

# Simultaneous Learning of Action and Space Hierarchies in Reinforcement Learning

Mehran Asadi and Afrand Agah  
Department of Computer Science and Engineering  
University of Texas at Arlington  
Arlington, TX 76019  
{asadi,agah}@cse.uta.edu

**Abstract**—This paper presents a new method for the autonomous construction of hierarchical action and state representations in reinforcement learning, aimed at accelerating learning and extending the scope of such systems. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks by analyzing the learned policy using Monte Carlo sampling. The agent is able to transfer this knowledge to subsequent tasks and to accelerate learning by creating corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning, adding a new level to the state space hierarchy. This level serves as the initial representation for new learning tasks. In order to ensure that tasks are learnable, value functions are built simultaneously at different levels of hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space.

## I. INTRODUCTION

Autonomous systems are often difficult to program. Reinforcement learning (RL) is an attractive alternative, as it allows the agent to learn behavior on the basis of sparse, delayed reward signals provided only when the agent reaches desired goals. However, standard reinforcement learning methods do not scale well to larger, more complex tasks. One promising approach to scaling is hierarchical reinforcement learning (HRL) [1], [2], [3], [4], [5].

One of the fundamental steps toward HRL is to automatically establish subgoals. Methods for automatically introducing subgoals have been studied in the context of adaptive production systems, where subgoals are created based on examinations of problem-solving protocols. For RL systems, several researchers have proposed methods by which policies learned for a set of related tasks are examined for commonalities or are probabilistically combined to form new policies. Subgoal discovery has

been addressed by several researchers [6], [7], [8]. The most closely related research is that of Digney [7] where states that are visited frequently or states where the reward gradient is high are chosen as subgoals.

The presented work introduces a new method for the autonomous construction of hierarchical actions and state representations in reinforcement learning. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks by analyzing the learned policy using Monte Carlo sampling. The agent is able to transfer knowledge and to accelerate learning of subsequent tasks by creating new subgoals and by off-line learning corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning. This representation forms a new level in the state space hierarchy and serves as the initial representation for new learning tasks. To ensure that tasks are learnable, value functions are built at different levels of hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space.

## II. REINFORCEMENT LEARNING

In the RL framework, a learning agent interacts with an environment over a series of time steps  $t = 0, 1, 2, 3, \dots$ . At each time  $t$ , the agent observes the state,  $s_t$ , and chooses an action,  $a_t$ , which causes a transition to state  $s_{t+1}$  and a reward,  $r_{t+1}$ . In a Markovian system, the likelihood of the next state and of the reward depend only on the preceding state and the action taken. The objective of the agent is to learn a (possibly probabilistic) mapping from states to actions which maximizes the expected discounted reward received over time. A common RL approach is to approximate the optimal state/action value function, or Q-function, which maps state/action

pairs to the maximum expected return starting from the given state and action and thereafter following the best policy.

To permit the construction of a hierarchical learning system, we model our learning problem as a Semi-Markov Decision Problem (SMDP) and use the option framework [1]. An option is a temporally extended action which, when selected by the agent, executes until a termination condition is satisfied. While an option is executing, actions are chosen according to the option’s own policy. More specifically, an option is a triple  $o_i = (I_i, \pi_i, \beta_i)$ , where  $I_i$  is the set of states in which the option can be initiated;  $\pi_i$  is the option’s policy defined over all states in which the option can execute; and  $\beta_i$  is the termination condition defining the probability with which the option terminates in a given state,  $s$ . Each option used in this paper bases its policy on its own internal value function. The value of a state  $s$  under an SMDP policy  $\pi^o$  is defined as [9], [1]:

$$V^{\pi^o}(s) = E[R(s, o_i) + \sum_{s'} F(s'|s, o_i) V^{\pi^o}(s')]$$

where

$$F(s'|s, o_i) = \sum_{k=1}^{\infty} P(s_{t+k} = s' | s_t = s, o_i) \gamma^k \quad (1)$$

and

$$R(s', o_i) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | \epsilon(\pi_i, s, t)] \quad (2)$$

where  $r_t$  denotes the reward at time  $t$  and  $\epsilon(\pi_i, s, t)$  denotes the event of an action under policy  $\pi_i$  being initiated at time  $t$  and in state  $s$  [1].

### III. AUTONOMOUS SUBGOAL DISCOVERY

An example that shows the importance of a subgoal is a room to room navigation task where the agent should discover the utility of doorways. If the agent discovers that a doorway is a subgoal it can learn an option to reach the doorway which, in turn, can accelerate learning of new navigation tasks. The idea of using subgoals is not, however, limited to grid worlds. For example, for a robot arm to pick up an object, an important subtask is the recognition of the object and thus being aware of its presence would be a subgoal.

The main goal of automatic subgoal discovery is to find useful subgoals in the agent’s state space. Once they are found, options to those subgoals can be learned and added to the behavioral repertoire of the agent. In the approach presented here, subgoals are identified as states with particular structural properties in the context of a

given policy. In particular, we define subgoals as states that, under a given policy, lie on a substantially larger number of paths than would be expected by looking at its successor states. In other words, we are looking for states that form a “funnel” for state space trajectories under the learned policy.

**Definition 1** A state  $s'$  is a direct predecessor of state  $s$ , if under a learned policy the action in state  $s'$  can lead to  $s$  i.e.,  $P(s|s', a) > 0$ .

**Definition 2** The count metric for state  $s$  under a learned policy,  $\pi$ , is the sum over all possible state space trajectories weighed by their accumulated likelihood to pass through state  $s$ .

Let  $C_{\pi}^*(s)$  be the count for state  $s$ , then:

$$C_{\pi}^1(s) = \sum_{s' \neq s} P(s|s', \pi(s')) \quad (3)$$

and

$$C_{\pi}^t(s) = \sum_{s' \neq s} P(s|s', \pi(s')) C_{\pi}^{t-1}(s') \quad (4)$$

$$C_{\pi}^*(s) = \sum_{i=1}^n C_{\pi}^i(s) \quad (5)$$

where  $n$  is such that  $C_{\pi}^n(s) = C_{\pi}^{n+1}(s)$  or  $n = |S|$ . The condition  $s' \neq s$  prevents the counting of self loops and  $P(s|s', \pi(s'))$  is the probability of reaching state  $s$  from state  $s'$  by executing action  $\pi(s')$ . The slope of  $C_{\pi}^*(s_t)$  along a path,  $\rho$ , under policy  $\pi$  is:

$$\Delta_{\pi}(s_t) = C_{\pi}^*(s_t) - C_{\pi}^*(s_{t-1}) \quad (6)$$

where  $s_t$  is the  $t^{th}$  state along the path. In order to identify subgoals, the gradient ratio  $\Delta_{\pi}(s_t) / \max(1, \Delta_{\pi}(s_{t+1}))$  is computed for states where  $\Delta_{\pi}(s_t) > 0$ . A state  $s_t$  is considered a potential subgoal candidate if the gradient ratio is greater than a specified threshold  $\mu > 1$ . Appropriate values for this user-defined threshold depend largely on the characteristics of the state space and result in a number of subgoal candidates that is inversely related to the value of  $\mu$ . This approach is an extension of the criterion in [10] with  $\max(1, \Delta_{\pi}(s_{t+1}))$  addressing the effects of potentially obtaining negative gradients due to nondeterministic transitions.

In order to reduce the computational complexity of the above method in large state spaces, the gradient ratio is here computed using Monte Carlo sampling.

**Definition 3** Let  $H = \{h_1, \dots, h_N\}$  be  $N$  sample trajectories induced by policy  $\pi$ , then the sampled count metric,  $C_H^*(s)$ , for each state  $s$  that is on the path of at least one path  $h_i$  can be calculated as the

average of the accumulated likelihoods of each path,  $h_i$ ,  $1 \leq i \leq N$ , rescaled by the total number of possible paths in the environment.

We can show that for trajectories  $h_i$  and sample size  $N$  such that

$$N \geq \frac{\max_t C_H^*(s_t)}{\epsilon_N^2} 2(1 + \epsilon_N) \log\left(\frac{2}{1-p}\right) \quad (7)$$

the following statement is true with probability  $p$ :

$$|C_H^*(s_t) - C_\pi^*(s_t)| \leq \epsilon_N$$

**Theorem 1** Let  $H = \{h_1, \dots, h_N\}$  be  $N$  sample trajectories induced by policy  $\pi$  with  $N$  selected according to Equation 7. If  $\frac{\Delta_H(s_t)}{\max(1, \Delta_H(s_{t+1}))} > \mu + \frac{2\epsilon_N(\mu+1)}{\max(1, \Delta_H(s_{t+1}))}$ , then  $\frac{\Delta_\pi(s_t)}{\max(1, \Delta_\pi(s_{t+1}))} > \mu$  with probability  $\geq p$ .

Theorem 1 implies that for a sufficiently large sample size the exhaustive and the sampling method predict the same subgoals with high probability.

### A. Example

Figure 1(a) shows a two-room example environment on a  $10 \times 6$  grid. For this experiment, the goal state is placed in the upper right hand portion (gray cell) and each trial is started from the same state in the lower left corner. The action space consists of eight primitive actions (North, East, South, West, Northwest, Northeast, Southwest and Southeast). The world is deterministic and each action succeeds in moving the agent in the chosen direction. With every action the agent receives a

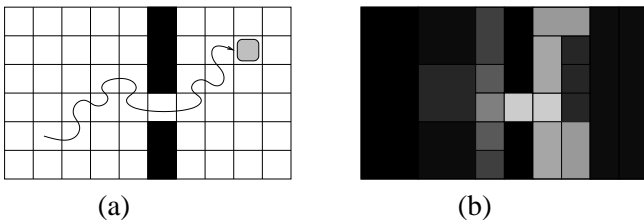


Fig. 1. (a) A two room environment with a connecting doorway. The goal is illustrated in gray and the line shows a sample trajectory. (b) Cells are shaded according to the gradient ratio over 10 trajectories with higher ratios indicated by lighter shading.

negative reward of  $-1$  for a straight action and  $-1.2$  for a diagonal action. In addition, the agent receives a reward of  $+10$  when it reaches the goal state. Policy  $\pi$  is learned using Q-learning and the count metric for every state is computed. The agent then evaluates the gradient ratio along the count curve by choosing 10 random trajectories according to  $\pi$ , and picks the states in which the ratio is higher than the specified threshold as subgoal states. Figure 1(b) shows the values of the gradient ratio for

each state. In this example, the gradient ratio is less than 4 in all states except the doorway where it is 5.232. The mean of the distribution of gradient ratios over the state space is 3.265 and the standard deviation is 0.208. The threshold is here chosen to be 4 resulting in one subgoal in the location of the doorway. This subgoal could now be used to learn similar tasks in this environment.

## IV. ACTION DEPENDENT PARTITIONS

Once potential subgoals are discovered, options that terminate in the subgoal states can be learned and added as abstract actions to the action hierarchy available to the agent. In addition, these options can be used to build a more compact representation of the state space. Abstraction is achieved here by partitioning the state space of the original MDP into blocks of states that have similar properties (i.e transition probabilities and reward values).

$\epsilon$ -reductions were introduced by Dean et al. [11] as a mechanism to derive state space partitions of a MDP that ensure approximately optimal policies to be learned. These partitions depend on the action space and the particular reward function of the task. Kim and Dean, [2] introduced an algorithm to derive a set of such partitions and used it to learn a policy for the task indicated by the reward function. The resulting policy is ensured to be within an  $\epsilon$ -dependent quality bound.

Two shortcomings of this algorithm are that it does not address temporally abstract actions and that it requires a complete re-computation of the partitions when a new action is introduced. In addition it requires knowledge of the reward function prior to partitioning, and thus no part of the partitioning transfers across tasks. To address these issues, the algorithm introduced in this section derives partitions with the same approximate optimality properties for the SMDP framework in two phases, the first of which is reward-independent and thus transfers across tasks. This method first constructs the options  $o_i = (I_i, \pi_i, \beta_i)$  according to the discovered subgoals. The transition probability function  $F(s|s', o_i)$  and the reward function  $R(s, o_i)$  can be computed using equations 1 and 2. The transition function can here be completely pre-computed at the time when the policy itself is learned and as a result, only the discounted reward estimate has to be re-computed for each new learning task.

In the two phase partitioning approach presented here we construct the initial blocks of the partition by distinguishing terminal states (subgoals) for available option from non-terminal states and then refine these blocks

based on the transition probability function.

Let  $\{s_1, \dots, s_n\}$  be  $n$  discovered subgoals and  $\{o_1, \dots, o_n\}$  be the corresponding options. We construct a partition  $P = \{B_1, \dots, B_n\}$  of state space  $S$  such that each set  $B_i$  contains all states  $s \in I_i$  such that  $F(s_i|s, o_i) > 0$ ,  $\beta(s_i) = 1$  and  $\cup_{i=1}^n B_i = S$ .

**Definition 4** A partition  $P = \{B_1, \dots, B_n\}$  of the state space of an MDP has approximate stochastic bisimulation homogeneity if and only if for each  $B_i, B_j \in P$  and for each  $s, s' \in B_i$ :

$$\left| \sum_{s'' \in B_j} F(s''|s, o_i(s)) - \sum_{s'' \in B_j} F(s''|s', o_i(s')) \right| \leq \delta \quad (8)$$

where  $0 \leq \delta \leq 1$ .

We say that a block  $B_i$  is  $\delta$ -stable with respect to block  $B_j$  if and only if Equation 8 holds.  $B_i$  is  $\delta$ -stable if  $B_i$  is  $\delta$ -stable with respect to all blocks of  $P$ .

To form partitions, each block is checked for  $\delta$ -stability and unstable blocks are split until no unstable blocks remain. When a block  $B_k$  is found to be unstable with respect to block  $B_l$ , we replace  $B_k$  by a set of sub-blocks  $B_{k_1}, \dots, B_{k_m}$  such that  $B_{k_i}$  is maximal sub-block of  $B_k$  that is  $\delta$ -stable with respect to  $B_l$ . To facilitate modifications in the action space, this process is first performed for each option individually. The blocks of the final partition are then formed by intersecting all blocks for each  $o_i$  that are used followed by a refining stage that achieves  $\delta$ -stability for the intersections [5]. This reduces the overhead required when the action set changes to the intersection and the final refinement step.

If the reward structure becomes available, the second phase of the partitioning technique further refines the partition with the following reward criterion:

$$|R(s, o_i) - R(s', o_i)| \leq \epsilon \quad (9)$$

Given a particular subset of options, an appropriate abstract state space representation for the learning task can thus be derived which is stable according to criteria in Equations 8 and 9. Furthermore, representation changes due to changes in the action set can be performed efficiently and a simple mechanism can be provided to use the previously learned value function as a starting point when such representation changes occur. This is particularly important if actions are added over time to permit refinement of the initially learned policy by permitting finer-grained decisions.

## V. LEARNING METHOD

Let  $P = \{B_1, \dots, B_n\}$  be a partition for state space  $S$  derived by the action-dependent partitioning method,

using subgoals  $\{s_1, \dots, s_k\}$  and options to these subgoals  $\{o_1, \dots, o_k\}$ . If the goal state  $G$  belongs to the set of subgoals  $\{s_1, \dots, s_k\}$ , then  $G$  is achievable by options  $\{o_1, \dots, o_k\}$  and the task is learnable according to Theorem 3. However, if  $G \notin \{s_1, \dots, s_k\}$  then the task may not be solvable using only the options that terminate at subgoals. The proposed approach solves this problem by maintaining a separate value function for the original state space while learning a new task on the partition space derived from only the subgoal options. During learning, the agent has access to the original actions as well as all options, but makes decisions only based on the abstract partition space information.

While the agent tries to solve the task on the abstract

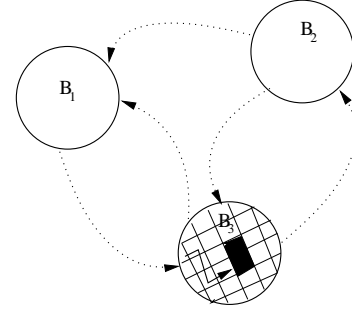


Fig. 2. An abstract state space with 3 blocks ( $B_1, B_2, B_3$ ). Options are shown using dotted curves and original action are illustrated with solid lines. The black cell in block  $B_3$  is the goal state. Since the Q-value in block  $B_3$  is significantly smaller than the one of the underlying goal state this block is refined using the options and the primitive actions.

partition space, it computes the difference in Q-values between the best actions in the current state in the abstract state space and in the original state space. If the difference is larger than a constant value (given by Theorem 2), then there is a significant difference between different states underlying the particular block that was not captured by the subgoal options. Theorem 2 [2] shows that if blocks are stable with respect to all actions the difference between the Q-values in the partition space and in the original state space is bounded by a constant value.

**Theorem 2** Given an MDP  $M = (S, A, T, R)$  and a partition  $P$  of the state space  $M_P$ , the optimal value function of  $M$  given as  $V^*$  and the optimal value function of  $M_P$  given as  $V_P^*$  satisfy the bound on the distance

$$\|V^* - V_P^*\|_\infty \leq 2 \left( 1 + \frac{\gamma}{1 - \gamma} \epsilon_p \right)$$

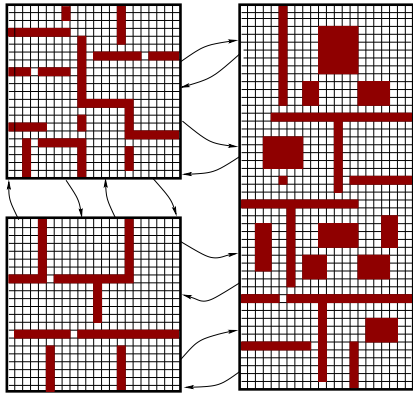


Fig. 3. A three room environment connected by stairways (arrows). Black cells indicate obstacles. The task for the agent is to navigate this environment, find an object, pick it up, move it to another locations, and drop it.

where  $\epsilon_p = \min_{V_p} \| V^* - V_p^* \|_\infty$  and

$$LV(s) = \max_a [R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V(s)]$$

When the difference between the Q-values for states in block  $B_i$  are greater than  $2(1 + \frac{\gamma}{1-\gamma}\epsilon_p)$ , then the primitive action that achieves the highest Q-value on the original state in the MDP will be added to the action space of those states that are in block  $B_i$  and block  $B_i$  is refined until it is stable for the new action set. Once no such significant difference exists, the goal will be achievable in the resulting state space according to Theorem 3. This procedure is illustrated in Figure 2.

**Theorem 3** For any policy  $\pi$  for which the goal  $G$  can be represented as a conjunction of terminal sets (subgoals) of the available actions in the original MDP  $M$ , there is a policy  $\pi_P$  in the reduced MDP,  $M_P$ , that achieves  $G$  as long as for each state  $s_t$  in  $M$  for which there exists a path to  $G$ , there exists a path such that  $F(G|s_t, \pi_P(s_t)) > \delta$ .

## VI. EXPERIMENTAL RESULTS

The main goal of this experiment is to demonstrate the potential of the proposed approach to hierarchical learning in accelerating learning in a stochastic environment. The task of the agent is to find an object in a randomly chosen cell, to pick it up and drop it in another randomly chosen location. Figure 3 illustrates the environment for this experiment. The state space consists of three grid worlds that are connected through stair ways (arrows). The dark cells represent obstacles and the actions are GoNorth, GoEast, GoSouth, GoWest, GoUp, GoDown, OpenArm, CloseArm, Pickup and Drop. Actions for

stairways are defined as sequences of length 10 of GoUp or GoDown. The cost for each single step action is  $-1$  and each action for navigation succeeds with probability 0.5 while leading to either side with probability 0.25. Actions OpenArm, CloseArm, Pickup and Drop always succeed. The reward in the goal state is 100.

To demonstrate the power of hierarchical learning, the agent is first given the task of learning a policy to move from a fixed starting location to a particular goal point. It then uses this policy to extract subgoals by generating random samples according to the learned policy. The samples are paths of length 40 and the subgoals are discovered as described in Section III. Figure 4(b) illustrates the number of subgoals that are discovered by Monte Carlo sampling. As illustrated in Figure 4(b), the

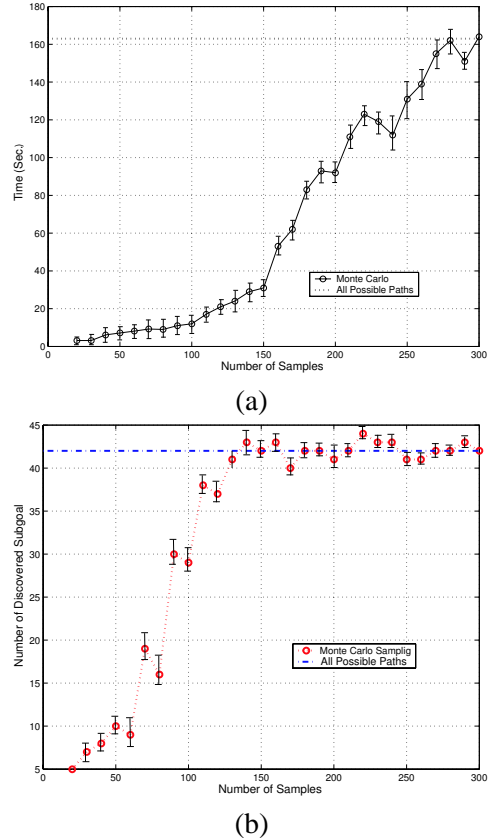


Fig. 4. (a) Comparison between the run times for subgoal discovery using the entire state space and Monte Carlo sampling. (b) Number of samples that are needed to discover all useful subgoals.

total number of samples needed to learn almost all of the subgoals is approximately 127. Figure 4(a) shows that the total time for subgoal using 127 samples is less than 30 seconds, which is 5 times faster than using the entire state space. The extracted subgoals in this experiment consist of a set of doorways, entry points of stairways

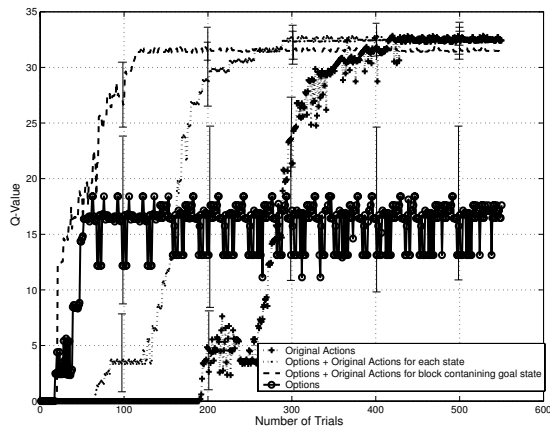


Fig. 5. Comparison of policies derived on the partition and the original state space. The policies using the options to subgoals on the partition space with primitive action refinement converge significantly faster than the optimal policy in the original state space.

and states where the agent acquired the object. Once subgoals are extracted, options for these subgoals are derived and the agent is given the final learning task described in the beginning of the section. Figure 5 shows the learning speed and quality of the learned policies with and without refinement of abstract states based on Q-value inconsistencies. This experiment shows that while the goal state is here not reachable using only the subgoal actions, almost equal performance to the original MDP is achieved with refinement based on primitive actions. Furthermore, learning on the partition space with on-line refinement of partitions according to Q-value differences in the abstract and original state representations significantly outperforms both learning on an a priori refined representation and on the original state space.

## VII. CONCLUSION

This paper presents an efficient method for autonomously constructing a hierarchical state and action space for SMDPs. To do this, it first discovers subgoals by analyzing previously learned policies for states with particular structural properties. Once subgoals are derived, it learns options to achieve these subgoals off-line and includes these into the action hierarchy available to the agent. Using the subgoal options, it then uses action-dependent state space partitioning to derive an abstract state space. Learning of subsequent tasks is addressed on the abstract state space. To ensure that the new task is learnable, the system maintains a separate value function for the original state space and refines the representation

when significant inconsistencies between this value function and the one derived on the abstract partition space are detected. Experiments using this approach show that this approach significantly accelerates learning of even complex task strategies while maintaining the quality of the found solution within predetermined bounds.

## REFERENCES

- [1] R. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.
- [2] K. Kim and T. Dean, "Solving Factored MDPs using Non-Homogeneous Partitions," *Artificial Intelligence*, vol. 147, pp. 225–251, 2003.
- [3] T. G. Dietterich, "An Overview of MAXQ Hierarchical Reinforcement Learning," *Lecture Notes in Computer Science*, vol. 1864, pp. 26–44, 2000.
- [4] R. Parr, "Hierarchical Control and Learning for Markov Decision Processes," Ph.D. dissertation, University of California, Berkeley, CA, 1998.
- [5] M. Asadi and M. Huber, "State Space Reduction For Hierarchical Reinforcement Learning," in *Proceedings of the 17th International FLAIRS Conference*. AAAI, 2004, pp. 509–514.
- [6] A. McGovern and A. Barto, "Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density," in *Proceedings of the 18th International Conference on Machine Learning*, 2001, pp. 361–368.
- [7] B. Digney, "Emergent hierarchical control structures: Learning reactive / hierarchical relationships in reinforcement environments," in *Proceedings of the Fourth Conference on the Simulation of Adaptive Behavior*, 1996, pp. 363–372.
- [8] C. Drummond, "Using a Case Base of Surfaces to Speed-Up Reinforcement Learning," in *Proceedings of the Second International Conference on Case-Based Reasoning*, 1997, pp. 435–444.
- [9] C. Boutilier, T. Dean, and S. Hanks, "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.
- [10] S. Goel and M. Huber, "Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies," in *Proceedings of the 16th International FLAIRS Conference*. AAAI, 2003, pp. 346–350.
- [11] T. Dean, R. Givan, and S. Leach, "Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes," in *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann Publishers, 1997, pp. 124–131.