Automated Discovery of Options in Factored Reinforcement Learning

Keywords: options, temporal abstraction, factored reinforcement learning

Olga Kozlova OLGA.KOZLOVA@THALESGROUP.COM
ISIR, Université Pierre et Marie Curie - Paris 6, CNRS UMR 7222, 4 pl. Jussieu, F-75252 Paris Cedex 05, France
Thales Security Solutions & Services, Simulation, 1 rue du Général de Gaulle, 95523 Cergy-Pontoise, France

Olivier Sigaud OLIVIER.SIGAUD@UPMC.FR
ISIR, Université Pierre et Marie Curie - Paris 6, CNRS UMR 7222, 4 pl. Jussieu, F-75252 Paris Cedex 05, France

Christophe Meyer CHRISTOPHE.MEYER@THALESGROUP.COM
Thales Security Solutions & Services, ThereSIS, Campus Polytechnique, 1, av. Augustin Fresnel, 91767 Palaiseau, France

Abstract

Factored Reinforcement Learning (FRL) is a method to solve Factored Markov Decision Processes when the structure of the transition and reward functions of the problem must be learned. In this paper, we present TEXDYNA, an algorithm that combines the abstraction techniques of Semi-Markov Decision Processes to perform the automatic hierarchical decomposition of the problem with an FRL method. The algorithm is evaluated on the taxi problem.

1. Introduction

The Markov Decision Process (MDP) framework is a standard framework for learning and planning under uncertainty. However, due to the "curse of dimensionality", standard exact algorithms cannot address large scale problems in this framework mostly because they have to enumerate all states (Sutton & Barto, 1998). The Factored MDPs (FMDPs) framework improves over MDPs by representing their structure compactly (Boutilier et al., 1995). In this approach, a state is implicitly described by an assignment of values to a collection of state variables. A Dynamic Bayesian Network (DBN) representation (Dean & Kanazawa, 1989) exploits the dependencies between the state variables to avoid the explicit state space enumeration.

Another way to reduce the size of a problem is the use of Semi-MDPs (SMDPs) where the number of time steps between two decisions is a random variable. The SMDP framework is the basis for hierarchical reinforcement learning (HRL) algorithms as it decomposes the original task into smaller pieces (subtasks) that are easier to solve individually (Barto & Mahadevan, 2003).

Generally, the methods used to solve FMDPs and SMDPs assume that the structure of the problem is known in advance. But, in practice, a perfect knowledge of the transition and reward functions of the problem is seldom available. The SDYNA framework (Degris et al., 2006b) is intended to solve such FMDPs while learning the structure. In other respects, algorithms like HEXQ (Hengst, 2002), VISA (Jonsson & Barto, 2006) or the approach recently proposed by (Vigorito & Barto, 2008b) are designed to discover the hierarchical structure of SMDPs. In this paper, we propose TEXDYNA, an algorithm that combines the benefits of HEXQ and SDYNA.

The paper is organized as follows. In sections 2 and 3, we present FMDPs, SMDPs and some standard algorithms in the domain. In section 4, we describe TEXDYNA. In section 5, we compare our results within three contexts: learning without hierarchical structure, learning with a given hierarchical structure and simultaneously learning the hierarchical structure and the policy. This comparison is based on the taxi problem. Finally, in section 6, we discuss the benefits of the proposed approach and conclude on the possibilities to extend this work to more complex problems.

2. Factored Reinforcement Learning

Factored Reinforcement Learning (FRL) is a model-based reinforcement learning approach combining Structured Dynamic Programming (SDP) and model learning.

Appearing in Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning, Montreal, Canada, 2009.

2.1. Structured Dynamic Programming

In the FMDP framework, the state space of the problem is represented as a collection of random variables $X = \{X_1, \ldots, X_n\}$. A state is then defined by a vector $x = (x_1, \ldots, x_n)$ with $\forall i, x_i \in \text{Dom}(X_i)$. FMDPs exploit the structure of the problem to represent large MDPs compactly. For each action a, the model of transitions is defined by a separate Dynamic Bayesian Network (DBN) model (Dean & Kanazawa, 1989). The model G_a is a two-layer directed acyclic graph whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ with X_i a variable at time t and X'_i the same variable at time $t+1$. The parents of X'_i are noted $\text{Parents}_a(X'_i)$. The model of transitions is quantified by Conditional Probability Distributions (CPDs), noted $P^a(X'_i|\text{Parents}_a(X'_i))$, associated to each node $X'_i \in G_a$. SDP algorithms such as SVI (Structured value Iteration) are exact planning algorithms that make profit of this structured representation (Boutilier et al., 2000).

2.2. SDYNA and SPITI

Reinforcement Learning in FMDPs is generally about the case where the structure of the DBNs is given, but the parameters of the CPDs are learned from experience. By contrast, we call Factored Reinforcement Learning (FRL) the case where the structure of the DBNs itself is learned.

An implementation of FRL is expressed in SDYNA, a model-based reinforcement learning framework (Degris et al., 2006a; Degris et al., 2006b), inspired from the DYNA architecture (Sutton, 1991). In SDYNA, the models of transition and reward functions are learned from experience under a compact form. The inner loop of SDYNA is decomposed into three phases: (i) *Acting*: choosing an action according to the current policy, including some exploration; (ii) *Learning*: updating the model of the transition and reward functions of the FMDP from $\langle X, a, X', R\rangle$ observations; (iii) *Planning*: updating the value function $Tree(V)$ and policy $Tree(\pi)$ using sweeps of SDP algorithms.

SPITI is a particular instance of SDYNA using ϵ-$greedy$ as exploration method, the Incremental Tree Induction (ITI) algorithm (Utgoff, 1989) to learn the model of transitions and reward functions as a collection of decision trees, and an incremental version of SVI as planning method. An algorithmic description is given in (Degris et al., 2006b).

3. Hierarchical Reinforcement Learning

The SMDP formalism takes advantage of hierarchical representations, using HRL methods to decompose the original task into smaller tasks by introducing temporally-extended actions that require a variable number of time steps until the system reaches the next state. The availability of these temporally-extended activities can exponentially improve the performance of the agent. It also enables learning or planning on multiple levels of temporal abstraction.

Among various SMDP formalisms, we use the options framework (Sutton et al., 1999). Options are a generalization of primitive actions including temporally extended courses of actions. Even if options are added to the primitive actions set, resulting in a bigger core MDP representation, options provide a decomposition of the original task into subtasks, leading to a simplification of the global problem. Options can also facilitate the transfer of learned local policies to related tasks.

An option is a tuple $\langle \mathcal{I}, \pi, \beta\rangle$, where $\mathcal{I} \subseteq S$ is an initiation set, that is a subset of states in which it is possible to execute o, $\pi : S \times O \rightarrow [0,1]$ is a policy executed in o, and $\beta : S \rightarrow [0,1]$ is a termination condition function, that is the probability of terminating the option in each state. A primitive action $a \in A$ of the original MDP is also an option, called one-step option, with $\mathcal{I} = \emptyset$ and $\beta(s) = 1$. If the option is executed, then sub-options are selected according to π until the option terminates in state $s\prime$ with probability $\beta(s\prime)$. When the option terminates, the agent can select another option. Therefore the SMDP model is represented by a hierarchy of options, in which options on one level select their actions among lower level options.

In this perspective, an option o can be viewed as a subtask given by the option SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi, R_o, T_o\rangle$ where $S_o \in S$ is the option state set, O_o is the set of sub-options that o selects from, Ψ is the set of admissible state-option pairs, i.e. a set of pairs including states determined by the initiation sets of options in O_o, R_o is the option reward function and T_o is a transition probability function.

Well known algorithms such as HEXQ (Hengst, 2002) and VISA (Jonsson & Barto, 2006) call upon the option framework to perform state and temporal abstractions. Similarly to the work presented here, an approach coupling the VISA algorithm with incremental learning of the model of transitions was proposed recently in (Vigorito & Barto, 2008a). As will become clear hereafter, our work is more based on HEXQ than on VISA, though it builds upon some ideas coming from VISA.

4. TeXDYNA: hierarchical decomposition in FRL

TeXDYNA[1] hierarchically decomposes an FMDP by automatically splitting it into a set of options. Meanwhile, the local policy of each option is incrementally improved by SPITI. The central contribution comes from the fact that the discovery of options and the learning of the model of the FMDP are simultaneous.

[1] for *Temporally Extended* SDYNA

The global planning algorithm, described in Algorithm 1, is decomposed into two phases: (1) recursive call of SPITI for model learning, planning and acting over options and (2) discovery of the options and their hierarchy. These two phases are presented hereafter.

Algorithm 1: TeXDYNA

input: FMDP $\mathcal{F}$, options hierarchy $\mathcal{M}$

for each *time step* t

1.a **if** *no option is running* **then**
  choose option o from $\mathcal{M}$ accessible in the current state s according to the current policy π

1.b **if** *terminal condition of* o *is satisfied* **then**
  (i) execute exit action a; observe next state $s\prime$ and immediate reward r
  (ii) update FMDP $\mathcal{F}$ with (s,a,$s\prime$,r)
  (iii) update local policy π_o

1.c **else**
  choose sub-option i according to local policy π_o
  if *sub-option* i *is primitive action* **then**
    (i) execute i; observe $s\prime$ and r
    (ii) update FMDP $\mathcal{F}$ with (s,i,$s\prime$,r)
    (iii) update local policy π_o
  else
    call TeXDYNA over sub-option i

2.a update exits set $\mathcal{E}$

2.b update options hierarchy $\mathcal{M}$

4.1. FRL over options

To achieve simultaneous SMDP structure learning, FMDP structure learning and policy computation, the algorithm executes options recursively by going down the options hierarchy up to primitive actions that can be executed by the agent in its environment. In practice, high level options often form a partition of the state space, thus the option selection procedure returns the only admissible option. Note that when choosing options, the preference is given to options with a higher level of abstraction. The root node of the option hierarchy represents the overall MDP. The resolution of each sub-MDP, represented by an option, follows the three phases of the SDYNA framework.

- Acting (choosing a option o and eventually a sub-option to execute). The first option o is chosen according to the global policy while its initiation set contains the current state s. The sub-options are selected according to the internal policy π_o of the option o augmented with an *ϵ-greedy* exploration policy.
- Learning the transition function of the overall FMDP (updating the FMDP model) using ITI.
- Planning (updating the local policy π_o) using an incremental version of SVI.

Moreover, to propagate the external rewards to the local policies of options, when a high level option is discovered, an additional reward, named "internal reward" r_i (by contrast with the external reward received from the environment) is assigned to its exit action. We set $r_i = \frac{r_{max}}{2}$, where r_{max} is the maximal immediate external reward that the agent can get. This heuristics is inspired by the "salient event" heuristics introduced in (Singh et al., 2005).

Finally, in part 2 of the algorithm, exits set $\mathcal{E}$ and options hierarchy $\mathcal{M}$ are learned according to the procedures presented in the next section.

4.2. Discovery of the options and their hierarchy

Exits discovery Our approach of options discovery is similar to the one used in HEXQ: we define $\mathcal{E}$ - a set of "exits" corresponding to the changes of values of the variables linked to the reward function. Then we associate to each exit a context corresponding to the set of states where this change of value can occur. Finally, we introduce an option for each exit.

However, unlike HEXQ, where exits are state-action pairs, we define exits as a tuple $\langle v, c, a, v_{ch} \rangle$, where v is the variable whose value is changed by this exit, $c = \{x_1, ..., x_n\}$ is the context, that is the set of constraints (i.e. assignment of values to a subset of state variables) that makes this exit available, a is the exit action that makes the value of v change at $s\prime$ and v_{ch} is a variable change, i.e. a pair of values $\langle x, x' \rangle$ where x is the value before a is executed and $x\prime$ the value after a is executed[2]. In this representation, the primitive actions have an empty context.

Algorithm 2 describes the procedure for discovering exits. To ensure the relevance of discovered exits, they are updated every time the model of transitions changes, taking advantage of its tree structure. Besides the fact that most of the exits would be discovered earlier than the complete structure of the problem, some exits might be incomplete or incorrect. To handle this issue, the algorithm checks if the $\mathcal{E}$ already contains an exit defined by the same action and variable, but with a different context definition. If so, it updates it (line 10 in Algorithm 2).

Options discovery The options hierarchy $\mathcal{M}$ is built upon the set of exits discovered at the previous step. Each option is introduced using the following procedure:

1. Create (or update) an option o for each exit;
2. Initialize local policy π_o;
3. Add sub-options;
4. Compute Initialization set I_o.

[2] In the stochastic case, the variable change is a probability distribution over v_{ch} for each exit.

Algorithm 2: Update Exits

init : exits set $\mathcal{E} = \emptyset$
input: FMDP $\mathcal{F}[Tree(P(x'|x,a))]$

forall *transition tree* $Tree(P(x'|x,a)) \in \mathcal{F}$ **do**
  forall *leaf* l *of the* $Tree(P(x'|x,a))$ **do**
    if *action* a *modifies the value of the variable* x *in leaf* l **then**
      if *if* $\mathcal{E}$ *does not contain a definition of exit* e *corresponding to variable* x *and action* a **then**
        introduce new exit $e : \langle v, c, a, v_{ch} \rangle$ with :
        • variable $v \leftarrow x$
        • context $c \leftarrow$ variables of the branch that leads to the leaf l
        • action $a \leftarrow$ current action a
        • variable change $v_{ch} \leftarrow \langle$value in the branch, value in the leaf$\rangle$
      else if $\mathcal{E}$ *contains a partial definition of* e **then**
        update e with new information

The sub-options are added in the following way: for each variable in the context of the option, if $\mathcal{E}$ contains an exit that modifies the value of this variable, a sub-option corresponding to this exit is added. Note that, when computing the context of an exit, the exit variables are excluded from the context to avoid cross-dependencies between options.

The initiation set I_o of option o is an union of its own exit context and all the exit contexts of its sub-options. If a sub-option is a primitive action, its exit variable is added to I_o with all possible values. That way, I_o contains the initiation sets I_i of each sub-option i. By convention, a sub-option with an empty initiation set is admissible everywhere. This is particularly true for primitive actions. Therefore, all the values of the corresponding exit variables of these options are accepted. Thus, I_o contains all the states from which the exit of the option is reachable. This property of direct reachability is ensured by the fact that the exit context copies the constraints of the corresponding branch in the transition tree.

An option terminates by executing the exit action a as soon as it reaches the context c of its associated exit e or as soon as it can no longer reach c. More precisely, an option o can no longer reach the exit as soon as the state is not listed in I_o anymore. The probability of terminating an option o is $\beta(s) = 1$ for states s consistent with the context c and for states $s \notin I_o$. In all other cases, $\beta(s) = 0$.

The structure of $\mathcal{M}_o$ evolves during the learning process according to the structure of the model of transitions, which can be erroneous in the first stages of learning. That may result in the discovery of inaccurate options. The quality of an option is related to the number of times this option has been updated. In fact, as the model of the transition function stabilizes, the exits corresponding to inaccurate dependencies are not discovered anymore. Therefore, the options introduced for these exits are not updated either. Thus one can discard options relying on a criterion based on the number of times it is updated. The exact process is the following. At the creation of a new option, its number of updates is initialized at the average number of updates of all options at the same level of the hierarchy. Then, removal is based on confidence intervals on these numbers of updates: the statistical confidence intervals are calculated over the qualities of neighboring options in the same level of hierarchy. The options whose number of updates is inferior to the average of more than 60 $\%$ are considered irrelevant and removed.

An example of hierarchy of options obtained on the taxi problem is given in figure 2.

5. Experiments

The algorithms are coded in C# and run on Intel Core2Duo 1.80GHz processor with 2Go RAM. All results presented below are averaged over 20 runs where each run performs 100 episodes limited to 300 steps. The ϵ-$greedy$ exploration method uses $\epsilon = 0.1$. The internal reward used to learn the structure of options is 10.

5.1. The taxi problem

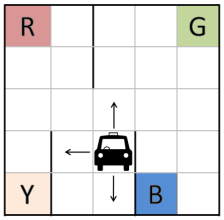

Figure 1. The taxi problem

The taxi problem presented in figure 1 was first proposed in (Dietterich, 1998). A taxi is in a 5-by-5 grid world. There are four special locations, named R, G, Y and B. The taxi problem is episodic, there are 800 possible states. In each episode, the taxi starts in a randomly-chosen state. There is a passenger at one of the four special locations (chosen randomly), and that passenger wishes to be transported to one of the three other locations (also chosen randomly). The taxi must go to the passenger's location, pick her up, go to the destination location, and drop her off there. The episode ends when the passenger is at her destination location or when a predefined number of steps has been reached. At each time step, the taxi can execute one possible action out of six: $Move$ the taxi one square $North$, $South$, $East$, or

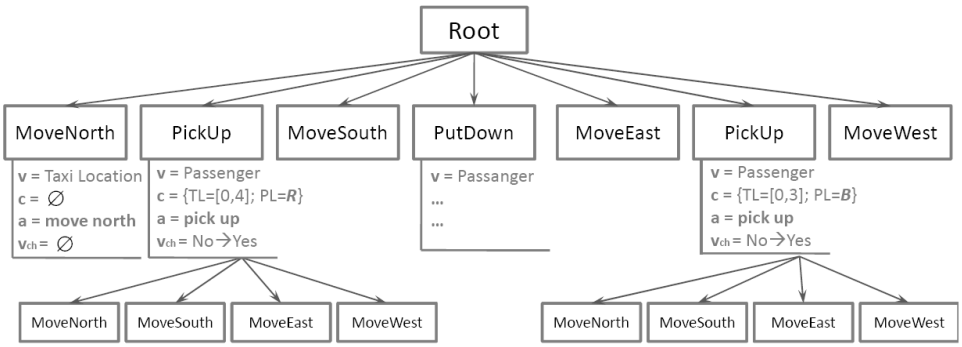

Figure 2. Example of options discovered in the taxi problem. The option *PickUp* changes the value of *Passenger* from *No* (not in the taxi) to *Yes* (in the taxi). Its exit context contains 2 variables: *Taxi Location* and *Passenger Location*. It has 4 sub-options.

$West$, $PickUp$ or $PutDown$ the passenger. In a stochastic version, instead of moving in the selected direction, a $Move$ action moves in a random direction with probability 0.2. There is a penalty of -1 for each action and an additional reward of 20 for successfully dropping off the passenger. There is a penalty of -10 if the taxi attempts to execute the $PutDown$ or $PickUp$ actions illegally.

5.2. Results

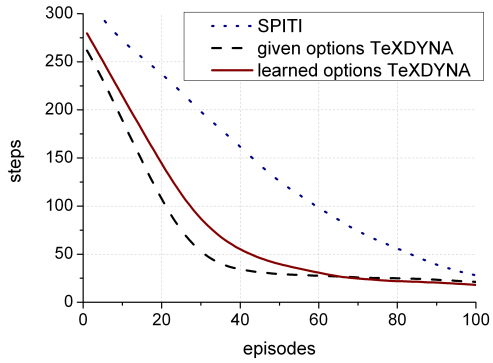

Figure 3. Convergence over episodes on stochastic taxi problem.

Figure 3 shows the performance in number of time steps required to complete one episode of the stochastic version of the taxi problem. The curves are smoothed by computing the moving average weighted over ten neighboring values. The results below are given for SPITI, for a simplified version of TeXDYNA where options are given and for TeXDYNA where options are learned. Table 1 recaps the average time in seconds per step within this three experimental contexts.

TeXDYNA clearly outperforms SPITI in computation time and the number of learning episodes needed to converge but also in memory requirements. It needs about 40 episodes to converge when options are given, almost 100 episodes with primitives actions only and 60 episodes when options are learned. As expected, this result is intermediate between the one with given options and the one without options.

Table 1. Performance on the stochastic taxi problem.

	Time/step(sec)
SPITI	1.1 ± 0.4
given options TeXDYNA	0.09 ± 0.03
learned options TeXDYNA	0.24 ± 0.08

As to the size on the value functions, SPITI builds the complete tree representing the 800 states of the problem, whereas TeXDYNA operates with 8 options each of which only considers 25 states. Thus TeXDYNA requires less time to perform one step since it works on a smaller representation. Even considering the options representing the primitives options, the hierarchy of options provides a simplification of the global structure. Moreover simultaneously discovering and refining options while learning the FMDP structure speeds up the global process since it builds most of the partial policies before the model of transitions is completely learned.

With respect to results found in the literature, our model performs better than HEXQ, which needs about 160 episodes to converge and than MAXQ-Q (Dietterich, 1998), which needs about 115 episodes. Furthermore, both MAXQ-Q and HEXQ converge slower than SPITI. That can be explained by the fact that the former does not explicitly use the factored structure of the problem and the latter spends a significant number of episodes evaluating the order of the variables before building the hierarchy. A detailed comparison with the approach described in (Vigorito & Barto, 2008a) remains to be performed.

6. Discussion and conclusion

We have presented the TEXDYNA algorithm combining incremental hierarchical decomposition with the FRL framework. The global architecture presented in section 4.1 appears similar to a Task option of the VISA framework (Jonsson & Barto, 2006). Nevertheless TEXDYNA builds an options hierarchy online and directly from the transition trees taking advantage of their structure, while VISA constructs a variable influence graph from the given DBNs and then builds transition graphs and reachability trees to determine the initiation sets of the options.

Although we have shown on the taxi problem that our approach performs better than reference algorithms, there are still many opportunities for improvement. First, we will make profit of the learning method used in SPITI to learn only a local model for each option. As a result, the models will be smaller and, therefore, easier to learn. Second, our current results are limited to the building of a two level hierarchy, mainly because we discovered in the taxi problem that relaxing this constraint may result in the discovery of options that are harder to solve than the global task. To solve this problem we are focusing on two points. First, we want to emphasize the reuse of options as sub-tasks in different contexts. Second, we are working on incremental reorganization of the options hierarchy. Furthermore, several heuristics used in (Vigorito & Barto, 2008a) may be helpful, such as waiting for an option to be "mature enough" before introducing it in the hierarchy or using a measure of entropy on the leaves of transition trees instead of numbering the times one option has been updated.

Since the taxi problem is limited as to hierarchy levels that can be built, we are currently working on applying TEXDYNA to the *LightBox* problem proposed in (Vigorito & Barto, 2008a), opening the possibility to a detailed comparison. Finally, we intend to apply our approach to more complex industrial simulation problems where building a greater number of hierarchy levels will help.

References

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

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting Structure in Policy Construction. *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1104–1111). Montreal.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, *121*, 49–10.

Dean, T., & Kanazawa, K. (1989). A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, *5*, 142–150.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006a). Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. *Proceedings of the 22nd Conference Uncertainty in Artificial Intelligence* (pp. 122–129). MIT, Cambridge: AUAI Press.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006b). Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. *Proceedings of the 23rd International Conference on Machine Learning* (pp. 257–264). Pittsburgh, Pennsylvania: ACM.

Dietterich, T. (1998). The MAXQ Method for Hierarchical Reinforcement Learning. *Proceedings of the 15th International Conference on Machine Learning* (pp. 118–126).

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. *Proceedings of the 19th International Conference on Machine Learning* (pp. 243–250).

Jonsson, A., & Barto, A. (2006). Causal Graph Based decomposition of Factored MDPs. *Journal of Machine Learning Research*, *7*, 2259–2301.

Singh, S., Barto, A., & Chentanez, N. (2005). Intrinsically Motivated Reinforcement Learning. *Advances in Neural Information Processing Systems*, *18*, 1281–1288.

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.

Sutton, R. S. (1991). DYNA, an Integrated Architecture for Learning, Planning and Reacting. *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*.

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

Utgoff, P. E. (1989). Incremental Induction of Decision Trees. *Machine Learning*, *4*, 161–186.

Vigorito, C., & Barto, A. (2008a). Hierarchical Representations of Behavior for Efficient Creative Search. *AAAI Spring Symposium on Creative Intelligent Systems, Palo Alto, CA*.

Vigorito, C. M., & Barto, A. G. (2008b). Autonomous Hierarchical Skill Acquisition in Factored MDPs. *Yale Workshop on Adaptive and Learning Systems*. New Haven, Connecticut.