IAC) [Oudeyer *et al.*, 2007]. IAC first measures the change in the agent's ability to perform the action over time and then chooses actions where that ability is increasing.

To use IAC to compute w_a for action a, we store the reliability $rel(a)$ each time action a terminates. We then let

$$w_a = max(0.001, rel_a^{new} - rel_a^{old}) \tag{6.1}$$

where rel_a^{new} is the average reliability of a after the most recent $\theta = 25$ calls, and rel_a^{old} is the average reliability of a over the $\theta = 25$ calls prior to the most recent $\theta = 25$ calls.[1]

To compute w_a using Equation 6.1, action a must have been called at least $2\theta = 50$ times. For actions where an accurate estimate of improvement cannot be calculated because they have been called fewer than 50 times, we want QLAP to choose actions that are not too reliable nor unreliable, yet likely to succeed in the current state s. To do this, QLAP chooses a plan o as described in Section 6.1.2 for action a and then computes w_a as

$$w_a = H(rel(a)) \cdot rel(o, s) \tag{6.2}$$

Recall that H is a measure of entropy. Entropy will be highest when the reliability of action a is 0.50.

QLAP chooses actions to practice so that it can get better at performing them and so that it can see the effects of the actions. Actions for direction of change variables sometimes need to

[1] IAC actually has two parameters: a smoothing parameter θ, and a time window parameter τ. QLAP sets $\theta = \tau$ for simplicity.

be maintained for a short while so that their effects can be observed. To do this, when the agent chooses a direction of change action to practice and that action is completed, that agent continues to give the last motor value for k timesteps.

6.1.2 Choosing the Best Plan to Perform an Action

When an action is called, it chooses a plan to perform the action. QLAP seeks to choose the plan that is most likely to be successful in the current state. To compare plans, QLAP computes a weight w_o^s for each plan o in state s. To compute the weight w_o^s for plan o in state s, QLAP computes the product of the reliability of the DBN r that led to the plan $rel(r, s)$ and the reliability of the second-order DBN $rel(o, s)$ so that

$$w_o^s = rel(r, s) \cdot rel(o, s) \tag{6.3}$$

To choose the plan to perform the action, QLAP uses ϵ-greedy action plan selection ($\epsilon = 0.05$). With probability $1 - \epsilon$, QLAP chooses the plan with the highest weight. And with probability ϵ it chooses a plan randomly. To prevent loops in the calling list, a plan whose DBN has its antecedent event already in the call list is not applicable and cannot be chosen.

6.1.3 Choosing an Action within a Plan

Recall from Chapter 5 that QLAP learns a Q-table for each plan that gives a value for taking each action a in state s. Here again, QLAP uses ϵ-greedy selection. With probability $1 - \epsilon$, in state s, QLAP chooses action a that maximizes $Q_i(s, a)$, and with probability ϵ, QLAP chooses a random action. This action selection method balances exploration with exploitation [Sutton and Barto, 1998].

6.2 Developmental Restrictions

The agent learns autonomously by constructing representations in the form of landmarks, DBNs, and plans.

adding landmarks leads to new events, new DBNs, and new actions.

adding DBNs leads to new plans and new landmarks.

adding plans leads to new policies and Q-tables

When each of these representations is added it leads to resource usage and to the possibility of new representations being added. There is a danger that the new representations will overwhelm the resources of the agent. This section describes restrictions that QLAP uses to keep resource usage manageable.

6.2.1 When an Action becomes Sufficiently Reliable

When an agent is able to do an action sufficiently well, QLAP is able to free up resources and stop allocating new resources to learn it. An action $a(v, q)$ is *sufficiently reliable* if at any time $rel(a) > \theta_{SR} = 0.75$. When an action $a(v, q)$ is sufficiently reliable, QLAP saves resources by

1. Not learning new event landmarks on event $v{\rightarrow}q$ (Chapter 4, Section 4.2).

Algorithm 2 The Qualitative Learner of Action and Perception (QLAP)

```
1: for t = 0 : ∞ do
2:    sense environment
3:    convert input to qualitative values using current landmarks
4:    update statistics for learning new contingencies
5:    update statistics for each DBN
6:    if mod(t, 2000) == 0 then
7:       learn new DBNs
8:       update contexts on existing DBNs
9:       delete unneeded DBNs and plans
10:      if mod(t, 4000) == 0 then
11:         learn new landmarks on events
12:      else
13:         learn new landmarks on DBNs
14:      end if
15:      convert DBNs to plans
16:   end if
17:   if current exploration action is completed then
18:      choose new exploration action and action plan
19:   end if
20:   get low-level motor command based on plan of current exploration action
21:   pass motor command to robot
22: end for
```

2. Not learning additional plans for action $a(v, q)$ (Chapter 5, Section 2).
3. Not learning new DBNs with the consequent event $v{\rightarrow}q$ (Chapter 4, Section 2).
4. Deleting DBNs with the consequent event $v{\rightarrow}q$ that are not currently plans for action $a(v, q)$.

6.2.2 Limiting the Number of Plans

The plans that QLAP creates are relatively small because they only include the variables necessary to carry out the plan. But with MDP planning, even small plans consume resources, so QLAP limits their number. Each action may have at most three plans. If each of these plans has been called fewer than 30 times or is sufficiently reliable, then no plans will be replaced. But, if this is not the case, then the plan with the lowest reliability $rel(o)$ that has been called more than 30 times can be replaced by a new plan if there is one to be added to the action. Also, any plan whose overall reliability $rel(o)$ falls below 0.05 is removed.

When to Convert a Change DBN to a Plan

As described in Chapter 5, Section 2, a DBN r must be sufficiently reliable to be converted to a plan. In addition to this, QLAP adds two other criteria:

1. The agent must be able to achieve the antecedent event of DBN r with sufficient reliability. This is true if there is a sufficiently reliable plan to achieve the antecedent event of DBN r. This saves resources because if the agent cannot reliably achieve the antecedent event, then the plan cannot reliably be executed.

2. It must not create a cycle in the action graph. The action graph is constructed by creating a vertex for each qualitative value. Then for each DBN that has been made into a plan, a directed edge is added to the vertex that matches the antecedent event coming from the vertex that matches the consequent event. If there is a cycle when the edge for the proposed DBN is added, then it is not made into a plan. This helps to simplify paths within actions and plans.

6.2.3 Limiting when Change DBNs are Added

As discussed in Section 6.2.1, a DBN cannot be learned if the action to bring about the consequent event is sufficiently reliable. Additionally, the learning of DBNs also follows a developmental progression starting with the motor variables as antecedent events. This progression occurs because A DBN r' can only be learned if there exists a sufficiently deterministic DBN r that predicts the antecedent event of r'. (If the antecedent event for r' is a magnitude variable v, then there must exist a sufficiently deterministic DBN that can increase or decrease v.) Finally, if the DBN does not become sufficiently deterministic or a plan after 100,000 timesteps, it is deleted.

6.3 Targeted Learning

Since QLAP creates an action for each variable and qualitative value combination, a QLAP agent is faced with many potential actions that could be learned. QLAP can choose different actions to practice based on the learning gradient, but what about the thresholds to learn predictive DBN models and plans? Some actions might be more difficult to learn than others, so it seems reasonable that the requirements for learning representations that lead to learning such actions should be loosened.

QLAP does targeted learning for difficult actions. To learn a plan for an action chosen for targeted learning, QLAP

1. **Lowers the threshold needed to learn a contingency.** Recall from Chapter 4, Section 1.2, that a contingency is learned when

$$Pr(soon(t, E_2)|E_1(t)) - Pr(soon(t, E_2)) > \theta_{pen} = 0.05 \tag{6.4}$$

   If event E_2 is chosen for targeted learning, QLAP makes it more likely that a contingency will by learned by setting $\theta_{pen} = 0.02$.
2. **Lowers the threshold needed to learn a plan.** Recall from Chapter 5, Section 2 that one of the requirements to convert a change DBN r into a plan is that

$$brel(r) > \theta_{SR} = 0.75 \tag{6.5}$$

   If event E_2 is chosen for targeted learning, QLAP makes it more likely that a DBN will be converted to a plan by setting $\theta_{SR} = 0.25$.

This leaves the question of when to use targeted learning of actions. An event is chosen as a goal for targeted learning if the probability of being in a state where the event is satisfied is less than 0.05; we call such an event *sufficiently rare*. This is reminiscent of Bonarini *et al.* [2006]. They consider desirable states to be those that are rarely reached or are easily left once reached.

6.4 Self

One step towards tool use is making objects in the environment part of "self" so that they can be used to perform useful tasks. The representation of "self" is straightforward in QLAP. A change variable is part of "self" if it can be quickly and reliably manipulated. QLAP learns what is part of "self" by looking for variables that it can reliably control with low latency.

Marjanovic [1996] enabled a robot to identify what was part of "self" by having the robot wave its arm and having the robot assume that the only thing moving in the scene was itself. The work of Metta and Fitzpatrick [2003] is similar but more sophisticated because it looks for optical flow that correlates with motor commands of the arm. Gold and Scassellati [2006] note the time between giving a motor command and seeing movement to denote self. Our method for learning self is similar to that of Gold and Scassellati, but we learn what is part of self while learning actions.

A direction of change variable $\dot{v}$ is part of self if:

1. the average time it takes for the action to set $\dot{v} = [+]$ and $\dot{v} = [-]$ is less than k, and
2. the actions for both $\dot{v} = [+]$ and $\dot{v} = [-]$ are sufficiently reliable.

6.5 Conclusion

QLAP is not given a goal, but instead explores the environment autonomously. It uses Intelligent Adaptive Curiosity (IAC) to determine what aspects of the environment should be explored. In Chapter 7, we will see a comparison between using IAC action selection and choosing actions to practice randomly.

QLAP also uses various methods to prune the number of representations learned. In Chapter 7, we will see that these methods free up resources without significantly hindering learning. Additionally, QLAP performs targeted learning to bring about rare events. We will also see in Chapter 7 that this helps QLAP to perform the difficult task of picking up the block.

Chapter 7

Evaluation

The central claim is that QLAP enables an agent in a continuous environment to autonomously learn useful abstract state representations and effective higher-level actions. QLAP learns autonomously as part of its developmental progression. Evaluating autonomous learning is difficult because there is no pre-set task on which to evaluate performance. The approach taken in this thesis to evaluate QLAP is to have the agent learn autonomously in an environment, and then to see if the agent is able to perform a set of tasks. It is important to note that during learning the agent does not know on which tasks it will be evaluated.

The evaluations are performed in a simulated environment that uses real physics. In this environment, the robot is sitting at a table that contains one or two blocks. The robot explores autonomously with one arm, and through exploration it learns to perform a set of QLAP actions. Some of the learned actions we evaluate QLAP on include hitting a block in a specified direction, hitting a block off the table, picking up a block with a magnetic hand, and using one block to hit another block.

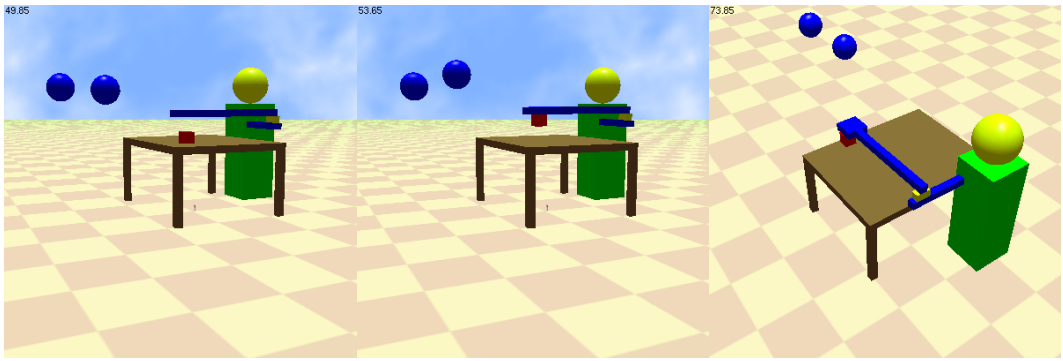
We evaluate the quality of the learned landmarks by using the landmarks to discretize the environment for reinforcement learning. We also perform various ablation studies to test QLAP under different conditions. Additionally, QLAP is evaluated in an environment based on the video game Pong. The agent autonomously learns the dynamics of the game, and then it is evaluated on how well it can play.

The results indicate that QLAP learning autonomously was able to do as well or better than a supervised learner on various tasks. Additionally, the agent was able to learn to use one block to hit another block if they are aligned. Also, the landmarks that QLAP learns are broadly useful because the agent was able to do reinforcement learning better using the QLAP landmarks than with random landmarks.

7.1 Core Evaluation Environment

QLAP is evaluated in multiple environments. This section presents the *core environment*, which is used for many of the experiments. Other environments are used, as well as modifications to the core environment. Those will be discussed where they are relevant.

The core environment is implemented in Breve [Klein, 2003] and has realistic physics. Breve simulates physics using the Open Dynamics Engine (ODE) [Smith, 2004]. The simulation consists of a robot at a table with one or more blocks and floating objects. The robot has an orthogonal arm that can move in the x, y, and z directions. The core environment is shown in Figure 7.1, and the variables perceived by the agent for the core environment are shown in Table 7.1. The block has a width that varies between 1 and 3 units. The block is replaced when it is out of reach and



(a) Not grasping (b) Grasping (c) Above view

Figure 7.1: The Core Environment (shown here with floating objects)

not moving, or when it hits the floor.[1]

The robot can grasp the object in a way that is reminiscent of both the palmer reflex [Payne and Isaacs, 2007] and having a sticky mitten [Needham *et al.*, 2002]. The palmer reflex is a reflex that is present from birth until the age 4-6 months in human babies. The reflex causes the baby to close its hand when something touches the palm [Payne and Isaacs, 2007]. In the sticky mittens experiments [Needham *et al.*, 2002], three-month-old infants wore mittens covered with Velcro that allowed them to more easily grasp objects.

Grasping is implemented on the robot to allow it to grasp only when over the block. When the hand is placed directly on top of the block, the block is grasped. Specifically, the block is grasped if the hand and block are colliding, and the Euclidean 2D distance in the x and y directions is less than half the width of the palm, $3/2 = 1.5$ units. For every timestep that the block is grasped, it is let go with probability 0.1. And to keep the robot from repeatedly picking it up and dropping it in the same place, when the block is let go, the block is moved with probability 0.5.

7.2 Experimental Setup

During the experiments, the agent first explores autonomously for 250,000 timesteps (about 3.5 hours of physical experience) as described in Chapter 6. During this exploration, the state of the agent is saved every 10,000 timesteps (about every 8 minutes of physical experience). The agent is then evaluated on how well it can do the learned task using the representations from each stored state. The next two sections explain the task setup and the specific goals to be achieved by performing the task.

7.2.1 Task Setup

For each task, the agent will have to achieve a goal. A goal is a qualitative value of some variable. At the beginning of each trial, a block is placed in a random location within reach of the agent and the hand is moved to a random location. Then, the goal is given to the agent. The agent makes and executes plans to achieve the goal. If the agent cannot make a plan to achieve the goal, it moves randomly. The trial is terminated after 300 timesteps or if the goal is achieved. The agent

[1]The physics simulator sometimes exhibits odd behavior. So the block is also reset if it out of reach for 100 timesteps (it occasionally flies off when hit just right) or if the robot torso moves. If the block flies up above the robot then the learning or evaluation is began again at the last saved checkpoint.

receives a penalty of -0.01 for each timestep it does not achieve the goal and a reward of 9.99 on the timestep it achieves the goal. Each evaluation consists of 100 trials. The rewards over the 100 trials are averaged, and the average reward is taken as a measure of ability.

7.2.2 Goals of the Core Environment

There are three goals in the core environment on which the agent's performance is evaluated. These are referred to as the *core tasks*.

move the block The evaluator picks a goal to move the block *left* ($\dot{T}_L = [+]$), *right* ($\dot{T}_R = [-]$), or *forward* ($\dot{T}_T = [+]$). The goal is chosen randomly based on the relative position of the hand and the block.[2] A trial is terminated early if the agent hits the block in the wrong direction.

hit the block to the floor The goal is to make *bang* = true.

pick up the block The goal is to get the hand in just the right place so the robot can grasp the the block and make T = true. A trial is terminated early if the agent hits the block out of reach.

7.3 Tests for Statistical Significance

The statistics for the evaluation are done using repeated measures analysis of variance (ANOVA) [Harris, 1995]. This is the method for statistical hypothesis testing typically used in longitudinal studies. The main test of ANOVA is the F test. The higher the value of F, the more likely it is that there is a significant result. The statistics for this chapter were calculated using the statistics package SPSS. Often, when many experiments of the same type are performed, it is standard practice to modify the test for statistical significance to make it less likely that statistically significant findings will result from simply having performed many experiments. Since, in this thesis, each of the experiments seeks to address a different question, no such adjustments were made to the significance values.

Often in the experiments in this chapter, a bar graph is displayed showing the average performance over time. The first 100,000 timesteps are excluded in these bar graphs, because doing so better highlights the differences in performance. The statistics were also calculated excluding the first 100,000 timesteps. In the line graphs, all error bars are standard error. Twenty agents were evaluated for each experimental condition (except for Pong).

7.4 Compare Undirected QLAP with Supervised Learning

Claim: QLAP, doing undirected exploration and creating high-level actions on low-level foundations, performs as well or better than a supervised learning method with a performance goal pre-specified.

We evaluate this claim by comparing the performance of QLAP to the performance of reinforcement learning using tile coding on the core tasks. The tile coding learner 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.

[2] If the block is very close to the robot, then the goal is to hit the block to the left if the block is on the left side of the arm, and right otherwise. Otherwise, if the block is on the extreme left, then the goal is chosen randomly to hit the block left or forward (right is analogous). Otherwise, the goal is chosen randomly to hit the block left, right, or forward.

Tile coding is a way to discretize continuous input for reinforcement learning. The tile coder was trained using linear, gradient-descent Sarsa(λ) with binary features [Sutton and Barto, 1998] where the binary features came from tile coding. See Appendix H for implementation details. For each experiment, 20 QLAP agents and 20 tile-coding agents were trained. The QLAP agents autonomously explored the environment, and the tile-coding agents continually repeated the specified core task.

7.4.1 Experimental Environment

In addition to the core environment, QLAP is also evaluated with distractor objects. This is done using the *floating extension environment*, which adds two floating objects that the agent can observe but cannot interact with. The purpose of this environment is to see if the robot can learn in the presence of distractor objects. The objects float around in an invisible box. The variables added to the core environment to make the floating extension environment are shown in the following table:

Variable	Type	Meaning
f_x^1, f_y^1,f_z^1	magnitude	location of first floating object in x, y, and z directions
$\dot{f}_x^1$,$\dot{f}_y^1$,$\dot{f}_z^1$	change	derivative of f_x^1, f_y^1,f_z^1
f_x^2, f_y^2,f_z^2	magnitude	location of second floating object in x, y, and z directions
$\dot{f}_x^2$,$\dot{f}_y^2$,$\dot{f}_z^2$	change	derivative of f_x^2, f_y^2,f_z^2

7.4.2 Experimental Conditions

Both the QLAP and the tile coding agents are evaluated on the core tasks in both the core environment and the floating extension environment. There are three experimental conditions.

QLAP The QLAP algorithm.

Tile-1 Tile coding choosing an action every time step.

Tile-10 Tile coding choosing an action every 10 time steps.

Tile-1 and Tile-10 are both used because Tile-1 had difficulty learning the core tasks due to high task diameter. Tile-10 gets a penalty of -0.1 every 10th timestep it does not reach the goal and a reward of 9.99 on the timestep it reaches the goal. QLAP learned autonomously for 250,000 timesteps as described in Section 7.2. The tile coding agents repeatedly performed trials of a particular core task for 250,000 timesteps. At the beginning of each trial, the core task that the tile coding agent would practice was chosen randomly.

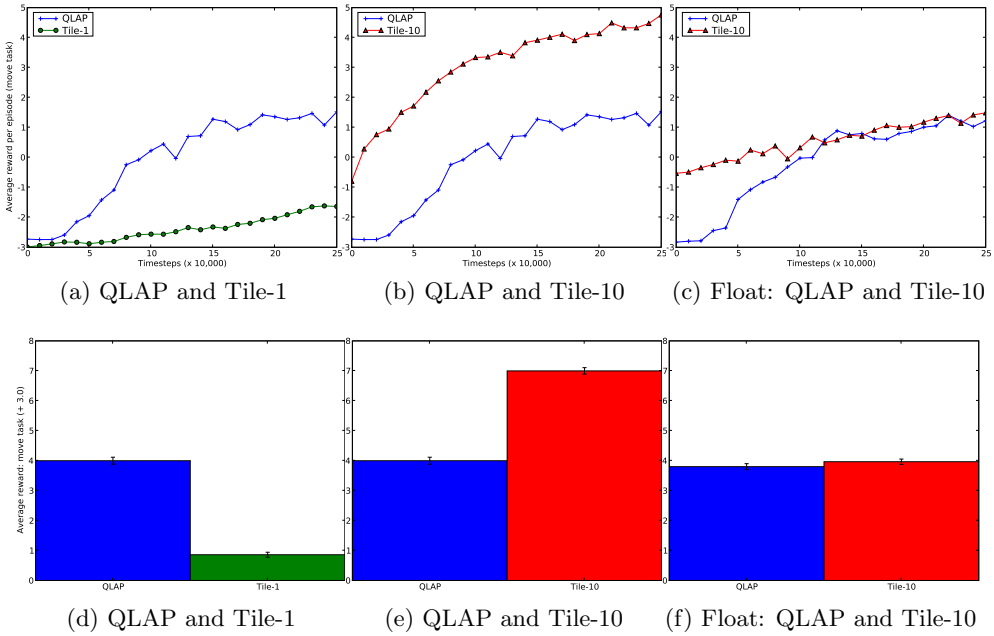
7.4.3 Results

The results are shown in Figures 7.2, 7.3, and 7.4. As can be seen in Figures 7.2(a), 7.3(a), and 7.4(a), Tile-1 was not able to do any of the tasks well compared to QLAP due to the high task diameter (the number of timesteps needed to complete the task).

QLAP does better relative to tile coding as the task gets more difficult, as can be seen in Figures 7.2(b), 7.3(b), and 7.4(b). On the easiest task of hitting the block, Tile-10 does better. They do almost equally well on the task of making the block hit the floor, and QLAP does better on the task of picking up the block.

As can be seen in Figures 7.2(c), 7.3(c), and 7.4(c), the performance of tile coding degrades when the distractor objects are added, but the performance of QLAP does not degrade, or degrades only slightly.

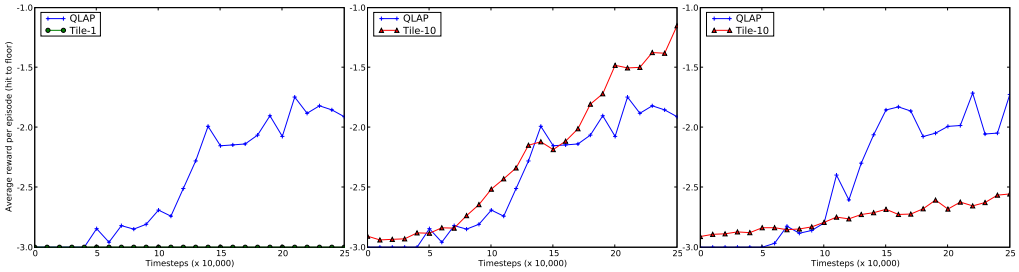
The sub-figures (d), (e), and (f) of Figures 7.2, 7.3, and 7.4 show the average value of the performance after the first 100,000 timesteps.



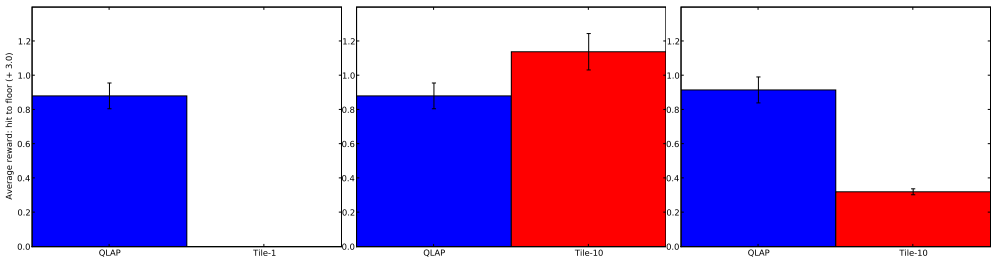
(a) QLAP and Tile-1 (b) QLAP and Tile-10 (c) Float: QLAP and Tile-10

(d) QLAP and Tile-1 (e) QLAP and Tile-10 (f) Float: QLAP and Tile-10

Figure 7.2: Moving the block. (a) QLAP does better than Tile-1 because of the high task diameter. (b) Tile-10 does better than QLAP. (c) When the floating objects are added, the performance of Tile-10 drops off but that of QLAP does not. (d) $F = 71.740$, $sig. = 0.000$. (e) $F = 94.142$, $sig. = 0.000$. (f) $F = 2.358$, $sig. = 0.133$. (Bar graphs show average value after 100,000 timesteps.)

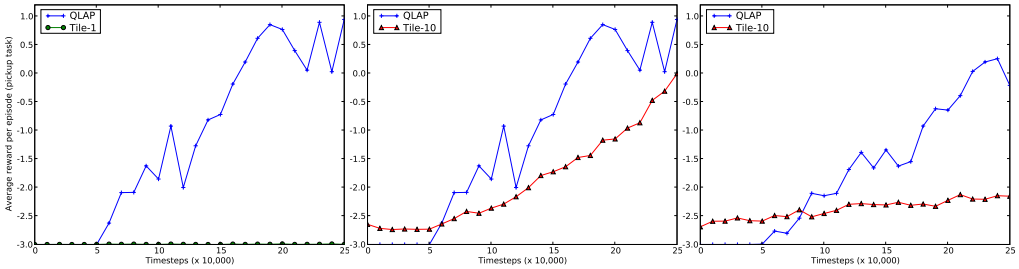


(a) QLAP and Tile-1 (b) QLAP and Tile-10 (c) Float: QLAP and Tile-10

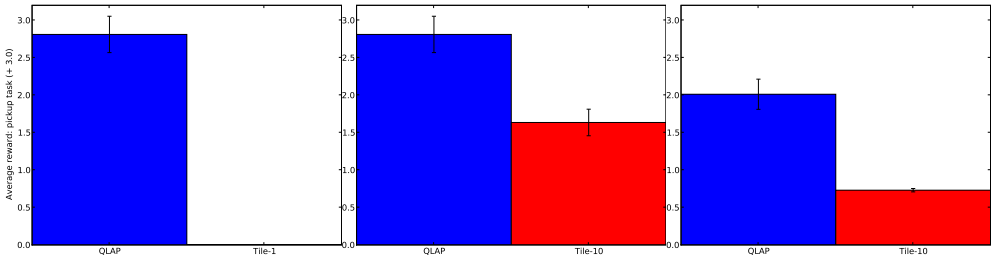


(d) QLAP and Tile-1 (e) QLAP and Tile-10 (f) Float: QLAP and Tile-10

Figure 7.3: Hit the block to the floor. (a) Tile-1 is not able to hit the block to the floor. (b) Tile-10 does the task well. (c) Tile coding has trouble with the floating objects. The performance of QLAP does not degrade. (d) $F = 49.703$, $sig. = 0.000$. (e) $F = 0.583$, $sig. = 0.450$. (f) $F = 19.779$, $sig. = 0.000$.



(a) QLAP and Tile-1 (b) QLAP and Tile-10 (c) Float: QLAP and Tile-10



(d) QLAP and Tile-1 (e) QLAP and Tile-10 (f) Float: QLAP and Tile-10

Figure 7.4: Picking up the block. (a) Tile-1 is not able to pick up the block. (b) Tile-10 does the task well. (c) Tile coding has trouble with the floating objects. The performance of QLAP does degrade some, but its end performance degrades only slightly. (d) $F = 18.403$, $sig. = 0.000$. (e) $F = 3.144$, $sig. = 0.084$. (f) $F = 10.531$, $sig. = 0.002$.

Table 7.1: Variables of the Core Environment

Variable	Type	Meaning
u_x	motor	force in x direction
u_y	motor	force in y direction
u_z	motor	force in the z direction
u_{UG}	motor	ungrasp the block
h_x	magnitude	global location of hand in x direction
$\dot{h}_x$	change	derivative of h_x
h_y	magnitude	global location of hand in y direction
$\dot{h}_y$	change	derivative of h_y
h_z	magnitude	global location of hand in z direction
$\dot{h}_z$	change	derivative of h_z
y_{TB}	magnitude	top of hand in frame of reference of bottom of block (y direction)
$\dot{y}_{TB}$	change	derivative of y_{TB}
y_{BT}	magnitude	bottom of hand in frame of reference of top of block (y direction)
$\dot{y}_{BT}$	change	derivative of y_{BT}
x_{RL}	magnitude	right side of hand in frame of reference of left side of block (x direction)
$\dot{x}_{RL}$	change	derivative of x_{RL}
x_{LR}	magnitude	left side of hand in frame of reference of right side of block (x direction)
$\dot{x}_{LR}$	change	derivative of x_{LR}
z_{BT}	magnitude	bottom side of hand in frame of reference of top of block (z direction)
$\dot{z}_{BT}$	change	derivative of z_{BT}
z_F	magnitude	distance to the floor
$\dot{z}_F$	change	derivative of z_F
T_L	magnitude	location of nearest edge of block in x direction in coordinate frame defined by left edge of table
$\dot{T}_L$	change	derivative of T_L
T_R	magnitude	location of nearest edge of block in x direction in coordinate frame defined by right edge of table
$\dot{T}_R$	change	derivative of T_R
T_T	magnitude	location of nearest edge of block in y direction in coordinate frame defined by top edge of table
$\dot{T}_T$	change	derivative of T_T
c_x	magnitude	location of hand in x direction relative to center of block
$\dot{c}_x$	change	derivative of c_x
c_y	magnitude	location of hand in y direction relative to center of block
$\dot{c}_y$	change	derivative of c_y
T	nominal	block is grasped, true or false. Becomes true when the hand is touching the block and the 2D distance between the center of the hand and the center of the block is less than 1.5.
bang	nominal	true when block hits the floor

7.5 QLAP enables transfer learning

Claim: QLAP can do transfer learning by learning actions that can later be treated as primitive actions to build yet higher-level actions. To evaluate this claim, the agent trains with some variables missing that are needed to make the block hit the floor in the core environment. Then, once these variables are added, the agent can use what it learned previously to learn faster than starting from scratch.

7.5.1 Experimental Environment

The experiment is performed in the core environment on the task of hitting the block to the floor.

7.5.2 Experimental Conditions

There are two experimental conditions. For each experimental condition, 20 agents were trained (20 agents were trained for each experimental condition in this chapter unless stated otherwise).

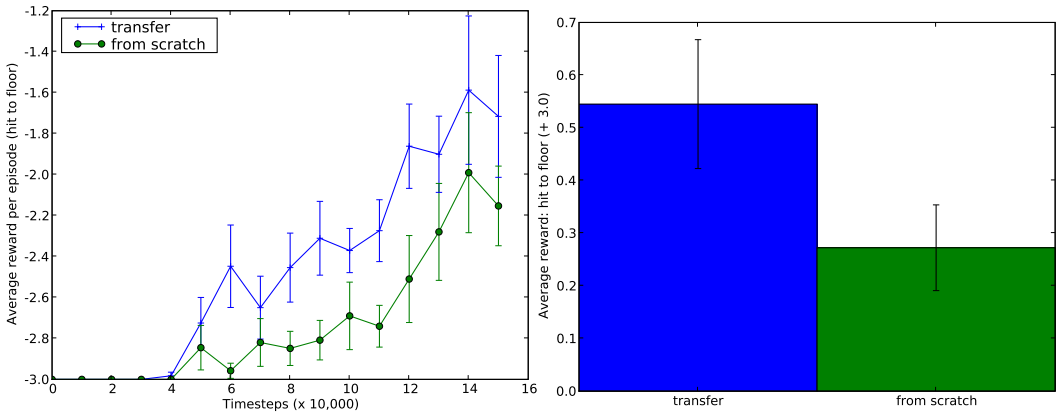
1. **transfer** The agent learns for 150,000 timesteps without the variables relevant to the task (*bang* and z_f in Table 7.1). It then learns for another 150,000 timesteps using all of the variables.

2. **from scratch** The agent explores for 150,000 timesteps and has access to all of the variables.

We call the conditions **transfer** and **from scratch** because the **transfer** agent should use what it learned during the first 150,000 timesteps to more quickly learn to perform the evaluation task compared with the **from scratch** agent that must begin learning from scratch.

7.5.3 Results

The results are shown in Figure 7.5. We see in Figure 7.5 that the **transfer** agent can reuse the skills learned during exploration with the limited set of variables to learn to perform the task faster. For this experiment, the statistics and bar graph, Figure 7.5(b), were computed over 150,000 timesteps.



(a) Average reward over time

(b) Summed average reward (+3.0)

Figure 7.5: The transfer agent can use the previously acquired knowledge to perform the task better. (b) $F = 6.739$, $sig. = 0.013$ over all 150,000 timesteps.

7.6 QLAP can ignore extraneous variables

Claim: Because of its bottom-up construction of DBNs and plans, QLAP can ignore extraneous variables. We saw in Section 7.4 that the performance of QLAP degrades very little in the presence of distractor objects. In this experiment, we explore that more deeply. The experiments in this section add extra distractor variables and demonstrate that the performance degrades gracefully on the core tasks.

7.6.1 Experimental Environment

There are two kinds of extraneous variables; there are irrelevant variables and relevant variables. *Irrelevant variables* are variables that are not in any meaningful way related to the dynamics of the environment. For example, random values. *Relevant variables* are variables related to the dynamics of the environment and so are more likely to become part of learned contingencies, DBNs, and plans.

Irrelevant Variables

There are three environments with irrelevant variables, each with an increasing number of variables. The first is the floating extension environment discussed previously. The second environment is called the *random-1 extension environment*. The random-1 environment uses all of the variables of the floating extension environment, plus it adds the variables: r_1, $\dot{r}_1$, r_2, $\dot{r}_2$, r_3, and $\dot{r}_3$. The values for the variables r_1, r_2, and r_3 are chosen randomly from a uniform distribution over the range $[0, 1)$. The third is the *random-2 extension environment*, which uses all of the variables of the random-1 extension environment and adds r_4, $\dot{r}_4$, r_5, $\dot{r}_5$, r_6, and $\dot{r}_6$. The values for the variables r_4, r_5, and r_6 are also chosen randomly from a uniform distribution over the range $[0, 1)$.

Relevant Variables

There are two environments with relevant variables; the second environment has more variables than the first. The *relevant-1 extension environment* extends the core environment by adding the variables in the table below.

Variable	Type	Meaning
y_{TT}	magnitude	top of hand in frame of reference of top of block (y direction)
$\dot{y}_{TT}$	change	derivative of y_{TT}
x_{LL}	magnitude	left side of hand in frame of reference of left side of block (x direction)
$\dot{x}_{LL}$	change	derivative of x_{LL}
b_{TY}	magnitude	the top of the block in global frame (y direction)
$\dot{b}_{TY}$	change	derivative of b_{TY}
b_{LX}	magnitude	the left side of the block in global frame (x direction)
$\dot{b}_{LX}$	change	derivative of b_{LX}
h_{TY}	magnitude	the top of the hand in global frame (y direction)
$\dot{h}_{TY}$	change	derivative of h_{TY}
h_{LX}	magnitude	the left side of the hand in global frame (x direction)
$\dot{h}_{LX}$	change	derivative of h_{LX}

The *relevant-2 extension environment* extends the relative-1 extension environment by adding the variables in the table below.

Variable	Type	Meaning
y_{BB}	magnitude	bottom of hand in frame of reference of bottom of block (y direction)
$\dot{y}_{BB}$	change	derivative of y_{BB}
x_{RR}	magnitude	right side of hand in frame of reference of right side of block (x direction)
$\dot{x}_{RR}$	change	derivative of x_{RR}
b_{BY}	magnitude	the bottom of the block in global frame (y direction)
$\dot{b}_{BY}$	change	derivative of b_{BY}
b_{RX}	magnitude	the right side of the block in global frame (x direction)
$\dot{b}_{RX}$	change	derivative of b_{RX}
h_{BY}	magnitude	the bottom of the hand in global frame (y direction)
$\dot{h}_{BY}$	change	derivative of h_{BY}
h_{RX}	magnitude	the right side of the hand in global frame (x direction)
$\dot{h}_{RX}$	change	derivative of h_{RX}

7.6.2 Experimental Conditions

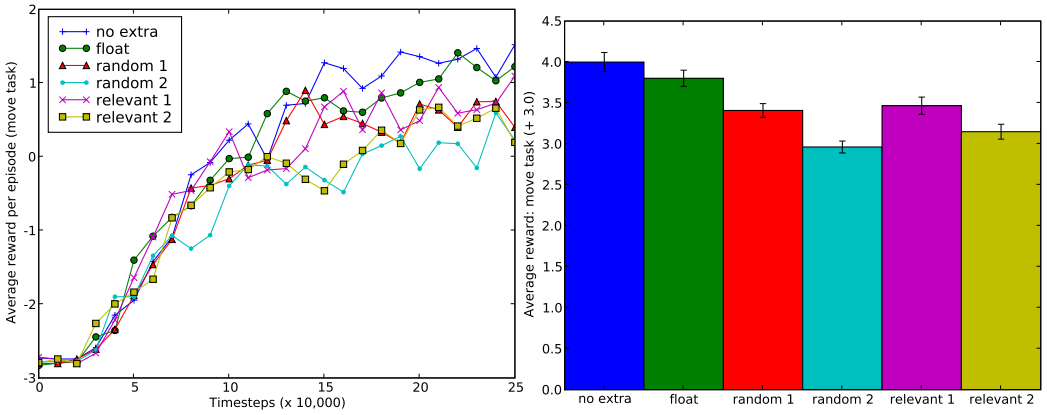
1. **no extra** The core environment
2. **float** The float extension environment

3. **random 1** The random-1 extension environment
4. **random 2** The random-2 extension environment
5. **relevant 1** The relevant-1 extension environment
6. **relevant 2** The relevant-2 extension environment

7.6.3 Results

The results on the core tasks are shown in Figures 7.6, 7.7, 7.8. We see that adding irrelevant variables reduces performance, but not by much. Table 7.2 shows the results of the statistical tests comparing the average performance of each condition with the **no extra** condition. We see that the performance degrades significantly but gracefully for the move task under conditions **random 2** and **relevant 2**

Figure 7.9 shows the number change DBNs learned for each condition. We see, as one would expect, that the relevant variables increase the number of DBNs, but the irrelevant variables do not.

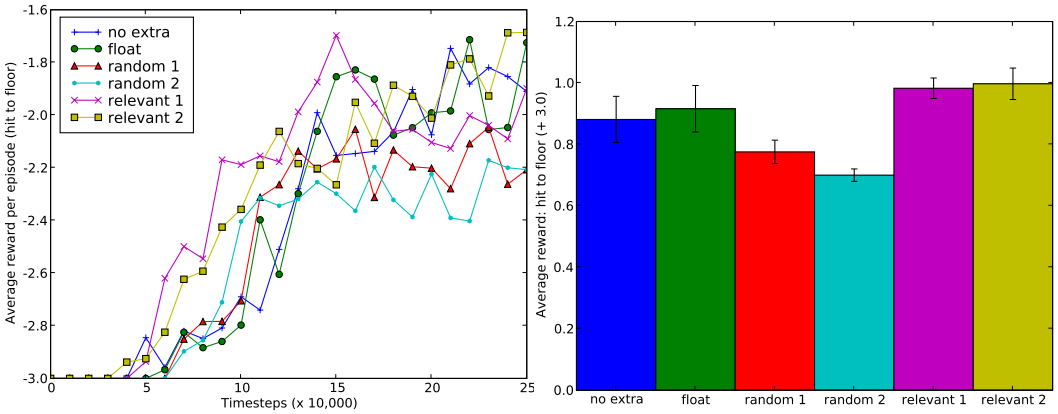


(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.6: Performance seems to go down slightly as new variables are added.

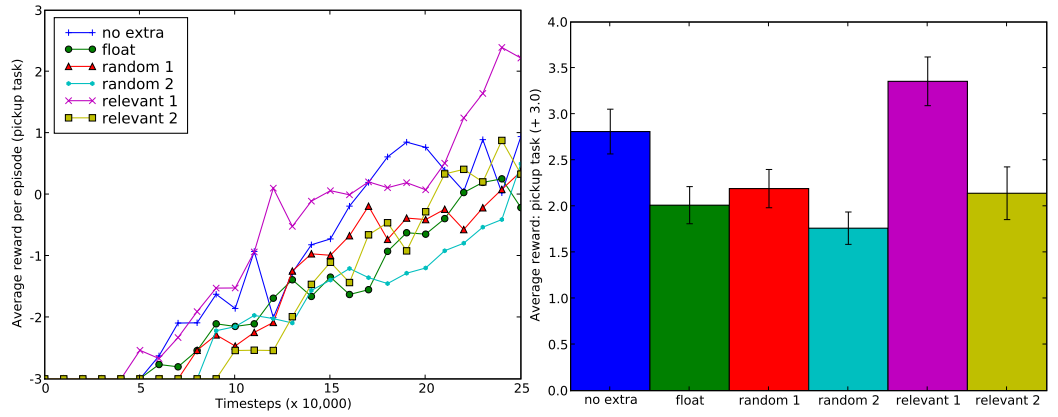
Table 7.2: Tests for Statistical Significance (compared with **no-extra**)

	move task	hit to floor	pickup task
float	$F = 0.303$, $sig. = 0.585$	$F = 0.038$, $sig. = 0.847$	$F = 1.002$, $sig. = 0.323$
random 1	$F = 2.447$, $sig. = 0.126$	$F = 0.465$, $sig. = 0.499$	$F = 0.578$, $sig. = 0.452$
random 2	$F = 7.648$, $sig. = 0.009$	$F = 0.713$, $sig. = 0.404$	$F = 1.862$, $sig. = 0.180$
relevant 1	$F = 1.595$, $sig. = 0.214$	$F = 0.337$, $sig. = 0.565$	$F = 0.382$, $sig. = 0.540$
relevant 2	$F = 5.597$, $sig. = 0.023$	$F = 0.453$, $sig. = 0.505$	$F = 0.730$, $sig. = 0.398$



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.7: Performance seems to go down when irrelevant variables are added.



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.8: Performance seems goes down with irrelevant variables, but appears to improve with some relevant variables. Most likely because the agent has more opportunities (different ways) to learn actions.

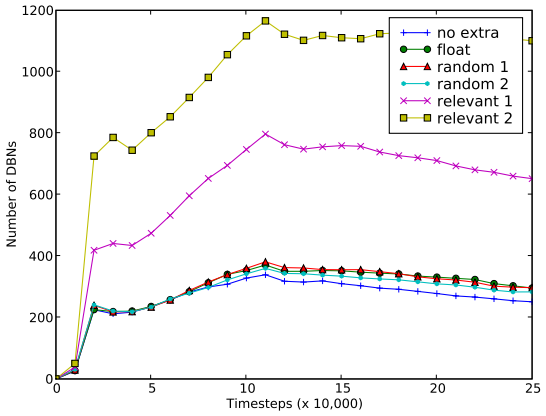


Figure 7.9: Number of change DBNs over time. This graph shows that the number of DBNs does not increase without bound. We see two drops in the number of DBNs. The first drop corresponds to learning to move the hand and those actions becoming sufficiently reliable. The second drop corresponds to contingencies being deleted after 100,000 timesteps because they did not become plans to perform actions.

7.7 QLAP learns landmarks that are generally useful

Claim: QLAP learns landmarks that are generally useful. The purpose of this experiment is to show that the learned landmarks really do represent the "natural joints" in the environment. To evaluate this claim, we will compare the results of tabular Q-learning using landmarks learned with QLAP with the results of tabular Q-learning using randomly generated landmarks. If the landmarks do represent the joints in the environment, then the tabular Q-learner using learned landmarks should do better than the one using random landmarks.

7.7.1 Experimental Environment

Tabular Q-learning does not generalize well. During exploratory experiments, the state space of the core environment was so large that tabular Q-learning rarely visited the same state more than once. We therefore evaluate this claim using a smaller environment and a simple task. We use the *2D core environment* where the hand only moves in two dimensions. It removes the variables of the z direction from the core environment. It subtracts u_z, u_{UG}, h_z, $\dot{h}_z$, z_{BT}, $\dot{z}_{BT}$, c_x, $\dot{c}_x$, c_y, $\dot{c}_y$, T, and *bang*.

7.7.2 Experimental Conditions

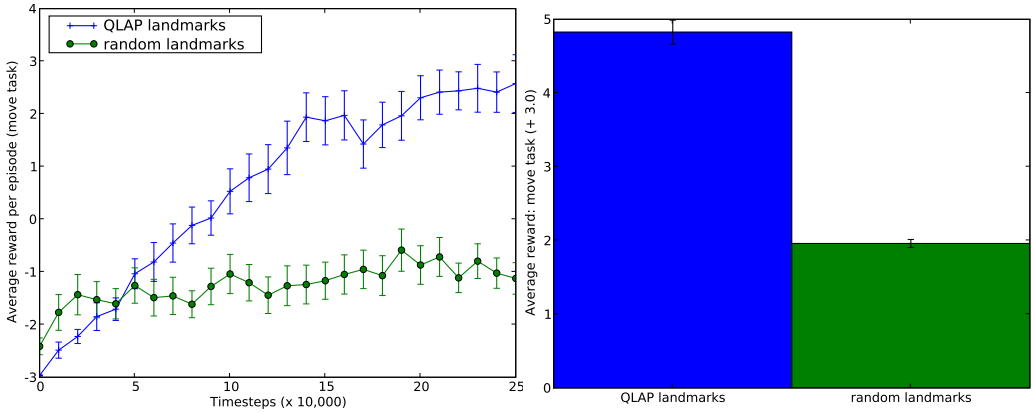
QLAP landmarks Tabular Q-learning using landmarks learned using QLAP after a run of 100,000 timesteps on the 2D core environment.

random landmarks Tabular Q-learning using randomly generated landmarks.

To generate the random landmarks, for each magnitude or motor variable v, a random number of landmarks between 0 and 5 is chosen. Each landmark is then placed in a randomly chosen location within the minimum and maximum range observed for v during a typical run of QLAP. Note that motor variables already have a landmark at 0, so each motor variable had between 1 and 6 landmarks.

7.7.3 Results

The results are shown in Figure 7.10. Tabular Q-learning works much better using the learned landmarks than using the random ones.



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.10: QLAP landmarks enable the agent to learn the task better than do random landmarks. (b) $F = 39.210$, $sig. = 0.000$.

7.8 QLAP: Limitations and Steps towards Tool Use

Currently, QLAP appears to have a limitation in the complexity of actions it can learn. To demonstrate this limitation, we set up an experiment where QLAP uses one block to hit another.

In this experiment there are two blocks. A *primary block* that the agent can interact with directly, and a *secondary block* that the agent cannot reach. The goal is to hit the secondary block with the primary block. This is a step towards tool use because QLAP can use one object to affect another.

The experiments in this section demonstrate that QLAP can learn to do this if the blocks are placed near each other, but it can not learn it if they are separate and QLAP first has to move the primary block to be near the secondary one.

7.8.1 Experimental Environment

This experiment takes place in the *secondary block environment*. The secondary block environment removes z_F, T_L, T_R, T_T, and their derivatives and *bang* from the core environment and it adds variables related to the distances between the blocks. It adds three variables x_{BB}, y_{BB}, and z_{BB}, which is the distance in the x, y, and z directions, respectively, between the two blocks. It also adds a Boolean variable m_{BB} that is true if the blocks are touching.

Variable	Type	Meaning
x_{BB}, y_{BB}, z_{BB}	magnitude	distance in the x, y, and z directions between the two blocks
$\dot{x}_{BB}$, $\dot{y}_{BB}$, $\dot{z}_{BB}$	change	derivative of above
m_{BB}	nominal	true if the blocks are touching

7.8.2 Experimental Conditions

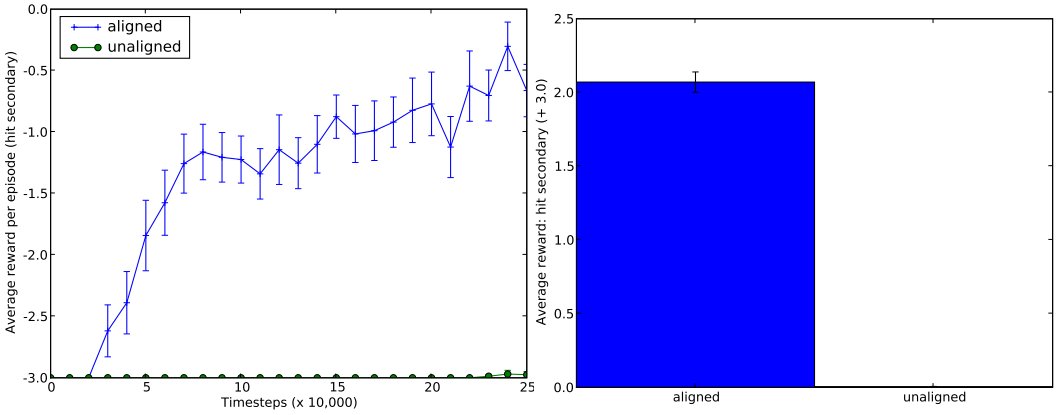
aligned The secondary block is lined up with the primary block.

unaligned The agent has to pick up the primary block and move it so that it is aligned with the secondary block.

In the aligned case, the secondary block is put at the same y location as the primary block and is placed at a random x location between the primary block and the edge of the table. For the unaligned condition, the secondary block is placed randomly on the left or right edge of the table at a random x location and a random y location.

7.8.3 Results

The results are shown in Figure 7.11. QLAP can learn to use the primary block to hit the secondary block when they are aligned, but not when they are not aligned. In the unaligned case, the agent was not able to learn any sufficiently reliable DBNs that led to plans. To learn the DBN, QLAP needs the situation to be set up correctly. But QLAP did not learn to align the blocks when they were unaligned. See Chapter 10 for a further discussion of this phenomenon.



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.11: Two Blocks. QLAP can use one block to hit another when they are aligned, but QLAP cannot learn to align then when they are not aligned.

7.9 QLAP can learn to do unintuitive tasks

Claim: QLAP can learn to perform a task that does not match a human's notion of intuitive physics. The purpose of this experiment is to show that QLAP is not limited to tasks that involve interacting with an object through physical contact.

In this environment, when the hand moves up, one of the floating balls moves up. We can call this ball the *contingent ball*. The goal of this task is for the agent to notice this contingency and then to learn to use it to move the contingent ball up. It is worth noting that this task was introduced after the development of the algorithm, so the QLAP algorithm was not altered to be able to do the task.

7.9.1 Experimental Environment

This experiment uses the floating extension environment with the enhancement that the contingent ball rises when the hand rises. This environment is split into two sub-environments. In the *continuous rising environment*, the contingent ball moves up continuously with the hand. It is worth noting that the force is not very much, if the contingent ball is going down, it does not immediately start to go up. The other environment is the *jump up environment*. When the hand reaches a certain point along the z axis, the contingent ball immediately jumps up. In both environments the contingent ball moves slowly down otherwise when it is not moved up by the hand.

In both environments, the goal is to get the contingent ball to move up ($\dot{f}_z^1 = [+]$). The contingent ball moves randomly in the x and y directions, but gradually falls in the z direction unless the hand moves up. These balls float around in an invisible box. When evaluating, we change how bouncy the ball is because we want to minimize it hitting the bottom of the box and bouncing up without the agent having to do anything. During the evaluation, if the agent does not have a plan, it sets the motor value to 0 instead of a random value. This is because a random value for u_z value might be too likely to affect the results.

7.9.2 Experimental Conditions

continuous rising The red ball moves up continuously with the hand.

jump up When the hand reaches a certain point ($h_z \in (13.75, 14.25)$), the contingent ball immediately jumps up.

7.9.3 Results

The results are shown in Figure 7.12. In both cases the agent learns to do the task. This task ended up being easy for the robot, so to see a learning curve, the step cost was increased to 0.1 per timestep from 0.01 per timestep. The reward is lower in the continuous rising case because the contingent ball starts off falling down and often the force is not enough to get it going up before it hits the bottom (Because it only gets a force of 50 units, this is often not enough to overcome momentum.) The jump up condition is not affected by momentum because the contingent ball jumps up.

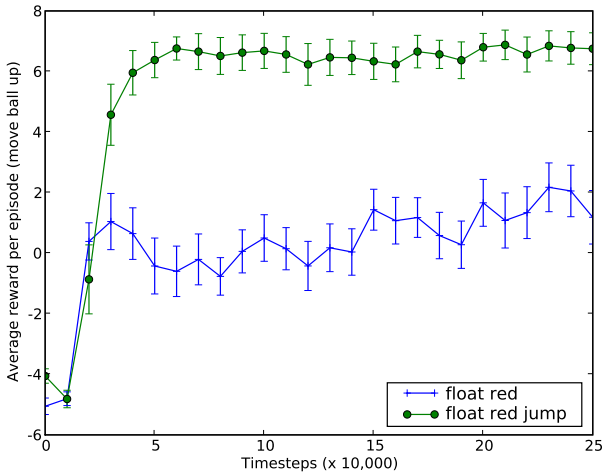


Figure 7.12: QLAP learns to make the contingent ball move up.

7.10 QLAP is not specific to a particular environment

Claim: QLAP is not specific to a particular environment. This section will show that QLAP can learn to hit a ball back and forth in the pong environment. Like with the Breve environments, QLAP explores the environment autonomously. It is not told to learn how to play Pong. Its learning is unsupervised.

7.10.1 Experimental Environment

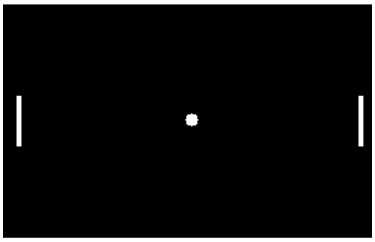


Figure 7.13: The pong environment.

The pong environment [Stober and Kuipers, 2008] is shown in Figure 7.13. The pong environment consists of a left paddle l, a right paddle r, and a ball b. The left paddle and the right paddle move in synchrony. The paddles only move in the y direction making l_x and r_x constants. The discrete motor variable moves the paddles down, steady, or up. The variable c_x^l gives the location of the left paddle in the x direction in the frame of reference of the ball. The variables c_y^l, c_x^r, and c_y^r are analogous. The variables are shown in Table 7.3.

Two changes needed to be made to QLAP for the pong environment. Because there is no latency between commands to move the paddles and the paddle moving (this environment is not built on a simulated physics engine), the window parameter k was not learned dynamically, but rather set to $k = 6$ so that *soon* is a window of 0 to 5 timesteps. (This is the value for k that was used in the Breve environment before the functionality was added to QLAP to set k dynamically.) Additionally, the Pong environment differs in an interesting way from the Breve environment. In this environment, the most interesting object is the ball (although QLAP does not know this), but it cannot be controlled directly. As a consequence, to run in this environment, QLAP was modified so that a plan can be added to QLAP without having to have a reliable plan for the antecedent event (see Chapter 6). No other modifications to QLAP were made.

QLAP learns autonomously in the pong environment, just as it does in the Breve-based environments. During learning, every time the ball goes off the screen it starts in the middle and the paddles start in the middle. During the evaluation, the QLAP agent is told to make the ball go left if it is currently going right, and vice versa. The evaluation metric is how long the agent can keep a volley going. As usual, during learning, QLAP did not know that it eventually would be told to make the ball change direction. To make the evaluation more difficult and to correspond with how the evaluations are done in the other environments where the hand is moved to a random location before each trial, each time a new volley begins, the paddles are moved to a random location.

7.10.2 Results

The results are shown in Figure 7.14. The y-axis is the average number of volleys for the four agents. The results show that the QLAP agent is able to learn to play pong somewhat well by

Table 7.3: Variables of the Pong Environment

Variable	Type	Meaning
u	motor	-1 down, 0 steady, 1 up
l_x	magnitude	left paddle location in x
$\dot{l}_x$	change	derivative of l_x
l_y	magnitude	left paddle location in y
$\dot{l}_y$	change	derivative of l_y
r_x	magnitude	right paddle location in x
$\dot{r}_x$	change	derivative of r_x
r_y	magnitude	right paddle location in y
$\dot{r}_y$	change	derivative of r_y
b_x	magnitude	ball location in x
$\dot{b}_x$	change	derivative of b_x
b_y	magnitude	ball location in y
$\dot{b}_y$	change	derivative of b_y
c_x^l	magnitude	location of left paddle in x direction relative to center of ball
$\dot{c}_x^l$	change	derivative of c_x^l
c_y^l	magnitude	location of left paddle in y direction relative to center of ball
$\dot{c}_y^l$	change	derivative of c_y^l
c_x^r	magnitude	location of right paddle in x direction relative to center of ball
$\dot{c}_x^r$	change	derivative of c_x^r
c_y^r	magnitude	location of right paddle in y direction relative to center of ball
$\dot{c}_y^r$	change	derivative of c_y^r

autonomously learning the dynamics of the environment.

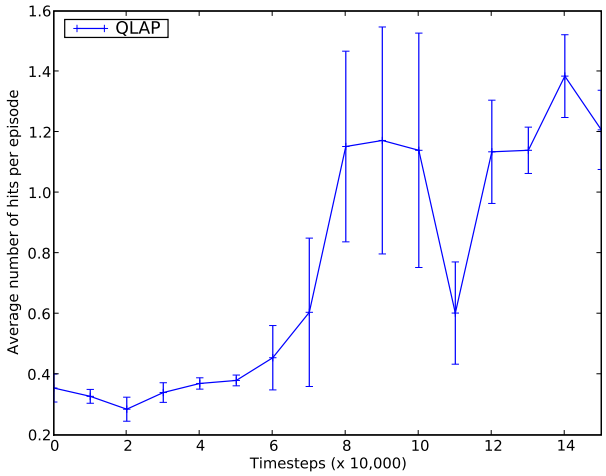


Figure 7.14: QLAP learns to play Pong.

7.11 Ablation Studies

This section discusses the results of exploring various questions using ablation studies. To perform the ablation studies, different parts of the algorithm are altered, and then QLAP is evaluated on the core tasks.

7.11.1 Experimental Conditions

QLAP normal QLAP run

no IAC Does not use IAC as described in Chapter 6. Instead, it chooses actions to practice randomly.

no Dyna Does not use Dyna as described in Chapter 5.

no Sarsa Does not use Sarsa as described in Chapter 5.

entropy Uses only entropy for hill climbing instead of the combination of entropy and best reliability as discussed in Chapter 4.

no targeted Does not do targeted learning, which lowers the thresholds needed to learn contingencies and plans for rare events as described in Chapter 6.

all targeted Does targeted learning for all actions.

NRC No restriction on contingencies for DBNs that the antecedent must be predicted by a sufficiently reliable DBN as described in Chapter 6.

NRP No restrictions on converting DBNs to plans that the antecedent event of the DBN must be able to be achieved by some sufficiently reliable plan as described in Chapter 6.

7.11.2 Results

The results are shown in Figures 7.15, 7.16, 7.17, and in Figures 7.18, 7.19, 7.20, and in Table 7.4. Additionally, Figures 7.21, 7.22, and 7.23 show the cumulative number of exploratory (practice) calls to different sets of actions for various conditions that occurred during autonomous exploration. And Figures 7.24 and 7.25 show the number of change DBNs learned under different conditions.

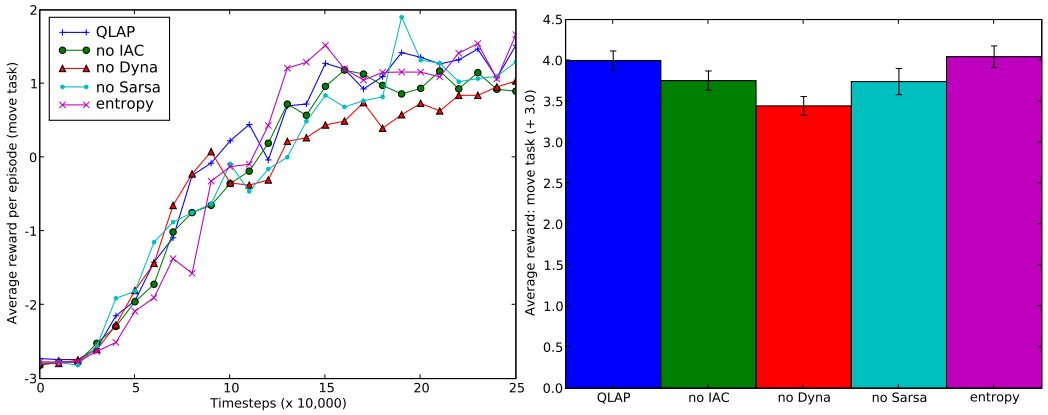
1. *How does the exploration method affect QLAP?* This question was tested using the condition **no IAC** where exploration actions were chosen randomly. The condition **no IAC** did not do as well on the task of picking up the block (although it was not statistically significant). Interestingly, Figure 7.21 shows that the condition **no IAC** makes the most calls to moving the hand. Since moving the hand is easy, the agent appears to be wasting exploration. This wasted exploration may be related to the result that **no IAC** uses more resources by learning more DBNs as shown in Figures 7.24 and 7.25.

2. *How is QLAP affected by developmental restrictions?* This question was tested using the condition **NRC** that limits when contingencies can be learned and the condition **NRP** that limits when plans can be learned. The results showed that adding these restrictions did not hinder performance. And figures 7.24 and 7.25 show that **NRC** uses more resources in the form of DBNs.

3. *How does QLAP do with different aspects ablated?* This question was tested using the condition **no Dyna** that removed the updating of plans using new statistics and the condition **no Sarsa** that removed the updating of plans using experience. When either of these were removed, the agent seemed to lose some ability to pick up the block, but the results were not statistically significant.

4. *Does the agent need to hillclimb on best reliability?* This question was tested using the condition **entropy** that only did hillclimbing on reduction in entropy. Without using hillclimbing on best reliability, the agent was significantly diminished in its ability to pick up the block.

5. *Does the agent need targeted learning?* This question was tested using the condition **no targeted** that did not do targeted learning and the condition **all targeted** that did targeted learning of all actions. The results show that targeted learning is important for learning to pick up the block (statistically significant) but does hurt slightly for moving the block (not statistically significant). We see in Figures 7.22 and 7.23 that **no targeted** makes fewer exploratory calls to the actions to move the hand or pick up the block. And figures 7.24 and 7.25 show that targeted learning affects the number of DBNs learned. So that doing **all targeted** uses more resources in the form of DBNs, but does not improve performance.



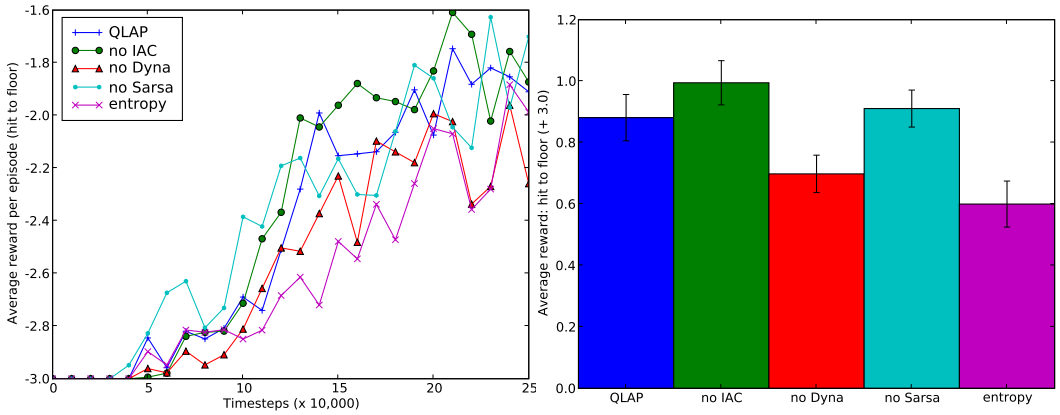
(a) Average reward over time

(b) Summed average reward (+3.0)

Figure 7.15: Moving the block. All conditions appear to do equally well on this easy task.

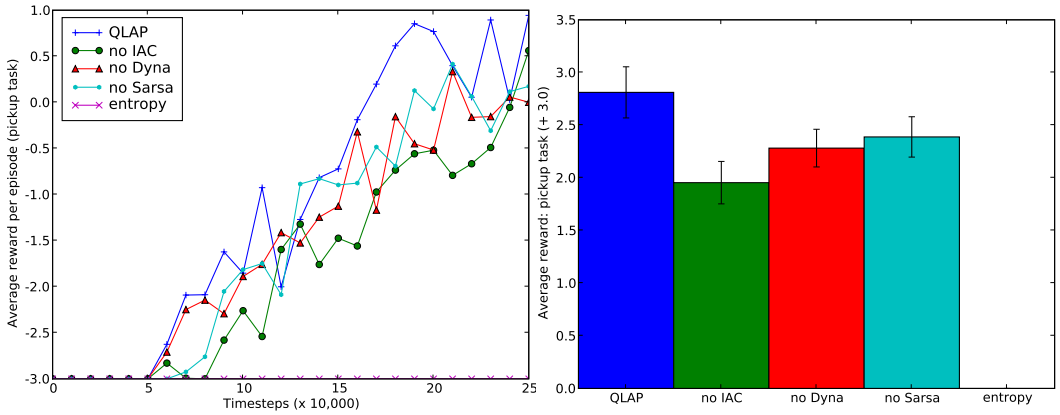
Table 7.4: Tests for Statistical Significance (compared with **QLAP**)

	move task	hit to floor	pickup task
no IAC	$F = 0.251$, $sig. = 0.619$	$F = 0.232$, $sig. = 0.633$	$F = 1.128$, $sig. = 0.295$
no Dyna	$F = 2.355$, $sig. = 0.133$	$F = 1.131$, $sig. = 0.294$	$F = 0.530$, $sig. = 0.471$
no Sarsa	$F = 0.398$, $sig. = 0.532$	$F = 0.022$, $sig. = 0.870$	$F = 0.332$, $sig. = 0.568$
entropy	$F = 0.009$, $sig. = 0.927$	$F = 2.812$, $sig. = 0.102$	$F = 18.455$, $sig. = 0.000$
no targeted	$F = 2.185$, $sig. = 0.148$	$F = 0.427$, $sig. = 0.517$	$F = 8.227$, $sig. = 0.007$
all targeted	$F = 2.183$, $sig. = 0.148$	$F = 0.055$, $sig. = 0.816$	$F = 1.295$, $sig. = 0.262$
NRC	$F = 0.774$, $sig. = 0.384$	$F = 2.178$, $sig. = 0.148$	$F = 0.002$, $sig. = 0.964$
NRP	$F = 1.491$, $sig. = 0.230$	$F = 0.441$, $sig. = 0.511$	$F = 0.149$, $sig. = 0.702$



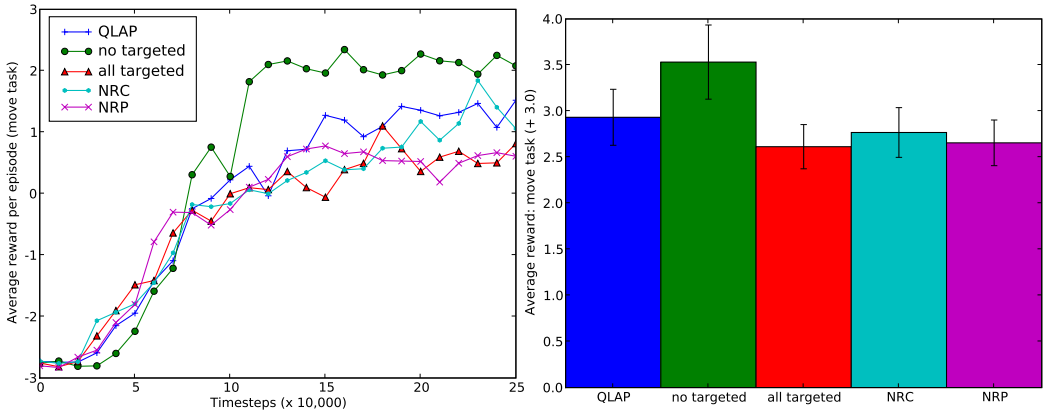
(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.16: Hit the block to the floor. In this experiment, both Dyna and using best reliability appear to be important (although neither effect was statistically significant).



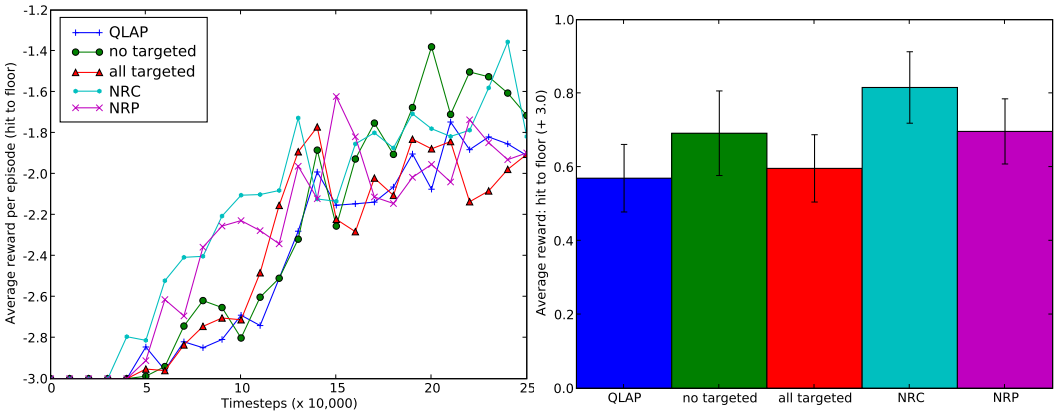
(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.17: Pick up the block. On this more difficult task, we see that not using Intelligent Adaptive Curiosity (IAC) for exploration or using entropy both hurt performance.



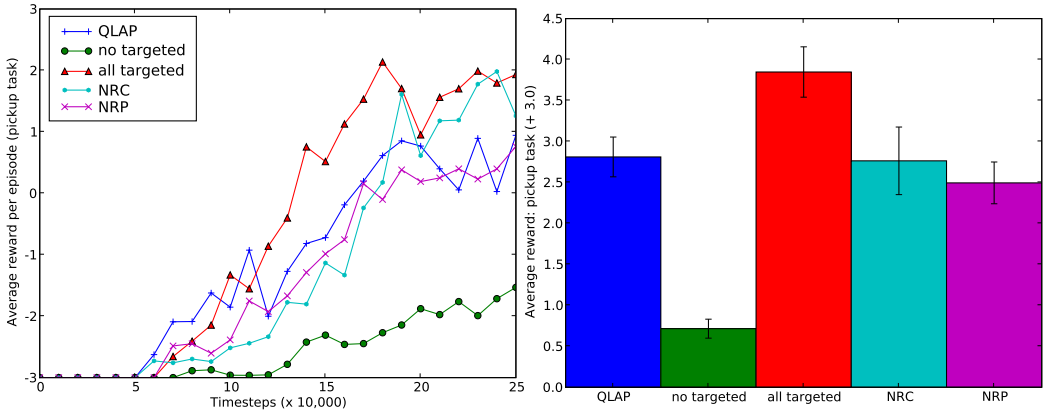
(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.18: Moving the block. It appears to do better when not using targeted learning.



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.19: Hit the block to the floor. All conditions appear to do equally well.



(a) Average reward over time (b) Summed average reward (+3.0)

Figure 7.20: Pick up the block. Targeted learning is particularly important on this difficult task.

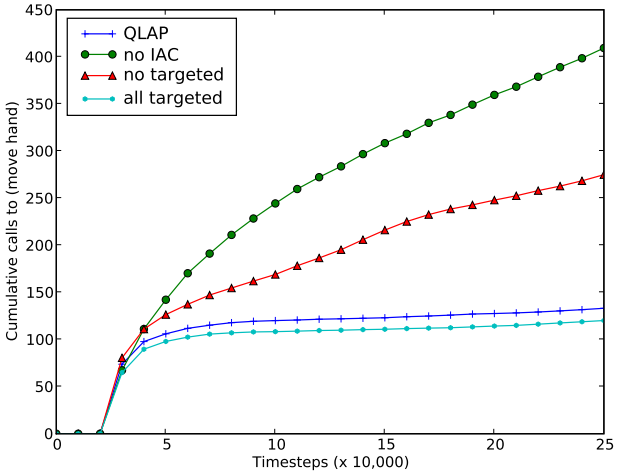


Figure 7.21: Move the hand. Cumulative number of exploratory calls to actions to each of the variables $\dot{h}_x$, $\dot{h}_y$, and $\dot{h}_z$ to positive and negative.

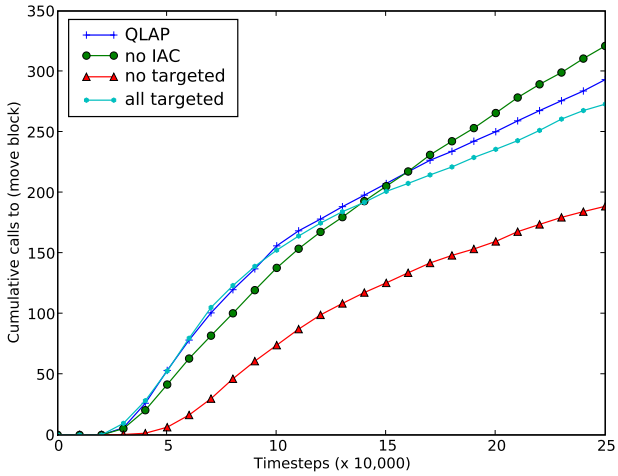


Figure 7.22: Move the block. Cumulative number of exploratory calls to actions to each of the variables $\dot{T}_L$, $\dot{T}_R$, and $\dot{T}_T$ to positive and negative.

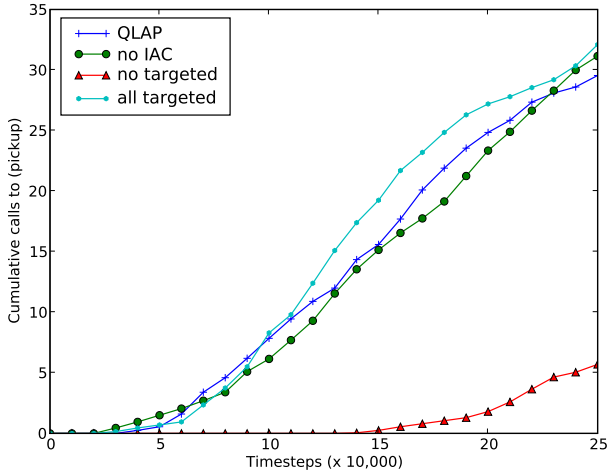


Figure 7.23: Pick up the block. Cumulative number of exploratory calls to the action $a(T, \mathsf{true})$.

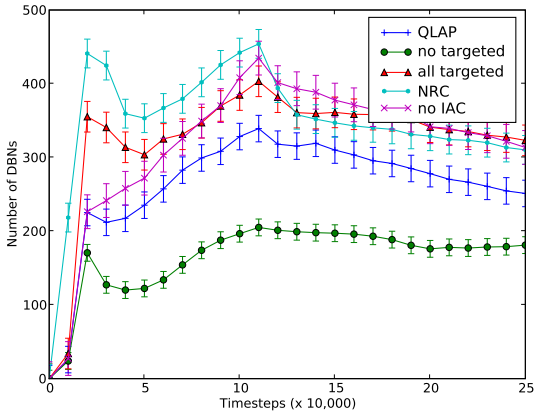


Figure 7.24: Number of change DBNs over time. (Error bars are standard error.)

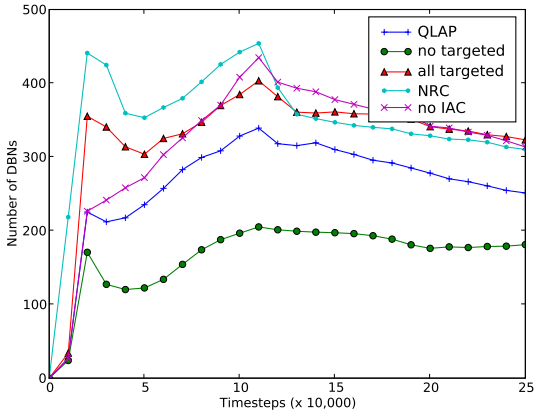


Figure 7.25: Number of change DBNs over time. In this graph, the error bars have been removed to more clearly show the trends.

Chapter 8

Discussion

This chapter discusses different aspects of the QLAP algorithm. First it provides examples of what is learned during a typical run of QLAP on the core environment. Then, it discusses the types of noise experienced by the QLAP agent in the core environment. Following this, there is a discussion of theoretical bounds and thresholds. Finally, there is a discussion about the learned action hierarchy in QLAP.

8.1 Examples of What QLAP Learns

This section shows the landmarks, and some of the DBNs, plans, and actions of a typical QLAP run in the core environment.

8.1.1 Landmarks Learned

Table 8.1 shows the landmarks that were learned. For motor variables u_x and u_y, it takes a force of at least 300 units to move the hand, and QLAP has learned these landmarks. QLAP has also learned the force necessary to move the hand up in the z direction for motor variable u_z. QLAP has learned the limits of movement for the hand on h_x, h_y, and h_z. For the variables y_{TB}, x_{RL}, x_{LR}, and z_{BT}, which give the distances between the edges of the hand and the edges of the block, QLAP has learned that zero is an important landmark because when those variables have a value of zero, the block moves.

For the distance from the floor, z_F, QLAP has learned a landmark at zero. Additionally, for T_L and T_T, the distance between the block and the left and top edges of the table respectively, QLAP learned that zero is an important landmark. As for T_r, the block rarely goes off the table to the right because the robot is using its left hand that is on the left side of the table.[1] And for the variables c_x and c_y, QLAP learns landmarks that help it center the agent's hand over the block.

8.1.2 DBNs, Plans, and Actions Learned

Different agents learn different representations. Some agents do not learn some actions very well. Some agents learn better plans for different actions, especially the more difficult actions such as picking up the block or knocking it off the table. We see in Table 8.2 that the action to move the hand to the left (which is the positive x direction) has one plan that has high reliability. In Table 8.3, we see that the action to move the hand left relative to the block has three plans. The first two plans indicate in their CPT tables for the DBNs (not shown in Table 8.3) that they are

[1] Note that left, right, and top are from the point of view of the robot.

Table 8.1: Learned Landmarks for Core Environment

Variable	Initial Landmarks	Final Landmarks
u_x	$\{[-0.05, 0.05]\}$	$\{[-335.03, -333.53], [-0.05, 0.05], [288.89, 331.27]\}$
u_y	$\{[-0.05, 0.05]\}$	$\{[-310.26, -301.04], [-0.05, 0.05], [300.00, 306.60]\}$
u_z	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05], [570.59, 575.89]\}$
u_{UG}	Boolean	Boolean
h_x	$\{\}$	$\{[-2.54, -2.47], [-1.52, -1.32], [0.62, 0.93], [2.50, 2.55]\}$
$\dot{h}_x$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
h_y	$\{\}$	$\{[-2.01, -1.99], [-1.03, -1.03], [2.93, 3.01]\}$
$\dot{h}_y$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
h_z	$\{\}$	$\{[11.92, 12.00], [12.40, 12.41], [13.76, 13.89]\}$
$\dot{h}_z$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
y_{TB}	$\{\}$	$\{[-2.55, -2.34], [0.05, 0.17], [2.65, 2.75]\}$
$\dot{y}_{TB}$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
y_{BT}	$\{\}$	$\{[-5.57, -5.55], [-4.73, -4.71]\}$
$\dot{y}_{BT}$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
x_{RL}	$\{\}$	$\{[0.00, 0.21]\}$
$\dot{x}_{RL}$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
x_{LR}	$\{\}$	$\{[-0.20, 0.01], [3.60, 3.80]\}$
$\dot{x}_{LR}$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
z_{BT}	$\{\}$	$\{[-0.08, 0.20]\}$
$\dot{z}_{BT}$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
z_F	$\{\}$	$\{[-0.00, 0.45], [10.50, 10.51]\}$
$\dot{z}_F$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
T_L	$\{\}$	$\{[-1.81, -1.45], [-0.09, 0.23]\}$
$\dot{T}_L$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
T_R	$\{\}$	$\{[12.28, 15.12], [20.76, 21.07]\}$
$\dot{T}_R$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
T_T	$\{\}$	$\{[-2.81, -1.64], [-0.01, 0.24]\}$
$\dot{T}_T$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
c_x	$\{\}$	$\{[-1.65, -1.61], [1.40, 1.47], [1.93, 2.00]\}$
$\dot{c}_x$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
c_y	$\{\}$	$\{[-2.12, -2.11], [-2.07, -1.92], [0.54, 0.83]\}$
$\dot{c}_y$	$\{[-0.05, 0.05]\}$	$\{[-0.05, 0.05]\}$
T	Boolean	Boolean
bang	Boolean	Boolean

reliable if the hand is not on the block. The first DBN is most reliable if the hand is not higher than the block, and the second DBN is most reliable if the block is not grasped. This makes sense, since if the block is grasped then when the hand moves the block will move, and x_{LR} will be unchanged.

Table 8.4 shows the action for moving the block to the left edge of the table. The third plan says that the robot should pick up the block ($T = \mathsf{true}$). During many runs, the agent also learns that if it sets x_{LR} to zero, then the block will move to the left. This means it needs to hit it with the left side of its hand. The plans for grabbing the block shown in Table 8.5 all specify that the agent should put its hand over the block and move it down. The plans for making the block hit the floor shown in Table 8.6 specify that the block should hit the floor (z_F) in the context. A better plan would be to make z_F be zero and then cause $bang \rightarrow \mathsf{true}$, but this particular agent was not able to reliably set z_F to zero, and so that plan was not available to it.

Table 8.2: Action $a(\dot{h}_x, [+])$: move hand toward the left ($rel(a) = 0.97$)

DBN r_i	$brel(r_i)$	$H(r_i)$	$\mathcal{C}_{o_i}$	$rel(o_i)$
$\langle \{h_x\} : u_x \rightarrow (331.27, +\infty) \Rightarrow \dot{h}_x \rightarrow [+] \rangle$	0.97	0.27	{}	0.97

Table 8.3: Action $a(\dot{x}_{LR}, [+])$: move hand left w.r.t. block ($rel(a) = 0.97$)

DBN r_i	$brel(r_i)$	$H(r_i)$	$\mathcal{C}_{o_i}$	$rel(o_i)$
$\langle \{z_{BT}\} : u_x \rightarrow (331.27, +\infty) \Rightarrow \dot{x}_{LR} \rightarrow [+] \rangle$	0.96	0.40	{}	0.96
$\langle \{T\} : \dot{h}_x \rightarrow [+] \Rightarrow \dot{x}_{LR} \rightarrow [+] \rangle$	0.96	0.26	{}	0.99
$\langle \{\} : \dot{x}_{RL} \rightarrow [+] \Rightarrow \dot{x}_{LR} \rightarrow [+] \rangle$	0.95	0.28	{}	0.78

Table 8.4: Action $a(\dot{T}_L, [+])$: move block towards left edge of table ($rel(a) = 0.28$)

DBN r_i	$brel(r_i)$	$H(r_i)$	$\mathcal{C}_{o_i}$	$rel(o_i)$
$\langle \{z_{BT}, c_x\} : \dot{Y}_{BT} \rightarrow [+] \Rightarrow \dot{T}_L \rightarrow [+] \rangle$	0.26	0.38	{}	0.21
$\langle \{z_{BT}, h_x\} : \dot{z}_{BT} \rightarrow [-] \Rightarrow \dot{T}_L \rightarrow [+] \rangle$	0.24	0.31	{}	0.08
$\langle \{T, z_{BT}\} : \dot{h}_x \rightarrow [+] \Rightarrow \dot{T}_L \rightarrow [+] \rangle$	0.86	0.28	$\{x_{LR}, h_y\}$	0.37

Table 8.5: Action $a(T, [+])$: grab the block ($rel(a) = 0.26$)

DBN r_i	$brel(r_i)$	$H(r_i)$	$\mathcal{C}_{o_i}$	$rel(o_i)$
$\langle \{c_y, c_x\} : u_z \rightarrow [-0.05, 0.05] \Rightarrow T \rightarrow [+] \rangle$	0.69	0.32	{}	0.06
$\langle \{x_{LR}, h_y\} : u_z \rightarrow (-\infty, -0.05) \Rightarrow T \rightarrow [+] \rangle$	0.80	0.48	{}	0.39
$\langle \{c_x, c_y\} : h_z \rightarrow [13.76, 13.89] \Rightarrow T \rightarrow [+] \rangle$	0.80	0.50	{}	0.43

8.2 Dynamics, Hidden State, Probability, and Noise

Noise can come from any form of nondeterminism. QLAP handles nondeterminism well because it uses statistical learning and probabilistic models. In the environments discussed in Chapter 7, there is noise from three sources:

Table 8.6: Action $a(bang, [+])$: hit the block to the floor $(rel(a) = 0.04)$

DBN r_i	$brel(r_i)$	$H(r_i)$	$\mathcal{C}_{o_i}$	$rel(o_i)$
$\langle \{z_F\} : \dot{T}_L \rightarrow [+] \Rightarrow bang \rightarrow [+] \rangle$	0.93	0.10	{}	0.08
$\langle \{z_F\} : \dot{T}_R \rightarrow [+] \Rightarrow bang \rightarrow [+] \rangle$	0.93	0.08	{}	0.10

1. hidden state from dynamics,
2. incomplete or incorrect models, and
3. quirks in the simulator.

8.2.1 Noise from Dynamics

Dynamics is how a physical process changes over time. Dynamic environments tend to have aspects that are not directly observable. In QLAP, acceleration is not observable and neither is momentum. And in Laplace's conception of natural phenomena, hidden state leads to nondeterminism [Pearl, 2000]. By contrast, in grid worlds like the taxi domain [Dietterich, 2000] or in Drescher's grid world [Drescher, 1991], all variables that affect the dynamics of the environment are observable.

8.2.2 Noise from Incomplete or Incorrect Models

QLAP builds models from the bottom up using a hillclimbing process. This means that models will often be incomplete. For example, QLAP learns the DBN

$$\langle \emptyset : u_x \rightarrow [+] \Rightarrow \dot{h}_x \rightarrow [+] \rangle \tag{8.1}$$

that says that if the agent applies positive force, then the hand will move in the positive x direction. This DBN is reliable only about a third of the time because it requires a force of 300 units to move the hand. Once the agent learns that distinction, and the distinction that if the hand is already at the farthest point on the x axis it can no long move, then the DBN becomes more reliable

$$\langle h_x : u_x \rightarrow (300, +\infty) \Rightarrow \dot{h}_x \rightarrow [+] \rangle \tag{8.2}$$

Discretizing the environment allows for generalization as discussed in Chapter 1, but if the discretization is not perfectly correct, it also leads to noise. For example, because of statistical sampling and noise from hidden state, QLAP may learn a landmark on u_x at 285 instead of 300. This will lead to noise because motor values are chosen randomly within the interval, and the hand will not move each time a value of less than 300 is chosen. Randomly choosing values within intervals can also lead to other noise. How long it takes for the hand to begin moving depends on where the chosen value for u_x falls in the range $(300, +\infty)$.

There is also noise from unusual situations that may not be accounted for by learned models. For example, when the hand is on top of the block, the hand may not move even if $u_x > 300$ because it may be stuck. Table 8.7 shows an example of the hand being stuck and moving almost as slowly as u_x being just barely above 300 (shown in Table 8.8).

8.2.3 Noise from the Simulator

The Breve environment based on ODE displays some unexpected behavior. For example, the robot sometimes shakes when the hand hits a limit of movement. This event causes the variable values to fluctuate as shown in Tables 8.9 and 8.10.

Table 8.7: Friction impedes movement when the hand is on top of the block ($u_x = 500$).

t	$\tilde{h}_x$	$\dot{\tilde{h}}_x$	$\dot{h}_x$
940	−2.34	0.07	[+]
941	−2.33	0.07	[+]
942	−2.33	0.07	[+]
943	−2.33	0.07	[+]
944	−2.32	0.07	[+]
945	−2.32	0.07	[+]
946	−2.32	0.07	[+]
947	−2.31	0.07	[+]
948	−2.31	0.07	[+]

Table 8.8: The hand moves slowly when given a relatively small force ($u_x = -305$).

t	$\tilde{h}_x$	$\dot{\tilde{h}}_x$	$\dot{h}_x$
9	0.00	0.00	[0]
10	0.00	−0.01	[0]
11	0.00	−0.02	[0]
12	0.00	−0.03	[0]
13	−0.01	−0.04	[0]
14	−0.01	−0.06	[−]
15	−0.01	−0.07	[−]
16	−0.02	−0.08	[−]
17	−0.02	−0.09	[−]
18	−0.03	−0.11	[−]
19	−0.03	−0.12	[−]
20	−0.04	−0.13	[−]

8.2.4 QLAP could implement smoothing

One type of noise not handled by the current implementation of QLAP is large Gaussian noise. Imagine a variable $\tilde{v}$ whose trend is increasing, but $\dot{\tilde{v}}$ is jumping up and down. Such a situation would be difficult for QLAP because the event $\dot{v}\rightarrow[-]$ would be triggered when it was not relevant. QLAP could handle such a situation by segmenting the time series. For example, by using a piecewise linear representation [Keogh *et al.*, 2001]. A related issue is that the current implementation does not distinguish between a direction of change event that lasts for a long time and one that lasts for a short time. In some applications, if the value of a variable increases or decreases for a short number of timesteps, it may not be a noteworthy event. A smoothing mechanism could also handle this case, but one would have to determine what amount of smoothing was appropriate for the environment.

8.3 Theoretical Bounds

QLAP is reasonable efficient in time and space. Let V be the number of variables. There are fewer than 9 landmarks learned per variable (usually between 2 and 4), which means there are fewer than 19 qualitative values per variable. The following table lists some values that are used in the subsequent discussion.

Table 8.9: The hand dips in the y direction when the hand hits the limit of movement in the x direction.

t	$\tilde{h}_x$	$\tilde{\dot{h}}_x$	$\dot{h}_x$	$\tilde{h}_y$	$\tilde{\dot{h}}_y$	$\dot{h}_y$
158	1.11	8.23	[+]	3.00	0.00	[0]
159	1.55	8.72	[+]	3.00	0.00	[0]
160	2.01	9.22	[+]	3.00	0.00	[0]
161	2.49	9.72	[+]	3.00	0.00	[0]
162	2.49	−0.13	[−]	2.93	−1.34	[−]
163	2.49	0.15	[+]	2.91	−0.55	[−]
164	2.50	0.12	[+]	2.91	0.00	[0]
165	2.50	0.00	[+]	2.91	0.00	[0]

Variable	Definition
N_Q	Number of qualitative values, which is fewer than $19V$.
N_R	Number of DBNs. Theoretically, the number of DBNs is $O((N_Q)^2)$ and therefore is $O(V^2)$. Practically, we observe about 250 DBNs on average on 34 variables in the core environment.
N_P	Number of Plans. This is $3N_Q$, which is $O(V)$.

8.3.1 Learning Contingencies

QLAP tracks statistics for each pair of events.[2] This means that learning contingencies is $O((N_Q)^2)$ (and therefore $O(V^2)$) in both time and space.

8.3.2 Adding Context to DBNs

For each DBN, QLAP looks at each variable to see if it should be added to the context. That makes context learning $O(VN_R)$, and therefore $O(V^3)$, time. For space, each DBN has to store the qualitative values of all the variables each time its antecedent event occurs. This storage is limited to the last 200 times the antecedent event occurred. So storage is $O(200N_RV)$ and therefore is $O(V^3)$.

8.3.3 Learning Landmarks on DBNs

For each DBN, QLAP saves the real value of each variable the last 200 times the antecedent event of the DBN occurred (although in the implementation, the data trace is saved and the DBN just stores the line number). To learn DBN landmarks, for each DBN, QLAP looks for a landmark on

[2]QLAP does not track statistics on consequent events of magnitude variables because those are covered by magnitude DBNs.

Table 8.10: The hand dips in the z direction when the hand hits the limit of movement in the y direction.

t	$\tilde{h}_y$	$\tilde{\dot{h}}_y$	$\dot{h}_y$	$\tilde{h}_z$	$\tilde{\dot{h}}_z$	$\dot{h}_z$
795	−1.01	−9.16	[−]	14.95	−0.01	[0]
796	−1.49	−9.72	[−]	14.95	−0.01	[0]
797	−2.01	−10.27	[−]	14.95	−0.01	[0]
798	−2.00	0.16	[+]	14.94	−0.13	[−]
799	−2.00	0.01	[0]	14.93	−0.13	[−]
800	−2.00	0.00	[0]	14.93	−0.09	[−]
801	−2.00	0.00	[0]	14.93	−0.04	[0]
802	−2.00	0.00	[0]	14.92	−0.01	[0]

each variable v in $200-1$ locations. So space and time on learning DBN landmarks is $O(200N_RV)$ and therefore is $O(V^3)$.

8.3.4 Learning Landmarks on Events

For each event E, QLAP creates a histogram with a fixed number of buckets for each variable V and looks at each bucket as a potential landmark. Therefore, learning event landmarks is $O(V^2)$ in time and space.

8.3.5 MDP Planning

MDP planning is known to be computationally expensive because the state space grows exponentially with the number of variables. MDP planning using dynamic programming is $O(S^2A)$, where S is the number of states, and A is the number of actions. Fortunately, the state spaces for MDPs in QLAP are small. Each MDP can have no more than 6 variables, so $S \leq 6 \times 19 = 114$. QLAP actions bring the agent to a qualitative value of a variable. This means that $A < 6 \times 19$ (not all actions are applicable). Time and storage needed for an MDP plan therefore is less than 114^3. There are $N_P \leq 3V$ plans. So time and storage for MDP planning is $O(114^3 \times 3V)$. This is still a big number, but most plans have fewer than 6 variables.

8.4 Thresholds and Parameters

In addition to the parameters that come from reinforcement learning as discussed in Chapter 5, QLAP has four kinds of parameters:

1. Thresholds to determine if enough statistics have been gathered (see Appendix B).
2. Parameters to determine how much resources should be used. For example, a threshold to determine how much history to save. The settings of these are manually determined by the computational and storage resources available to the agent.

3. Thresholds to determine if a decision should be made. For example, a threshold to determine if a landmark should be created or if a DBN should be created. These are the most critical. None of these thresholds (or any other thresholds) needed to be reset when QLAP was run on the Pong environment. A table of these parameters is shown in Appendix G.

4. Parameters for exploration. These thresholds include how long to motor babble before starting exploration using actions, and how often to take a random action, and how many timesteps random actions should last. It is important that the agent does not get stuck on one small part of the state space. This can happen, for example, if the agent learns a landmark on a upper limit of a variable v and no other landmarks on v. In this case, if nothing else in the dynamics of the environment changes the value of v, then the agent may spend all of its time at the upper limit of v.

8.5 Hierarchy

QLAP learns a hierarchy of plans and actions, but this hierarchy is not strict. The hierarchy is loosely maintained because the agent is required to be able to reliably achieve the antecedent event before a new plan can be added. But there is no guarantee about context variables being "below" in the hierarchy, or even having plans. QLAP is designed this way because exploratory experiments demonstrated that learning a strict hierarchy was too brittle. This brittleness came from race conditions in the learning of representations.

The brittleness of learning a strict hierarchy in continuous environments is an interesting finding. Most previous work on learning hierarchy has been done on grid-like environments, e.g. [Vigorito and Barto, 2008; Dietterich, 2000] where the desired hierarchy is generally clear. An interesting direction for future work would be to determine in what types of environments is a strict hierarchy possible or desired. And when a strict hierarchy is not possible or desired, what kinds of non-strict hierarchies should be used.

Chapter 9

Related Work

QLAP is the only algorithm that we are aware of that learns states and hierarchical actions in continuous, dynamic environments with continuous motors through autonomous exploration. The closest direct competitor to QLAP is the work of Barto, Jonsson, and Vigorito. Given a DBN model of the environment, the VISA algorithm [Jonsson and Barto, 2006] creates a causal graph which it uses to identify state variables for options. Like QLAP, the VISA algorithm performs state abstraction by finding the relevant variables for each option. Jonsson and Barto [2007] learn DBNs through an agent's interaction with a discrete environment by maximizing the posterior of the DBN given the data by building a tree to represent the conditional probability. Vigorito and Barto [2008] extends [Jonsson and Barto, 2006; 2007] by proposing an algorithm for learning options when there is no specific task.

This work differs from QLAP in that learning takes place in discrete environments with events that are assumed to occur over one-timestep intervals. The work also assumes that the agent begins with a set of primitive actions. Because QLAP is designed for continuous environments with dynamics and does not assume a set of primitive actions, QLAP uses a qualitative representation. This qualitative representation leads to a novel DBN learning algorithm for learning predictive models, and a novel method for converting those models into a set of hierarchical actions.

In the remainder of this chapter we survey the related literature and discuss how particular aspects of different approaches compare with QLAP.

9.1 Autonomous Learning

We say that an agent learns autonomously if it learns models and actions without being directed what to learn. We also assert that autonomous learning should happen over a long, continuous time period, as opposed to learning that occurs as a sequence of episodes set by the experimenter. Drescher's Schema Mechanism [Drescher, 1991] and the work on Neo by Cohen *et al.* [1997] are important precursors to QLAP. Neo learns common sequences of events in a discretized environment where there was no explicit goal.

Drescher's system learns STRIPS-like rules called *schemas* in a discrete environments that link together a context, an action, and a result. The example schema

$$\langle InFrontOfDoor | OpenDoor | DoorOpen \rangle \tag{9.1}$$

given in [Chaput, 2004] states that if the robot is in front of the door, and it opens the door, then the door will be open. Learning begins by creating a *bare schema* $\langle |a| \rangle$ for each action a. Schemas are then grown by a method Drescher calls *marginal attribution*. The schema $\langle |a| \rangle$ is spun off to $\langle |a| r \rangle$ if the result event r is more likely to follow a than otherwise. A context item c is added to

$\langle |a| r \rangle$ resulting in $\langle c|a|r \rangle$ if the schema is more reliable when c is true. Because of the "spinning off" of new schemas, there can be multiple schemas that predict the same result. In QLAP, the learning of multiple small models to predict events, and the method for adding context variables to DBNs were directly inspired by Drescher.

Shen's LIVE algorithm [Shen, 1994] learns a set of rules in first-order logic and then uses goal regression to perform actions. The algorithm assumes that the agent already has basic actions, and the experiments presented are in environments without dynamics such as the Tower of Hanoi. Another method for learning planning rules in first-order logic is [Zettlemoyer *et al.*, 2005; Pasula *et al.*, 2007]. The rules they learn are probabilistic, given a context and an action their learned rules provide a distribution over results. This algorithm assumes a discrete state space and that the agent already has basic actions such as *pick up*.

Oudeyer, Kaplan, and Hafner [Oudeyer *et al.*, 2007] created a system for autonomous developmental learning driven by the agent's ability to predict its environment. The history of the agent is divided into cells based on the state vector $\mathbf{S}(t)$, the motor vector $\mathbf{M}(t)$, and the next state vector $\mathbf{S}(t+1)$. Each cell has an expert that predicts the next state $\mathbf{S}(t+1)$ given a state and action $\mathbf{SM}(t)$ in that cell. To choose actions, the agent takes the current state $\mathbf{S}(t)$ and finds the action that would put it in the cell whose expert is improving the fastest in its ability to predict the next state. When a cell has seen 200 data points it is split into two new cells. The split is done by finding a cutpoint on an input variable $s \in \mathbf{S}$ that splits the cell so that the variance of the next states $\mathbf{S}(t+1)$ in the new cells is minimized. In a simulated experiment with a two-wheeled robot, and with an experiment with a physical Aibo robot in a playground environment, the robots were able to learn interesting behaviors. In both cases the input vectors to the robots were small (one continuous variable for the two-wheeled robot, and three Boolean variables for the playground experiment) and it is unclear if the variance-based cutpoint splitting mechanism would be able to find sensible splits in environments with higher dimension sensory input. In a more recent paper [Baranes *et al.*, 2009], the splitting mechanism is improved to maximize the dissimilarity of learning progress.

9.2 Learning Models

One classic bootstrapping algorithm for learning models is Structural EM [Friedman, 1997; 1998]. Structural EM is used when there are hidden variables or missing values. QLAP assumes that the agent is able to observe its variables. In the case where the variables are observable, the standard algorithm for learning structure in probabilistic models is to keep counts of co-occurrences of variables values and to define a scoring function (typically BIC or MDL) for a network and parameters and then to perform a hillclimb search to try to maximize that score [Heckerman, 1995; Friedman *et al.*, 1998]. In iid environments, this method can even be used to find discretizations of variables by inserting a discretization step in the search and scoring method [Friedman and Goldszmidt, 1996].

There has been work to adapt this process of structure learning to learning in sequential decision processes where the environment can be modeled as a factored MDP. Degris *et al.* [2006] proposed a method called SDYNA that learns a structured representation in the form of a decision tree and then uses that structure to compute a value function. Strehl *et al.* [2007] learn a DBN to predict each component of a factored state MDP. Hester and Stone [2009] learn decision trees to predict both the reward and the change in the next state. All of these methods are evaluated in discrete environments where transitions occur over one-timestep intervals.

A method for learning probabilistic planning rules is proposed by Schmill [Schmill, 2002].

Schmill's algorithm learns planning operators that given a context and an action provide a distribution over results. The context is learned using a decision tree and is different from a context in QLAP because it only gives the probability of the result for single range of values for each variable. The results predicted by operators come from increases or decreases in variable values [Schmill *et al.*, 1998] or by clustering of the sensory information [Schmill *et al.*, 2000]. This method differs from QLAP because QLAP considers any distinctions found in the context of any contingency as broadly useful to the agent allowing it to form the antecedent event of new contingencies. Additionally, all operators are on motor variables, where QLAP allows arbitrary events.

9.2.1 The Approach Taken by QLAP

QLAP uses a qualitative representation to discretize the environment. QLAP learns landmarks and then uses those landmarks to define special variables to create the DBN. This allows QLAP to handle transitions that occur over more than one timestep.

Contrary to [Degris *et al.*, 2006; Vigorito and Barto, 2008; Strehl *et al.*, 2007], QLAP does not learn a different DBN for each action-result combination. Instead, QLAP learns DBNs by first observing a contingency, and then adding context variables using marginal attribution. If the antecedent event for a contingency is not on a motor variable, QLAP still treats it like a primitive because QLAP creates high-level actions for each known event. Treating the antecedent event like a primitive action leads to a hierarchy, because it can be called as part of a plan without having to know the details of how it is implemented.

The approach to learning models taken by QLAP may result in multiple models that predict a consequent event (each with a different antecedent event). This is important because since we want to make these models into plans, it is imperative that models be small so that the resulting plans have a small state space. So instead of having fewer larger models covering many situations, QLAP learns many, smaller models that are situation specific.

9.3 Learning a State Abstraction

As described in Chapter 1, QLAP builds on the motor primitive abstraction learned by [Pierce and Kuipers, 1997] and the identified-objects abstraction learned by [Modayil and Kuipers, 2007]. QLAP builds on these abstractions in two ways. (1) QLAP identifies states by learning landmarks for a qualitative representation (2) QLAP identifies a small set of important variables for each learned plan.

9.3.1 Discretizing the Space

In our experiments, a simulated robot learns to hit a block while exploring the world. In the process of learning how to hit a block, the robot learns that if its hand is on the left side of the block, then the block will go to the right. This is interesting because initially the robot has no concept of "the left side of the block." It *learns* that there is such a thing as having its hand on the left side of the block, and this knowledge allows it to map the infinite number of possible states where its hand is on the left to a single state. By learning to hit the block, the robot has learned to see the world in a more useful way.

QLAP discretizes the space by iteratively learning new distinctions that make existing models more reliable. It finds these discretizations using the method of Fayyad and Irani [Fayyad and Irani, 1992] originally designed for finding discretizations in decision trees [Quinlan, 1993]. Each new discretization allows new models to be learned, and so there is a synergy between the models and

the discretization. Friedman and Goldszmidt [1996] also does a loop between model learning and discretization, and they also use the method of Fayyad and Irani [Fayyad and Irani, 1993] (although they use the 1993 version that uses the minimum description length principle to determine the cut points, and in our experiments this produced too many landmarks). Friedman and Goldszmidt [1996] learn one large model instead of many small models as QLAP does, and their method is not for sequential decision tasks. We found that learning Bayesian networks purely for prediction was different than learning Bayesian networks that would later be used for planning. For example, we found that we needed to use the idea of best reliability (*brel*) described in Chapter 4 as a scoring metric because it might be important to find some state in which the transition is reliable, even if the transition cannot be predicted in other states.

Goodman *et al.* [2007] also simultaneously learn a model and a discretization. They treat each discretized value as a Boolean variable, and then use those variables to learn Bayesian models. The discretization comes by finding square regions of a two-dimensional space such that models with high posterior probability can be built.

Clustering

Another way to discretize the input space is to use clustering. Cohen *et al.* [1997] did work with abstracting sensor input using clustering of time series data. Provost and Kuipers [2007] use a growing neural gas self-organizing map (SOM) [Fritzke, 1995] to discretize the environment. They then learned control laws to allow a robot to get from one winning SOM node to another. Clustering is an important abstraction tool when the statistics of the sensory input are such that clustering can find meaningful states for the robot. However, clustering can miss subtle but important distinctions that may be small in the statistics of the data, but large with respect to the goals of the robot. For example, consider a Skinner box [Skinner, 1961]. To a clustering algorithm all of the small details of the box may be equally important, but the important feature indicating the coming electrical shock may be a small red light (a similar point was made in [Goodman *et al.*, 2007]).

U-Tree

McCallum [1996] created the U-Tree algorithm. Given a set of discretized variables, the U-Tree algorithm learns a state space that includes relevant history. In U-Tree, a new state is created if it leads to a difference in reward. In QLAP, new states are learned not based on reward, but rather based on the ability to make predictions.

Jonsson and Barto [2001] use U-Tree to define a state space and a policy for a set of options defined for the Taxi task [Dietterich, 2000]. They assigned a different U-Tree for each task. Uther and Veloso [2003] extend U-Tree to work with continuous variables by finding cutpoints that maximize differences in the reward received on each side of the cutpoint.

Adaptive Partitioning

Reynolds [2000] proposed a way to refine the resolution of a Q-function. Regions are split if the best action is different in adjacent regions. Like QLAP, the representation begins course and is progressively more refined. But the refinements are based on reward, and not the ability to predict the next state as is done in QLAP. Stone *et al.* have also done work on learning representations. Sherstov and Stone [2005] enable a learning algorithm to autonomously set the generalization parameter for learning using tile coding. Whiteson and Stone [2006] propose a method for learning of representations for TD-learning using evolutionary function approximation.

9.3.2 Finding the Right Variables

As mentioned previously, given a DBN model of the environment, the VISA algorithm [Jonsson and Barto, 2006] creates a causal graph which it uses to identify state variables for options. And there has been work on state abstraction [Jong and Stone, 2005] that finds the relevant variables by using hypothesis testing or Monte Carlo simulation to determine which variables should be included to express an optimal policy.

9.4 Learning Actions

QLAP does not assume a set of actions, it learns the agent's first actions from a learned representation. This representation includes a learned discretization of the motor space. Another way of learning in continuous environments is Binary Action Search (BAS) [Pazis and Lagoudakis, 2009]. BAS allows many known RL algorithms to learn to execute continuous action policies. The key to moving to continuous actions is that instead of choosing an action at each timestep, BAS chooses to increase or decrease the current action (thus using a binary policy). To determine how much to increase or decrease the current action, BAS uses binary search to query the current binary policy to find the best amount it should be modified.

In addition to the continuous action learners discussed in the introduction, Deisenroth and Rasmussen [2009] learn transition dynamics using a Gaussian process. These learned transition dynamics are then used to learn a value function and a policy. Another method for learning actions is Parti-game [Moore and Atkeson, 1995]. Parti-game allows an agent to reach a goal by breaking up the state space into cells, but it requires a controller to get from one cell to another. Building on this, [Munos and Moore, 1999] does fine-grained discretization for control, but it needs to be provided a model.

Modayil and Kuipers [2007] learn actions by first motor babbling and then extracting actions that measure high on a scoring function that consists of the geometric mean of the precision, recall, and reliability of the action. This work is similar to QLAP in that actions are learned with no specific goal in mind, and can later be used to achieve specific goals. This work differs from QLAP in that the actions do not use MDP planning and are not hierarchical, but instead are based on control laws.

9.5 Learning Hierarchy

The hierarchy in QLAP comes from plans calling learned QLAP actions instead of built-in action primitives.

9.5.1 The MAXQ Value Function Decomposition

QLAP's learned structure of actions and plans is reminiscent of the MAXQ value function decomposition [Dietterich, 1998]. In MAXQ, the agent's capabilities are represented by a hierarchy of subtasks. The hierarchy consists of two types of nodes: *Max nodes* and *Q nodes*. Each Max node corresponds to a separate subtask M_i, and the children of M_i are Q nodes that are the actions available to M_i. Each Q node in turn has a single child that is a Max node to carry out the action. Ghavamzadeh and Mahadevan [2001] extend MAXQ to continuous-time SMDPs. Jong and Stone [2008] combine MAXQ with a model-based approach. They learn the transition and reward function models and use that to compute the decomposed value function.

Each task node of MAXQ roughly corresponds to a plan in QLAP, and each action node in MAXQ roughly corresponds to an action in QLAP. One difference is that in QLAP, an action can choose from multiple plans and choose the best one for the current situation. One significant difference in objective is that QLAP is not trying to represent a single, underlying MDP. A QLAP agent receives no external reward, and instead is trying to autonomously learn a set of skills. And so there is no single root node as in MAXQ. QLAP defines its own actions and plans as it learns, and as the agent learns a more refined discretization the hierarchy changes.

9.5.2 Other Work on Learning Hierarchy

There has also been much work on learning hierarchy. Like QLAP, Digney [1996] creates a task to achieve each discrete value of each variable. However, QLAP learns the discretization.

Work has been done on learning a hierarchical decomposition of a factored Markov decision process by identifying *exits*. Exits are combinations of variable values and actions that cause some state variable to change its value [Jonsson and Barto, 2006]. Exits roughly correspond to the DBNs found by QLAP except that there is no explicit action needed for QLAP DBNs. Hengst [2002] determined an order on the input variables based on how often they changed value. Using this ordering, he identified exits to change the next variable in the order and created an option for each exit.

9.6 Creating Reinforcement Learning Problems

QLAP carves out important states in the environment by finding states that make predictive models more reliable. And given the current state abstraction, QLAP generates reinforcement learning problems to get to desired states. Many methods exist to learn options from a given discretization. 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. Our focus on learning distinctions also distinguishes our work from the hierarchical reinforcement work of Bakker and Schmidhuber [2004], which clusters low-level observations.

A learning agent can also define options to achieve salient states [Barto *et al.*, 2004; Stout *et al.*, 2005; 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.

Konidaris and Barto [2010] propose an algorithm that learns options in a chain. It first learns an option to get to the goal within 250 timesteps of the goal. Since this is a continuous environment, the algorithm uses regression to learn what parts of the space fall within this 250 timesteps (to be the set of initiation states for the next option). Once it has learned this option, the agent learns an option to reach this first set of initiation states, and so on. The agent uses function approximation to learn a policy for each option. This allows the function approximator to focus on a smaller part of the space.

Chapter 10

Future Work

Future work in QLAP entails enabling the algorithm to handle more complex environments. For example, how can a developing agent learn complex actions in huge environments? And, what kind of developmental milestones would we want to lay out for the agent? And to achieve these milestones, how can an agent learn more complex representations? Also of interest, is how can QLAP be used as scaffolding for continuous methods such as regression. And how can QLAP be scaled up to very large numbers of variables through active perception. This chapter discusses these topics.

10.1 Navigating the Space of Environmental Configurations

A developing agent has to search through a huge space of environmental configurations. For example, in the secondary block environment in Chapter 7, the environment had to be configured so that the blocks were aligned in order to use one block to hit another. How can an agent navigate this large space? And since learning actions requires practice, it is not sufficient to just arrive at the configuration once. The agent needs to be at the configuration repeatedly. This section presents two possible approaches to this problem. The first is social learning, and the second is intrinsic valuation of events.

10.1.1 Social Learning

Other agents can point out what is worth learning, can guide the agent to achieve the prerequisites for actions, and can guide the agent on how to perform the actions. One way that the benefits of social learning can be achieved is through imitation. But learning through imitation is a challenge because the "student" agent has to map the actions and movements of the "teacher" agent onto its own body. Since QLAP learns representations autonomously, it might have an advantage in such learning because it could map what the teacher agent is doing using its learned representation.

An interesting first experiment would be to have one agent guess the intention of another agent. The two agents would be allowed to develop independently, each running QLAP. Then we could allow a teacher agent to explore and see if the student agent can guess the goal of the teacher agent using the data trace from the teacher agent. The student agent would guess the goal by picking its own action that would be most likely given the data trace. The performance of the student agent on this task could be compared with supervised learning algorithms. Once it was established that the student agent could guess the goal of the teacher agent, the teacher agent could be used to guide the development of the student agent.

10.1.2 Intrinsic Valuation of Events

If there is some way, such as social interaction, of specifying events worthy of special attention, then the agent may be able to do intrinsically-motivated exploration. Intrinsically-motivated exploration is done in a grid world by Barto *et al.* [2004], where interesting events are specified by the experimenter and the agent creates an option to achieve each interesting event. An analog to this process could be implemented in QLAP. When an interesting event occurred, the agent could note the qualitative state and try to get back to that state in the future. The agent would have to figure out how to narrow this qualitative state down to the important variables. These variable values could then have intrinsic value to the agent.

10.2 Reaching Developmental Milestones

QLAP provides an algorithm for autonomous learning in continuous environments. This algorithm could allow us to further explore what it takes to autonomously learn complex behavior. With children and animals, we can modify the environment and experimental conditions, but we cannot directly modify the learning algorithm. QLAP allows us to perform experiments by manipulating the algorithm. Some interesting tasks for further research would be:

1. Design a cup with a coffee-cup like handle and see if the agent can learn to pick up the cup by putting a stick through the loop.
2. Provide a box and small objects and see if an agent can learn to put small objects in the box, move the box, and spill out the objects. See if the agent can learn that the objects move when the box moves.
3. Provide the agent with a stick and see if it can use the stick to move a block.
4. Provide the agent with multiple blocks and see if the agent can learn to:
   (a) push other blocks with the grasped block
   (b) knock other blocks with the grasped block
   (c) push one block away with the grasped block
   (d) let go of the grasped one, and push both with that one
   (e) push a "train" of blocks until the end one falls off the table
5. Provide blocks of different shapes, such as long blocks or L-shaped blocks, and see if the agent can use them as tools. For example, see if the agent can use the L-shaped block to bring a distant block closer.

10.3 Creating New Representations

QLAP learns DBNs, plans, and actions based on the variables given, but there is nothing in QLAP that prevents it from using new variables acquired during development. This section proposes some ways that could be done.

10.3.1 Learning new variables by trying combinations

One could use a method such as [Stober and Kuipers, 2008] to generate many potential variables from combinations of existing variables. For example, it could be that if for two existing variables A and B that if $A_t - B_t = 90$ then a DBN r successful. In this case it would make sense to create a new variable $V = A - B$ with a landmark at 90. This method could result in many variables, but this problem should be mitigated somewhat if variables are only added when they do some measurable good. In Chapter 7, we saw that variables could be added without significantly degrading performance. This ability of QLAP to work with additional variables should help to mitigate the stagnation problem of the algorithm being overwhelmed by variables [Lenat and Brown, 1984]. One possibility entails seeing how QLAP could be used in a massively parallel environment. Many variables and transforms of variables could be generated and tested, and added to the system if they are found to be valuable.

10.3.2 Learning more complex representations

Consider the example of one block sitting on top of another block. How could this state be represented in QLAP? Currently, it would have to be represented as a conjunction of variables. For example, for two blocks b^1 and b^2, to represent the state that b^1 is on top of b^2, we would need the cumbersome representation

$$b^1_x = b^2_x \quad \wedge \quad b^1_y = b^2_y \quad \wedge \quad b^1_z > b^2_z \tag{10.1}$$

This formula abstracts many low-level states, and we would like to refer to all of those states using a single symbol. Using a single symbol, we can start to predict results of being in this state or when this state will occur. We refer to this problem of creating a symbol to represent an important state as the *state abstraction problem*, and we call such states *high-level states*.

There is another representational issue, the *event abstraction problem* is the problem of abstracting time-series data into a single event. Consider the event of block b^1 falling off block b^2. This event takes more than one timestep and involves a change in multiple variables. We refer to such events as *high-level events*.

If we can define high-level states, then we can define high-level events as transitions between high-level states. If we do this, the set of possible solutions to the state representation problem is the set of ways to group variables (and variable values) into high-level states.

To aggregate these states, we can consider both bottom-up and top-down approaches. One bottom-up method is to identify high-level states as those that are *stable*. We define states as stable as those that come up often and stay that way for a relatively long time. One block on top of another is stable, and two blocks sitting on a table is also stable. Another example is a block sitting on the floor after it has fallen off the table. Defining a high-level state as a block sitting on the floor allows us to define the high-level event of the block falling off the table without specifying an additional variable. The high-level event is the transition from the high-level state of the block being on the table to the high-level state of the block being on the floor. By contrast, a top-down approach would be to look for "important" states that are associated with positive outcomes for the agent.

However, sometimes we may be interested in the way a transition takes place. For example, the agent may want to learn the difference between slide and tumble. We might want the agent to learn that pushing sideways from the middle of the block will cause it to slide, but pushing from the upper edge will cause it to tumble.

10.4 Qualitative Model as Scaffolding

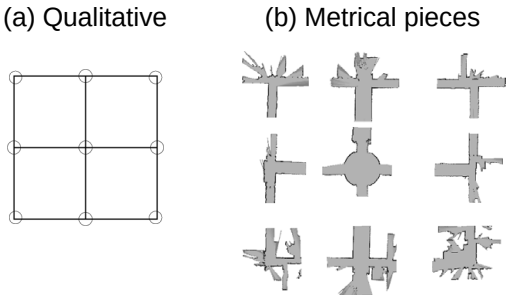


Figure 10.1: Metaphor for how QLAP can break up an environment. Chapter 1 discusses how QLAP is analogous to learning a topological map that breaks up the environment instead of a metrical map. But once QLAP breaks up the environment, continuous methods can be used within the pieces to get the best of both the qualitative approach used in QLAP and continuous approaches such as regression. (a) Metaphor for how the environment is represented by QLAP. (b) Metaphor for how the pieces of the environment are linked together. Continuous learning can then be done within each piece.

We can think of the qualitative model of the environment learned by QLAP as a type of scaffolding, as shown in Figure 10.1. QLAP breaks up the world in three different ways. First, QLAP breaks up the world by learning landmarks. This creates a discrete state space. Second, QLAP breaks the world up into small clusters of variables through the DBN and MDP learning. And third, QLAP learns actions which result in qualitative trajectories through space.

Breaking up the world into a state space and small sets of variables could be useful for regression. Regression has no notion of state, and so while regression may be able to predict a little way ahead with high accuracy, it can have trouble where the landscape changes. The factoring of the environment provided by QLAP may allow prediction regression to be extended to more complex environments. This might be useful, for example, for predicting when components of a complex system will fail.

Breaking up the world into trajectories could be useful for other learning methods. For example, QLAP is able to move the arm through space, but may not do this with the kind of precision that other methods may be able to attain. However, other methods, for example regression [Vijayakumar *et al.*, 2005], often need a trajectory to follow to be effective. The qualitative movements of QLAP could provide this trajectory.

10.5 Scaling up through Active Perception

QLAP could use active perception [Ballard, 1991] to scale up to environments with significantly more variables. The agent could acquire the values of the variables by actively choosing which variables were needed for the current situation. This would allow the agent to ignore the irrelevant variables.

As a first experiment, consider a scenario with a hand and two blocks. The agent could choose to know the values of the variables related to two of the three objects at any timestep. If the agent could also choose to see only the hand, this would result in four possible choices:

1. see block 1 and block 2,

2. see the hand and block 1,

3. see the hand and block 2, or

4. see only the hand.

If the agent could learn to make this choice well, it could reduce computation. For example, seeing only the hand would be useful at the beginning for learning how to move the hand because the relational variables between the hand and the blocks would not get in the way. Later on, it could choose to see the hand and the block its hand was closest to.

10.6 Conclusion

This chapter outlined possibilities for expanding QLAP in the future. It would also be interesting to apply QLAP to problems unrelated to developmental learning. For example, QLAP could be applied to the problem of fault detection or anomaly detection. QLAP creates many models of the environment in the form of DBNs. When one of these DBNs are no longer predictive, it could be an indication that there has been an anomalous event. QLAP could be used to determine when a system has been hacked or there has been malicious activity on a network.

Another interesting possible future direction would be allowing QLAP to take advantage of specific algorithms when they were applicable. For example, if the variables in the MDP were directly connected to the motor variables, QLAP could use movement planning (for example, RRT [Kuffner Jr and Lavalle, 2000]). As another example, QLAP currently assumes that the method of Pierce and Kuipers [1997] can provide orthogonal motor primitives. Another possibility is first learning the motor primitives using a preprocessing method such as [Sun and Scassellati, 2005].

Chapter 11

Summary and Conclusion

The Qualitative Learner of Action and Perception (QLAP) is an unsupervised learning algorithm that allows an agent to autonomously learn states and actions in continuous environments. Learning actions from a learned representation is significant because it moves the state of the art of autonomous learning from grid worlds to continuous environments. Another contribution of QLAP is providing a method for factoring the environment into small pieces. Instead of learning one large predictive model, QLAP learns many small models. And instead of learning one large plan to perform an action, QLAP learns many small plans that are useful in different situations.

QLAP discretizes the environment using a qualitative representation. A qualitative representation allows the agent to generalize and to focus on important events. Using a qualitative representation, QLAP learns a discretization of the environment and a set of predictive models simultaneously. QLAP begins with a very broad discretization that only allows the agent to know if a sensory input variable is increasing or decreasing in value. Then, given the current discretization, QLAP learns a set of possibly unreliable predictive models. For each model, QLAP can look for a new discretization (qualitative landmark) that makes the model more reliable. QLAP assumes that such a landmark represents an inherent discontinuity in the environment, and the landmark is added to the current discretization. The agent then is able to learn more models, and this process of learning of landmarks and models creates a cycle leading to increasingly reliable models and an increasingly fine-grained representation.

The predictive models that QLAP learns are in the form of Dynamic Bayesian Networks (DBNs). QLAP uses a novel method for learning DBNs in continuous environments. This method is based on learning contingencies. A contingency is a pair of events where the occurrence of the antecedent event leads to the occurrence of the consequent event. To identify contingencies, QLAP performs a search over pairs of events. Once a contingency is found, it is converted into a DBN. QLAP then adds context variables to make the DBN more reliable. Adding context variables and adding new landmarks are two ways that QLAP has for making DBNs more reliable.

Planning in QLAP uses both goal regression and Markov Decision Process (MDP) planning. Goal regression is advantageous when the plan only uses some of the available variables. But goal regression has difficulty with nondeterministic transitions. MDP planning handles nondeterminism well, but grows exponentially with the number of variables in the state space. QLAP takes advantage of the best of both by creating many small MDPs and linking them together in goal-regression fashion. Each MDP in QLAP is created from a learned predictive model. This ensures that the MDP is small because the predictive model has already identified the important variables.

QLAP creates an action to achieve each qualitative (discrete) value of each variable. To perform an action, QLAP uses the MDP plans just described. An action can have more than one plan. This allows different plans to be called in different situations. The actions within each MDP plan

are QLAP actions, and this links the plans and actions together in a hierarchical, goal-regression like process.

QLAP explores autonomously and tries to learn to achieve each qualitative value of each variable. To explore, the agent continually chooses an action to practice. To choose which action to practice, QLAP uses Intelligent Adaptive Curiosity (IAC). IAC motivates the agent to practice actions that it is getting better at, and IAC motivates the agent to stop practicing actions that are too hard or too easy.

QLAP was evaluated in environments with simulated physics. The evaluation was performed by having QLAP explore autonomously and then measuring how well it could perform a set of tasks. The agent learned to hit a block in a specified direction and to pick up the block as well or better than a supervised learner trained only on the task. The evaluation also showed that the landmarks learned by QLAP were broadly useful. Future work will consist of incorporating continuous learning methods within the discretized representation learned by QLAP. This should enable QLAP to leverage both best of discrete learning and the best of continuous learning.

Appendix A

Nominal Variables

QLAP can handle nominal variables, such as Boolean variables. Nominal variables can appear anywhere in DBNs. DBNs with nominal variables as the consequent event are learned and treated just like direction of change DBNs. Nominal actions are treated like direction of change actions, and nominal options are treated the same as direction of change options. Additionally, there are two details

1. To compute $\mathcal{A}_r^s$ for state s, QLAP subtracts those actions on nominal variables whose goal is already achieved in state s.
2. For a nominal variable to be part of self, all of its values must meet the two criteria for self.

Appendix B

Computing Probability

QLAP computes statistics by maintaining counts on successes and failures. To determine a probability p_s of success, we let

$$p_s = \frac{\#success + \alpha}{\#fail + \alpha + \#success + \beta} \tag{B.1}$$

where $\alpha = \beta = 1$ is added as a prior. (There are two exceptions: the prior is $\alpha = \beta = 0$ for determining which plan should be replaced (Chapter 6) because it requires 30 activations, and $\alpha = \beta = 2$ when determining if an action is sufficiently reliable (Chapter 6).)

To determine if p_s is greater than some threshold, we use a beta distribution. We do this because we want to consider the number of observations. With more observations, we can be more sure that a threshold is exceeded.

1. To determine if p_s is less than a threshold θ we use the cumulative probability distribution

$$Pr(p_s < \theta) \equiv \texttt{beta.cdf}(\theta, \#success + \alpha, \#fail + \beta) \tag{B.2}$$

   If $Pr(p_s < \theta)$ is at least 0.8, then we consider p_s to be less than the threshold θ.

2. We consider p_s to be greater than a threshold θ if $1 - Pr(p_s < \theta) \geq 0.8$.

B.1 Computing Probabilities for Best Reliability

The equation for *brel* is based on the calculation of reliability in Equation 4.13 so that *brel* is

$$brel(r) = \operatorname*{argmax}_{q \in \mathcal{C}} reliability(r|q) \tag{B.3}$$

where reliability is calculated as

$$p_s = \frac{\#success + 1}{\#fail + \#success + 2} \tag{B.4}$$

In experiments we found that if there were few data points, and one of those points was a success, then best reliability may not reflect the true reliability. To overcome this, we determined that 5 successes must be observed. If there are not at least 5 successes for any q, then $brel = \perp$. Any *brel* value for any DBN not equal to $\perp$ is always greater than $\perp$, and $brel = \perp$ is never greater than any value.

B.2 Requirement of 30 Data Points

QLAP is an online learning algorithm that makes decisions as information comes in. It is important that these decisions are based on a sufficient amount of information. To make the following decisions, we require that there be 30 data points:

1. Section 4.1.2 learning new DBNs. We also require that $Pr(soon(t, E_2)|E_1(t))$ have 30 data points.
2. Section 4.4.1 learning new landmarks from models. Variable v and open interval q must have at least 30 data (activations) for DBN r to be considered for a landmark.
3. Section 4.4.2 learning new landmarks from events. There must be at least 30 data points in $[lb, ub]$ of $Pr(\tilde{v}_{t-1} \in [lb, ub]|E(t))$.

Appendix C

Learning the DBN Window Size

Since different environments have different granulations of time, we want the algorithm to be able to set k based on the environment. QLAP sets k to be the average amount of time it takes for a motor command to make a noticeable change in the environment.

To find this noticeable change, QLAP notes the average time k_r between each *successful* transition between a motor variable and a direction of change variable. A transition is successful if the consequent event becomes satisfied before the antecedent event stops being satisfied. QLAP then keeps the same statistics that it keeps to learn DBNs, except that k_r is used for k for each potential DBN r.

Then at the periodic processing interval every 2000 timesteps, QLAP finds if any DBN r can be learned using this current value of k_r. If any can be learned, QLAP takes the set of those DBNs that can be learned and finds the average value for k_r, and the resulting average is k.

Appendix D

Hillclimbing Algorithms

This appendix lists the pseudocode and discusses the details of the algorithms for improving DBN models.

D.1 Adding Context Variables

The process of adding context variables is shown in Algorithm 3. For each DBN r, the algorithm makes a copy r' with an empty context. The algorithm then iteratively adds context variables to r' that improve r'. Finally, if r' is an improvement over r, then the context of r' is used for r.

Valid context variables are magnitude variables or nominal variables. The variable on the consequent event cannot be part of the context. Interestingly, a valid context variable must also have at least one landmark. It seems impossible that a context variable with no landmarks could ever improve a DBN, but because statistics can be recalculated from the last 200 times the antecedent event occurred, a variable with no landmarks can be added as the second variable of a context if the first variable was more predictive in the last 200 occurrences than before.

Algorithm 3 Update Context

Require: A DBN $r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$

let r' be a copy of r but with an empty context, i.e. let $r' = \langle \mathcal{C}' : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ where $\mathcal{C}' = \emptyset$
while context of r' is different from previous iteration **do**
  if r' is sufficiently reliable **then**
    let $r'' = \langle \mathcal{C}' \cup v : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ where $v \neq X$ is the magnitude variable minimizing $H(r'')$
  else
    let $r'' = \langle \mathcal{C}' \cup v : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ where $v \neq X$ is the magnitude variable maximizing $brel(r'')$
  end if
  if $isModelImprovement(r', r'')$ **then**
    $\mathcal{C}' \leftarrow \mathcal{C}' \cup v$
  end if
end while
if $isModelImprovement(r, r')$ **then**
  $\mathcal{C} \leftarrow \mathcal{C}'$
end if

D.2 Learning Landmarks on DBNs

Algorithm 4 gives the pseudocode for learning landmarks on DBNs. To limit landmarks during the search for landmarks over DBNs, we require that if variable v is in the antecedent event of r, then v must be a motor variable. We also require that during a single execution of Algorithm 4:

1. A DBN r can not be already improved upon by another new landmark.
2. A DBN r adds at most one landmark. If there is more than one landmark found for r, the landmark with the highest weighted gain is added.
3. There can be only one new landmark per combination of variable v and open interval q.

Algorithm 4 Finding landmarks on DBNs

let $\theta_{IG} = 0.30$ be the required information gain
for each DBN r **do**
  for each magnitude variable v **do**
    for each open interval $q \in \mathcal{Q}(v)$ **do**
      find the best cutpoint c with gain G_c^*
      if $G_c^* > \theta_{IG}$ and $G_c^* \cdot Pr(v = q) > \theta_{IG}/2$ **then**
        create the candidate landmark v^* and the potential new DBN r'
        if $isModelImprovement(r, r')$ **then**
          add v^* as a new landmark
        end if
      end if
    end for
  end for
end for

D.3 Learning Landmarks to Predict Events

Algorithm 5 gives the pseudocode for learning landmarks to predict events. We require that during an execution of Algorithm 5 that

1. Any found landmark within two buckets of an existing landmark is discarded.
2. Only one landmark per v and q combination be found. It takes the one with the highest Δ.
3. Only one landmark per event E. It takes the one with the highest Δ.

Algorithm 5 Finding landmarks on events

let $\theta_E = 0.30$ be the needed difference across buckets
for each event E on a direction of change variable **do**
  for each motor or magnitude variable v **do**
    find the bucket $[lb, ub]$ that maximizes $\Delta = Pr(\tilde{v}_{t-1} \in [lb, ub] | E(t)) - Pr(\tilde{v}_{t-1} \in [lb, ub])$
    if $\Delta > \theta_E$ **then**
      add $v^* = [lb, ub]$ as a new landmark
    end if
  end for
end for

Appendix E

Motor Babbling

When motor babbling, we want the agent to explore the space. To best explore the space, we want to maintain the same motor command for more than one timestep. Additionally, we want the values of motor variables to change at different times to avoid spurious correlations for the model learner. Therefore, for motor babbling we treat each motor variable separately. Each motor variable has a counter that counts the number of timesteps it will maintain its randomly chosen value; this counter has a randomly chosen maximum between 0 and 40 timesteps. When the counter of some motor variable hits its maximum, a new random value is chosen and a new counter maximum is chosen randomly.

When a single motor babbling command is issued during exploration, the command continues until one of the motor variables' counter reaches its maximum.

Appendix F

Performing Actions

Both algorithms either return a motor value or *success* or *fail*. An action is first called. Once it is called, it is processed on subsequent timesteps until it is terminated.

Algorithm 6 Calling an Action

Require: action $a(v, q)$, state s
if v is a motor variable **then**
  let a.motor be a continuous motor value within the range covered by q
  return a.motor
end if
let a.plan be o_r chosen as described in Chapter 6
if no plan found **then**
  return fail
end if
let a.subaction be subaction a' based on policy π_r
call a'
if a' fails **then**
  return fail
else
  a.motor $\leftarrow a'$.motor
  return a.motor
end if

Algorithm 7 Processing an Action

Require: action $a(v, q)$, state s

if v is a motor variable **then**
  return a.motor
else if exceeds resource constraints (β_r from a.plan) **then**
  return fail
else if action is completed ($v = q$) **then**
  return success
else
  if waiting for k timesteps **then**
    if waiting period over **then**
      let a.subaction be subaction a' based on policy π_r
      call a'
    else
      return a.motor
    end if
  else
    process a.subaction $= a'$
    if subaction a' returns fail or success **then**
      if v is change variable and a.subaction to achieve antecedent of a.plan and success **then**
        begin waiting for k timesteps
        return a.motor
      else
        let a.subaction be subaction a' based on policy π_r
        call a'
      end if
    else
      return a'.motor
    end if
  end if
  if a' fails **then**
    return fail
  else
    a.motor $\leftarrow a'$.motor
    return a.motor
  end if
end if

Appendix G

Decision Parameters

Table G.1 shows the decision parameters in QLAP.

Table G.1: Decision Parameters and Values in QLAP

Parameter	Value	Location	Description
θ_{SR}	0.75	Sec. 4.2.2	required reliability for DBN to be sufficiently reliable
θ_{pen}	0.05	Sec. 4.1.2	penalty for rule improvement
θ_{IG}	0.30	Sec. 4.4.1	required information gain
θ_{E}	0.30	Sec. 4.4.2	required difference for entropy landmark

Appendix H

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. The remaining variables were treated as continuous. They were normalized to the range $[0, 1]$ using the minimum and maximum values observed during a typical run of QLAP. The generalization was 0.25, and 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].

Bibliography

[Arkin, 1998] Ronald C. Arkin. *Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1998.

[Atkeson *et al.*, 1997a] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1/5):11–73, 1997.

[Atkeson *et al.*, 1997b] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1/5):75–113, 1997.

[Ayres, 1994] R.U. Ayres. *Information, entropy, and progress: a new evolutionary paradigm*. American Institute of Physics, 1994.

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

[Ballard, 1991] D. H. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.

[Baranes *et al.*, 2009] A. Baranes, P.Y. Oudeyer, and F. INRIA. R-IAC: Robust Intrinsically Motivated Exploration and Active Learning. *IEEE Transactions on Autonomous Mental Development*, 1(3):155–169, 2009.

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

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

[Bedau, 2003] M.A. Bedau. Artificial life: organization, adaptation and complexity from the bottom up. *Trends in cognitive sciences*, 7(11):505–512, 2003.

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

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94, 1999.

[Brafman and Tennenholtz, 2003] R.I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[Brooks, 1986] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Trans. on Robotics and Automation*, RA-2(1):14–23, 1986.

[Chaput, 2004] Harold Chaput. *The Constructivist Learning Architecture: A Model of Cognitive Development for Robust Autonomous Robots*. PhD thesis, University of Texas at Austin, Department of Computer Sciences, 2004. Also available as UT AI TR04-34.

[Cohen *et al.*, 1997] P. R. Cohen, M. S. Atkin, T. Oates, and C. R. Beal. Neo: Learning conceptual knowledge by sensorimotor interaction with an environment. In *Agents '97*, Marina del Rey, CA, 1997. ACM.

[Cohen *et al.*, 2002a] L. B. Cohen, H. H. Chaput, and C. H. Cashon. A constructivist model of infant cognition. *Cognitive Development*, 17:1323–1343, 2002.

[Cohen *et al.*, 2002b] P. R. Cohen, T. Oates, C. R. Beal, and N. Adams. Contentful mental states for robot baby. In *Proc. 18th National Conf. on Artificial Intelligence (AAAI-2002)*. AAAI/MIT Press, 2002.

[Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(2):142–150, 1989.

[DeCasper and Carstens, 1981] A. J. DeCasper and A. Carstens. Contingencies of stimulation: Effects of learning and emotions in neonates. *Infant Behavior and Development*, 4:19–35, 1981.

[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.

[Deisenroth and Rasmussen, 2009] M.P. Deisenroth and C.E. Rasmussen. Efficient Reinforcement Learning for Motor Control. In *10th International PhD Workshop on Systems and Control*, 2009.

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

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Digney, 1996] B.L. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In *From animals to animats 4: proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, page 363. The MIT Press, 1996.

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

[Duda *et al.*, 2000] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[Fayyad and Irani, 1992] U.M. Fayyad and K.B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8(1):87–102, 1992.

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

[Fitzpatrick *et al.*, 2003] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through action-initial steps towards artificial cognition. In *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03*, volume 3, 2003.

[Friedman and Goldszmidt, 1996] Nir Friedman and Moises Goldszmidt. Discretizing continuous attributes while learning bayesian networks. In *Int. Conf. on Machine Learning*, pages 157–165, 1996.

[Friedman *et al.*, 1998] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI98)*, pages 139–147. Citeseer, 1998.

[Friedman, 1997] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *ML*, pages 125–133. Citeseer, 1997.

[Friedman, 1998] N. Friedman. The Bayesian structural EM algorithm. In *Proc. UAI*, volume 98. Citeseer, 1998.

[Fritzke, 1995] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[Gergely and Watson, 1999] G. Gergely and J.S. Watson. Early socio-emotional development: Contingency perception and the social-biofeedback model. *Early social cognition: Understanding others in the first months of life*, pages 101–136, 1999.

[Ghavamzadeh and Mahadevan, 2001] M. Ghavamzadeh and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[Gibson, 1988] EJ Gibson. Exploratory behavior in the development of perceiving, acting, and the acquiring of knowledge. *Annual review of psychology*, 39(1):1–42, 1988.

[Gold and Scassellati, 2006] K. Gold and B. Scassellati. Learning acceptable windows of contingency. *Connection Science*, 18(2):217–228, 2006.

[Goodman *et al.*, 2007] N.D. Goodman, V.K. Mansinghka, and J.B. Tenenbaum. Learning grounded causal models. In *Proceedings of the Twenty-Ninth Annual Conference of the Cognitive Science Society*. Citeseer, 2007.

[Harris, 1995] M.B. Harris. *Basic statistics for behavioral science research*. Allyn & Bacon, 1995.

[Heckerman, 1995] D. Heckerman. A Tutorial on Learning Bayesian Networks. *Technical Report MSR-TR-95-06*, 1995.

[Hengst, 2002] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250, 2002.

[Hester and Stone, 2009] T. Hester and P. Stone. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 717–724. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[Hill, 1990] W.F. Hill. *Learning: a survey of psychological interpretations*. HarperCollins Publishers, 1990.

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

[Johnson, 1987] Mark Johnson. *The body in the mind: The bodily basis of meaning, imagination, and reason*. University of Chicago Press, Chicago, Illinois, USA, 1987.

[Jong and Stone, 2005] N.K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. *Proc. Joint Conf. on Artificial Intelligence*, pages 752–757, 2005.

[Jong and Stone, 2008] Nicholas K. Jong and Peter Stone. Hierarchical model-based reinforcement learning: Rmax + MAXQ. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, July 2008.

[Jonsson and Barto, 2001] A. Jonsson and A.G. Barto. Automated state abstraction for options using the U-tree algorithm. *Advances in neural information processing systems*, pages 1054–1060, 2001.

[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.

[Jonsson and Barto, 2007] A. Jonsson and A. Barto. Active learning of dynamic bayesian networks in markov decision processes. *Lecture Notes in Artificial Intelligence: Abstraction, Reformulation, and Approximation - SARA*, pages 273–284, 2007.

[Jordan and Rumelhart, 1992] M.I. Jordan and D.E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[Keogh *et al.*, 2001] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings IEEE International Conference on Data Mining, 2001. ICDM 2001*, pages 289–296, 2001.

[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.

[Knox and Stone, 2010] W.B. Knox and P. Stone. Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. 2010.

[Konidaris and Barto, 2010] G. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, 2010.

[Kuffner Jr and Lavalle, 2000] J.J. Kuffner Jr and S.M. Lavalle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robot. Autom.(ICRA*, pages 995–1001, 2000.

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

[Kuipers, 2000] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(1-2):191–233, 2000.

[Lakoff and Johnson, 1980] George Lakoff and Mark Johnson. *Metaphors We Live By*. University of Chicago Press, Chicago, 1980.

[Lenat and Brown, 1984] D. B. Lenat and J. S. Brown. Why AM and EURISKO appear to work. *Artificial Intelligence*, 23(3):269–294, 1984.

[Mandler, 2004a] Jean Mandler. *The Foundations of Mind, Origins of Conceptual Thought*. Oxford University Press, New York, New York, USA, 2004.

[Mandler, 2004b] Jean Mandler. A synopsis of the foundations of mind: Origins of conceptual thought. *Developmental Science*, 7(5):499–505, 2004.

[Marjanovic *et al.*, 1996] MJ Marjanovic, B. Scassellati, and MM Williamson. Self-taught visually guided pointing for a humanoid robot. In *From Animals to Animats 4: Proc. Fourth Int l Conf. Simulation of Adaptive Behavior*, pages 35–44, 1996.

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

[McCallum, 1996] A.K. McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1996.

[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.

[Metta and Fitzpatrick, 2003] G. Metta and P. Fitzpatrick. Early integration of vision and manipulation. *Adaptive Behavior*, 11(2):109–128, 2003.

[Miller *et al.*, 1960] G. A. Miller, E. Galanter, and K. H. Pribram. *Plans and the Structure of Behavior*. Holt, Rinehart and Winston, 1960.

[Modayil and Kuipers, 2007] J. Modayil and B. Kuipers. Autonomous development of a grounded object ontology by a learning robot. In *Proc. 22nd Conf. on Artificial Intelligence (AAAI-2007)*, 2007.

[Moore and Atkeson, 1995] A.W. Moore and C.G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, 1995.

[Munos and Moore, 1999] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, 1999.

[Needham *et al.*, 2002] A. Needham, T. Barrett, and K. Peterman. A pick-me-up for infants' exploratory skills: Early simulated experiences reaching for objects using 'sticky mittens' enhances young infants' object exploration skills. *Infant Behavior and Development*, 25(3):279–295, 2002.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[Nolfi and Floreano, 2002] S. Nolfi and D. Floreano. Synthesis of autonomous robots through evolution. *Trends in Cognitive Sciences*, 6(1):31–37, 2002.

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

[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.

[Payne and Isaacs, 2007] V. Gregory Payne and Larry D. Isaacs. *Human Motor Development: A Lifespan Approach*. McGraw-Hill Humanities/Social Sciences/Languages, 2007.

[Pazis and Lagoudakis, 2009] J. Pazis and M.G. Lagoudakis. Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 793–800. ACM, 2009.

[Pearl, 2000] Judea Pearl. *Causality: Modeling, Reasoning, and Inference*. Cambridge University Press, Cambridge, 2000.

[Piaget, 1952] Jean Piaget. *The Origins of Intelligence in Children*. Norton, New York, 1952.

[Pierce and Kuipers, 1997] D. M. Pierce and B. J. Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–227, 1997.

[Price and Boutilier, 2003] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19(1):569–629, 2003.

[Provost *et al.*, 2007] Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen. Self-organizing distinctive state abstraction using options. In *Proc. of the 7th Int. Conf. on Epigenetic Robotics*, volume 7, 2007.

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

[Puterman, 1994] M.L. Puterman. *Markov Decision Problems*. Wiley, New York, 1994.

[Quinlan, 1993] J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

[Rasmussen, 2006] C.E. Rasmussen. Gaussian processes in machine learning. *Advanced Lectures on Machine Learning*, pages 63–71, 2006.

[Reynolds, 2000] S.I. Reynolds. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 783–790. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000.

[Saxena *et al.*, 2007] A. Saxena, J. Driemeyer, J. Kearns, and A.Y. Ng. Robotic grasping of novel objects. *Advances in neural information processing systems*, 19:1209, 2007.

[Schmidhuber, 1991] J. Schmidhuber. Curious model-building control systems. In *Proc. Int. Joint Conf. on Neural Networks*, volume 2, pages 1458–1463, 1991.

[Schmill *et al.*, 1998] M.D. Schmill, M.T. Rosenstein, P.R. Cohen, and P. Utgoff. Learning what is relevant to the effects of actions for a mobile robot. *Proceedings of the second international conference on Autonomous agents*, pages 247–253, 1998.

[Schmill *et al.*, 2000] M.D. Schmill, T. Oates, and P.R. Cohen. Learning Planning Operators in Real-World, Partially Observable Environments. *Artificial Intelligence*, pages 246–253, 2000.

[Schmill, 2002] M.D. Schmill. *Learning the Structure of Activities for a Mobile Robot*. PhD thesis, University of Massachusetts Amherst, 2002.

[Shannon, 1948] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[Shen, 1994] Wei-Min Shen. *Autonomous Learning from the Environment.* W. H. Freeman and Company, 1994.

[Sherstov and Stone, 2005] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and I. Saitta, editors, *SARA 2005*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag, Berlin, 2005.

[Simsek and Barto, 2004] O. Simsek and A.G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *ICML*, pages 751–758, 2004.

[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.

[Sinapov and Stoytchev, 2007] J. Sinapov and A. Stoytchev. Learning and generalization of behavior-grounded tool affordances. In *Proc. of the Int. Conf. on Development and Learning*, 2007.

[Singh *et al.*, 2005] S. Singh, A.G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems*, 17:1281–1288, 2005.

[Skinner, 1961] B.F. Skinner. *Cumulative record.* Appleton-Century-Crofts New York, 1961.

[Smith, 2004] R. Smith. *Open dynamics engine v 0.5 user guide.* http://ode.org/ode-latest-userguide.pdf., 2004.

[Stober and Kuipers, 2008] J. Stober and B. Kuipers. From pixels to policies: A bootstrapping agent. In *Proc. of the Int. Conf. on Development and Learning*, 2008.

[Stout *et al.*, 2005] Andrew Stout, George Konidaris, and Andrew Barto. Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning. In *Proceedings of AAAI Symposium on Developmental Robotics*, 2005.

[Stoytchev, 2005a] Alexander Stoytchev. Behavior-grounded representation of tool affordances. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3071–3076, 2005.

[Stoytchev, 2005b] Alexander Stoytchev. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, 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.

[Sun and Scassellati, 2005] G. Sun and B. Scassellati. A fast and efficient model for learning to reach. *International Journal of Humanoid Robotics*, 2(4):391–414, 2005.

[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.

[Uther and Veloso, 2003] W. Uther and M. Veloso. Adversarial reinforcement learning. Technical Report CMU-CS-03-107, Carnegie Mellon University, 2003.

[Vigorito and Barto, 2008] Christopher M. Vigorito and Andrew G. Barto. Autonomous hierarchical skill acquisition in factored mdps. In *Yale Workshop on Adaptive and Learning Systems*, New Haven, Connecticut, 2008.

[Vijayakumar and Schaal, 2000] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, pages 288–293, 2000.

[Vijayakumar *et al.*, 2005] S. Vijayakumar, A. D'souza, and S. Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005.

[Watson, 2001] J. S. Watson. Contingency perception and misperception in infancy: Some potential implications for attachment. *Bulletin of the Menninger Clinic*, 65:296–320, 2001.

[Whiteson and Stone, 2006] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[Whiteson *et al.*, 2005] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving soccer keepaway players through task decomposition. *Machine Learning*, 59(1):5–30, 2005.

[Zettlemoyer *et al.*, 2005] Luke S. Zettlemoyer, Hanna Pasula, and Leslie Pack Kaelbling. Learning planning rules in noisy stochastic worlds. In *AAAI*, pages 911–918, 2005.