Hierarchical Reinforcement Learning Using Graphical Models

Victoria Manfredi VMANFRED@CS.UMASS.EDU
Sridhar Mahadevan MAHADEVA@CS.UMASS.EDU
Computer Science Dept, 140 Governor's Drive, University of Massachusetts, Amherst, MA 01003-9264 USA

Abstract

The graphical models paradigm provides a general framework for automatically learning hierarchical models using Expectation-Maximization, enabling both abstract states and abstract policies to be learned. In this paper we describe a two-phased method for incorporating policies learned with a graphical model to bias the behaviour of an SMDP Q-learning agent. In the first reward-free phase, a graphical model is trained from sample trajectories; in the second phase, policies are extracted from the graphical model and improved by incorporating reward information. We present results from a simulated grid world Taxi task showing that the SMDP Q-learning agent using the learned policies quickly does as well as an SMDP Q-learning agent using hand-coded policies.

1. Introduction

Abstraction is essential to scaling reinforcement learning (RL) (Barto & Mahadevan, 2003; Dietterich, 2000; Parr & Russell, 1998; Sutton et al., 1999). Temporal abstraction permits structured initial exploration by RL agents, allowing reuse of learned activities, and simplifying human interpretation of the learned policy. Spatial abstraction decreases the number of states that need to be experienced, reducing the amount of memory needed, and capturing regularities in the policy structure. Given predefined state and policy abstractions, for instance a task hierarchy, existing methods (Dietterich, 2000; Parr & Russell, 1998; Sutton et al., 1999) can be used to learn the corresponding hierarchical policy. One of the most difficult problems in hierarchical reinforcement learning, however, is how to automatically learn the abstractions. For instance, suppose a MAXQ hierarchy is desired: What should the tasks be? How should the termination conditions be defined?

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

The graphical models framework provides a powerful approach to automatically learning abstractions for hierarchical RL. For example, the abstract hidden Markov model (AHMM) proposed by Bui *et al.* (2002) is a hierarchical graphical model that encodes abstract policies; these policies are derived from the options framework (Sutton et al., 1999) and have associated initiation and termination states. Alternatively, the hierarchical hidden Markov model (HHMM) (Fine et al., 1998) encodes abstract states.

In this work, we describe the use of graphical models to automate hierarchical reinforcement learning using imitation. The method we propose takes advantage of a mentor who provides examples of optimal behaviour in the form of state transitions and primitive actions. We believe humans exploit similar guidance when learning complex skills, such as driving.

Previous approaches to automatic abstraction in RL can be divided into two groups: methods that identify subgoals, i.e., states or clusters of states, and then learn policies to those subgoals (McGovern & Barto, 2001; Şimşek & Barto, 2004; Mannor et al., 2004) and methods that explicitly build a policy hierarchy (Hengst, 2002). Key differences between our method and previous work are that first, by modifying the structure of the graphical model, different abstractions can be learned; second, we learn sub-goals (terminations) and policies simultaneously, rather than separately; finally, our method provides a mechanism for coping with partially observable state through the use of hidden variables.

Previous work on imitation in the context of RL has focused on learning a flat policy model. In Price and Boutilier (2003) a mentor's state transitions are used to help the learner converge more quickly to a good policy. In Abbeel and Ng (2004), the observer's reward function is unknown; instead, a mentor's state transitions and feature expectations are used to identify a policy with similar feature expectations, where features are a mapping over states and feature expectations partially encode the value of the policy. In comparison, graphical models provide a mechanism for learning by imitation where the mentor learns not to just imitate the teacher but learns the task structure implicit in the higher level subgoals in the mentor's policy. This inference involves computing a distribution over the mentor's higher-level policies from its state transitions and actions with higher level policy variables treated as hidden variables. While not studied here, rewards could additionally be incorporated into the graphical model, as in (Samejima et al., 2004).

In the rest of this paper, we define the graphical model that we use and investigate its effectiveness in automating hierarchical RL.

2. Dynamic Abstraction Networks

Previous work in graphical models has largely focused on studying temporal abstraction or state abstraction in isolation. Intuitively, abstract policies are intricately tied to abstract states. For instance, New York City is both a temporal and a spatial abstraction: its geographical location permits you to both execute such abstract policies as visit the Metropolitan Museum of Art or attend a Broadway play, and to define such spatial abstractions as the five boroughs of New York City or the state of New York.

In other work (Manfredi & Mahadevan, 2005) we have proposed a new type of hierarchical graphical model, *dynamic abstraction networks* (DANs), that combines state and temporal abstraction. Jointly encoding state and temporal abstraction permits abstract states and policies to be learned simultaneously, unlike in the AHMM or HHMM alone. We showed in (Manfredi & Mahadevan, 2005) that empirically DANs seem to learn better policy abstractions than do AHMMs.

Figure 1 shows the dynamic Bayesian network (DBN) representation of a 2-level DAN. Informally, we can think of DANs as a merging of a state hierarchy, represented by the HHMM, and a policy hierarchy represented by a modified version of the AHMM which we refer to as an mAHMM. Technically we use the dynamic Bayesian network representations of the HHMM and AHMM defined in (Murphy & Paskin, 2001) and (Bui et al., 2002) respectively. We merge the mAHMM and HHMM by adding edges from state nodes at time t on level i to policy and policy termination nodes at time t on level i and from policy nodes at time t on level i to state nodes at time $t+1$ on

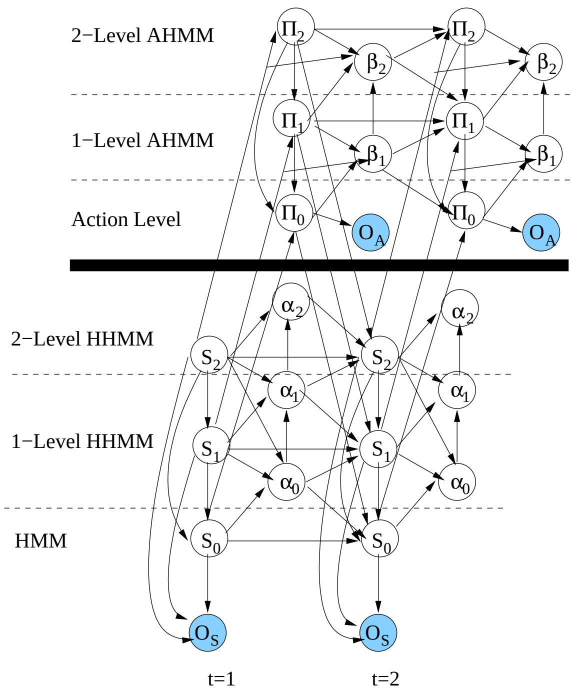


Figure 1. DBN representation of a dynamic abstraction network. We emphasize that this is just one realization of a dynamic abstraction network, and other configurations are possible.

level i. One of the key ideas for developing this model is that abstract states are useful for learning abstract policies. Consequently, abstract states are fed into all policy levels including the actions. We formally define a K-level DAN as comprised of an intertwined state hierarchy and policy hierarchy defined as follows.

Policy Hierarchy. A policy hierarchy H_π with K levels is given by the ordered tuple $H_\pi = (O_A, \Pi_0, \Pi_1, \ldots, \Pi_K)$.

- O_A is the set of action observations. O_A is only necessary if actions are continuous or partially observable.
- Π_0 is the set of primitive (one-step) actions.
- Π_i is the set of abstract policies at level $1 \leq i \leq K$, defined in more detail below.

State Hierarchy. A state hierarchy H_s with K levels is given by the ordered tuple $H_s = (O_S, S_0, S_1, \ldots, S_K)$.

- O_S is the set of state observations.
- S_j is the set of abstract states at levels $0 \leq j \leq K$, defined in more detail below.

Abstract Policies. At level i, each abstract policy $\pi_i \in \Pi_i$ is given by the tuple $\pi_i =< S_{\pi_i}, B_{\pi_i}, \beta_{\pi_i}, \sigma_{\pi_i} >$.

- *Selection set.* $S_{\pi_i} \subset S_i$ is the set of states in which π_i can be executed.
- *Termination probability.* $B_{\pi_i} \subset S_i$ is the set of states in which π_i can terminate. $\beta_{\pi_i} : B_{\pi_i} \rightarrow (0,1]$ is the probability that policy π_i terminates in state B_{π_i}. Note that for all termination states that are not also selection states, policy π_i terminates with probability one.
- *Selection probability.* $\sigma_{\pi_i} : \Pi_{i+1:K} \times S_{\pi_i} \rightarrow [0,1]$ is the probability with which a policy $\pi_i \in \Pi_i$ can be initiated when executing abstract policies $\pi_{i+1} \in \Pi_{i+1}, \ldots, \pi_K \in \Pi_K$ in state $s_{\pi_i} \in S_{\pi_i}$.

Abstract States. At level j, each abstract state $s_j \in S_j$ is given by the tuple $s_j =< \Pi_{s_j}, \alpha_{s_j}, \sigma_{s_j} >$. Note that an abstract state may be part of more than one higher level state; for instance, states at level j may be letters while states at level $j+1$ are words.

- *Entry set.* $\Pi_{s_i} \subset \Pi_i$ is the set of policies which enter state s_i.
- *Exit probability.* $\alpha_{s_j} : S_j \rightarrow (0,1]$ is the probability that the agent exits state $s_j \in S_j$ when in states $s_{j+1} \in S_{j+1}, \ldots, s_K \in S_K$.
- *Transition probability.* $\sigma_{s_j} : S_{j+1:K} \times \Pi_j \times S_j \rightarrow [0,1]$ is the probability that the agent transitions to state $s_j^{t+1} \in S_j$ from state $s_j^t \in S_j$ when in parent states $s_{j+1}^t \in S_{j+1}, \ldots, s_K^t \in S_K$ and executing policy $\pi_j^t \in \Pi_j$.

For the policy hierarchy, Π nodes encode the *selection sets* S_π and the *selection probabilities* σ_π, while β nodes encode the *termination probabilities* β_π. For the state hierarchy, S nodes encode the *entry sets* Π_s and the *transition probabilities* σ_s, while α nodes encode the *exit probabilities* α_s. At level i, choosing policy $\pi_i \in \Pi_i$ depends in part on state $s_i \in S_i$ but executing $\pi_i \in \Pi_i$, by choosing policy $\pi_{i-1} \in \Pi_{i-1}$, depends in part on state $s_{i-1} \in S_{i-1}$. That is, choosing a policy at level i depends on the state at level i, but executing a policy at level i depends on the state at level $i-1$.

3. Hierarchical Reinforcement Learning

In this section, we show how DANs can be applied to the problem of automating hiearchical RL.

3.1. Approach

There are two phases to our approach. In the first phase, the DAN is constructed and trained. Like MAXQ (Dietterich, 2000), our approach requires that the number of levels and the number of policies and states at each level be specified. However, unlike MAXQ, the dependencies between policies and states at different levels are unknown. Instead, all policies (states) at one level are connected to all policies (states) in the adjacent levels to permit any possible set of dependencies to be learned. Sequences of state-action pairs obtained from a mentor are then used to train the model with Expectation-Maximization (EM) (Dempster et al., 1977). Unlike other approaches for learning hierarchies, by reducing the problem to parameter estimation, all levels of the state and policy hierarchies are learned simultaneously through joint inference on the model. Learning in the first phase is on-policy; consequently, the quality of the sample trajectories used to train the model will affect the quality of the policies learned.

In the second phase, the learned policies are obtained from the DAN and improved using RL. Once the DAN has been trained, the policy hierarchy is extracted. The options framework (Sutton et al., 1999) fits most naturally with the DAN policy hierarchy. Changing the policy hierarchy would permit other types of policy hierarchies, such as HAMs (Parr & Russell, 1998) or MAXQ task graphs (Dietterich, 2000), to be used. An option consists of (1) a set of states in which it can be initiated, (2) a set of states in which it terminates, (3) a probability distribution over termination states for each option, and (4) a probability distribution over actions (or lower-level options) for each state. Define a 1-level option to be a policy over options. Then an i-level option is encoded within an DAN: Π nodes encode (1), (2), and (4), while β nodes encode (3). Various methods could be used, but we use reward to improve the extracted policies by using semi-Markov decision process (SMDP) Q-learning (Sutton et al., 1999) to estimate the value function. We discuss this further in the Experiments section.

We note that like DANs, MAXQ similarly encodes state abstractions with associated policy abstractions (task decompositions). However, unlike DANs, these abstractions are hand-specified rather than learned.

3.2. Experiments

The *Taxi* domain (Dietterich, 2000) is used to illustrate the proposed approach. The *Taxi* domain (Dietterich, 2000) consists of a five-by-five grid with four taxi terminals, R, G, Y, and B, see Figure 2. The goal

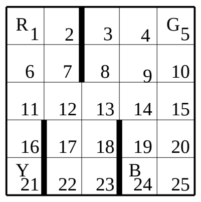


Figure 2. Taxi grid.

of the agent is to pick up a passenger from one terminal, and deliver her to another one (possibly the same one). There are six actions: north (N), south (S), east (E), west (W), pick up passenger (PU), and put down passenger (PD). 80% of the time N, S, E, and W work correctly; for 10% of the time the agent goes right and 10% of the time the agent goes left. The agent's state consists of the taxi location (TL), the passenger location (PL), and the passenger destination (PD). Note that $PL = 1$ when the passenger has been picked up and $PD = 1$ when the passenger has been delivered.

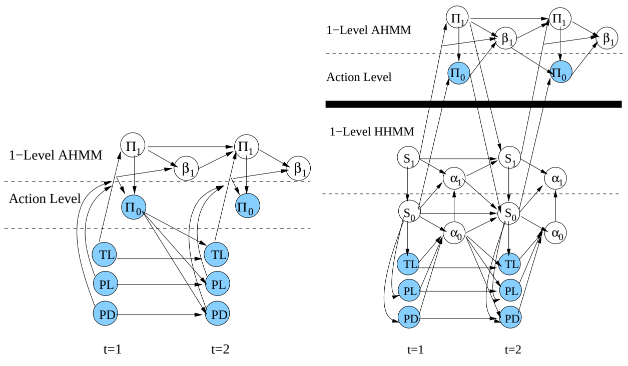
We generated 1000 training sequences from an hierarchical RL mentor trained in the *Taxi* domain using SMDP Q-learning over hand-coded policies. Each sequence is the trajectory of states visited and actions taken in one episode as the RL mentor uses its learned policy to reach the goal. Examples were of variable lengths. A learning rate of $\alpha = 0.1$, an exploration rate of $\epsilon = 0.01$, and a discount rate of $\gamma = 0.9$ were used. Bayes Net Toolbox (Murphy, 2001) was used to implement and train the mAHMM and DAN models in Figure 3 using Expectation-Maximization (Dempster et al., 1977). All distributions were multinomials and except for the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions, were initialized randomly. For the *Taxi* data we set $|S_1| = 5$, $|S_0| = 25$, $|TL| = 25$, $|PL| = 5$, $|PD| = 5$, $|\Pi_1| = 6$, $|\Pi_0| = 6$, $|\alpha_0| = 2$, $|\alpha_1| = 2$, and $|\beta_1| = 2$.

For all experiments, higher level states and policies were biased to change more slowly than lower level states and policies; e.g., the floor you are on should not change more frequently than the room you are in. This was done by initializing the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions as follows, where $i = 0, 1$ and $O_S^t = \{TL^t, PL^t, PD^t\}$.

$$P(\beta_1^t \mid \Pi_0^t, \Pi_1^t, S_1^t) = \begin{cases} 0.95 & \text{if } \beta_1^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_0^t \mid S_0^t, O_S^t) = \begin{cases} 0.95 & \text{if } \alpha_0^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_1^t \mid \alpha_0^t, S_0^t, S_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \alpha_0^t = \text{continue} \\ 0.5 & \text{if } \alpha_1^t = \text{continue and} \\ & \quad \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$



(a) mAHMM (b) DAN

Figure 3. (a) 1-Level mAHMM and (b) 1-Level DAN for Taxi domain; shaded nodes are observed. Note that Π_0 represents actions.

$$P(\Pi_1^{t+1} \mid \Pi_1^t, S_1^t, \beta_1^t) = \begin{cases} 1 & \text{if } \beta_1^t = \text{continue and } \Pi_1^{t+1} = \Pi_1^t \\ 1/|\Pi_1| & \text{if } \beta_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

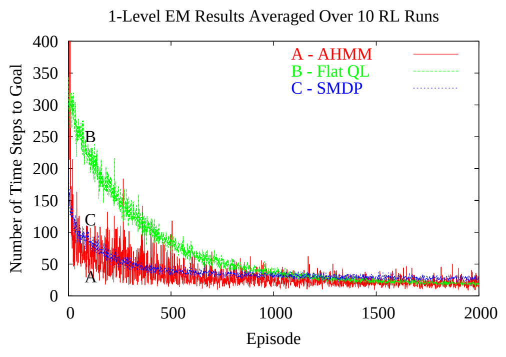
$$P(S_0^{t+1} \mid \Pi_0, S_0^t, S_1^{t+1}, \alpha_0^t, \alpha_1^t) = \begin{cases} 1 & \text{if } \alpha_0^t = \text{continue and } S_0^{t+1} = S_0^t \\ 1/|S_0| & \text{if } \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(S_1^{t+1} \mid \Pi_1^t, S_1^t, \alpha_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \text{continue and } S_1^{t+1} = S_1^t \\ 1/|S_1| & \text{if } \alpha_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

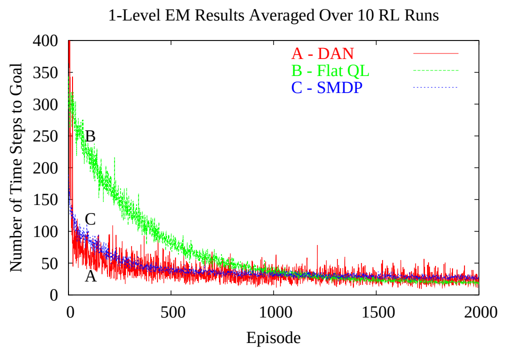
Given the trained mAHMM, policies were extracted and improved using SMDP Q-learning as follows. Once an option (policy) was chosen using an ϵ-greedy exploration strategy, the learned probability distribution $P(\Pi_0|\Pi_1, TL, PL, PD)$ from the mAHMM was used to probabilistically select an action, π_0. Given the trained DAN, policies were again extracted and improved using SMDP Q-learning. However, in order to use the learned probability distributions, we must first compute the most likely abstract state, s_0, with,

$$P(S_0|TL, PL, PD) = \frac{\sum_{S_1} P(TL|S_0)P(PL|S_0)P(PD|S_0)P(S_1)P(S_0|S_1)}{\sum_{S_0,S_1} P(TL|S_0)P(PL|S_0)P(PD|S_0)P(S_1)P(S_0|S_1)}$$

Then given the abstract policy π_1 that was selected using the ϵ-greedy strategy, we can select an action π_0 directly from the conditional probability distribution,



(a) mAHMM



(b) DAN

Figure 4. (a) 1-Level mAHMM results. (b) 1-Level DAN results.

$P(\Pi_0|\Pi_1 = \pi_1, S_0 = s_0)$. Other approaches could be used as well: for instance the full machinery of inference could be used to ascertain how the selected action will affect the predicted distributions over future states. We note that for both the mAHMM and DAN, we permitted options (policies) to be interrupted during SMDP Q-learning. This prevented looping behaviour due to bad policies.

3.3. Results

Figure 4 compares how well the learned mAHMM and DAN policies do against the hand-coded policies and a flat Q-learning agent. What Figure 4 shows is that within about the first fifty timesteps, the SMDP Q-learning agent using the learned policies does as well or better than the SMDP Q-learning agent using the hand-coded policies. We note that the performance of the agent is slightly noisier when it uses the mAHMM rather than the DAN policies.

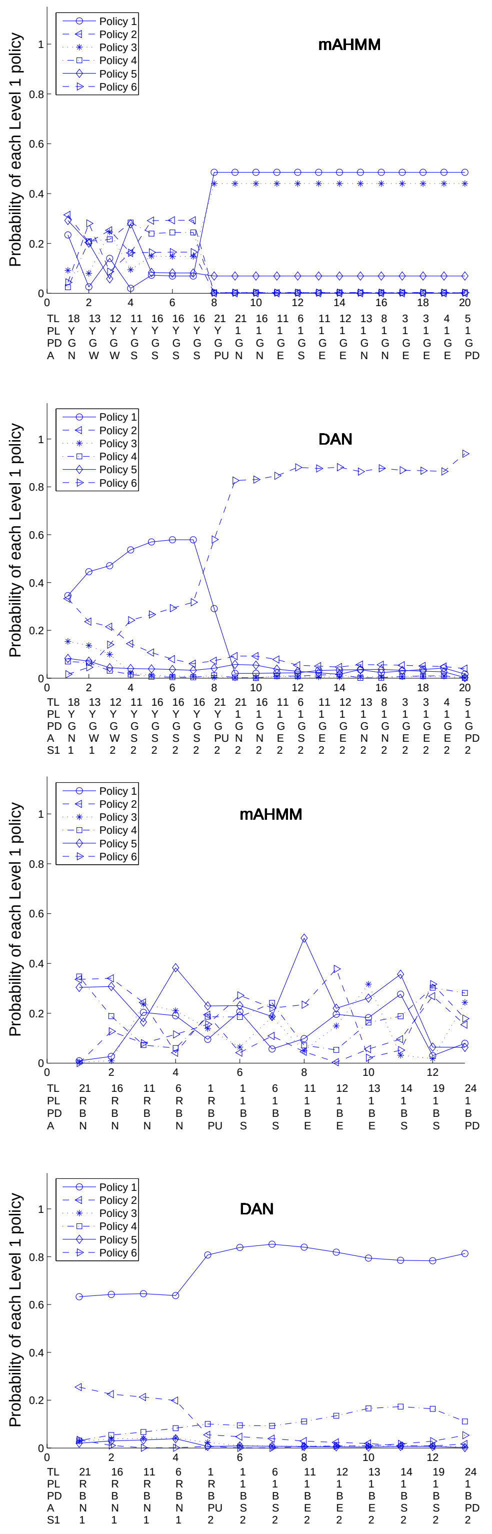


Figure 5. Level 1 policies and states for two sample sequences from the *Taxi* data. The TL, PL, and PD state values and Π_0 actions are shown for both models; the S_0 abstract states are shown for the DAN.

Figure 5 shows the probability of each level 1 policy and state for two sample Taxi sequences for the mAHMM and DAN. What we see from Figure 5 is that the mAHMM has difficulty identifying a single most likely policy with high probability, while the DAN model is able to identify a unique most likely policy with high probability. As shown in Figure 5, the mAHMM performs particularly poorly on the second sequence, never identifying a single policy as most likely for more than a couple of timesteps. Note that while the mAHMM has difficulty identifying a single policy as most likely, this is not a consequence of the policies themselves being poorly learned. Rather, because the mAHMM has learned every policy over the entire state space (due to the structure of the model), all policies are equally good, so any can be used. In essence only a single global policy has been learned. The problem with this is that it cannot be reused, unlike the more local policies learned by the DAN. In particular we see from the DAN graphs in Figure 5 that Policy 1 is used for part or all of both sequences.

4. Conclusions

We have presented a general method for automating hierarchical reinforcement learning. The first phase of our method trains a hierarchical graphical model; the second phase uses the learned policies in an hierarchical reinforcement learning agent. Assuming the graphical model in the first phase encodes the appropriate policy structure, other hierarchical reinforcement learning methods besides SMDP Q-learning can be used for the second phase. In future work we would like to incorporate both phases into an actor-critic architecture. The main disadvantages to our approach are the cost of Expectation-Maximization and having to specify the number of levels and the number of parameters within each level. In future work we plan to explore methods for approximate inference and model selection as applied to dynamic abstraction networks.

Acknowledgments

This work was supported in part by the National Science Foundation under grant ECS-0218125. Any opinons, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Victoria Manfredi was supported by a National Science Foundation graduate research fellowship.

References

Abbeel, P., & Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. *ICML'04* (pp. 506–513).

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Special Issue on Reinforcement Learning, Discrete Event Systems Journal*, *13*, 41–77.

Bui, H., Venkatesh, S., & West, G. (2002). Policy recognition in the abstract hidden Markov model. *Journal of Artificial Intelligence Research*, *17*, 451–499.

Şimşek, O., & Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. *ICML'04* (pp. 751–758).

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

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

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, *32*, 41–62.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. *ICML'02* (pp. 243–250).

Manfredi, V., & Mahadevan, S. (2005). Dynamic abstraction networks. *Technical Report 05-33.* Dept of Computer Science, U of Massachusetts Amherst.

Mannor, S., Menache, I., Hoze, A., & Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. *ICML'04* (pp. 751–758).

McGovern, A., & Barto, A. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *ICML'01* (pp. 361–368).

Murphy, K. (2001). The Bayes net toolbox for Matlab. *Computing Science and Statistics*, *33*.

Murphy, K., & Paskin, M. (2001). Linear time inference in hierarchical hmms. *NIPS'01.*

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *NIPS'98.*

Precup, D. (2000). *Temporal abstraction in reinforcement learning.* Doctoral dissertation, University of Massachusetts, Amherst, Department of Computer Science.

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

Samejima, K., Doya, K., Ueda, Y., & Kumura, M. (2004). Estimating internal variables and parameters of a learning agent by a particle filter. *NIPS'04.*

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