Autonomous Hierarchical Skill Acquisition in Factored MDPs

Christopher M. Vigorito and Andrew G. Barto
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01002
{vigorito,barto}@cs.umass.edu

***Abstract*— Learning hierarchies of reusable skills is essential for efficiently solving multiple tasks in a given domain. Understanding the causal relationships between one's actions and various dimensions of one's environment can facilitate learning of abstract skills that may be used subsequently in related tasks. 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 useful skills that exploit this structure. As new structure is discovered, more complex skills are learned, which in turn allow the agent to discover more structure, and so on. Because of this bootstrapping property, our approach can be considered a developmental process that results in steadily increasing domain knowledge and behavioral complexity.**

I. Introduction

Much research in reinforcement learning (RL) has focused on efficient learning of optimal behavior policies for single sequential decision tasks in a given domain [1], [2]. The body of literature applying RL to ensembles of related tasks in the same or similar domains is considerably smaller. Part of the reason for this is the difficulty of defining relatedness between tasks. Without providing a strict definition, we adopt the notion that two or more tasks are related if the transition dynamics of their domains are either identical or overlap considerably in terms of their structure. In the former case, the tasks would only differ in their reward functions, while in the latter it is assumed that there are certain aspects of the dynamics that are common among the tasks. These commonalities can often be exploited to learn policies for each task more efficiently than by learning each task from scratch [3].

An essential component of learning systems designed for solving ensembles of tasks efficiently is a mechanism for representational abstraction. That is, agents must be able to compactly represent policies and models of skills in order to learn feasibly a library of skills that can be reused in multiple tasks to solve similar sub-problems. If the representations for each skill is sparse in the sense of being defined only over relevant environmental variables, a skill can be applied in multiple contexts that differ along irrelevant dimensions without having to relearn the skill in each of those contexts. Abstract representations like this greatly facilitate learning of such skills as well, since the number of relevant variables is generally much smaller than the total number of environmental variables, greatly reducing the amount of experience and computation needed to find good policies.

Hierarchy is also a necessity 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 learning of complex skills as well as planning at multiple levels of abstraction. If an agent can construct a useful hierarchy of abstract skills in a given environment, then the search space of policies for similar tasks within that environment effectively shrinks. This is because selecting between alternative abstract actions allows the agent to take larger, more meaningful steps through the search space of policies than does selecting between more primitive actions [4].

In the framework presented here we focus on the model-based approach whereby an agent accumulates knowledge of the dynamical structure of its environment as it explores. Using this structural knowledge, the agent incrementally generates abstract skills, each composed of a policy for reliably changing certain aspects of its environment and a compact model representing the long term effects that skill has on the environment. As these skills are added to the agent's skill set, they become available as primitive actions to be used when computing policies and models of more complex skills. This bootstrapping scenario of steadily increasing behavioral complexity built upon existing knowledge and behavioral repertoires can be considered an instance of an autonomous developmental learning system [5]. As an agent in this framework continues to add new skills to its behavioral repertoire it becomes more of an expert at manipulating its environment. This increasing behavioral expertise is essentially the high-level goal of an agent in our approach.

The following section describes our formalism for this framework and presents relevant background material. 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, both of which are discussed below. We use incremental Bayesian network learning techniques [6] to accumulate structural knowledge of the environment and, given this knowledge, employ structured dynamic programming methods [2], [7] to compute abstract, closed-loop control policies and their corresponding models in the form of options, the formalization of skills we adopt.

II. Background

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 the 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 defining the reward and transition dynamics of the environment.

When the task of an RL agent is formulated as an MDP, the goal of the agent is to learn a policy $\pi : S \rightarrow A$, 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 value function for the MDP, then the agent can act optimally by selecting actions in each state that maximize Q^π. The Q-learning algorithm [8] is one method for estimating this function online from experience.

When the transition dynamics of the environment are known or estimated from experience, model-based RL [9] can be employed to expedite value function learning in the sense of requiring less experience for Q^π to converge to Q^*. 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 from the model [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 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 [2].

B. Factored MDPs

A factored MDP (FMDP) is an MDP in which the state space 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 [10]). However, if the FMDP contains relatively sparse inter-variable dependencies, we can exploit this structure to reduce the effect this exponential growth has on computing optimal policies.

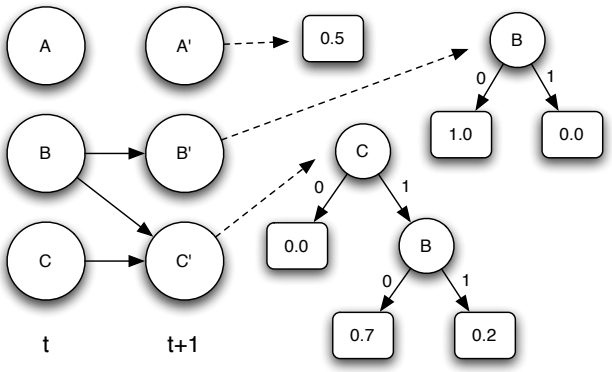

Fig. 1. A simple DBN for a given action with corresponding conditional probability trees.

FMDPs can be represented as a set of Dynamic Bayesian Networks (DBN) [11], one for each action. A DBN 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 a probability distribution 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 decision trees, with the probability that $S_i = 1$ displayed at the leaves.

When the transition and reward models of an FMDP are known, one can use Structured Value Iteration (SVI) [2] 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 on many FMDPs. However, for very large FMDPs, 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 [4] 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. 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, with some subtle differences. 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 Sutton, Precup, and Singh [4]. 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 [7] 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 conditional probability distributions that define the DBN, 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 decision trees the probability distributions over the variables of the FMDP expected once the option finishes executing from 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 compute new option policies in terms of existing options very efficiently. 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 incrementally and construct options as sufficient relevant structure is obtained. This is the subject of the following sections.

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 the corresponding CPTs, one for each variable. 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 [6], described next.

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 $\mathbf{X}$. We thus denote the projection of an assignment $\mathbf{s}$ to $\mathbf{S}$ onto the parents of a variable S_i as $f_{\mathbf{Pa}(S_i)}(\mathbf{s})$. Data points in our framework will take the form of assignment pairs $\langle \mathbf{s}^t, \mathbf{s}^{t+1} \rangle$ denoting the agent's state at times t and $t+1$.

One way to find the best network for a given data set 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 [12], 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 networks.

To do this we maintain at each leaf of each CPT a set of data points that are distributed, one at each time step, to the 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_j, resulting in a new child leaf for each value of S_j, to which the data instances of l are distributed accordingly. If the sum of the BIC scores associated with any refinement of a leaf is greater than the current BIC score of that leaf, then the refinement is kept. Refinements of a leaf l on a variable S_j are not considered if S_j is already on the path from the root of the tree to l.

F. Caching Options

The framework outlined in [13] proposes to learn the full structure of an FMDP given a specified reward function and 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 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 we would hope 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 (described in section II-D) 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 exit option 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 $\mathcal{C}$.

The process described here is the equivalent of running the VISA algorithm on a task whose reward function is 1 for executing action a in the refinement's context, and zero everywhere else. The only difference is that the algorithm now has at its disposal the set of options (and their corresponding models) for setting each variable in $\mathcal{C}$ to each of its values. Additionally, since we are defining the reward function for the option, it need not be learned and we can therefore use SVI to compute the policies for the new options (as distinguished from [13], in which RL was used to learn the option policies). Once the policy is computed we can then compute the option model as well.

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 use VISA to construct an exit option at this point, the option would likely be incorrect (both policy and 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 stochastic, this can only be done to within some confidence factor, since it is impossible to determine with certainty whether a distribution at a CPT leaf has non-zero entropy because of inherent stochasticity or because of incomplete structural knowledge. However, if we make the assumption that 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. For the experiments presented here we make this assumption, although we are currently in the process of extending the approach to the stochastic case by analyzing the distributions and BIC scores at each of the leaves in various ways.

III. Experiments

A. The Light Box Domain

We ran preliminary experiments in a simple artificial domain called the Light Box (Figure 2). 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 and $2^{20} \approx 1$ million states. 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. It should be noted, however, that

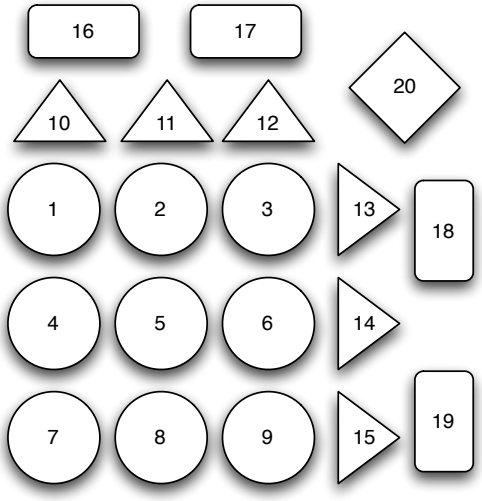

Fig. 2. A visual rendering of the Light Box domain.

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 at any given time. The structure must be discovered from the state transitions the agent experiences while interacting with the environment. Exploitation of this structure is essential for efficient computation of option policies and models in a large domain such as this.

B. Results

We ran an agent in this domain using a random policy and allowed it to collect statistics to incrementally refine its transition model. The model was initialized such that there were no dependencies between variables—each CPT for each variable consisted of a single leaf node. The structure learning techniques discussed in section II-E were applied at every time step and when a refinement was made to a leaf, the method given in section II-F was used to determine whether to construct a new option or not.

The agent was able to learn the correct structure of the domain on each of 30 runs, with an average of 15,783 time steps per run. This is a considerably smaller number than than the number of states in the domain, illustrating the advantage of using factored representations. Learning a tabular representation of the transition model of this domain would require a few orders of magnitude more experience. The amount of computation time spent computing option policies and models during each run was on average 1 minute 55 seconds on an Apple MacBook Pro with a 2.16GHz dual core processor and 2GB of RAM. Figure 3 shows two example option policies that were computed, one for toggling light 10 (O_{10}), and one for light 16 (O_{16}). Note how compact option models allow the nesting of policies, so that O_{16} makes use of O_{10} in its policy.

To illustrate the utility of computing hierarchies of skills

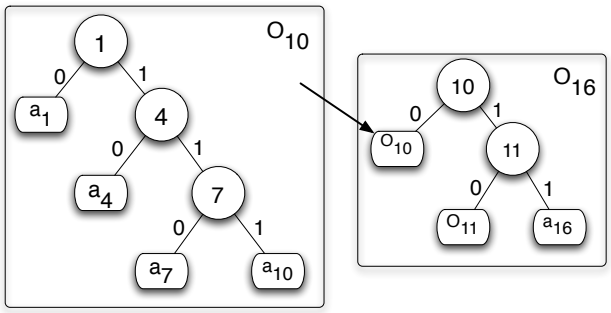

Fig. 3. The policies of two options constructed in the Light Box domain. Note the nested policies.

in large factored domains such as the Light Box, we compared the time it took to compute policies for various different tasks (i.e., different reward functions) by 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 0 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).

Results are shown in Figure 4. For the lowest level, where the tasks can be solved by one primitive action, the two agents take very little time to compute policies, with the options agent being slightly slower due to having a larger action set. 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, but very little increase for the options agent. The overhead of computing the options in the first place is thus made up for once the agent has been confronted with just a few different higher-level tasks such as these. The savings become very substantial above level 2 (note the log scale). The level 4 task took the options agent just 0.05 seconds, but we ran out of time trying to run the primitives-only agent, and so it is not shown.

IV. Discussion

We have presented a framework for autonomous, incremental learning of skill hierarchies in FMDPs. Our preliminary results show that the construction of abstract policies and models of skills in this framework can provide drastic reductions in the computational costs of computing policies for new tasks when compared with flat, or unstructured policy representations. Our approach also has the appeal of being developmental in nature, allowing for steadily increasing behavioral complexity through bootstrapping of existing structural knowledge and behavior.

Although our results suggest that our framework is sound, there are clearly many points on which this work can be

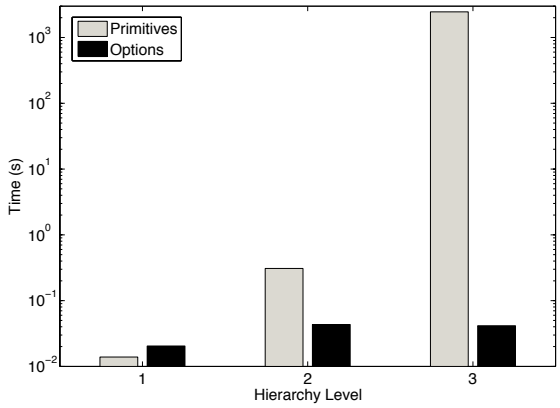

Fig. 4. Policy computation times for tasks at varying levels of the Light Box hierarchy for an agent with primitive actions only and for one with options + primitives.

improved and extended. The first and most obvious is to extend the mechanism for deciding when to construct an option to the stochastic case, as mentioned earlier. This is a relatively straightforward extension and we are currently experimenting with a few different possibilities. Even if a good mechanism is designed, however, additional mechanisms for evaluating the effectiveness of an option at performing its subtask may be required in the case where the mechanism makes a wrong decision and constructs an incorrect option.

While the Light Box domain illustrates how a set of options in a fixed domain can be used to compute policies efficiently for many different reward functions, it does not afford the potential to illustrate transfer between tasks in similar, but non-identical domains. In future work we plan to test the framework in environments with multiple domains that differ somewhat in their transition structure but share common dynamical properties (e.g., "physics" that stay constant over different domains in the same environment). It is in these types of setting that we expect our approach to provide an even greater improvement in computational efficiency than presented here.

We are also currently experimenting with active learning techniques for speeding up the acquisition of environmental structure. Rather than execute a random explorative policy, we would like the agent to use its current skill set to compute plans to move to areas of the state space for which its structural knowledge is lacking. We expect that this will speed up structure learning in complex domains where a random policy has low probability of reaching many areas of the state space. While Jonsson and Barto [6] develop a greedy approach to active learning of this sort, they obtain mixed results in large domains similar to the Light Box. This may be because of the myopic way in which actions are selected. Applying longer-term planning methods may alleviate this problem.

The approach presented in [13] as well as our work constructs options to set every environmental variable to each of its possible values. For environments with large numbers of variables and/or variables with many values this may not be feasible or desirable. Rather we would like to consider ways of selectively constructing options based on some metric evaluating the utility of being able to set a certain variable to a certain value. In the case where the agent has a specific task it is clear that this metric should take the task's reward function into account. However, in the 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, which makes certain types of variable-value combinations inherently more interesting to the agent [14].

V. Acknowledgements

We would like to thank Anders Jonsson for his helpful insights and especially for making his code available to us. The work presented here was supported by the National Science Foundation under Grant No. IIS-0733581. 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 National Science Foundation.

References

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

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

[3] G. D. Konidaris and A. G. Barto, "Building portable options: Skill transfer in reinforcement learning," in *The Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007, pp. 895–900.

[4] 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, pp. 181–211, 1999.

[5] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, "Autonomous mental development by robots and animals," *Science*, vol. 291, no. 5504, pp. 599–600, 2001.

[6] 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, pp. 273–284.

[7] ——, "Causal graph based decomposition of factored mdps," *Journal of Machine Learning Research*, vol. 7, pp. 2259–2351, 2006.

[8] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[9] R. S. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming," in *Advances in Neural Information Processing Systems 3*, 1991.

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

[11] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Computational Intelligence*, vol. 5, pp. 142–150, 1989.

[12] D. Chickering, D. Geiger, and D. Heckerman, "Learning bayesian networks: search methods and experimental results," in *Proceedings of Artificial Intelligence and Statistics*, vol. 5, 1995, pp. 112–128.

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

[14] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *The 3rd International Conference on Developmental Learning*, La Jolla, California, 2004.