Behavioral Hierarchy: Exploration and Representation

Andrew G. Barto[†] George Konidaris[‡]

Christopher Vigorito[†]

[†]Autonomous Learning Laboratory
University of Massachusetts Amherst

[‡]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Abstract

Behavioral modules are units of behavior providing reusable building blocks that can be composed sequentially and hierarchically to generate extensive ranges of behavior. Hierarchies of behavioral modules facilitate learning complex skills and planning at multiple levels of abstraction and enable agents to incrementally improve their competence for facing new challenges that arise over extended periods of time. This chapter focusses on two features of behavioral hierarchy that appear to be less well recognized: its influence on exploratory behavior and the opportunity it affords to reduce the representational challenges of planning and learning in large, complex domains. Four computational examples are described that use methods of hierarchical reinforcement learning to illustrate the influence of behavioral hierarchy on exploration and representation. Beyond illustrating these features, the examples provide support for the central role of behavioral hierarchy in development and learning for both artificial and natural agents.

1 Introduction

Many complex systems found in nature or that humans have designed are organized hierarchically from components—modules—that have some degree of independence. Herbert Simon called such systems “nearly decomposable” and suggested that complex systems tend to take this form because it enhances evolvability due to module stability (Simon 1996, 2005). The subject of modularity has attracted significant attention in psychology, neuroscience, evolutionary biology, computer science, and philosophy (Callebaut and Rasskin-Gutman 2005).

Although it is widely accepted that modularity is critically important for understanding and constructing complex systems, no single concept of modularity adequately covers all the cases. Callebaut (2005), for instance, distinguishes modularity of structure from modularity of process, further pointing out significant differences between developmental, evolutionary, neural, and cognitive modularity. This chapter focuses on a type of modularity in which modules are viewed as "building blocks" that can be combined and connected to span large spaces of structured entities, whether they are physical entities or behaviors.

Specifically, this chapter focuses on *behavioral hierarchy.* A behavioral hierarchy is composed of behavioral modules: units of behavior by which an agent interacts with its environment. Hierarchical structure results when a module is itself composed of other modules. Human behavior has long been recognized to exhibit hierarchical structure, with tasks being comprised of subtask sequences, which in turn are built out of simpler actions. Behavioral hierarchy has been an enduring theme in psychology, from the advent of cognitive views (e.g., Lashley 1951, Miller et al. 1960, Newell et al. 1963) to more recent models and cognitive architectures (e.g., Anderson 2004, Botvinick and Plaut 2004, Langley and Rogers 2004, Langley et al. 2009, Schneider and Logan 2006). Behavioral hierarchy has also been of longstanding interest in artificial intelligence (e.g., Sacerdoti 1974), control engineering (e.g., Antsaklis and Passino 1993), software design and analysis (e.g., Alur et al. 2002), and neuroscience (e.g., Botvinick et al. 2009).

In this chapter we use the framework of hierarchical reinforcement learning (HRL) to illustrate the benefits of behavioral hierarchy in addressing problems of *exploration* and *representation* that arise in designing capable learning agents. Although these benefits are well recognized by some, they are not as widely appreciated as are more obvious benefits, and yet they also lie at the heart of the utility of behavioral hierarchy. The relevance of behavioral hierarchy to exploration and representation may also help explain why we see hierarchical structure in the behavior of humans and other animals.

After discussing behavioral hierarchy and HRL, we describe four computational experiments. The first two experiments illustrate how the influence of behavioral hierarchy on exploration can greatly improve learning in structured environments (Sections 4.3 and 4.4). The third and fourth experiments respectively illustrate how behavioral hierarchy can address representational challenges by allowing the use of low-complexity function approximation methods (Section 5.1) and how an agent can select different abstractions for each behavioral module to make learning easier in problems with high-dimensional state spaces (Section 5.2). Our conclusion emphasizes the generality of the principles illustrated by these examples and their relevance to the design of agents that accumulate competence over extended time periods. The chapter ends with a discussion of prospects for future research. Material in this chapter has appeared previously in Konidaris and Barto (2009a, 2009b), Konidaris (2011), and Vigorito and Barto (2010).

2 Behavioral Hierarchy

The most widely appreciated aspect of behavioral hierarchy is that actions can make use of lower-level actions without concern for the details of their execution. This facilitates both learning complex skills and planning at multiple levels of abstraction. If an agent can construct a useful hierarchy of behavioral modules, which here we think of as *skills*, then the search space of behaviors effectively shrinks. This is because selecting between alternate higher-level skills allows the agent to take larger, more meaningful steps through the search space of behavioral strategies than does selecting between more primitive actions.

Another salient feature of behavioral modularity and hierarchy is that it facilitates the transfer of results of learning in one task to other related tasks. This has been called *transfer learning*, and is recognized to be significant for both artificial and natural agents (Taylor and Stone 2009). Rather than acquiring skills from scratch for each problem it faces, an agent can acquire "portable skills" that it can deploy when facing new problems. Illustrating this benefit of behavioral modularity is beyond the scope of this chapter, and we refer the reader to Guestrin et al. (2003), Konidaris et al. (2012a), Konidaris and Barto (2007), Liu and Stone (2006), Mehta et al. (2008), Taylor et al. (2007), and Torrey et al. (2008).

The influence of behavioral hierarchy on exploratory behavior is less well recognized. As new skills are added to an agent's behavioral repertoire, they become available as atomic behavioral modules that may be used when computing behavioral strategies and models of more complex skills. The agent's growing skill set allows it to reach increasingly many areas of its state space that were previously not easily accessible. This in turn allows for learning about more complex environmental dynamics and consequently enables further skill discovery. In this sense, behavioral hierarchy can provide a means for continual, developmental learning in which the acquisition of new skills is bootstrapped on existing structural and procedural knowledge.

Another aspect of behavioral hierarchy, and of behavioral modularity in general, is the opportunity it affords to reduce the representational challenges of planning and learning in large, complex domains. Each module can incorporate its own module-specific representation that includes what is needed for its operation while excluding information that is irrelevant to that operation. When we perform a skill like throwing a ball, for instance, we do not have to take into account the vast range of information available to us that is relevant to other skills, but not to ball throwing. This can make it feasible to plan and learn successfully in complex domains. This benefit of behavioral modularity is well recognized across artificial intelligence, including the various approaches to HRL (e.g., Barto and Mahadevan 2003, Dietterich 2000a, Parr 1998, Parr and Russell 1998, Sutton et al. 1999), but its profound importance may not be widely appreciated.

3 Hierarchical Reinforcement Learning

This chapter is concerned with behavioral hierarchies implemented and learned using computational reinforcement learning (RL, Sutton and Barto 1998). RL algorithms address the problem of how a behaving agent can learn to approximate an optimal behavioral strategy while interacting with its environment. More technically, RL is about the on-line approximation of solutions to stochastic optimal control problems, usually under conditions of incomplete knowledge of the system being controlled. By emphasizing incremental on-line algorithms instead of batch-style algorithms, RL not only relates well to the kind of learning we see in animals, it is also useful for engineering control problems where it can have some advantages over more conventional approaches (Lewis and Vrabie 2009).

RL problems consist of four elements: a set of environmental states; a set of actions available to the agent in each state; a transition function, which specifies the probability of the environment transitioning from one state to another in response to each of the agent's actions; and a reward function, which indicates the amount of reward associated with each such transition. Given these elements, the objective for learning is to discover a behavioral strategy that maximizes cumulative long-term reward. Behavioral strategies are called *policies*, which are rules, or functions, that associate with each possible state an action to be taken in that state. Policies are like stimulus-response rules of learning theories except that a state is a broader concept than a stimulus. A state characterizes relevant aspects of the learning system's environment, which includes information about the world with which the learning agent interacts as well as information about the internal status of the agent itself.

In the usual RL scenario, an agent learns how to perform a "task" specified by a given reward function—for instance, learning how to win at playing backgammon (Tesauro 1994), where a win is rewarded and a loss punished.[1] In these scenarios, there is a conflict between *exploitation* and *exploration*: in deciding which action to take, the agent has to exploit what it has already learned in order to obtain reward, and it has to behave in new ways—explore—to learn better ways of obtaining reward. RL systems have to somehow balance these objectives. However, in other scenarios the goal is to learn a predictive model of the environment, the environment's causal structure, or a collection of widely useful skills. In these scenarios, exploration is itself part of an agent's task. Scenarios like this play a role in some of the illustrations described below.

A key focus of RL researchers is the problem of scaling up RL methods so they can be effective for learning solutions to large-scale problems. Artificial Intelligence researchers address the need for large-scale planning and problem solving by introducing various forms of abstraction into problem solving and planning systems (e.g., Fikes et al. 1972, Korf 1985, Sacerdoti 1974). Abstrac-

[1] In RL, the reward signal usually handles both rewards and punishments, which are respectively represented by positive and negative values of the numerical reward signal. This abstraction is widely used despite the fact that it is at odds with the differences between appetitive and aversive systems in animals.

tion allows a system to ignore details that are irrelevant for the task at hand. For example, a macro—a sequence of operators or actions that can be invoked by name as if it were a primitive operator or action—is one of the simplest forms of abstraction. Macros form the basis of hierarchical specifications of action sequences because macros can include other macros in their definitions: a macro can "call" other macros. A macro is a type of module: a reusable component of a program.

HRL uses a generalization of a macro that is often referred to as a "temporally-abstract action," or just an "abstract action" (Sutton et al. 1999). Conventional macros are *open-loop* behavioral policies in the sense that their unfolding does not depend on information sensed during their execution. In contrast, HRL uses *closed-loop* policies as modules, meaning that the course of execution can depend on input from the agent's environment. Unlike a macro, then, which specifies a specific action sequence, an abstract action in HRL generates a sequence of actions that depends on how the actions influence the environment. Thus, in addition to specifying a single "primitive action" to execute in a given state, a policy in HRL can specify a multi-step abstract action to execute in a state, which is characterized by its own policy that can specify both primitive actions and other abstract actions. Once a temporally abstract action is initiated, execution of its policy continues until a specified termination condition is satisfied. Thus, the selection of an abstract action ultimately results in the execution of a sequence of primitive actions as the policy of each component abstract action is "expanded."

Although several approaches to HRL have been developed independently, they all use closed-loop policies as behavioral modules: the *options* formalism of Sutton, Precup, and Singh (1999), the *hierarchies of abstract machines* (HAMs) approach of Parr and Russell (Parr 1998, Parr and Russell 1998), and the MAXQ framework of Dietterich (2000a). These approaches are reviewed by Barto and Mahadevan (2003).

The illustrations in this chapter are based on the theory of options, and we use the terms option and skill interchangeably. An option consists of 1) an *option policy* that directs the agent's behavior when the option is executing, 2) an *initiation set* consisting of all the states in which the option can be initiated, and 3) a *termination condition*, which specifies the conditions under which the option terminates. In addition, a system can maintain, for each option, an *option model*, which is a probabilistic description of the effects of executing an option. As a function of an environment state where the option is initiated, it gives the probability with which the option will terminate at any other state, and it gives the total amount of reward expected over the option's execution. Option models can be learned from experience (usually only approximately) using standard methods. Option models allow algorithms to be extended to handle learning and planning at higher levels of abstraction (Sutton et al. 1999).

The question of where options come from has occupied a number of researchers. In many cases a system designer can define a collection of potentially useful options by hand based on prior knowledge about the agent's environment and tasks. To do this, the designer specifies the policy, initiation set, and termination condition for each of these "native options," which are analogous to action patterns and their triggering conditions given to an animal through evolution, such as swallowing, the fight-or-flight-or-freeze response, and many others. In other instances, it may be possible to define an option by creating only a reward function for that option, and let the agent learn a policy, together with its initiation and termination conditions, through RL.[2] This is most commonly done by identifying potentially useful option goal states and rewarding the agent for reaching them. These goal states are potential subgoals for problems the agent will face over its future.

Option goal states have been selected by a variety of methods, the most common relying on computing visit or reward statistics over individual states to identify useful subgoals (Digney 1996, McGovern and Barto 2001, Şimşek and Barto 2004, 2009). Graph-based methods (Mannor et al. 2004, Menache et al. 2002, Şimşek et al. 2005) build a state-transition graph and use its properties (e.g., local graph cuts, Şimşek et al. 2005) to identify option goals. Other methods create options that allow the agent to alter otherwise infrequently changing features of its environment (Hengst 2002, Jonsson and Barto 2006). Mugan and Kuipers (2009) developed a related system, which they called Qualitative Learner of Actions and Perception (QLAP), that creates a discrete qualitative representation of a continuous state space using "landmarks" to partition the space. Options are created to change the values of the resulting qualitative variables. A related method was presented by Bakker and Schmidhuber (2004) that relies on unsupervised clustering of low-level sensations to define subgoals. Clustering was also used to cluster subgoals to prevent the creation of multiple options that all correspond to the same underlying skill (Niekum and Barto 2011). Konidaris and colleagues (Konidaris and Barto 2009b, Konidaris et al. 2011a, 2012b) illustrated the utility of setting the goal of an option to be reaching the initiation set of an already-formed option in a process called "skill chaining." This method is used in the example described in Section 5.1 below. Still other methods extract options by exploiting commonalities in collections of policies over a single state space (Bernstein 1999, Perkins and Precup 1999, Pickett and Barto 2002, Thrun and Schwartz 1995).

Barto and colleagues (Barto et al. 2004, Singh et al. 2005) proposed that option goals can be identified through reward signals unrelated to a specific task, such as signals triggered by unexpected salient stimuli. This allows an agent to create options that have the potential to be useful for solving *many different tasks* that the agent might face in its environment over the future (see Section 4.4 below). Hart and Grupen (2011) proposed a comprehensive approach in which a reward signal identifies behavioral affordances (J.Gibson 1977) that expose possibilities for action in the environment. Mugan and Kuiper's (2009) QLAP system similarly adopts this "developmental setting" in creating options outside the context of a specific task. Reward signals that do not specify an explicit problem to solve in an environment are related to what psychologists

[2] Option reward functions have been called "pseudo reward functions" (Dietterich 2000a) to distinguish them from the reward function that defines the agent's overall task, a distinction not emphasized in this chapter.

call *intrinsic motivation*. Whereas *extrinsic motivation* means doing something because of some specific rewarding outcome, intrinsic motivation means "doing something because it is inherently interesting or enjoyable" (Ryan and Deci 2000). Intrinsic motivation leads organisms to engage in exploration, play, and other behavior driven by curiosity in the absence of externally-supplied rewards. Schmidhuber (1991a, 1991b) proposed that an RL agent could be given a kind of curiosity by being rewarded whenever it improved its environment model. Although he didn't use the term "intrinsic reward," this was the first contribution to what we now call "intrinsically-motivated RL." An extended discussion of intrinsically-motivated RL is beyond the scope of this chapter, and the reader is referred to Baldassarre and Mirolli (2012) for multiple perspectives on the subject.

Overall, the problem of how to create widely useful options—or more generally, how to create widely useful behavioral modules—is of central importance for progress in HRL, and much research remains to be done. The illustrations described in this chapter are relevant to creating useful options, but since our purpose here is to highlight the utility of behavioral modularity, we do not deeply discuss option creation algorithms.

4 Exploration in Structured Environments

Exploration is indispensable for an RL system's operation. To learn in the absence of explicit instructional information, meaning information that directly tells the agent how it should act, an agent has to try out actions to see what effect they have on reward signals and other aspects of the agent's environment. In other words, behavior has to exhibit some form of *variety*. The simplest RL systems inject randomness into their action-generation procedures so that, for example, they sometimes choose an action uniformly at random from the set of all possible actions instead of taking an action that appears to be one of the best based on what they have learned so far. But exploration does not have to be random: trying something new can be directed intelligently, the only requirement being that an exploratory action is not one of the actions currently assessed to be best for the current situation. Behavioral hierarchy provides an important means for exploring more intelligently than acting randomly because it allows exploration to take advantage of an environment's structure.

In this section we present two examples that illustrate how behavioral hierarchy is implicated in intelligent exploration in structured environments. The first example illustrates how hierarchically-organized skills can be discovered that facilitate learning an environment's structure. The second example shows how a skill hierarchy can make it possible to solve a collection of learning problems that would be essentially impossible to solve otherwise. Both examples use learning problems posed in what Vigorito and Barto (2010) called the "Light Box Environment," described next.

4.1 The Light Box Environment

Figure 1 shows the Light Box Environment in which many different problems can be posed. There is a set of twenty "lights," each of which can be on or off. For each, the agent has an action that toggles the light on or off. Thus there are twenty actions, $2^{20} \approx 1$ million states, and approximately 20 million state-action pairs.

The nine circular lights are simple toggle lights that can be turned on or off by executing their corresponding action. The triangular lights are toggled similarly, but only if certain configurations of circular lights are active, with each triangular light having a different set of dependencies. Similarly, the rectangular lights depend on certain configurations of triangular lights being active, and the diamond-shaped light depends on configurations of the rectangular lights. In this sense there is a strict hierarchy of dependencies in the structure of this environment.

Figure 2 is the causal graph of the Light Box Environment, showing the dependencies between the variables that represent the states (on or off) of the lights. To remove clutter, the dependencies of the variables on themselves are not drawn, but the state of each light obviously depends on its own value at the previous time step. With the exception of these reflexive dependencies, each link in the causal graph indicates that the parent light must be "on" in order to satisfy the dependency.[3]

The Light Box Environment is also stochastic (each action taken fails to produce its intended effect with probability 0.1), and it is even more complicated because if an action is taken to toggle a light whose dependencies are not currently satisfied, the environment's entire state is reset to all lights being off.

This environment emulates scenarios in which accurate lower-level knowledge is essential for successfully learning more complex behaviors and their environmental effects. Because of the "reset" dynamics, random action selection is extremely unlikely to successfully turn on any of the lights at the top of the hierarchy. An agent must learn and make use of specific skills in order to reach and remain in the more difficult-to-reach areas of the state space. We emphasize that the agent does not perceive any structure directly: it only senses strings of twenty bits. The structure must be discovered solely from the state transitions the agent experiences, which is a non-trivial problem.

Skills, in the form of options, discovered in the Light Box Environment may have nested policies, the relationship between two of which is shown in Figure 3. The policies are represented as trees, with internal nodes representing state

[3] More technically, the complete structure of an environment like the Light Box can be represented as a set of Dynamic Bayesian Networks (DBNs, Dean and Kanazawa 1989), one for each of the agent's actions. A DBN is a directed acyclic graph with nodes in two layers representing the environment's features at time steps t and $t+1$ respectively. A DBN also includes conditional probability tables that give the state-transition probabilities. A causal graph as shown in Figure 2 summarizes the feature dependencies by including a directed edge from one feature's node to another's if and only if there is an agent action whose DBN has an edge from that feature's node at step t to the second feature's node at step $t+1$ (Jonsson and Barto 2006).

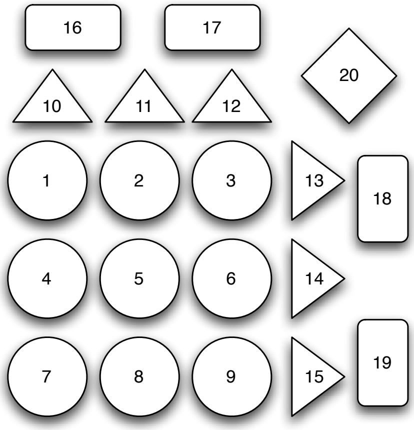

Figure 1: The Light Box Environment. ©2010 IEEE. Reprinted, with permission, from Vigorito and Barto (2010).

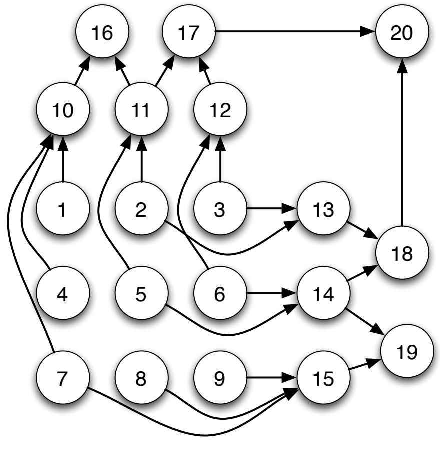

Figure 2: The Causal Graph of the Light Box Environment. ©2010 IEEE. Reprinted, with permission, from Vigorito and Barto (2010).

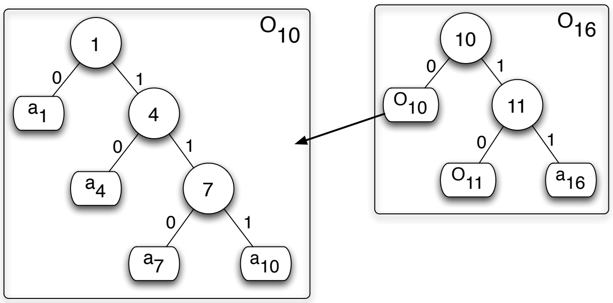

Figure 3: Examples of Option Policies in the Light Box Environment. Internal nodes represent state variables, leaves represent action (option) choices. Branches are labeled with state variable values. Notice the nested policies. ©2010 IEEE. Reprinted, with permission, from Vigorito and Barto (2010).

variables and leaves representing action choices, which may be either primitive actions or options. Branches are labeled with the possible values of their parent variables. In the example shown, the policy for the option to turn on light number 16 (O_{16}) contains at one of its leaves another option (O_{10}) to turn on light number 10, which is one of the dependencies for light number 16. This nesting of policies is a direct result of the hierarchical nature of the environment.

4.2 Exploration for Learning Structure

If the causal structure of an agent's environment is known, in the form of a causal graph like that shown in Figure 2, it can be used to create a hierarchical collection of skills that decompose a task into sub-tasks solved by these skills. Jonsson and Barto (2006) presented an algorithm called VISA (Variable Influence Structure Analysis) that creates options by analyzing the causal graph of an environment. VISA identifies context-action pairs, called *exits* (Hengst 2002), that cause one or more variables to be set to specific values when the given action is executed in the corresponding context. A context is a setting of a subset of the environment's descriptive variables to specific values from which the exit's action has the desired effect; it is like a production system's precondition (Waterman and Hayes-Roth 1978). By searching through the structured representation of the environment, VISA constructs exit options that allow the agent to reliably set collections of variables to any of their possible values. Variables for which such options have been formed are called *controllable variables*. The result of executing VISA is a hierarchy of skills that together represent a solution to the original task. VISA also takes advantage of structure in the environment to learn compact policies for options by eliminating variables on which a policies do not depend, a topic we address in detail in Section 5 below.

But it is unrealistic to assume that a causal model of the environment is available to an agent before it has to tackle specific tasks in that environment. Although discovering causal structure is a very subtle problem in general situations (Pearl 2000), the assumptions built into an environment such as the Light Box, in which actions are the complete causes of state changes, make it possible to apply a number of different approaches that have been developed to learn the structure of Bayesian networks. Many of these apply to the case where there is a training set of possible observations (assignments of values to the environment's descriptive variables, or sequences of such assignments) that can be accessed without restriction (e.g., Buntine 1991, Friedman et al. 1998, Heckerman et al. 1995). Some algorithms accelerate learning by selecting the most informative data instances from the training set through a process called active learning (e.g., Murphy 2001, Steck and Jaakkola 2002, Tong and Koller 2001). These methods effectively perform experiments by fixing subsets of the variables to specific values and sampling over the remaining variables.

In an RL setting where an agent learns while interacting with its environment, it may not be easy for the agent to access data necessary for learning the environment's causal structure. For example, a mobile robot attempting to learn the structure of a new environment cannot transport itself to any location instantaneously, making it difficult to perform all the experiments that might be useful. A collection of exit options formed by an algorithm like VISA would be valuable for allowing the agent to perform useful experiments, but such an algorithm requires knowledge of the environment's causal structure to begin with. Degris at al. (2006), Diuk et al. (2009), Jonsson and Barto (2007), Mugan and Kuipers (2009), and Strehl et al. (2007) proposed methods for learning causal graphs of certain types of dynamic environments as part of the RL process, where it is not possible to sample the process in arbitrary states but only along trajectories.

Here we focus on the structure learning method of Jonsson and Barto (2007), which is an active learning method that takes advantage of the growing structural knowledge to guide behavior so as to collect useful data for refining and extending this knowledge. It follows other structure-learning methods by estimating the probability of observing the data given a particular graphical representation of the system's causal structure, and incrementally searches the space of structures by adding and deleting the edges in the graph in an attempt to find the structure that maximizes this probability. This is not guaranteed to find the best graph (the general problem is NP-complete Chickering et al. 1995), but it usually succeeds in finding high-scoring graphs.

Details of Jonsson and Barto's (2007) active learning scheme are fairly complicated and beyond the scope of this chapter, but for current purposes the essential point is that the agent collects sample experiences so that the amount of evidence for each possible graph refinement is roughly equalized. This is accomplished by the agent choosing actions that maximize the entropies of the probability distributions the system uses for its refinement criteria. This is advantageous because having a more uniform distribution over the input values of a given refinement variable makes the evaluation of that refinement more accurate. Thus, correct refinements get discovered more quickly than they do with uniformly random action selection.

This approach does produce faster learning in some environments, but in more complex environments it can fail to discover a significant portion of the environment's structure. This is because this active learning scheme is myopic, only considering the effects of primitive actions at each step, and thus can cause the agent to become stuck in "corners" of the state space that are difficult to get out of. To partially remedy this the agent can learn skills, in the form of options, while it is exploring. Selecting skills can allow the agent to reach configurations of environmental variables that would be difficult to reach using only primitive actions. In this approach, an agent uses its current skill collection to perform experiments so as to expedite structure learning. An experiment in this scheme, like an exit, is composed of a context and an associated primitive action. Vigorito and Barto (2010) studied a method that selects actions according to Jonsson and Barto's (2007) active structure learning method, but while interacting with its environment it constructs options with hierarchically-defined policies like that shown in Figure 3 and uses them to improve active structure learning. Instead of selecting only from primitive actions, it selects from the agent's entire suite of primitive actions and available options.

Although this method takes advantage of acquired skills to explore more broadly than possible with just primitive actions, in complex tasks with large state spaces it still may fail to explore many regions of the state space, preferring to maintain uniformity in the current region. This happens because the agent still uses only local information in selecting actions: it only considers how the distributions change locally as a result of executing single actions or single options. This method is illustrated in the Light Box Environment in Section 4.3 below, where we call the agent using it the Local agent.

One way to produce a more global method is to allow the agent to use its current environment model to *plan* to reach configurations of environmental variable values that will likely yield more relevant information. Here, executing a plan means executing the plan's policy that specifies actions—primitive actions or options—in response to observations from the environment. Vigorito and Barto (2010) produced an algorithm that does this by introducing a reward function that rewards the agent for achieving a context consisting only of controllable variables. This method is also illustrated in the Light Box Environment in Section 4.3 below, where we call the agent using it the Global agent.

Using this reward function, a policy is computed (using Structured Value Iteration, Boutilier et al. 2000) to reach the rewarding context and execute the action associated with it. The policy is executed to completion before the next best experiment is computed. In this sense, the agent defines its own problems as it continues to learn and explore, becoming "bored" with things that it understands well and focusing its attention on the parts of its environment about which it is uncertain. This is a property of intrinsically-motivated curiosity as pointed out by Schmidhuber (1991a). We think of the reward function used by the Global agent as producing intrinsic reward signals.

Since the Global agent starts out with no controllable variables, initial exploration is carried out according to the local active learning scheme like that described above. However, as enough structure is discovered and certain variables become controllable via construction of low-level options, the agent can use these new skills to reliably set contexts about which it has limited experience. When options happen to be created prematurely and are malformed, their lack of utility is discovered fairly quickly by the agent when it attempts to use these options in its experimental plans and they fail repeatedly. These options are removed from the agent's skill collection until the agent performs more experiments relevant to discovering their structure, at which point they will be re-created and tested in further experiments. Once a correct option is learned, its empirical success rate will on average match its expected success rate, and the option will remain in the agent's skill collection to be used in all further experiments. In this way, structure learning is bootstrapped on existing structural and procedural knowledge.

The next section describes a study by Vigorito and Barto (2010) that compares the performance of agents using different types of exploration strategies to guide behavior while learning the structure of the Light Box Environment. Although this environment is relatively simple and was designed with an explicit hierarchical structure, it clearly illustrates the advantages that skill hierarchies can confer to exploration for learning an environment's structure.

4.3 Active Learning Learning of Light Box Structure

Three agents were compared in Vigorito and Barto's (2010) illustration of the behavior of structure-learning agents in the Light Box Environment. Agent Random always selected a random action from its set of available actions, which included options previously acquired, and executed each to completion before choosing another. Agent Local also selected from its set of primitive actions and previously acquired options, but employed the local active learning scheme described above. Agent Global, on the other hand, used the global active learning scheme of described above. This agent was able to compute plans (via Structured Value Iteration) in order to select among primitive actions and previously acquired options so as to maximize future reward. It therefore used more global information than did agent Local and so could reach more informative areas of the state space.

Since Vigorito and Barto (2010) knew the true transition structure of the Light Box, they could compare the refinements made by each agent at any given time step to the set of refinements that define the correct model. Figure 4 shows the number of correct refinements discovered by each agent as a function of the number of time steps. The learning curves presented are averages of 30 runs for each agent. Clearly the hierarchical nature of the environment made structure learning very difficult for agents that could not plan ahead in order to reach more informative areas of the state space. Both Random and Local were able to learn what is essentially the bottom layer of the hierarchy, but once this structure was discovered they continually sampled the same areas of the state space and their learning rate leveled out.

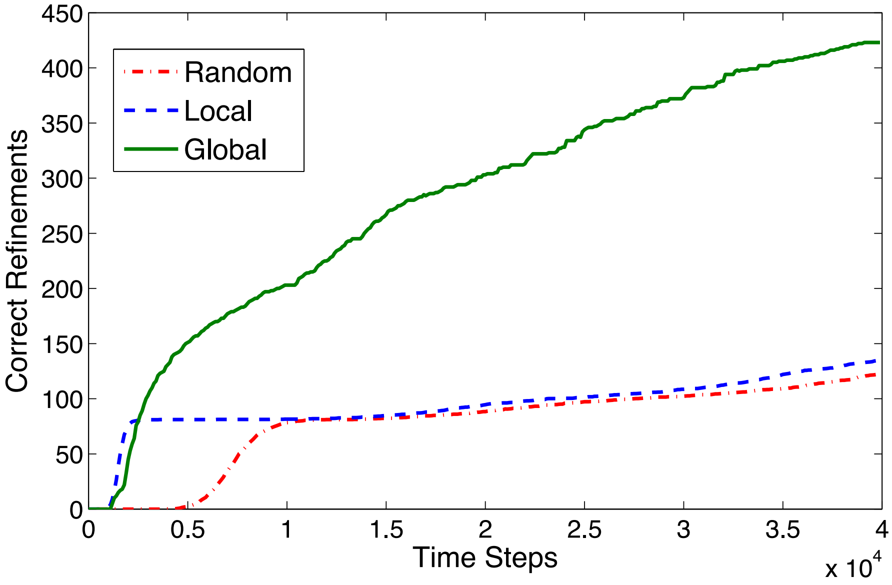

Figure 4: Structure Learning Performance for Three Exploration Policies. ©2010 IEEE. Reprinted, with permission, from Vigorito and Barto (2010).

Global, on the other hand, used the options constructed from this initial structure to plan useful experiments in its environment, allowing it to reach areas of the state space that the other agents could not reach reliably. This allowed it to uncover more of the environment's structure, which in turn allowed it to generate new skills that enabled further exploration not possible with only the previous set of options. This bootstrapping process continued until all of the domain structure had been discovered, at which point the agent possessed options to set each light to either on or off. The structured representation of the environment allowed the agent to uncover the transition dynamics without ever visiting a vast majority of the environment states, with Global reliably finding the correct structure in under 40,000 time steps.

This experiment clearly demonstrates the utility of behavioral modularity in a developmental framework. Using an initially learned set of skills as a basis for further exploration in hierarchically-complex environments like the Light Box is necessary for the discovery and learning of new, more complex skills. But the initial learning of low-level skills may not be sufficient for learning complex tasks. An agent must continue to learn new skills using those it has already discovered in order to make full use of a behavioral hierarchy. Unlike agents Random and Local, Global used those skills in a planning framework that drove it towards the more informative areas of the state space, and thus it performed much better than the others.

The learning curves for Random and Local in Figure 4 eventually flatten out, indicating that the agents reach a complexity plateau and are unable to learn beyond that point. By contrast, agent Global *is* able to learn more complex behavior, using its already acquired skills as building blocks. This kind of open-ended learning, where the agent is able to continually acquire ever more complex skills by progressively building on the skills it has already acquired, characterizes the utility of this approach in complex hierarchically-structured environments.

4.4 Skills for Multiple Tasks

Another benefit of behavioral modularity for exploration occurs when an agent has to face multiple tasks in an environment. Consider the following scenario. Suppose an agent finds itself in an environment in which it will—over the future—face a collection of tasks, where each task is specified by a different reward function. Further suppose that the environment has structure that can be exploited to accumulate reward, but that the agent does not know anything about this structure to begin with, or how it can be exploited. Assume also that at the start the agent does not know what tasks it will have to face. This scenario captures some features of the situation in which an infant finds itself, or in which an adult finds itself when confronting an environment in which it has little experience and where there is the luxury of a "developmental period" during which relatively few demands are made. What should an agent do to prepare for future challenges? Clearly, it should explore its environment and discover features of its structure that might be useful later on. Our perspective

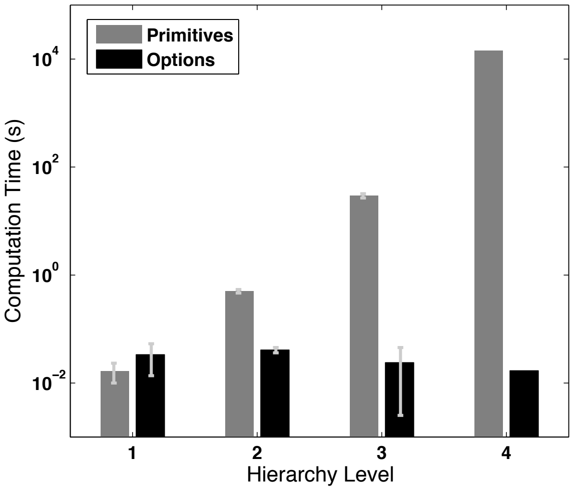

Figure 5: Policy Computation Times. Times are given for tasks at varying levels of the Light Box hierarchy for an agent with primitive actions only and for one with options plus primitives. Note the log scale. ©2010 IEEE. Reprinted, with permission, from Vigorito and Barto (2010).

is that it should not only accumulate knowledge of the environment's structure, but it should also acquire behavioral modules in the form of skills that will be on call as it faces multiple tasks over the future, thereby enabling it to learn to perform those tasks more efficiently than would be possible without those skills.

Vigorito and Barto (2010) conducted experiments to illustrate this scenario in the Light Box Environment. They compared the time it took to compute policies for various tasks (defined by different reward functions) for an agent with only primitive actions to the time taken by one with a full hierarchy of options (including primitives). For each of the twenty lights they computed a policy for a task whose reward function was 1 when that light was on and -1 otherwise. They averaged together the computation times of the tasks at each level of the Light Box hierarchy, i.e., all times for circular lights were averaged together, and similarly for triangular and rectangular lights, with only one task for the diamond light.

Results in Figure 5 show that for the lowest level of the hierarchy, where each task could be accomplished by one primitive action, the two agents took very little time to compute policies, with the options agent being slightly slower due to having a larger action set through which to search. However, once the tasks required longer sequences of actions to perform, there was a significant increase in the computation time for the primitives-only agent, and little or no increase for the options agent. The overhead of computing the options in the first place was thus compensated for once the agent had been confronted with just a few different higher-level tasks. The savings became very substantial above level 2 (note the log scale).

These results illustrate that even with as few as two or three dependencies-per-variable, the benefits of exploring with options can be dramatic. Lacking a hierarchically-structured search, the probability of the agent "stumbling upon" the solution to a higher-level task is extremely small. Indeed, in some environments, this probability is essentially zero. Caching previous experience in the form of reusable skills effectively restructures the search space, enabling the agent to experience the consequences of behavior that would otherwise be very unlikely.

5 Representational Advantages of Behavioral Modularity

In addition to facilitating searches for highly rewarding behavior by restructuring the search space, behavioral modularity presents the opportunity to greatly reduce the representation problem that is a major challenge in applying RL to problems with large or infinite state and/or action sets. RL algorithms work by learning certain functions of a problem's set of states. Depending on the type of RL algorithm, these functions can include value functions, which for each state (or possibly each state-action pair) provide a prediction of the reward the agent can expect to receive over the future after observing that state (and possibly performing a specific action in that state). Or an algorithm may require directly learning a policy, a function that for each state specifies an action that the agent should take when observing that state.

Learning in environments with large state sets requires compact representations of the needed functions when it is not feasible to explicitly store each function value. This is especially an issue when state sets are multi-dimensional since the size of the state set increases exponentially with the number of dimensions: the well-known "curse of dimensionality" (Bellman 1957). The problem is compounded when states are represented by multiple descriptive variables that can take real numbers as values, which is the case for many applications of RL. Hierarchical organizations of behavioral modules can be especially advantageous in these types of problems.

An important component of all the approaches to HRL is that each behavioral module, or skill, can operate with its own representations of states and actions, as well as its own manner of representing any needed functions. The representations on which a skill depends must be sufficient only for successfully learning and performing that skill, and thus can exclude wide ranges of information that, while perhaps relevant to other skills, are irrelevant for the skill in question. This implies that the problem of learning an individual skill can be much simpler than learning to solve a larger problem.

To be more specific about how skill-specific representations can differ from one another, consider the following common approach to representing the functions a learning agent may need to maintain. Suppose each environmental state has a "native representation" that is a vector of the values of a set of descriptive variables.[4] For example, the native representation of a state of the Light Box Environment of Section 4.1 is a tuple of 20 bits, each indicating the state of one of the lights. In the Pinball Task described in Section 5.1 below, the native representation of a state is a 4-tuple of real numbers giving the two-dimensional position of the ball and its two-dimensional velocity. These descriptive variables are usually called features, and we can call them "native features." Of course, what are considered native features are not immutable properties of an environment. In designing artificial agents, these features depend on design decisions and the nature of the problems the agent is supposed to handle; in nature, they depend on the animal's sensory repertoire and perceptual processes, both exteroceptive and interoceptive.

Given a native state representation, it is necessary to implement a function approximation method that receives as input a state's native representation and produces as output a function value, for example, an estimate of the amount of reward expected over the future after the agent visits that state. Linear function approximation is a simple and very well-studied way to do this. In this method, a state's native representation is transformed into another vector specified by a set of *basis functions*, and this new vector's components are linearly combined to produce the function's output value. Each basis function maps the space of native representations to the real numbers. The weights used to combine the components are adjusted by a learning algorithm.

The reason for using a set of basis functions is that although any function implemented by this method is a linear function of the weights—which makes it relatively easy to design and analyze learning algorithms—the overall function can be a very complex non-linear function of the native representation because the basis functions can be non-linear functions of the native representation. Further, it is usual practice to use many more basis functions than native features. For example, in continuous domains with many native features, an overall problem's value function or policy approximation may require hundreds of even thousands of basis functions to represent it. The examples described in Sections 5.1 and 5.2 below use Fourier bases consisting of varying numbers of multivariate sinusoidal basis functions (Konidaris et al. 2011b). Mahadevan (2009) provides a thorough account of the use of basis functions in RL.

Skill-specific representations can therefore differ in terms of native features, function approximation methods, basis functions, or all of these. For example, a skill may depend only on a subset of the environment's full set of native features, the rest being irrelevant to the skill. More generally, a skill may depend only on an *abstraction* of the full native representation produced by transforming the native representation in a many-to-one manner to produce a more compact description that removes some of the distinctions originally present while pre-

[4] Actions similarly have native representations, but we restrict attention to state representations to keep things simple.

serving relevant information (Dietterich 2000b, Li et al. 2006, Ravindran and Barto 2002). Abstraction is a key approach to solving high-dimensional RL problems, but it is typically difficult to find a single abstraction that applies to the entirety of a complex problem: the problem itself may simply be intrinsically high-dimensional and therefore hard to solve monolithically. Nevertheless, it may at the same time consist of several subproblems that can be captured as skills, each of which can be learned efficiently using a suitable abstraction.

Skill-specific representations can also differ in the function approximation methods they use. In the case of linear function approximation, for example, each skill may use its own set of basis functions. Alternatively, each skill might use the same type of basis functions, but can use fewer of them than would be required to provide sufficiently accurate function approximation over the environment's entire state space. This applies when skills generate activity that is concentrated on a subset of the environmental state space.

In the next section we illustrate the case in which all the options rely on the same native features, but the options are only required to achieve accuracy on subsets of the state space. Each option therefore uses a function approximator based on many fewer basis functions than would be required to achieve adequate accuracy over the entire state space. Konidaris and Barto (2009b) called these "lightweight options," in reference to the relative simplicity of their function approximation methods. In Section 5.2, we present an example in which each skill can select a different abstraction of the task's native representation.

5.1 Representing Complex Policies using a Collection of Lightweight Options

Konidaris and Barto (2009b) use a "Pinball Task" to illustrate lightweight options. A Pinball Task is a dynamic navigation task with a 4-dimensional continuous state space. Figure 6 shows an instance of a Pinball Task. Other instances have different obstructions and different goal locations.

The goal of a Pinball Task is to maneuver a small ball from its starting location into the large red hole in the upper right corner in Figure 6). The ball's movements are dynamic so its state is described by four real-valued variables: x, y, $\dot{x}$ and $\dot{y}$. Collisions with obstacles are fully elastic and cause the ball to bounce, so rather than merely avoiding obstacles the agent may choose to use them to efficiently reach the red hole. There are five primitive actions: incrementing or decrement $\dot{x}$ or $\dot{y}$ by a small amount (which incurs a reward of -5 per action), or leaving them unchanged (which incurs a reward of -1 per action); reaching the goal obtains a reward of $10,000$.

Learning policies for a Pinball Task is very difficult because its dynamic aspects and sharp discontinuities make it difficult for both control and for function approximation, and it requires very accurate control for long periods of time. Konidaris and Barto (2009b) compared Pinball agents that attempted to learn flat policies (that is, policies not hierarchically defined) with agents that used hierarchical policies. The latter agents used *skill chaining* to create skills such that from any point in the state space, the agent could execute a sequence

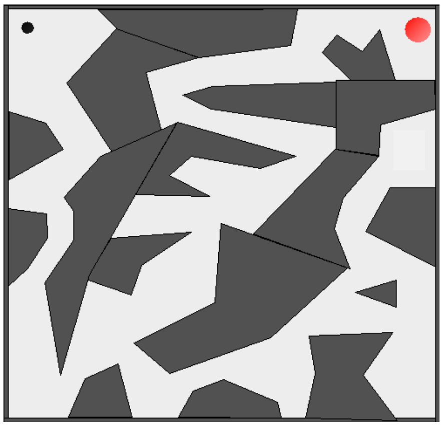

Figure 6: An Instance of a Pinball Task. The goal is to maneuver a small ball from a starting location into the large red hole in the upper right corner. One possible starting location is shown in the upper left corner.

of skills, implemented as options, to reach a given goal region of state space. This algorithm both discovers the skills and determines their initiation sets by learning the local region from which their policies consistently allow the agent to reach its goal. Skill chaining adaptively breaks the problem into a collection of subproblems whose sizes depend on the complexity of policy that they can represent. Its key mechanism is to treat the initiation sets of options already acquired as goal regions for forming new options.[5]

We omit details of skill chaining and refer the reader to Konidaris and Barto (2009b). It is related to what is known in robotics as pre-image backchaining or sequential composition (Burridge et al. 1999). In related work by Neumann et al. (2009), an agent learns to solve a complex task by sequencing motion templates. The most recent related work is by Tedrake (2010) in a model-based control setting.

The options created by skill chaining were lightweight because they used a function approximation scheme with many fewer basis functions than required to approximate the value function adequately over the entire state space. Specifically, whereas the flat policies were derived from value functions combining 1,296 basis functions for each action for the Pinball instance shown in Figure 6, the value functions of the options each combined only 256 basis functions for each

[5] In this respect, it is closely related to the algorithm used by agent Global described in Section 4.3 in which an intrinsic reward is generated for reaching a context consisting only of controllable variables, although that algorithm does not immediately extend to problems with continuous state spaces.

action.[6] Each option therefore implemented a "lightweight policy"—a much simpler policy than the overall task policy.

Figure 7 shows the performance (averaged over 100 runs) in the Pinball Task shown in Figure 6 for agents using a flat policy (without options) compared with agents employing lightweight options obtained through skill chaining, and agents starting with given (pre-learned) options. Performance is shown as an agent's return for each episode, which is the total amount of reward it received over that episode. Given the task's reward function, return is inversely related to the number of time steps required to maneuver the ball into the hole. Pre-learned options were obtained using skill chaining over 250 episodes in the same Pinball Task instance. The figure shows that the skill chaining agents performed significantly better than flat agents by 50 episodes, and obtained consistently better solutions by 250 episodes, whereas the flat agents did much worse and were less consistent. Konidaris and Barto (2009b) observed similar results for another instance of the Pinball Task.

Agents that started with pre-learned options did very well initially—with an initial episode return far greater than the average solution eventually learned by agents without options—and proceeded quickly to the same quality of solution as the agents that discovered the options themselves. This shows that it was the options acquired, and not a by-product of acquiring them, that were responsible for the increase in performance. Given this set of lightweight options, the problem became relatively easy to solve.

Figure 8 shows two sample solution trajectories from different start states accomplished by an agent performing skill chaining in the Pinball Task, with the options executed shown in different colors. The figure illustrates that this agent discovered options corresponding to simple, efficient policies covering segments of the sample trajectories.

The primary benefit of behavioral modularity in this problem is that it reduces the burden of representing the task's value function, allowing each option to focus on representing its own local value function and thereby achieving a better overall solution. Furthermore, using the strategy described here, an agent can create as many options as necessary, thereby adapting to fit the problem difficulty. Although these results illustrate a behavioral hierarchy consisting of only two levels (the level of the options and the level of the primitive actions) the benefit of lightweight options clearly extends to deeper hierarchies.

5.2 Selecting Skill-Specific Abstractions

The lightweight options described above illustrate the fact that a behavioral building block can have lower complexity than would be required for a monolithic solution to a given problem. Although the lightweight options in that example acquired different policies applicable to different parts of the problem's

[6] Konidaris and Barto (2009b) used Sarsa with linear function approximation using a 5th-order Fourier basis ($1,296$ basis functions per action). Option policy learning was done using Q-learning with a 3rd-order Fourier basis (256 basis functions per action). See Konidaris et al. (2011b) for details about using Fourier bases in RL.

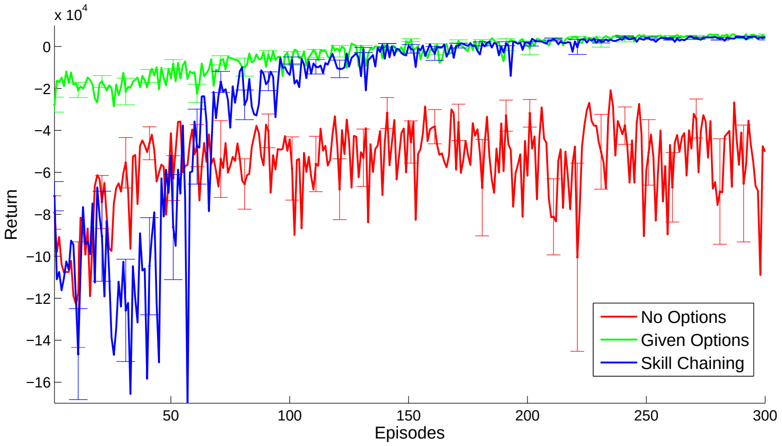

Figure 7: Performance in the Pinball Task. Graphs show performance for agents employing skill chaining, agents using given pre-learned options, and agents without options. Performance is shown as an agent's return for each episode, which is inversely related to the amount of time the agent took to complete the task. Results are averages of 100 runs.

state space, they were all built using the same, complete, set of native environmental features. It is also possible for each skill to be based on its own abstraction of the problem's state and action spaces. Here we describe an example due to Konidaris and Barto (2009a) and Konidaris (2011) in which each skill selects its own abstraction, which in this case is a subset of the environment's native features.[7] The goal is similar to that of earlier work by Jonsson and Barto (2002) in which each option implemented a separate instance of McCallum's (1996) U-Tree algorithm designed to synthesize state representations from past histories of observations and actions. The goal is also similar to that of Seijen et al. (2007) who studied a method that included special abstraction-switching actions.

In abstraction selection (Konidaris 2011, Konidaris and Barto 2009a), when an agent creates a new option it also selects an appropriate abstraction for learning that option from a library of existing abstractions. Once the agent has determined the option's termination condition—but not yet its policy or initiation set—the agent uses trajectories that reach the option's termination condition as sample trajectories to determine the appropriate abstraction for the new option. The new option policy is then defined using only the native features relevant to its selected abstraction.

Briefly, abstraction selection is treated as a model selection problem, where

[7] More technically, each abstraction is a projection of the state space onto the subspace spanned by a subset of the set of native features.

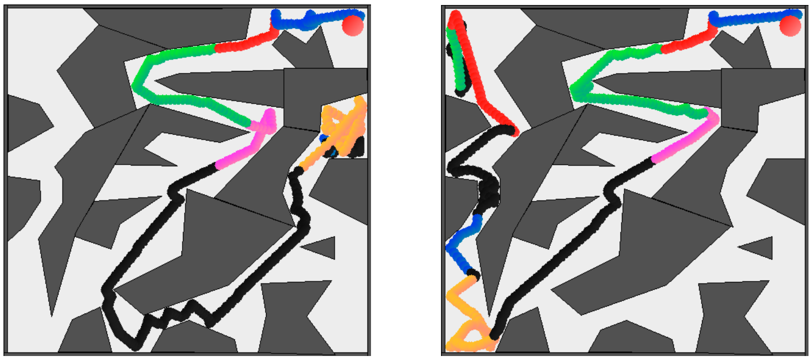

Figure 8: Sample Solution Trajectories from Different Start States in the Pinball Task. Acquired options executed along each sample trajectory are shown in different colors; primitive actions are shown in black.

the agent aims to find the highest likelihood model to fit the returns it receives along its sample trajectories. For the example described below, the class of models considered consists of linear combinations of a set of basis functions defined over the sets of native features corresponding to the abstractions in the library. Konidaris and Barto (2009a) used the Bayesian Information Criterion (BIC), a relatively simple metric that balances an abstraction's fit and its size while incorporating prior beliefs as to which abstractions might be suitable. The process returns a selection probability as output. These properties allow the agent, for example, to select an abstraction when it is only 95% sure it is the correct one. The abstraction selection algorithm runs on each abstraction in parallel and is incremental and online. It compiles information from the sample transitions into sufficient statistics and does not require them to be stored. For details, see Konidaris (2011), Konidaris and Barto (2009a).

Konidaris and Barto (2009a) illustrate abstraction selection using the Continuous Playroom, an environment where skill-specific abstractions can significantly improve performance. The environment, a continuous version of the environment of Barto et al. (2004) and Singh et al. (2005), consists of three effectors (an eye, a marker, and a hand), five objects (a red button, a green button, a light switch, a bell, a ball, and a monkey) and two environmental variables (whether the light is on, and whether the music is on). The agent is in a 1-unit by 1-unit room, and may move any of its effectors 0.05 units in one of the usual four directions. When both its eye and hand are over an object, the agent may additionally interact with the object, but only if the light is on (unless the object is the light switch). Figure 9 shows an example configuration.

Interacting with the green button switches the music on, while the red button switches the music off. The light switch toggles the light. Finally, if the agent interacts with the ball and its marker is over the bell, then the ball hits the bell. Hitting the bell frightens the monkey if the light and music are both on and

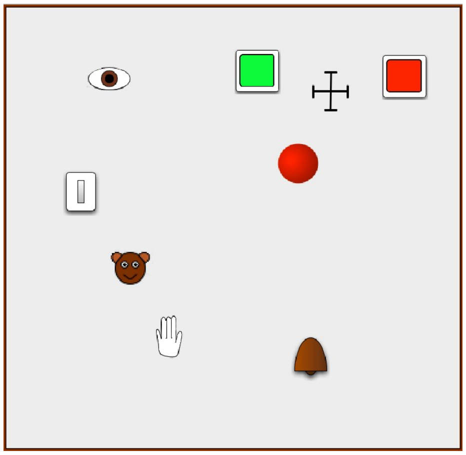

Figure 9: An Example Continuous Playroom. The agent can move its eye, hand, and marker (shown as the cross). The objects are randomly relocated at the start of each episode.

causes it to squeak, whereupon the agent receives a reward of $100,000$ and the episode ends. All other actions cause the agent to receive a reward of -1. At the beginning of each episode the objects are arranged randomly in the room so that they do not overlap.

The agent has 13 possible actions (3 effectors with 4 actions each, plus the interact action), and a full description of the Continuous Playroom requires 18 state variables: x and y pairs for three effectors and five objects (since the position of the monkey may be omitted) plus a variable each for the light and the music. Because the environment is randomly rearranged at the beginning of each episode, the agent must learn the relationships between its effectors and each object, rather than simply the absolute location for its effectors. Moreover, the settings of the light and music are crucial for decision making and must be used in conjunction with object and effector positions. Thus, for task learning the agent used 120 native state features—for each of the four settings of the lights and music it used a set of 30 features representing the difference between each combination of object and effector (Δx and Δy for each object-effector pair, so 5 objects $\times 3$ effectors $\times 2$ differences $= 30$).

The Continuous Playroom is a good example of a problem that should be easy—and is easy for humans—but is made difficult by the large number of features and the interactions between them that cannot all be included in an overall task function approximator: a 1st order Fourier basis over 120 variables that does not treat each variable as independent has 2^{120} basis functions. Thus, it is a problem in which options can greatly improve performance, but only if those options are themselves feasible to learn.

Konidaris and Barto (2009a) assumed that the agent starts with primitive actions only, and that a new option is created for moving each effector over each object when the agent first successfully does so. The task is then to efficiently learn the policies for these options using abstraction selection.

The agent was given an abstraction library consisting of 17 abstractions. Since the environment consists of objects and effectors, abstractions were included for each of the 15 object-effector pairs, with each abstraction consisting of just two features: Δx and Δy for the object-effector pair. Also included in the library was a random abstraction (two features with values selected uniformly at random over $[0, 1]$ at each step), and a null abstraction, which used all 120 native features.

For overall task learning in the Continuous Playroom, Konidaris (2011) used a 10th-order independent Fourier basis over the 120 native features ($1,320$ basis functions per action). For learning an option policy with an option-specific abstraction, a full 10th-order Fourier basis was used for each action (121 basis functions per option). For option learning without abstraction, the lights and music features were discarded and the agent used the 30 difference features and a 10th-order independent Fourier basis (330 basis functions per action). For the abstraction selection process, a 2nd-order Fourier basis was used.

Figure 10 compares the overall learning curves in the Continuous Playroom for agents that used abstraction selection with those that did not use abstractions when learning option policies. The figure also includes curves for agents that were given pre-learned optimal option policies, and agents that were immediately given the correct abstraction so that performance can be compared to the ideal case. The abstraction selection agents were constrained to only perform selection after 5 episodes.

Figure 10 shows that agents performing abstraction selection performed identically to those that did not use abstractions initially, until about 5 episodes, whereafter the agents were allowed to perform abstraction selection and their performance improved relative to agents that did not use abstractions, matching the performance of agents that were given the correct abstractions in advance by 10 episodes. The difference in performance between agents that did not use abstractions and agents that performed abstraction selection was substantial.

This example illustrates advantages of using abstractions, but it does not explain why it makes sense for an agent to select abstractions from a library instead of developing them itself from its experiences. According to Konidaris and Barto (2009a), the key advantage of abstraction selection is that it shifts the state-space representation problem out of the agent's control loop: instead of having to design a relevant abstraction for each skill as it is discovered, a library of abstractions can be provided and the agent can select a relevant one for each new skill. As the Continuous Playroom example shows, this can significantly bootstrap learning in high-dimensional state spaces, and it allows one to easily incorporate background knowledge about the problem into the agent.

However, providing a library of abstractions to the agent in advance requires both extra design effort and significant designer knowledge of the environment the agent is operating in. This immediately suggests that the abstraction library

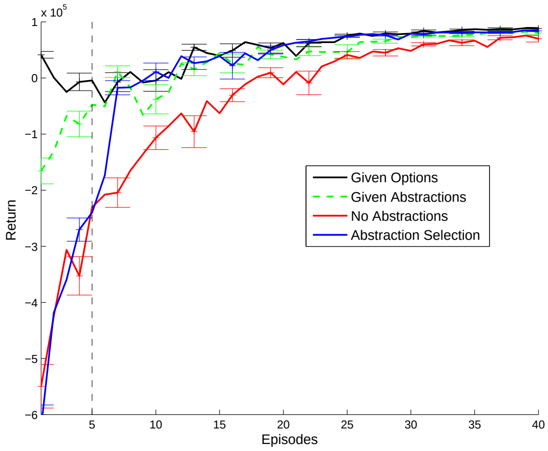

Figure 10: Learning with Abstraction Selection. Learning curves for agents given optimal option policies, agents given the correct abstraction in advance, agents using no abstractions, and agents that performed abstraction selection. The dashed line at 5 episodes indicates the first episode where abstraction selection was allowed; before this line, agents using abstraction selection learned option policies without abstractions.

should be *learned*, rather than given. Just as an agent should learn a library of applicable skills over its lifetime, so should it also learn a library of suitable abstractions over its lifetime—because an agent's abstraction library determines which skills it can learn quickly. We refer the interested reader to Konidaris (2011) for more on these issues.

6 Summary and Prospects

Modularity, in one form or another, is widely observed in both artificial and natural systems. This chapter has focused on behavioral modularity in which "units of behavior," which we call skills (or in the HRL framework, options), are reusable building blocks that can be composed to generate extensive ranges of behavior. Perhaps the most widely appreciated feature of behavioral modularity becomes apparent when hierarchical composition is possible, in which modules can be composed of other, lower-level modules. Behavioral hierarchies facilitate both learning complex skills and planning at multiple levels of abstraction. Another salient feature of behavioral modularity is that it facilitates transfer learning, in which results of learning in one task are transferred to other related tasks. In a developmental framework, this allows agents to incrementally improve their competence for facing new challenges that arise over extended time periods.

This chapter focusses on two features of behavioral hierarchy that appear to be less well recognized: its influence on exploratory behavior and the opportunity it affords to reduce the representational challenges of planning and learning in large, complex domains. Behavioral hierarchy not only allows exploration to take "bigger steps," it can make areas of a state space accessible that would be effectively inaccessible otherwise. Our example using the Light Box Environment illustrates this: without the ability to cache learned behavior into reusable skills, the agent would have very little chance of ever causing "higher-level" events to occur, and thus would have very little chance to learn about these aspects of its environment. This additionally illustrates how behavioral hierarchy can provide a means for continual, developmental learning in which the acquisition of new skills is bootstrapped on previously acquired structural and procedural knowledge.

Our other examples using the Pinball and Continuous Playroom environments, illustrate the utility of allowing each module to incorporate its own module-specific representation. Skills typically involve circumscribed aspects of an agent's environment, which allows irrelevant features to be omitted in the representations underlying the skill. Because each skill in a Pinball Task applies to a restricted subset of the problem's state space, skills can use mechanisms of lower complexity than they would need if they were more widely applicable. This was illustrated through the idea of a "lightweight option." In the Continuous Playroom, an example with a higher dimensional state space, the skill-creation mechanism selected the subset of state variables considered most relevant to learning and performing that skill. This is an example of how individual skills can take advantage of skill-specific abstractions. In this case, abstractions were selected from a pre-specified small library of abstractions, but the principle would be similar with other methods for creating and selecting abstractions.

The examples described in this chapter were developed to illustrate methods designed to scale RL to larger and more complex problems. However, we believe that key features of these examples are relevant to modularity in human and animal behavior as well. A guiding principle of all of this research has been that behavioral modules, especially when hierarchically organized, provide a powerful means for continual, developmental learning. It is plausible that the benefits of behavioral hierarchy that we have illustrated through these examples have counterparts that influenced the evolution of hierarchical modularity in humans and other animals.

Although the examples presented in this chapter are based on the options framework of Sutton et al. (1999), it would be a mistake to conclude that this framework is fully developed, that it provides an adequate account of all the issues associated with behavioral hierarchy, or that it is the only framework in which our main points could have been framed. The options framework has the advantages of resting on a strong theoretical base and affording a collection of principled algorithms, but much more development is needed to make it better suited to engineering applications and to improve its account of salient features of behavioral modularity observed in animals. We conclude by briefly describing several avenues for continued development.

1. *Option Creation.* The question of where options come from was briefly discussed in Section 3. Although many different approaches are being studied, significant challenges remain. For example, most of the methods that have been developed focus on identifying state space regions that act as useful option goals, and therefore as subgoals for higher-level tasks. These options terminate when the goal region is entered. How to identify useful option goals is far from settled. Further, many skills involve ongoing repetitive behavior, such as running, swinging, stirring, etc. These kinds of skills are not characterized by goal regions but rather by desired dynamic behavior, such as desired limit cycles. Little work has been done on creating options of this type, although much work in robotics is relevant, such as the methods used in the biped walker of Tedrake et al. (2004) and the control-basis approach of Grupen and colleagues (Hart and Grupen 2011, 2012, Huber and Grupen 1997).

2. *Parameterized Options.* One limitation of the option model of a skill is that most skills, as the term is ordinarily used, seem more flexible than options. What we may think of as a single skill, for instance, throwing a ball, would correspond to many different options whose policies would differ depending on the type of ball, its desired trajectory, etc. Because option policies are closed-loop, the behaviors that options specify are reactive to state information, which could include desired trajectory specifics. However, it might be better to model a skill as a *family* of options that are parametrically related to one another in the sense that they share a basic structure and differ by the settings of only a few parameters. For instance, the ball's speed might be a parameter for a ball-throwing skill. Initial investigations of parameterized options have been undertaken by Soni and Singh (2006) and da Silva et al. (2012).

3. *Representation and Abstraction.* It is well-known that how a problem is represented critically influences the performance of problem solving methods Amarel (1981). The problems of designing and/or learning representations are intimately related to the problem of forming useful abstractions: all representations involve abstraction in one guise or another. The forms of representation and abstraction illustrated by the examples in this chapter only touch the surface of these topics, which have long occupied researchers in many fields. There is ample opportunity for integrating the approaches to representation and abstraction taken in RL and HRL , such as those described by Mahadevan (2009) and Osentoski and Mahadevan (2010), with those developed by researchers in other areas.

4. *Many-Level Skill Hierarchies.* There is a shortage of examples in which an agent automatically constructs a many-level skill hierarchy. The GLOBAL agent in the Light Box illustration constructed a four-level skill hierarchy for turning on the highest-level light, and the agent solving a Continuous Playroom problem constructed a multi-level skill hierarchy, but

these illustrations were accomplished in relatively simple environments with explicitly-designed hierarchical structure. Other examples, such as the Pinball example described here, create two-level hierarchies (primitives actions plus options that do not invoke other options). Compelling support remains to be developed for the claim that truly open-learning can emerge from an HRL system with the ability to autonomously form deep skill hierarchies.

5. *Ensembles of Tasks: Competence and Transfer.* A theme only briefly touched upon in this chapter is the importance of considering ensembles of tasks instead of single tasks. The example in Section 4.4 illustrates how appropriate skills can make it much easier for an agent to learn any one of several tasks posed by different reward functions in the Light Box Environment. The right kinds of skills—based on the right kinds of representations and abstractions—can make an agent "competent" (White 1959) in an environment, meaning that it is able to efficiently solve many different problems that can come up in that environment. Beyond this, competence can extend from multiple tasks in a single environment to multiple tasks in multiple environments, where the environments have features in common that make it possible to transfer knowledge and skills from one to another. Developing algorithms for learning and problem solving across task ensembles continues to be a challenge. Some perspectives on this theme are discussed by Barto (2012) and Singh et al. (2010).

6. *Real-World Applications.* The illustrations described in this chapter all involved simulated environments that were artificially structured to test and demonstrate the capabilities of various HRL algorithms. Although these are considered to be "toy" environments, the problems they pose are not trivial. For example, a Pinball Task can be very difficult to solve for both humans and conventional control methods. But in many respects HRL theory has far outpaced its applications. Although there are many applications of related techniques to real-world problems (too many to attempt to cite here), there is a shortage of instances that demonstrate the full potential of HRL in important real-world tasks. This is a direction in which progress is much needed.

Acknowledgements

The authors thank Sridhar Mahadevan, Rod Grupen, and current and former members of the Autonomous Learning Laboratory who have participated in discussing behavioral hierarchy: Bruno Castro da Silva, Will Dabney, Anders Jonsson, Scott Kuindersma, Scott Niekum, Özgür Şimşek, Andrew Stout, Phil Thomas, and Pippin Wolfe. This research has benefitted from Barto's association with the European Community 7th Framework Programme (FP7/2007-2013), "Challenge 2 - Cognitive Systems, Interaction, Robotics", grant agreement No. ICT-IP-231722, project "IM-CLeVeR - Intrinsically Motivated Cumulative Learning Versatile Robots." Some of the research described here was supported by the National Science Foundation under Grant No. IIS-0733581 and by the Air Force Office of Scientific Research under grant FA9550-08-1-0418. Any opinions, findings, conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

R. Alur, M. McDougall, and Z. Yang. Exploiting behavioral hierarchy for efficient model checking. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification: 14th International Conference, Proceedings (Lecture Notes in Computer Science)*, pages 338–342. Springer, 2002.

S. Amarel. Problems of representation in heuristic problem solving: Related issues in the development of expert systems. Technical Report CBM-TR-118, Laboratory for Computer Science, Rutgers University, New Brunswick NJ, 1981.

J. R. Anderson. An integrated theory of mind. *Psychological Review*, 111: 1036–1060, 2004.

P. J. Antsaklis and K. M. Passino, editors. *An Introduction to Intelligent and Autonomous Control*. Kluwer Academic Publishers, Norwell MA, 1993.

B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, and B. Kröse, editors, *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8*, pages 438–445. IOS Press, Amsterdam, The Netherlands, 2004.

G. Baldassarre and M. Mirolli, editors. *Intrinsically motivated learning in natural and artificial systems*. Springer-Verlag, Berlin, 2012.

A. G. Barto. Intrinsic motivation and reinforcement learning. In G. Baldassarre and M. Mirolli, editors, *Intrinsically Motivated Learning in Natural and Artificial System*. Springer-Verlag, Berlin, 2012.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamcal Systems: Theory and Applications*, 13: 341–379, 2003.

A.G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In J. Triesch and T. Jebara, editors, *Proceedings of the 2004 International Conference on Development and Learning*, pages 112–119. UCSD Institute for Neural Computation, 2004.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton NJ, 1957.

D. S. Bernstein. Reusing old policies to accelerate learning on new MDPs. Technical Report Technical Report UM-CS-1999-026, Department of Computer Science, University of Massachusetts Amherst, 1999.

M. M. Botvinick and D. C. Plaut. Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review*, 111:395–429, 2004.

M. M. Botvinick, Y. Niv, and A. G. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement-learning perspective. *Cognition*, 113:262–280, 2009.

C. Boutilier, R. Dearden, and M. Goldszmdt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

W. Buntine. Theory refinement on Bayesian networks. In B. D'Ambrosio and P. Smets, editors, *UAI '91: Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann, San Francisco CA, 1991.

R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dextrous robot behaviors. *International Journal of Robotics Research*, 18:534–555, 1999.

W. Callebaut. The ubiquity of modularity. In W. Callebaut and D. Rasskin-Gutman, editors, *Modularity: Understanding the Development and Evolution of Natural Complex Systems*, pages 3–28. MIT Press, Cambridge MA, 2005.

W. Callebaut and D. Rasskin-Gutman, editors. *Modularity: Understanding the Development and Evolution of Natural Complex Systems.* MIT Press, Cambridge MA, 2005.

D. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 112–128, 1995.

B. C. da Silva, G. Konidaris, and A. G. Barto. Learning parameterized skills. In *Machine Learning, Proceedings of the 29th International Conference (ICML 2012)*. Edinburgh Scotland, 2012. To appear.

T. L. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

T. Degris, O. Sigaud, and P. H. Wuillemin. Learning the structure of factored Markov decision processes in reinforcement learning problems. In W. W. Cohen and A. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006)*, volume 148 of *ACM International Conference Proceeding Series*, pages 257–264. ACM Press, New York NY, 2006.

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

T. G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 994–1000. MIT Press, Cambridge MA, 2000b.

B. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In P. Meas, M. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 363–372. MIT Press, Cambridge MA, 1996.

C. Diuk, L. Li, and B.R. Leffler. The adaptive k-meteorologists problems and its application to structure learning and feature selection in reinforcement learning. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 249–256. ACM Press, New York NY, 2009.

R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In G. F. Cooper and S. Moral, editors, *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 139–147. Morgan Kaufmann, San Francisco, CA, 1998.

C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1003–1010. Morgan Kaufmann, San Francisco, 2003.

S. Hart and R. Grupen. Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development*, 3:216–231, 2011. Special Issue on Representations and Architectures for Cognitive Systems.

S. Hart and R. Grupen. Intrinsically motivated affordance discovery and modeling. In G. Baldassarre and M. Mirolli, editors, *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer-Verlag, Berlin, 2012.

D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20: 197–243, 1995.

B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In C. Sammut and A. G. Hoffmann, editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 243–250. Morgan Kaufmann, San Francisco CA, 2002.

M. Huber and R. A. Grupen. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22:303–315, 1997.

J. J.Gibson. The theory of affordances. In R. Shaw and J. Bransford, editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, pages 67–82. Lawrence Erlbaum, Hillsdale NJ, 1977.

A. Jonsson and A. G. Barto. Automated state abstraction for options using the U-tree algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Neural Information Processing Systems (NIPS) Conference*, pages 1054–1060. MIT Press, Cambridge MA, 2002.

A. Jonsson and A. G. Barto. Causal graph based decomposition of factored mdps. *Journal of Machine Learning Research*, 7:2259–2301, 2006.

A. Jonsson and A. G. Barto. Active learning of dynamic Bayesian networks in Markov decision processes. In I. Miguel and W. Rumi, editors, *Abstraction, Reformulation, and Approximation, 7th International Symposium, SARA 2007, Whistler, Canada, July 18-21, 2007, Proceedings*, volume 4612 of *Lecture Notes in Computer Science: Abstraction, Reformulation, and Approximation*, pages 273–284. Springer, 2007.

G. Konidaris, A. Barto, and I. Scheidwasser. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13:1333–1371, 2012a.

G. D. Konidaris. *Autonomous Robot Skill Acquisition.* PhD thesis, Computer Science, University of Massachusetts Amherst, 2011.

G.D. Konidaris and A.G. Barto. Building portable options: Skill transfer in reinforcement learning. In M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 895–900, 2007.

G.D. Konidaris and A.G. Barto. Efficient skill learning using abstraction selection. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1107–1112, 2009a.

G.D. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22, Proceedings of the 2009 Conference*, pages 1015–1023. NIPS Foundation, 2009b.

G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Autonomous skill acquisition on a mobile manipulator. In W. Burgard and D. Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011*, pages 1468–1473. AAAI Press, 2011a.

G.D. Konidaris, S. Osentoski, and P.S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In W. Burgard and D. Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011*, pages 380–385. AAAI Press, 2011b.

G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31:360–375, 2012b.

R. E. Korf. *Learning to Solve Problems by Searching for Macro-Operators*. Pitman, Boston MA, 1985.

P. Langley and S. Rogers. Cumulative learning of hierarchical skills. In J. Triesch and T. Jebara, editors, *Proceedings of the 2004 International Conference on Development and Learning*, pages 1–8. UCSD Institute for Neural Computation, 2004.

P. Langley, D. Choi, and S. Rogers. Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, 10:316–332, 2009.

K. S. Lashley. The problem of serial order in behavior. In L. A. Jeffress, editor, *Cerebral Mechanisms in Behavior: The Hixon Symposium*, pages 112–136. Wiley, New York NY, 1951.

F. L. Lewis and D. Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. In *IEEE Circuits and Systems Magazine*, volume 9, pages 32–50. IEEE Circuits and Systems Society, 2009.

L. Li, T.J. Walsh, and M.L. Littman. Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006), Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

Y. Liu and P. Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 415–420. AAAI Press, 2006.

S. Mahadevan. *Learning representation and control in Markov Decision Processes: New frontiers*, volume 1 of *Foundations and Trends in Machine Learning*. Now Publishers Inc., Hanover MA, 2009.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In C. E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004)*, volume 69 of *ACM International Conference Proceeding Series*, pages 560–567. ACM Press, New York NY, 2004.

A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996.

A. McGovern and A. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In C. E. Brodley and A. P. Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 361–368. Morgan Kaufmann, San Francisco CA, 2001.

N. Mehta, S. Natarajan, and P. Tadepalli. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73:289–312, 2008.

I. Menache, S. Mannor, and N Shimkin. Q-Cut – Dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002, 13th European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2002.

G. A. Miller, E. Galanter, and K. H. Pribram. *Plans and the Structure of Behavior*. Holt, Rinehart & Winston, New York, 1960.

J. Mugan and B. Kuipers. Autonomously learning an action hierarchy using a learned qualitative state representation. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1175–1180, 2009.

K. Murphy. Active learning of causal Bayes net structure. Technical report, Computer Science Division, University of California, Berkeley CA, March 2001.

G. Neumann, W. Maass, and J. Peters. Learning complex motions by sequencing simpler motion templates. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 753–760. ACM Press, New York, NY, 2009.

A. Newell, J. C. Shaw, and H. A. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York NY, 1963.

S. Niekum and A. G. Barto. Clustering via Dirichlet process mixture models for portable skill discovery. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24 (NIPS)*, pages 1818–1826. 2011.

S. Osentoski and S. Mahadevan. Basis function construction for hierarchical reinforcement learning. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 747–754. International Foundation for Autonomous Agents and MultiAgent Systems (IFAAMAS), 2010.

R. Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California, Berkeley CA, 1998.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, pages 1043–1049. MIT Press, Cambridge MA, 1998.

J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, Cambridge UK, 2000.

T. J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, University of Massachusetts Amherst, 1999.

M. Pickett and A. G. Barto. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In C. Sammut and A. Hoffmann, editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 506–513. Morgan Kaufmann, San Francisco CA, 2002.

B. Ravindran and A. G. Barto. Model minimization in hierarchical reinforcement learning. In S. Koenig and R. C. Holte, editors, *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2002.

R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25:54–67, 2000.

E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J.-A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (Complex Adaptive Systems)*, pages 222–227. MIT Press, Cambridge MA, 1991a.

J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität München, Arcisstr. 21, 800 München 2, Germany, 1991b.

D. W. Schneider and G. D. Logan. Hierarchical control of cognitive processes: Switching tasks in sequences. *Journal of Experimental Psychology: General*, 135:623–640, 2006.

H. A. Simon. *The Sciences of the Artificial, 3rd Edition.* MIT Press, Cambridge MA, 1996.

H. A. Simon. The structure of complexity in an evolving world: The role of near decomposability. In W. Callebaut and D. Rasskin-Gutman, editors, *Modularity: Understanding the Development and Evolution of Natural Complex Systems*, pages ix–xiii. MIT Press, Cambridge MA, 2005.

Ö. Şimşek and A.G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In C. E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004)*, volume 69 of *ACM International Conference Proceeding Series*, pages 751–758. ACM Press, New York NY, 2004.

Ö. Şimşek and A.G. Barto. Skill characterization based on betweenness. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, pages 1497–1504. Curran Associates, Inc., Red Hook NY, 2009.

Ö. Şimşek, A. P. Wolfe, and A.G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In L. De Raedt and S. Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*, volume 119 of *ACM International Conference Proceeding Series*, pages 816–823. ACM Press, New York NY, 2005.

S. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17: Proceedings of the 2004 Conference*, pages 1281–1288. MIT Press, Cambridge MA, 2005.

S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2:70–82, 2010. Special issue on Active Learning and Intrinsically Motivated Exploration in Robots: Advances and Challenges.

V. Soni and S. Singh. Reinforcement learning of hierarchical skills on the Sony Aibo robot. In L. Smith, O. Sporns, C. Yu, M. Gasser, C. Breazeal, G. Deak, and J. Weng, editors, *Fifth International Conference on Development and Learning (ICDL)*. Bloomington IN, 2006.

H. Steck and T. Jaakkola. Unsupervised active learning in large domains. In A. Darwiche and N. Friedman, editors, *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence*, pages 469–476. Morgan Kaufmann, San Francisco CA, 2002.

A. L. Strehl, C. Diuk, and M. L. Littman. Efficient structure learning in factored-state MDPs. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 2007.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA, 1998.

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

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8: 2125–2167, 2007.

R. Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. In J. Trinkle, Y. Matsuoka, and J. A. Castellanos, editors, *Robotics: Science and Systems V: Proceedings of the Fifth Annual Robotics: Science and Systems Conference*, pages 17–24. MIT Press, Cambridge MA, 2010.

R. Tedrake, T. W. Zhang, and H. S. Seung. Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2849–2854. Sendai, Japan, 2004.

G. J. Tesauro. TD–gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.

S. B. Thrun and A. Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7: Proceedings of the 1994 Conference*, pages 385–392. MIT Press, Cambridge MA, 1995.

S. Tong and D. Koller. Active learning for structure in Bayesian networks. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 863–869. Morgan Kaufmann, San Francisco CA, 2001.

L. Torrey, J. Shavlik, J. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In H. Blockeel, J. Ramon, J. Shavlik, and P. Tadepalli, editors, *Inductive Logic Programming 17th International Conference, ILP 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2008.

H. van Seijen, S. Whiteson, and L. Kester. Switching between representations in reinforcement learning. In R. Babuska and F. C. A. Groen, editors, *Interactive Collaborative Information Systems*, volume 281 of *Studies in Computational Intelligence*, pages 65–84. Springer, 2007.

C. Vigorito and A. G. Barto. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development*, 2:83–90, 2010. Special issue on Active Learning and Intrinsically Motivated Exploration in Robots: Advances and Challenges.

D. A. Waterman and F. Hayes-Roth. *Pattern-Directed Inference Systems.* Academic Press, New York NY, 1978.

R. W. White. Motivation reconsidered: The concept of competence. *Psychological Review*, 66:297–333, 1959.