Intrinsically Motivated Hierarchical Skill Learning in Structured Environments

Christopher M. Vigorito and Andrew G. Barto, *Fellow, IEEE*

***Abstract*—We present a framework for intrinsically motivated developmental learning of abstract skill hierarchies by reinforcement learning agents in structured environments. Long-term learning of skill hierarchies can drastically improve an agent's efficiency in solving ensembles of related tasks in a complex domain. In structured domains composed of many features, understanding the causal relationships between actions and their effects on different features of the environment can greatly facilitate skill learning. Using Bayesian network structure-learning techniques and structured dynamic programming algorithms, we show that reinforcement learning agents can learn incrementally and autonomously both the causal structure of their environment and a hierarchy of skills that exploit this structure. Furthermore, we present a novel active learning scheme that employs intrinsic motivation to maximize the efficiency with which this structure is learned. As new structure is acquired using an agent's current set of skills, more complex skills are learned, which in turn allow the agent to discover more structure, and so on. This bootstrapping property makes our approach a developmental learning process that results in steadily increasing domain knowledge and behavioral complexity as an agent continues to explore its environment.**

***Index Terms*—reinforcement learning, planning, options, structure learning, active learning, intrinsic motivation**

I. Introduction

DESIGNING artificial agents that behave robustly on ensembles of related tasks is a challenging open problem in machine learning. One way to approach this problem is to specify a developmental learning framework that motivates agents to incrementally learn a hierarchical set of abstract behaviors, or skills, through interaction and experimentation with their environment. These skills can then be used as modular solutions to commonly encountered sub-problems, increasing the efficiency with which agents can solve novel tasks. The work presented here details our specification of such a framework for reinforcement learning (RL) agents. While there has been much research in RL focusing on efficient learning of optimal behavior policies for single sequential decision tasks in a given domain [1], the body of literature applying RL to ensembles of related tasks is considerably smaller.

One important component of learning systems designed for solving ensembles of tasks efficiently is a mechanism for representational abstraction. If policies and models of the long-term effects of each of an agent's skills can be represented solely in terms of relevant subsets of environmental variables, each skill may be applied in multiple contexts, including novel ones, that differ along irrelevant dimensions with little or no relearning necessary [2]. When the number of relevant variables in such abstract representations is much smaller than the total number of environmental variables, the amount of experience and computation needed to find good policies is often greatly reduced.

Hierarchy is also crucial in such systems, allowing more abstract skills to make use of lower-level skills as atomic actions without concern for the details of their execution. This facilitates both learning of complex skills and planning at multiple levels of abstraction. If an agent can construct a useful hierarchy of abstract skills in a given domain, then the search space of policies for similar tasks within that environment effectively shrinks. This is because selecting between alternate abstract actions allows the agent to take larger, more meaningful steps through the search space of policies than does selecting between more primitive actions [3].

Acquiring models of the effects of actions in structured environments can facilitate learning of skill hierarchies in complex domains as well [4]. In the framework presented below, an agent incrementally accumulates knowledge of the dynamical structure of its environment as it explores. Using this structural knowledge, the agent generates both abstract skills that reliably change specific aspects of its environment, and compact models representing the long-term effects of those skills. Because of their structured representations, these skills and models can be computed much more efficiently than can their unstructured counterparts.

As new skills are added to an agent's behavioral repertoire in our framework, they become available as

C. Vigorito and A. Barto are with the Department of Computer Science, University of Massachusetts, Amherst, MA, 01003 USA e-mail: ({vigorito,barto}@cs.umass.edu)

atomic behavioral modules that may be used when computing policies 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, our framework provides a mechanism for continual, developmental learning in which the acquisition of new skills is bootstrapped on existing structural and procedural knowledge. This is a key feature of our approach and a novel contribution to the RL community.

While there are many possible exploration strategies for such a framework, we focus here on a novel active learning scheme aimed at maximizing the rate at which an agent learns its environment's dynamics and, consequently, learns appropriate skills for controlling it. Active learning refers to methods of exploration that seek out the most informative data available when choosing the next training example for a given model [5]. In particular, we extend recent work on intrinsically motivated RL [6], [7], which can provide a mechanism for active learning in sequential decision problems. Agents in our framework are intrinsically motivated to generate plans, or "experiments" that take them to areas of their environment for which their models are inaccurate and from which they stand to gain the most information.

Since we are focused on strategies for learning hierarchies of skills that can be applied over ensembles of tasks, agents in our experiments are not posed with specific tasks during their developmental period, which means that there is no *extrinsic* reward the agents seek to maximize, as in standard RL problems, but instead they seek to maximize only *intrinsic* reward. We show that our novel, intrinsically motivated, bootstrapped methods of model- and skill-learning result in dramatic increases in the rate of knowledge and skill acquisition over previous active-learning methods that do not use existing skills to explore.

The following section describes our formalism for this framework and presents relevant background material and previous work in this area. In particular, we make the assumption that an agent's environment can be modeled as a Markov Decision Process (MDP), more specifically a factored MDP. We use incremental Bayesian network learning techniques [8] to accumulate structural knowledge of the environment. Given this knowledge, we employ structured dynamic programming methods [9], [10] to compute abstract, closed-loop control policies in the form of options (the formalization of skills that we adopt) and their corresponding models. Section III presents our bootstrapped active learning approach, in which these skills are added incrementally to an agent's skill set as they are discovered, and subsequently used in plans that guide the agent to under-explored areas of its environment. We present results in a large, factored domain that illustrates the advantages of our approach in Section IV, and discuss related work in Section V. Finally, we conclude with a discussion of important issues and future work in Section VI.

II. Background and Previous Work

A. Markov Decision Processes

A finite Markov decision process (MDP) is a tuple $\langle S, A, P, R\rangle$ in which S is a finite set of states, A is a finite set of actions, P is a one-step transition model that specifies a probability distribution over successor states given a current state and action, and R is a one-step expected reward model that determines the real-valued reward an agent receives for taking a given action in a given state. An MDP is assumed to satisfy the Markov property, which guarantees that the one-step models R and P are sufficient for predicting the distribution of rewards and successor states any number of time steps in the future given a current state and sequence of actions.

When the task of an RL agent is formulated as an MDP, the goal of the agent is to learn a policy $\pi : S \rightarrow A$ that mapping states to actions that maximize its expected sum of future rewards, also called expected return. It is often assumed that the transition and reward models are unavailable to the agent. When this is the case, a policy can be learned through estimation of an action-value function $Q^\pi : S \times A \rightarrow \Re$, which maps state-action pairs $(s, a) \in S \times A$ to real values representing the expected return for executing action a in state s and from then on following policy π. If $Q^\pi = Q^*$, where Q^* denotes the optimal action-value function for the MDP, then the agent can act optimally by selecting actions in each state that maximize Q^π.

When the transition dynamics of the environment are known or estimated from experience, model-based RL can be employed to expedite value function learning in the sense of requiring less experience for Q^π to converge to Q^* [11] . If the reward function is also known, dynamic programming techniques such as value iteration can be used to compute an optimal value function and corresponding policy directly [1]. However, even when model-based methods are used in this way to improve data efficiency, tabular representations of value functions and policies (i.e., those with one entry per state or state-action pair) become infeasible to learn or compute efficiently in large MDPs.

For this reason much work has focused on approximation techniques that allow for both generalization of

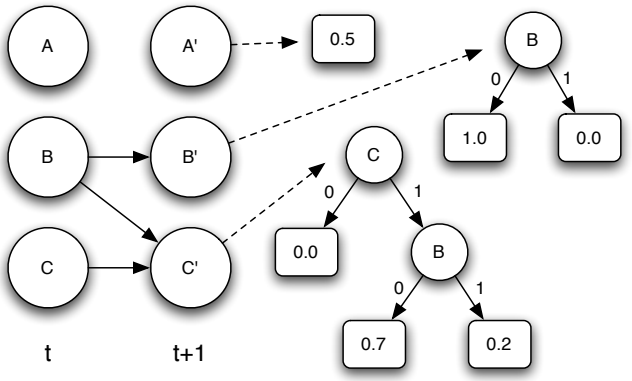


Fig. 1. A simple DBN for a given action with corresponding conditional probability trees.

value between similar states and compact representations of value functions [1]. One class of these methods is appropriate when the MDP can be represented in factored form, affording the potential for certain dimensions of the MDP to be irrelevant when predicting the effects of actions on other dimensions. In these cases, this structure can be exploited to learn or compute compact representations of value functions and policies efficiently [9].

B. Factored MDPs

A factored MDP (FMDP) is an MDP in which the state set is defined as the Cartesian product of the domains of a finite set of random variables $\{S_1, \ldots, S_n\} = \mathbf{S}$. While the variables in an FMDP can be either discrete or continuous, we restrict our attention to the discrete case such that each $S_i \in \mathbf{S}$ takes on one of finitely many values in $\mathcal{D}(S_i)$, the domain of S_i. States in factored MDPs are thus represented as vectors of assignments of specific values to the variables in $\mathbf{S}$. As the number of variables in an FMDP increases linearly, the number of states increases exponentially (a problem known as the curse of dimensionality [12]). However, if the transition dynamics of the FMDP contains relatively sparse inter-variable dependencies, it is possible to exploit this structure to reduce the effect this exponential growth has on computing optimal policies.

FMDPs can be represented as a set of Dynamic Bayesian Networks (DBN) [13], one for each action. A DBN in this case is a two-layer directed acyclic graph with nodes in layers one and two representing the variables of the FMDP at times t and $t+1$, respectively (Figure 1). Edges represent dependencies between variables given an action. We make the common assumption that there are no synchronic arcs in the DBN, meaning that variables within the same layer do not influence each other. The transition model for a given DBN can often be represented compactly as a set of conditional probability trees (CPTs), one for each variable S_i, each of which contains internal nodes corresponding to the parents of S_i and leaves containing probability distributions over $\mathcal{D}(S_i)$ at time $t+1$. Figure 1 shows a simple arbitrary DBN (for some action a) consisting of three binary variables and their corresponding CPTs, with the probability that $S_i = 1$ displayed at the leaves. We call the path from the root of a CPT to a given leaf the *context* for that leaf.

When the transition and reward models of an FMDP are known, one can use Structured Value Iteration (SVI) [9] to compute value functions and policies that exploit domain structure to represent these functions compactly. It has been shown that SVI can be much more computationally efficient, both in time and space, than the flat version of value iteration when solving FMDPs with sparse inter-variable dependencies. We assume that this type of sparse-dependency structure is present in our agents' environments, as many interesting real-world domains are so structured. For very large FMDPs, however, even this approach does not scale well in general. This is because the decision-theoretic regression approach taken by SVI regresses value functions through primitive actions, which have very short-term effects. If one could regress through longer sequences of actions in one step using temporally abstract models of long-term behaviors, then computational efficiency could be greatly improved. This requires a formalization of skills in MDPs, which we discuss next.

C. Hierarchical Reinforcement Learning

The options framework is a formalism for temporal abstraction in RL that details how to learn and use closed-loop control policies for temporally extended actions in MDPs [3]. An option is defined as a tuple $\langle I, \pi, \beta \rangle$, where $I \subseteq S$ is a set of states over which the option is defined (the initiation set), π is the policy of the option, defined over I, and $\beta : S \rightarrow [0, 1]$ is a termination condition function that gives the probability of the option terminating in a given state.

Options can also be understood as sub-MDPs embedded within a (possibly) larger MDP, and so all of the machinery associated with learning MDPs also applies to learning options. Thus, models for the transition and reward functions of an option can be learned as well. Algorithms for learning the policy, reward model, and transition model of an option from experience are given in [3]. The advantage of having access to the transition

and reward models of an option is that the option can be treated as an atomic action in planning or model-based RL methods. Additionally, since options can call other options in their policies, agents can construct deeply-nested policies with multiple levels of behavioral abstraction, leading to increased efficiency in both learning and planning as the hierarchy deepens.

While much attention has been devoted to learning options in MDPs, most of these approaches use the same state representation for every option, leading to temporal abstraction but not state abstraction. Less research has focused on learning options in FMDPs, where it is possible for different options to have different representations. The following section discusses the relevant work involved in constructing options in FMDPs, each with its own state abstraction.

D. Hierarchical Decomposition of Factored MDPs

Jonsson and Barto [10] present a framework for option discovery and learning in FMDPs. The VISA algorithm discovers options by analyzing the causal graph of a domain, which is constructed from the dependencies exhibited in the DBNs that define the FMDP. There is an edge from S_i to S_j in the causal graph if there exists an edge from S_i^t to S_j^{t+1} in the DBN model for any action. The algorithm identifies (in the causal graph) context-action pairs, called exits, that cause one or more variables to change value when the given action is executed in the corresponding context. By searching through the CPTs that define the DBNs of an FMDP, exit options are then constructed to reliably reach this context from any state and execute the appropriate action. The agent's overall task is then decomposed into sub-tasks solved by these options. VISA takes advantage of structure in the domain to learn compact policies for options efficiently by ignoring irrelevant variables.

Another feature of the framework is a method for computing compact option models from a given DBN model. The models are compact in that they take the same form as the models of primitive actions (DBNs) and represent with CPTs the expected probability distributions over the domains of the variables of the FMDP that would result from executing the option in a given state. Having option models in this form allows their use in planning as atomic actions, as mentioned above. This also means that one can use SVI to efficiently compute new option policies in terms of existing options. The VISA algorithm and option model construction techniques described here require knowledge of the transition structure of the environment. It is thus interesting to ask whether one can learn this structure online from experience. This is the subject of the following section.

Before moving on, however, it is worth pointing out that this scheme for hierarchical decomposition produces solutions to FMDPs that are *recursively optimal*, but not necessarily *hierarchically optimal*. This distinction, made by Dietterich [14], refers to the fact that a given hierarchically-decomposed solution to an MDP may be suboptimal even though each of the solutions to the subtasks into which the full solution was decomposed is optimal—this is referred to as recursive optimality. If the decomposed solution is also in fact optimal, then the solution is said to be hierarchically optimal.

E. Incremental DBN Structure-Learning

Recall that we model an agent's environment as a set of DBNs, each of which consists of a directed acyclic graph representing the dependencies between state variables (conditioned on an action) and its corresponding CPTs, one for each variable of the FMDP. The problem of Bayesian network structure-learning is to find the network $B = \langle G, \theta \rangle$ that best fits a data set $\mathbf{D}$, where G in our case represents the graphical structure of a DBN and θ represents the corresponding CPTs. To learn this structure incrementally, we take the approach given in [8], described next. Alternative approaches and justification for our choice are both discussed in Section V.

To simplify the description, we first introduce some notation building upon that in section II-B. Let S_i^t and S_i^{t+1} denote the value of variable $S_i \in \mathbf{S}$ at times t and $t+1$, respectively, and let $f_{\mathbf{X}}$, $\mathbf{X} \subseteq \mathbf{S}$, be a projection such that if $\mathbf{s}$ is an assignment to $\mathbf{S}$, then $f_{\mathbf{X}}(\mathbf{s})$ is $\mathbf{s}$'s assignment to $\mathbf{X}$. For example, if $\mathbf{X} = \{A, C\} \subseteq \mathbf{S} = \{A, B, C, D\}$, and $\mathbf{s} = \{A = 0, B = 1, C = 1, D = 1\}$, then $f_{\mathbf{X}}(\mathbf{s}) = \{A = 0, C = 1\}$. We thus denote the projection of an assignment $\mathbf{s}$ to $\mathbf{S}$ onto the parents of a variable S_i as $f_{\mathbf{Pa}(S_i)}(\mathbf{s})$, where the set of parents $\mathbf{Pa}(S_i)$ of S_i is determined by the structure of G. Data points in our framework will take the form of assignment tuples, $\langle \mathbf{s}^t, a^t, \mathbf{s}^{t+1} \rangle$, denoting the agent's state at times t and $t+1$, and the action, a^t, selected at time t which determines to which DBN the data should be applied.

One way to find the best network B for a given data set $\mathbf{D}$ is to compute the posterior probability distribution $P(B|\mathbf{D})$ over a set of networks and choose the one that maximizes this distribution. It is not feasible to compute this distribution directly, but there are approximation techniques that have been shown to perform well. It follows from Bayes theorem that $P(B|\mathbf{D}) \propto P(\mathbf{D}|B)P(B)$. One approximation technique, known as the Bayesian Information Criterion (BIC), makes the

approximation

$$\log[P(\mathbf{D}|B)P(B)] \approx L(\mathbf{D}|B) - \frac{|\theta|}{2}\log|\mathbf{D}|,$$

where $L(\mathbf{D}|B)$ is the log-likelihood of the data given the network. When all data values are observable, as we assume, this likelihood can be decomposed as

$$L(\mathbf{D}|B) = \sum_i \sum_j \sum_k N_{ijk} \log \theta_{ijk},$$

where N_{ijk} is the number of data points $x \in \mathbf{D}$ such that $f_{\mathbf{Pa}(S_i^{t+1})}(\mathbf{s}^t) = j$ and $f_{\{S_i^{t+1}\}}(\mathbf{s}^{t+1}) = k$, and $\theta_{ijk} = P(S_i^{t+1} = k|\mathbf{Pa}(S_i^{t+1}) = j)$. This quantity is maximized for $\theta_{ijk} = N_{ijk}/\sum_k N_{ijk}$. Although finding the network with the best BIC score is known to be NP-complete [15], the score decomposes into a sum of terms for each variable S_i and each value of j and k that only changes locally when edges between variables are added or deleted. Thus we can incrementally add or delete edges greedily to find high-scoring (though possibly sub-optimal) networks.

To do this, we maintain at each leaf of each CPT a set of data points which are distributed, one at each time step, to the appropriate leaves of each tree according to the assignment given by $\mathbf{s}^t$ in each data point $\langle \mathbf{s}^t, \mathbf{s}^{t+1} \rangle$. Each time a new data point is added to a leaf, we compute the BIC score of the data at the leaf and the scores associated with each possible refinement of that leaf. A *refinement* of a leaf l is a split of l on some variable S_i^t, resulting in a new child leaf for each value of S_i^t, to which the data instances of l are distributed accordingly.

To compute the BIC score for each possible refinement of a leaf, we maintain a *distribution vector*, $\mathbf{M}$, for each potential split variable S_i^t, each entry, $\mathbf{M}_k$, of which contains the number of data instances that would assign the value k to S_i^t. Note that this is not a distribution over *outcome* values (i.e., S_i^{t+1}), but rather *input* values (i.e., S_i^t), and that the frequencies in $\mathbf{M}$ can be normalized to sum to 1 and thus be represented as a probability distribution. This will be important for applying active learning techniques, as described in the following section.

Thus, when the algorithm evaluates a refinement over S_i^t, $\mathbf{M}$ determines how the data instances at the current leaf will be distributed to the new leaves. When evaluating a refinement at leaf l, if the sum of the BIC scores associated with the potential new leaves is greater than the current BIC score of l, then the refinement is kept. Refinements of a leaf on a variable S_j are not considered if S_j is already on the path from the root of the tree to the leaf. We only consider refinements at leaves that have collected at least k samples, where k is an integer design parameter.

This approach greedily adds edges to the DBN models of the environment according to the BIC metric in an incremental fashion. Occasionally an incorrect refinement is made. In contrast to [8], at each time step, for each non-leaf node, we perform a Chi-Squared test of significance between the distribution over the domain values of the current refinement (split) variable and the distribution that would result from eliminating the refinement. If the significance of this difference drops below a certain value (0.995 in all of our experiments) we remove the refinement by pruning the tree at that node and place all of the data from that subtree into the newly formed leaf node.

F. Active Structure Learning

Although a random policy could be used to collect the data necessary for structure-learning, it is interesting to consider exploration policies that attempt to maximize the rate at which environmental structure is learned. A learning algorithm in which the learning agent chooses training examples to maximize its information gain at each step is referred to as an *active learning* algorithm. One such algorithm for learning the structure of the transition model of an FMDP was presented in [8]. We build upon this algorithm in our work, and so discuss its behavior here. Some alternatives are discussed in Section V.

The basic idea of the algorithm is to have the agent collect samples to refine its model in a way that maximizes the entropy of each distribution vector in the model. Recall that each distribution vector is a histogram of samples that can be represented as a probability distribution over input values of a given potential refinement variable at a given leaf node. Maximizing the entropy of one of these distributions is equivalent to making the distribution as uniform as possible. 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 random action selection, which will tend not to produce these uniform distributions due to asymmetries in the difficulty of obtaining samples of differing input values; asymmetries that result from the dynamics of the environment.

An agent employing this algorithm thus chooses a primitive action at every step in order to maximize the sum of the changes in entropies of its model's distribution vectors. It does this by taking the current

state of the environment and checking to see, for each action, to which leaf the resulting sample will map for each CPT associated with that action, should the action be executed. The associated change in entropy of each distribution vector at each of those leaves is calculated for each action and the action with the largest total change in entropy is selected with probability $1 - \epsilon$, where $0 < \epsilon < 1$ is an exploration parameter. Otherwise a random action is selected. Note that the agent is maximizing the entropies of *input* distributions, not output distributions. That is, the outcome of a given action, which is obviously not known before the action is executed, is irrelevant to where the data sample will be placed in a given CPT. This is determined exclusively by the current state. It is for this reason that the agent can accurately compute and optimize the potential change in this quantity.

While this approach does produce faster learning in some domains, in more complex domains the approach still fails to discover a significant portion of the environmental structure [8]. This is because the active learning scheme used 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 without proper planning. This can be remedied if we allow the agent to learn options, which can then be used by a planning algorithm to have the agent reach configurations of domain variables that will yield more relevant information about the environment. Additionally, we will need a mechanism for motivating the agent to take purposeful, multi-step trajectories through its environment. Recent work in intrinsically motivated RL affords us one possibility for such a mechanism, and we discuss the relevant work in this area in the following section.

G. Intrinsically Motivated Reinforcement Learning

Early work on incorporating the concept of intrinsic motivation into artificial RL agents focused exclusively on efficient learning of world models in sequential decision problems, and were not specifically concerned with skill learning. These approaches provide intrinsic reward to agents proportional to errors in the predictions of their world model, leading the agent to areas of the environment which are unpredictable, thereby focusing learning on those areas so as to reduce that unpredictability [6]. In stochastic environments, however, this causes the agent to become "obsessed" with inherently unpredictable regions, since they provide high reward indefinitely. Thus, methods that reward agents for *progress* in improving model quality were proposed, causing agents to become "bored" with such inherently unpredictable areas (as well as predictable ones), since they afford no learning progress [16], [17]. These methods, however, were still largely focused on model learning, and not skill acquisition.

Barto et al. [7] were the first to suggest intrinsic motivation as a method for driving the accumulation of hierarchical sets of skills, and proposed an intrinsic reward that encouraged agents to develop skills that reliably cause certain (designer-specified) salient events to occur. Simsek and Barto [18] generalized this somewhat and presented an algorithm that rewards the agent for improvements in the value function of a given task (or option), which they show can speed up learning of that value function by focusing exploration on areas where learning will have the most influence. All of these methods, however, are employed in the MDP framework, and as such cannot take advantage of any potential state abstraction that might be afforded by a factored representation. Our approach, presented in the following section, will take the principles of existing work with intrinsically motivated model-learning and extend them to the case of structured environments. Our focus, however, will be on the use of these models for acquisition of abstract skills applicable across many tasks, and not simply accurate model-learning for its own sake.

III. Intrinsically Motivated Learning of Skill Hierarchies in FMDPs

Our approach to developmental acquisition of skill hierarchies in FMDPs will make use of and extend much of the work discussed in the preceding section. Our contributions consist of a framework for incremental construction of options, each with its own state abstraction, based on currently available structural knowledge, and an intrinsic reward mechanism for active structure-learning that uses these options by having the agent plan to reach the most informative areas of its environment. One can think of this mechanism as causing the agent to execute the best "experiments" it can, given its behavioral expertise and its current knowledge about the dynamics of its environment.

We also emphasize again here that the "task" of an agent in our framework is to learn a hierarchy of abstract skills in the absence of a specific, user-defined task. As a consequence, our approach does not make direct use of the standard extrinsic reward normally associated with RL problems. Of course the purpose of constructing a hierarchy of skills in a given domain is to improve learning performance on specific tasks that will involve extrinsic reward, but these rewards are not used to develop skills in this framework.

Agents behave exclusively to maximize intrinsic reward using the same computational mechanisms that are generally used to maximize extrinsic reward. The primary difference between the two in this context is that the intrinsic reward is a function of the agent's current state of knowledge, and thus the reward function is continually changing as the agent continues to learn. 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.

While the notion of balancing the maximization of intrinsic and extrinsic rewards is an interesting open problem closely related to the exploration/exploitation tradeoff, we do not address this issue in this work. In Section V, however, we cite some recent work in this area by other researchers. We next describe our method for incremental option construction and then explain our intrinsic reward mechanism and active learning scheme.

A. Caching Options

The framework outlined in [10] proposes to learn the full structure of an FMDP given a specified reward function and only then use the VISA algorithm to decompose the task into sub-problems solved by exit options. To extend this approach to the case in which we are interested, where there is no single specified task, we would like an agent to accumulate structural knowledge as it explores its environment and cache options for reaching various subgoals as enough structure becomes available to do so. For many options, this will occur long before the full structure of the environment is discovered. Indeed the point of our approach is that incrementally constructing options before the full structure is discovered will increase the probability of an agent being able to reach areas of the state space that would otherwise be quite difficult to reach, thereby enabling the agent to learn about the structural properties of those areas.

To do this we must monitor changes in the structure of an agent's model and, each time the structure is changed, evaluate the resulting model to decide whether a new option may be constructed. We maintain a set, $\mathcal{C}$, initially empty, of what we term *controllable* variables. These are variables for which the agent possesses options to set to each of its possible values. Every time a new refinement of a leaf in the CPT for variable S_i in the DBN for action a is made, if $S_i \notin \mathcal{C}$ we check the causal graph of the domain to see if each of its ancestors is controllable. This is to make sure that we can reliably reach the context given by the branch along which the new refinement has been made. If this is true, and the value of S_i is possibly changed by executing a in the branch's context, we construct an option (and its associated transition and reward models) to reach that context and execute action a. If the new option, coupled with all existing options, results in the agent's ability to set S_i to each of its possible values, we add S_i to $\mathcal{C}$.

As in [10], we define the reward function that specifies the subtask an option is constructed to solve (known as a pseudo-reward function) to be -1 for every state in which the option's exit context is not satisfied, and 0 when the context is satisfied. Because we are defining the pseudo-reward function for the options we create, we can use SVI to compute their policies, as distinguished from [10], in which unstructured RL algorithms were used to learn the option policies from experience. Additionally, the SVI algorithm now has at its disposal the agent's current set of options (and their corresponding models) for setting each variable in $\mathcal{C}$ to each of its values, which will in general lead to faster computation of new option policies for the reasons described earlier. Of course this means we must compute the transition and reward models for each option as they are constructed, which we do using the algorithm given in [10]. In contrast to [10], however, in which only primitive actions were used in option policies, the options constructed by our agents may contain recursive calls to other options in the agents' skill sets.

There is one more issue we have not addressed that must be considered when deciding whether to construct an option. It may be the case that a refinement is made in the CPT for S_i, and all ancestors of S_i are controllable, but the CPT is either incomplete or incorrect in some way. If we were to construct an option at this point, it would likely be incorrect (both its policy and its model). Thus we need a way to decide whether the correct CPT has been learned for S_i under action a. If the environment is deterministic, then once the entropy of the distribution at every leaf of the CPT has reached zero, no more refinements can be made and we know the correct structure has been discovered. This is a very strong assumption though, and applies only to less interesting domains.

When the environment is stochastic, our choice of structure-learning algorithm prevents us from being able to distinguish incomplete structural knowledge from inherent stochasticity, since greedy methods like it are not always guaranteed to find the correct structure. We discuss this disadvantage in Section V. Rather than attempting to make this distinction, however, agents in our framework construct options in the absence of knowledge about structural correctness, and instead monitor

the utility of their current set of options, abandoning those whose empirical success rates do not match their expected success rates.

More formally, each time the structure of a DBN is modified by the structure-learning algorithm, if the latest refinement results in a context-action pair that alters the value of some variable, an option to set that variable to the new value is created and added to the agent's set of options, $\mathcal{O}$. Each option in $\mathcal{O}$ is assigned a success rate, σ, initially equal to the expected success rate, σ^*, of the option. The expected success rate is obtained from the leaf of the CPT corresponding to the option's exit context in the DBN corresponding to the option's exit action, and is equal to the probability that the variable the option is intended to change will take on its intended value when the exit action is executed in the exit context. Every time an option is executed, it is allowed to run to completion for a maximum of M time steps. If within that number of steps the option's context is reached, its exit action is executed, and its objective is achieved (i.e., its associated variable is set to its desired value), then the execution is considered successful. Otherwise, the execution is considered unsuccessful.

After the k^{th} execution of option o, o's success rate, σ_o, is updated according to

$$\sigma_o \leftarrow \sigma_o + \frac{1}{k}(\delta - \sigma_o), \tag{1}$$

where δ is 1 if the option was successful, and 0 otherwise, so that σ_o always reflects the average empirical success rate of o. If at any time after at least N executions of o, σ_o drops below o's expected success rate, σ_o^*, by more than a factor η, the option is removed from $\mathcal{O}$, along with any options that reference o in their policies. The agent then continues to explore with its remaining skill set, the process of discovering new structure, constructing options, and testing their utility continuing until all variables are controllable. Should the agent reach that point, it then has a set of options it can use to efficiently compute a recursively optimal solution to a wide array of potential tasks in its domain via the SVI algorithm. With this machinery in place for incrementally adding options as enough structure becomes available to do so, we next describe our method for employing intrinsic motivation to maximize the rate at which DBN structure is learned.

B. Intrinsically Motivated Structure-Learning

In our approach, an agent uses its current skill set to perform "experiments" in its environment so as to expedite structure-learning. An experiment in our scheme, like an exit, is composed of a context and an associated primitive action. Similar to [8], and as described in Section II-F, we seek to find the best experiments to perform by calculating potential changes in distribution vector entropies at CPT leaves and picking the experiment that results in the largest change. Rather than only looking at leaves whose contexts are satisfied by the current state, however, we can also consider leaves whose contexts consist exclusively of controllable variables, since the agent possesses options to reliably set those variables to any of their values. Additionally, we can check to see which settings of the controllable variables that are not part of the leaf's context will yield the highest gain at that leaf.

For each leaf of each CPT in the agent's DBN model whose context consists only of controllable variables, we compute the total change in entropies, Δ, of the distribution vectors at that leaf that would result from taking the leaf's associated action, in the same way as described in [8]. The best experiment is then chosen to be the context-action pair associated with the largest Δ. An intrinsic reward function is then created that is 1 if the experiment's context is satisfied, and -1 otherwise. Using this reward function, a policy is computed using SVI to reach that context and execute the action associated with the leaf's CPT. The policy is executed to completion before the next best experiment is computed. This can always be done because we only consider leaves whose contexts are controllable.

Since the agent starts out with no controllable variables, initial exploration is carried out according to the local active learning scheme in [8]. However, as enough structure is discovered and certain variables become controllable via construction of low-level options as outlined in the previous section, the agent can use those new skills to reliably set contexts for which it has limited or uneven samples at the leaves of its CPTs. 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 those options in its experimental plans and they fail repeatedly. These options will be removed from the agent's skill set 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 set to be used in all further experiments.

In this way, structure-learning in our framework is bootstrapped on existing structural and procedural knowledge. For domains with hierarchical structure in which it is not necessary to know the full structure of

```
Initialize CPTs to single-leaf trees (no refinements)
t ← 1
s^t ← initial state
loop forever:
   a^t ← selectAction(s^t);
   Execute primitive action a^t and observe s^{t+1};
   update(s^t, a^t, s^{t+1});
   s^t ← s^{t+1};
   t ← t + 1;
```

Fig. 2. Pseudocode for our algorithm. See Figures 3 and 4 for pseudocode of referenced functions.

```
selectAction(s^t)
  if any option o is currently executing
    if o has been running for less than M steps
      return next primitive action of o's policy
      (may involve nested options);
    else
      Terminate o;
      Update σ_o using Equation 1 (with δ = 0);
      if σ_o < σ*_o − η and o has been executed at
      least N times
          Remove o and its descendants from O;
          Update C accordingly;
      return selectAction(s^t);
  else
    if O = ∅ \\no options available
      return best action given by method in [8];
    else
      Compute policy π to reach best context
      given O as described in Section III-B;
      return first primitive action of π(s^t) (may
      involve nested options);
```

Fig. 3. Pseudocode for the **selectAction** function.

```
update(s^t, a^t, s^{t+1})
 for each S_i ∈ S
    Add sample ⟨s^t, f_{S_i}(s^{t+1})⟩ to leaf of tree
    for S_i in DBN for a_t according to context
    given by s^t;
    Compute BIC scores for each potential
    refinement at leaf, as in [8];
    if refinement is made at leaf
      Distribute samples to appropriate children;
      if new context causes change in value of S_i
        Create option(s) to reach context and set
        S_i to appropriate value(s) as in [10];
        Update O and C accordingly;
    for each internal node in context given by s^t
      if refinement is no longer significant (e.g.,
      according to Chi-Squared test)
        Prune tree at node and collect all samples
        into new leaf;
        Delete options and descendants (if any)
        associated with former context;
        if new context causes change in value of S_i
         Create option(s) to reach context and set
         S_i to appropriate value(s) as in [10];
        Update O and C accordingly;
```

Fig. 4. Pseudocode for the **update** function.

the domain to compute lower level skills, this approach should offer a distinct advantage over active exploration schemes that use only local information to choose actions. Before reporting the empirical evaluations of our approach that lend support to this hypothesis in the following section, we present here the pseudocode for our algorithm, shown in Figures 2-4. The pseudocode references previous work for implementation details when appropriate.

IV. Experiments

A. The Light Box Domain

We conducted experiments in a simple but large artificial domain called the Light Box (Figure 5). The domain consists of a set of twenty "lights", each of which is a binary variable with a corresponding 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 domain. Figure 6 shows the causal graph of the instance of the Light Box domain we used in our experiments, illustrating the dependencies between each of the variables. To remove clutter, the reflexive dependencies are not drawn, but each light obviously depends on its own value at the previous time step. With the exception of reflexive dependencies, each link in the causal graph indicates that the parent light must "on" in order to satisfy the dependency. Each action has a 0.9 probability of toggling its associated light as long as the

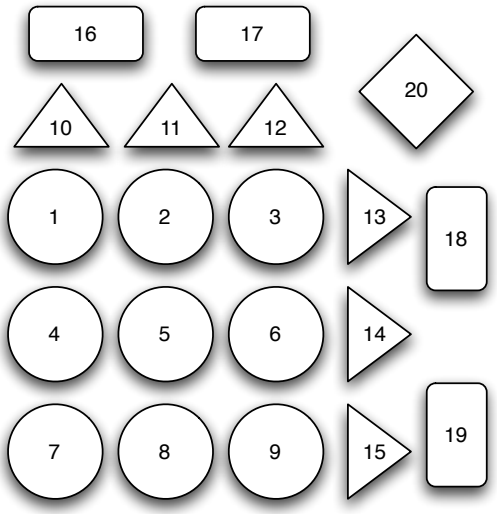


Fig. 5. A visual rendering of the Light Box domain.

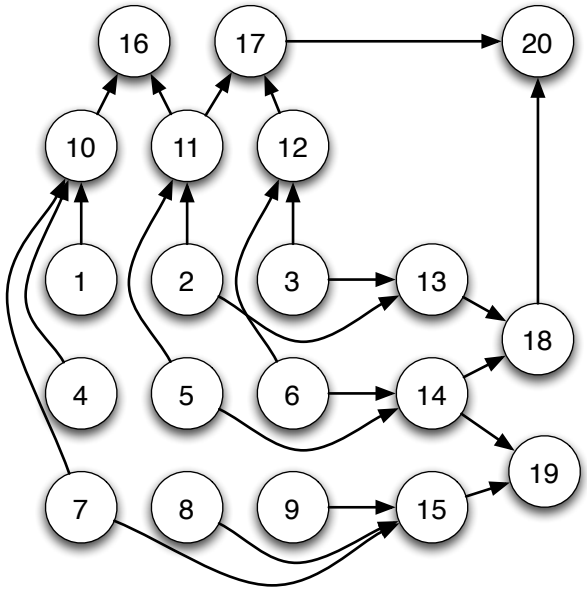


Fig. 6. The causal graph of the Light Box domain.

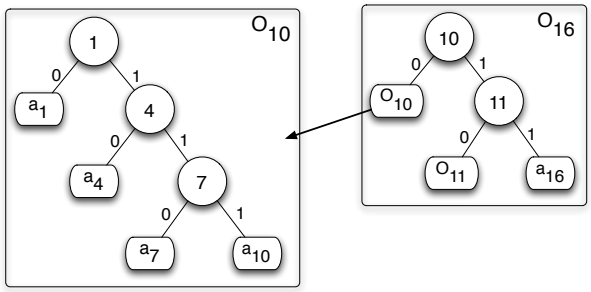


Fig. 7. Examples of compact option policies in the Light Box domain. Internal nodes represent state variables, leaves represent action (option) choices. Branches are labeled with state variable values. Notice the nested policies.

light's dependencies are satisfied, and a 0.1 probability of leaving the light unchanged. However, if an action is taken to toggle a light whose dependencies are not currently satisfied, the entire domain is reset to all lights being off.

The domain was designed to emulate scenarios in which accurate lower-level procedural knowledge is essential for successful learning of 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. Additionally, structure-learning is quite difficult using only local active learning schemes. An agent must learn and make use of low-level skills in order to be able to remain in the more difficult-to-reach areas of the state space in which it can learn higher-level skills. We also emphasize that the agent does not perceive any structure directly as may be evident in the visual rendering of the domain. Rather the agent perceives only a string of twenty bits as its state. The structure must be discovered from the state transitions the agent experiences while interacting with its environment, and thus the discovery of hierarchy is highly non-trivial.

The options that are discovered in the Light Box domain may have nested policies, the relationship between two of which is shown in Figure 7. The policies are represented as trees, with internal nodes representing state 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 domain.

B. Structure-Learning

To evaluate our proposed scheme for active structure-learning we compared the performance of agents using three different types of exploration policies to guide behavior while learning the structure of the Light Box domain. All agents executed the same structure-learning algorithm discussed above and incrementally created options according to the scheme described in the previous section, also deleting options based on their empirical success rate when appropriate. The number of samples

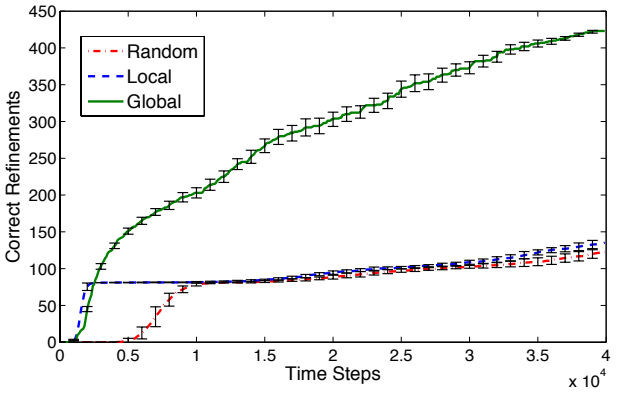


Fig. 8. Structure-learning performance for three different exploration policies.

k required to make a refinement at a leaf of a CPT was 20. The maximum number of steps M that options were allowed to execute was 50. The minimum number of executions N needed to evaluate the utility of an option was 20. And the maximum discrepancy η allowed between the empirical success rate σ and the expected success rate σ^* of an option was 0.1.

The Random agent selected a random action from the agent's set of actions (including options) and executed each one to completion before choosing another. The Local agent employed the active learning scheme presented in [8], except that when a random action was taken, the action was chosen randomly from the agent's entire set of available actions (including options) and executed to completion. The exploration parameter ϵ for the Local agent was set to 0.1. The Global agent employed our intrinsically motivated active learning scheme, which uses more global information when selecting actions and computes plans to reach more informative areas of the state space. The choices for parameter values in each agent were made via a rough search of parameter space and based on reported values in previous work when applicable. We did not notice much sensitivity in performance as a result of changing these values slightly, though of course the parameter M will in general be largely dependent on the domain, which is a limitation of this specific algorithm.

Since we had access to the true transition structure of the instance of the Light Box we used in our experiment, we could compare the refinements made by each agent at a given time step to the set of refinements that define the correct model and plot the accuracy of the model for each agent over time. Figure 8 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. Error bars show standard deviation. Clearly the hierarchical nature of the domain makes structure-learning very difficult for agents that cannot plan ahead in order to reach more informative areas of the state space. Both the Random and Local agents are able to learn what is essentially the bottom layer of the hierarchy, but once this structure is discovered they continually sample the same areas of the state space and their learning rate levels out.

The Global agent on the other hand uses the options constructed from this initial structure to perform useful experiments in its environment, allowing it to reach areas of the state space that the other agents cannot reach reliably, and thus uncover more of the domain structure. This structure is then used to generate new skills that enable further exploration not possible with only the previous set of skills. This bootstrapping process continues until all of the domain structure has been discovered, at which point the agent possesses options to set each light to either on or off. There are 423 refinements in the true DBN model of this instance of the Light Box, all of which the Global agent was able to find in each run. Note that the structured representation of the environment allows the agent to uncover the transition dynamics without even visiting a vast majority of the states in the domain, with the Global agent finding the correct structure in under 40,000 time steps reliably.

C. An Ensemble of Tasks

We also conducted experiments to illustrate the utility of computing hierarchies of skills for ensembles of tasks in large factored domains such as the Light Box. We compared the time it took to compute policies using the SVI algorithm for various tasks (i.e., 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 we computed a policy for a task whose reward function was 1 when that light was on and -1 otherwise. We 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). Experiments were run using unoptimized Java code on an Intel 2.4GHz quad core processor with 4GB of RAM. The time spent computing option policies and corresponding models for the agent with options was 21.76 seconds.

Results are shown in Figure 9. For the lowest level of the hierarchy, where the tasks can be solved by one primitive action, the two agents take very little time to

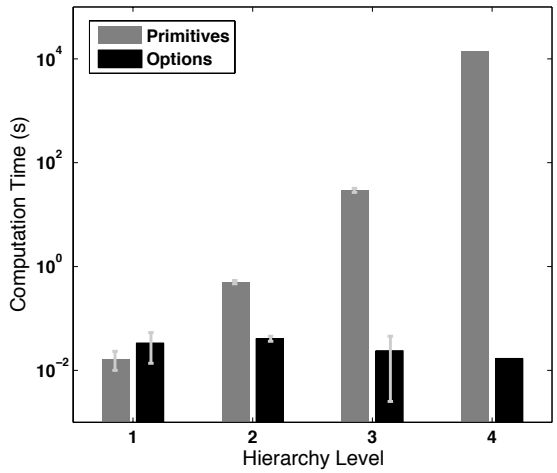


Fig. 9. Policy computation times for tasks at varying levels of the Light Box hierarchy for an agent with primitive actions only and for one with options + primitives. Note the log scale.

compute policies, with the options agent being slightly slower due to having a larger action set through which to search. However, once the tasks require longer sequences of actions to solve we see 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 is thus compensated for once the agent has been confronted with just a few different higher-level tasks. The savings become very substantial above level 2 (note the log scale). Of course the complexity of this domain can be increased by increasing the number of dependencies in its structure, but our results show that for even as few as two or three dependencies per variable the benefits of computing options are drastic.

V. Related Work

We outline here the related work on structure-learning in FMDPs and intrinsically motivated RL that has the most in common with our approach. Although the literature on Bayesian network structure-learning is substantial, many of these methods are not incremental and generally require that the data are drawn in an independent and identically distributed (i.i.d.) fashion [19]. For the case in which we are interested, namely learning the structure of a DBN online from experience with an FMDP, these methods are thus not applicable. There are, however, a few incremental methods developed recently that search for DBN structures that best fit an agent's experience with an FMDP, where the data are not drawn i.i.d. because of the temporal dependencies involved. We review these approaches here and explain their advantages and disadvantages, justifying our choice to use the approach given in [8].

Strehl et al. [20] present a unique incremental structure-learning algorithm that is actually composed of multiple instances of a "knows what it knows" (KWIK) algorithm [21]. The KWIK framework for self-aware learning is a formalism similar to the PAC formalism [22] for analyzing the class of hypotheses learnable by a given supervised learning algorithm. The structure-learning algorithm in [20] makes use of a set of KWIK algorithms to predict the value of each variable in the DBN representation of an FMDP's transition model given the previous state and action as input.

Although Strehl et al. prove that their algorithm has polynomial sample and computational complexity, their approach requires that the maximum number of parents D that a variable in the DBN may have be given a priori. In fact, the sample and computational complexity is exponential in D. This is because the algorithms keep statistics about each possible combination of values for each possible set of parents of size D or less. The structure-learning algorithm in this work was embedded in the Factored R_{max} framework [23], a factored version of the R_{max} algorithm [24], which is a PAC-MDP approach for efficient exploration in MDPs. The authors' focus was therefore on efficiently achieving near-optimal behavior on a single task by balancing exploration with exploitation, not on learning modular solutions to ensembles of related tasks, as ours is.

Diuk et al. [25] describe a novel KWIK structure-learning algorithm, called the adaptive k-meteorologists algorithm, that is more efficient than the algorithm presented in [20], but whose computational and sample complexity is still exponential in D, the maximum in-degree of the DBN. This exponential dependence on D is unavoidable in provably optimal (or PAC) Bayesian network structure-learning [19]. When D is large or no a priori information about the domain is known that allows one to specify a small D, these methods are therefore not feasible.

In contrast to these provably optimal approaches, there are greedy methods with only polynomial computational complexity that attempt to add dependencies to a DBN in an incremental fashion, and that have been shown to perform well empirically. They are, however, not guaranteed to find the best network. One approach to structure-learning in FMDPs that uses such a method is given in [8], which was described in Section II-B, and is the approach we chose to extend in our work. Another is given by Degris et al. [26], who present a structured form of the Dyna architecture for planning in MDPs [11] that makes use of an incremental version of the SVI

algorithm to handle planning and employs Utgoff et al.'s [27] incremental tree induction (ITI) algorithm to learn the CPTs of an FMDP's transition and reward models online. A χ^2 test of significance between candidate conditional distributions is applied at the leaves of each of the trees to determine whether to split that leaf on a given variable at each time step. The approach is used to speed up learning on a single task by making use of offline computation to simulate actual experience. They do not address skill learning or performance on ensembles of tasks, however.

Hart et al. [28] present an intrinsic reward mechanism that drives a bimanual robot to learn closed-loop, hierarchical control policies for various abstract behaviors (e.g., tracking, reaching, grasping). Their framework does not make use of the options formalism, but rather a similar scheme for closed-loop control in continuous dynamical systems, called the control basis. The control basis uses the convergence states of hand-engineered continuous controllers to produce a small, discrete state space in which standard RL algorithms may be applied. Intrinsic reward is given to the robot if the state of convergence of some controller that references an external set of stimuli switches from un-converged to converged, with the magnitude of the reward proportional to the number of externally referenced stimuli. This encourages the robot to learn behaviors that allow it to exercise specific types of stable control over its environment in various contexts. Although the learning of new skills is bootstrapped on existing skills, the addition of new skills in this work is controlled by the experimenter and not fully autonomous.

Mugan and Kuipers [29] present a framework for autonomous learning of abstract skill hierarchies in continuous domains. However, the mechanisms for skill learning and abstraction they employ are for discrete environments. They first discretize a continuous domain by extracting "landmarks," and then learn options to set the continuous variables that define the environment to values corresponding to these landmarks. The action (motor) variables are similarly discretized. They employ a modified version of DBNs as their representation of dynamics in terms of what they call "qualitative" variables and actions (the latter being options), but they do not utilize any of the structure-learning methods mentioned previously or any of the RL algorithms that exploit structural independence. The latter means that the option policies and value functions they learn using RL must be represented using a full lookup table, which in some cases can be much larger than the structured representations we employ. Additionally, while their focus is on learning skills applicable over ensembles of related tasks (and indeed there is no extrinsic reward in their framework), there is also no intrinsic reward. Rather, the agent simply chooses random actions (options) from its current skill set, which increases in complexity each time a new option is learned. Although they show that this can lead to increasingly complex behavior, there is no sense of the agent optimizing the rate at which this complexity increases.

VI. Conclusions and Future Work

We have presented a framework for autonomous, incremental learning of skill hierarchies in ensembles of finite FMDPs and active learning of domain structure using intrinsically motivated hierarchical RL. Our results show that the construction of policies and models of abstract skills in this framework can provide drastic reductions in the computational costs of computing policies for novel but related tasks in a given domain when compared with costs using flat policy representations. The addition of options and associated planning methods into our scheme for active learning of environmental dynamics was shown to outperform previous methods of active structure-learning that use only local information when guiding the agent to informative areas of its state space. Our novel method for incremental option construction also makes our approach developmental in nature, allowing for steadily increasing behavioral complexity via bootstrapping of existing structural knowledge and behavioral expertise.

In both the approach presented in [10] and in our work, an agent constructs options to set every environmental variable to each of its possible values. For environments with large numbers of variables and/or values this may not be feasible or desirable. Rather one would like to consider ways of selectively constructing options based on some metric evaluating the utility of being able to set a variable to a certain value. In the case where the agent has a specific task this metric would likely take the task's reward function into account. However, in the initially taskless scenario we outline here, it is less clear what this metric should depend on. One possibility is to incorporate a designer-specified salience function that makes certain variable settings inherently more interesting to the agent than others [7].

Our choice of intrinsic reward was based largely on a previous method for active structure-learning in FMDPs. This is clearly not the only possible intrinsic reward one could employ in this framework. Experimenting further with other ways to increase the rate at which new structure is acquired could yield new insights into more effective intrinsic rewards. Recent work has addressed

searching in the space of reward functions for intrinsic rewards that result in faster learning [30]. Perhaps methods such as these could be used to search for good intrinsic reward functions in our framework as well. Whatever form those rewards may take, however, they can be readily substituted into our developmental framework and make use of the incrementally increasing set of abstract skills generated by agents in the framework.

We took the approach of constructing potentially "premature" options because of the inability for our structure-learning algorithm to distinguish between inherent domain stochasticity and incomplete structural knowledge. As a result we had to add three design parameters to our framework that will in general be domain-dependent. In the absence of domain knowledge this is undesirable, and so it would be fruitful to consider other methods for structure-learning that could make this distinction reliably or to within some confidence factor without having to wait until the full structure of the domain is known. Although the alternative structure-learning methods presented in Section V have this property, in the absence of domain knowledge their complexity is prohibitively high.

Finally, the mechanics of our current approach limits its applicability to finite FMDPs. While it's not clear that certain components we make use of in our work are extensible to the case of continuous states and/or actions (e.g., the VISA algorithm), we are engaged in work exploring structure learning techniques in continuous FMDPs [31]. Applying the principles of our approach, especially intrinsically motivated exploration and skill-learning, to continuous domains is an important direction for future research.

ACKNOWLEDGMENT

The authors are very grateful to Anders Jonsson for his useful insights and for making his research code available to us. We would also like to thank George Konidaris, Scott Kuindersma, Scott Niekum, and Pippin Wolfe for helpful comments and discussions. The work presented here was supported in part by the National Science Foundation under Grant No. IIS-0733581 and in part by the AFOSR under Grant No. FA9550-08-1-0418. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press, 1998.

[2] G. D. Konidaris and A. G. Barto, "Efficient skill learning using abstraction selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, Jul. 2009.

[3] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[4] A. Jonsson, "A causal approach to hierarchical decomposition in reinforcement learning," Ph.D. dissertation, University of Massachusetts Amherst, Feb. 2006.

[5] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning*, vol. 15, no. 2, pp. 201–221, 1994.

[6] J. Schmidhuber, "A possibility for implementing curiosity and boredom in model-building neural controllers," in *International Conference on Simulation of Adaptive Behavior (SAB): From Animals to Animats*. MIT Press/Bradford Books, 1991, pp. 222–227.

[7] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *International Conference on Developmental Learning (ICDL)*, 2004.

[8] A. Jonsson and A. G. Barto, "Active learning of dynamic bayesian networks in markov decision processes," in *Lecture Notes in Artificial Intelligence: Abstraction, Reformulation, and Approximation - SARA 2007*, vol. 4612, 2007, pp. 273–284.

[9] C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artificial Intelligence*, vol. 121, no. 1, pp. 49–107, 2000.

[10] A. Jonsson and A. G. Barto, "Causal graph based decomposition of factored mdps," *Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, 2006.

[11] R. S. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming," in *Advances in Neural Information Processing Systems (NIPS)*, 1991, pp. 471–478.

[12] R. E. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.

[13] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Computational Intelligence*, vol. 5, no. 3, pp. 142–150, 1989.

[14] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research (JAIR)*, vol. 13, pp. 227–303, Nov. 2000.

[15] D. Chickering, D. Geiger, and D. Heckerman, "Learning bayesian networks: Search methods and experimental results," in *Artificial Intelligence and Statistics (AISTATS)*, vol. 5, 1995, pp. 112–128.

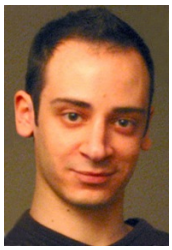
[16] F. Kaplan and P.-Y. Oudeyer, "Maximizing learning progress: An internal reward system for development," in *Embodied Artificial Intelligence*, ser. Lecture Notes in Artificial Intellgience (LNAI) 3139, F. Idia, R. Pfeifer, L. Steels, and Y. Kuniyoshi, Eds. Springer-Verlag, 2004, pp. 259–270.

[17] J. Schmidhuber, "Self-motivated development through rewards for predictor errors/improvements," in *American Association of Artificial Intelligence (AAAI)*, 2005.

[18] O. Simsek and A. G. Barto, "An intrinsic reward mechanism for efficient exploration," in *International Conference on Machine Learning (ICML)*, Jun. 2006.

[19] P. Abbeel, D. Koller, and A. Y. Ng, "Learning factor graphs in polynomial time and sample complexity," *Journal of Machine Learning Research*, vol. 7, pp. 1743–1788, 2006.

[20] A. L. Strehl, C. Diuk, and M. L. Littman, "Efficient structure learning in factored-state MDPs," in *American Association for Artificial Intelligence (AAAI)*, 2007.
[21] L. Li, M. L. Littman, and T. J. Walsh, "Knows What It Knows: A framework for self-aware learning," in *International Conference on Machine Learning (ICML)*, 2008.
[22] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[23] C. Guestrin, R. Patrascu, and D. Schuurmans, "Algorithm-directed exploration for model-based reinforcement learning in factored MDPs," in *International Conference on Machine Learning (ICML)*, 2002, pp. 235–242.
[24] R. I. Brafman and M. Tennenholtz, "R-Max - a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2003.
[25] C. Diuk, L. Li, and B. R. Leffler, "The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning," in *International Conference on Machine Learning (ICML)*. New York, NY, USA: ACM, 2009, pp. 249–256.
[26] T. Degris, O. Sigaud, and P.-H. Wuillemin, "Learning the structure of factored markov decision processes in reinforcement learning problems," in *The 23rd Annual International Conference on Machine Learning*, Pittsburgh, Pennsylvania, 2006.
[27] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision tree induction based on efficient tree restructuring," *Machine Learning*, vol. 29, no. 1, pp. 5–44, 1997.
[28] S. Hart, S. Sen, and R. Grupen, "Intrinsically motivated hierarchical manipulation," in *IEEE Conference on Robots and Automation (ICRA)*, 2008.
[29] J. Mugan and B. Kuipers, "Autonomously learning an action hierarchy using a learned qualitative state representation," in *International Joint Conference on Artificial Intelligence*, 2009.
[30] S. Singh, R. L. Lewis, and A. G. Barto, "Where do rewards come from?" in *Annual Conference of the Cognitive Science Society*, 2009, pp. 2601–2606.
[31] C. M. Vigorito and A. G. Barto, "Incremental structure learning in factored mdps with continuous states and actions," University of Massachusetts Amherst - Department of Computer Science, Tech. Rep., 2009.



Christopher M. Vigorito received his BA in Computer Science and Psychology with honors from Amherst College in Amherst, MA, USA in 2004 and an MS in Computer Science from the University of Massachusetts in Amherst, MA, USA in 2008. He is currently a PhD candidate at UMass Amherst under the advisorship of Andrew Barto. His research interests are in hierarchical reinforcement learning, developmental learning, and computational neuroscience.

Andrew G. Barto Andrew Barto is a Professor of Computer Science, University of Massachusetts, Amherst. He received a B.S. with distinction in mathematics from the University of Michigan in 1970, and a Ph.D. in Computer Science in 1975, also from the University of Michigan. He Co-Directs the Autonomous Learning Laboratory and is a core faculty member of the Neuroscience and Behavior Program of the University of Massachusetts.