Learning Options through Human Interaction

Abstract

Research in Hierarchical Reinforcement Learning shows how problems can be solved by breaking them down into a set of sub-tasks or Options. In this paper we develop a method of automatically extracting these Options by interacting with a human. We show how humans naturally think of and solve problems in a manner compatible with the Options framework. We develop an interaction mechanism combined with a supervised learning algorithm to extract human-derived Options that are generalizable across the state space of the problem. We instantiate our approach in the Taxi domain and Pac-Man (a stochastic environment) and show that the human-derived Options outperform automated methods of Option extraction both in terms of optimality and computation time. In some cases we observe that the learned Options are required to be interrupted before completion. We discuss the trade-offs of learning with Options under these circumstances and the potential use of Modular Reinforcement Learning to approach this problem in future work.

Introduction

Human beings have been known to solve complex problems. This can be attributed to our ability to perform a variety of decompositions, abstractions and generalizations on everyday problems making them easier to solve. For example, one can imagine that it would be easier to play a game of Mario by decomposing it into a set of sub-goals like "go the nearest coin", "kill the goomba", "get the flower" and so on. Such decompositions have been studied to a great extent in the field of Hierarchical Reinforcement Learning (HRL). Research in this field has taken advantage of the hierarchy present in all real-world domains and applied it to solve these problems. A few examples of applications of HRL are in the optimization of manufacturing processes (Wang and Mahadevan 1999), quadruped locomotion (Kolter, Abbeel, and Ng 2007), simulated flying (Ryan and Reid 2000) among others. These applications provide a glimpse into the advantages of using hierarchical techniques to approach real-world problems.

Within HRL, there have been several techniques proposed to take advantage of task hierarchies. In this paper, we make use of the Options framework (Sutton, Precup, and Singh 1999). An Option is a temporally extended action, which provides a layer of action abstraction to the underlying domain and offers a principled way of speeding up learning algorithms and of expressing prior domain knowledge. We believe that humans naturally tend to solve problems using this framework and that learning the humans options could significantly increase performance. Our motivation comes from the field of Interactive Machine Learning which takes advantage of humans input to speed up the performance of learning algorithms. Specifically we are interested in applying our algorithm in application domains where humans already exist and they can readily provide the necessary decomposition information.

In the Options framework, the components that make up each Option are either designed beforehand or learned automatically. Most automatic methods depend on the ability to find sub-goals by identifying bottleneck states. In large sparse domains, extracting these sub-goals can be computationally intensive, and the automated methods are prone to generating redundant options. Constructing the Options a priori by hand, if even possible, is a time-consuming task.

In light of this, we introduce an interactive method to leverage a human's ability to efficiently identify meaningful sub-goals for a given problem, and run a series of experiments to ask questions around this notion of interactive options learning.

1. **Is the way that humans decompose a problem compatible with the options framework?**

   In our first experiment, we analyze the ways in which humans decompose problems and compare this to automated approaches to Option extraction.

2. **Do the options that people provide lead to performance gain over just flat-learning the entire problem?**

   In our second experiment we develop an interaction technique and a learning mechanism to utilize humans to construct Options and their components. During the interaction, the human provides examples of an option in the form of trajectories and on this we use a supervised learning algorithm to learn the model of the option. We instantiate our framework on two domains and show how we can achieve significant speedup when using human-derived options.

3. **Do all the options execute till completion or are they required to be interrupted?**

   We briefly discuss the situation where Options are often interrupted, the methods used to approach this problem and their tradeoffs. We propose the use of Modular Reinforcement Learning as an alternative approach to this problem.

Related Work

The use of humans in collaboration with learning algorithms has been an area of extensive research. Argall et al. (2009) provides a broad overview of the use of humans in the Learning by Demonstration framework. Specifically within the RL domain, Apprenticeship Learning (Abbeel and Ng 2004), Inverse Reinforcement Learning (Ng and Russell 2000), TAMER (Knox and Stone 2009) and Social Learning (Thomaz and Breazeal 2006) describe ways in which we can use human input to learn an optimal policy for the given domain. These algorithms use the primitive one-step actions for planning and hence for complex domains are likely to experience significant computational complexity. The use of temporally extended actions or options might serve to mitigate the computational costs of planning with one-step actions.

In previous work on learning with options, they have been either carefully designed by the human or are automatically extracted. The automatic methods focus on the ability to find goals-states or bottleneck states within the domain. Heuristics for selecting goal states for options have been developed based on frequency of visit on successful trajectories (McGovern and Barto 2001), relative novelty (Simsek and Barto 2004), clustering algorithms (Mannor et al. 2004) and others. Work in Konidaris and Barto (2009) uses the concept of options to learn task hierarchies to extract task-relevant skills. In all these methods, the limitations arise due to constrained applicability of their heuristics on different domains. When using bottleneck states as a heuristic in sparse domains, it is not exactly clear whether these heuristics will work well. Also the human is required to understand the domain well enough to design and encode beforehand the heuristics useful to extract task-relevant options. In some cases the automated methods generate numerous options, making them susceptible to option redundancy. The redundancy arises when two or more options are used to perform the same sub-task or a part of it. Jong, Hester, and Stone (2008) highlight some of the disadvantages of planning with such options and their work motivates the need for more generalized options applicable to different parts of the state space.

There has been some recent work on leveraging human information into action abstraction that is close to our approach. Zang et al. (2009) describe a method of automatically extracting options from human trajectories. In their work, they assume that the features relevant to options are already known while our setup extracts these features from human interaction. Comanici and Precup (2010) develop a policy switching algorithm that iteratively decides how to switch from one option policy to another using human examples. They assume that the low-level behavioral policies are given as prior knowledge, while in our work we extract these policies from human examples. Research on extracting skills from humans in continuous domains has been looked at by Konidaris et al. (2010). In their work, they assume that best model for combining a pair of skills is the one in which they are represented individually. This gives rise to numerous skills where a single skill would be sufficient to complete the task.

Preliminaries - Reinforcement Learning

A reinforcement learning agent interacts with an environment described by a Markov Decision Process (MDP), which is a tuple $M = \langle S, A, \mathcal{T}, R, \gamma \rangle$ with states S, actions A, transition function $\mathcal{T} : S \times A \mapsto \Pr[S]$, reward function $R : S \times A \mapsto [R_{min}, R_{max}]$, and discount factor $\gamma \mapsto [0, 1]$. A policy $\pi : S \mapsto A$ is a relation that defines which action must be taken in a particular state. Every state $s \in S$ is represented by n-tuple of features, $s = (f_1, f_2, ..., f_n)$.

Within the framework of Reinforcement Learning, temporally extended actions are modeled using the Options framework (Sutton, Precup, and Singh 1999). An option consists of the following three components - $\langle I, \pi, \beta \rangle$ where $I \in S$ is the initiation set, which determines the states from which the option can be started, $\pi : S \times A \mapsto [0, 1]$ is the option's policy, a probabilistic distribution over each s, a pair that the option is defined in and $\beta(s)$ is the termination set which gives the probability of an option terminating in state s. Formally, a set of Options defined over an MDP constitutes a Semi-Markov Decision Process (SMDP) (Sutton, Precup, and Singh 1999). For an SMDP, the Markov policy over options $\mu : S \times O \mapsto [0, 1]$ maps the relation between the option to be taken in every state. Planning with options in an SMDP requires a model of their consequences - the Reward and Transition model. If $\mathcal{E}(o, s, t)$ denotes the event of option o being initiated in state s at time t, then the reward model of o for any state $s \in S$ is

$$R_s^o = E[r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} | \mathcal{E}(o, s, t)] \quad (1)$$

where $t + k$ is the random time at which o terminates. The transition model is the discounted probability of finishing option o in state s' after starting it in state s.

$$P_{ss'}^o = \sum_{k=1}^{\infty} \gamma^k P_{ss',k}^o \quad (2)$$

Here $P_{ss',k}^o$ represents the probability that option o will terminate in s' in exactly k steps. Methods of planning and computing the optimal policy with options have been discussed in Sutton, Precup, and Singh (1999). They describe methods of planning using Synchronous Value Iteration and Q-learning for SMDPs.

Domains

We apply human-option learning on two domains, the Taxi Domain (Dietterich 2000) and Pac-Man.

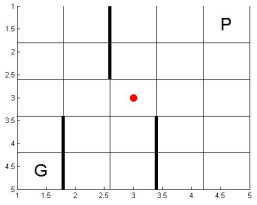

Figure 1: Taxi domain showing a sample set of passenger and destination Locations

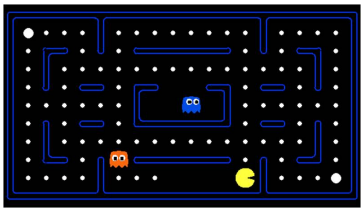

Figure 2: Pac-Man domain with two ghosts

Taxi Domain

In the Taxi domain (Dietterich 2000) shown in Figure 1 , the agent is a taxi that navigates in a grid with obstacles that the taxi cannot pass through. The goal is to pick up and deliver a passenger at one of four predefined locations. The taxi can move deterministically in one of the four cardinal directions, pick up the passenger when the taxi is in the same cell, and drop off the passenger. The taxi starts in a random cell. A 5x5 grid is used. Every state is defined by a set of 10 features, some of which include the taxi location in the grid, location of the passenger, location of the destination, passenger picked up, present on the passenger cell and so on. The domain has a total of 500 states. Every state has a reward of -1 except for the goal state which has a reward of 1.

Pac-Man

In this game, the agent, Pac-Man [1] shown in Figure 2 navigates through the maze (using the 4 cardinal directions) and eats all the white dots (food). Pac-Man must avoid the ghosts (two of them) who can kill him. On eating a power pill (two are used), Pac-man can eat the ghosts and can earn extra points by eating fruit which appears in the maze on occasion. Eating the power pills changes the color of the ghosts, their direction and makes them move slowly. The game ends when all the lives have been lost. Pac-Man starts with three lives. The features for Pac-Man include agent position, ghost position, nearest food pellet, position power pill and ghost runaway timer. We used a discretized grid of size 9×18.

Extraction of Domain Options

In the first experiment we focus on acquiring the different ways in which humans decompose problems and to understand their relation to Options. Our hypothesis is that when solving problems, humans naturally think in terms of options and not the low level actions space. Our study involved 10 participants. Each participant was assigned one of the above mentioned domains. We described the domains to the participants as an interactive game with buttons representing the low level actions. They were allowed to play the respective game until they were familiar with the controls. On completing this, the participants were asked to modify the buttons in a way that would make winning the game easier and faster. They were restricted to creating a fixed number of buttons, at most 3 for Taxi and at most 4 for Pac-Man.

The results of this study are reported in Table . The button names given in the table appear to be similar to Options. Each of those buttons can thought of as a sequence of primitive actions. We also compared Human Options to automated means of extracting Options (Stolle and Precup 2002). The automatic option extraction algorithm works by extracting bottleneck states in the domain. A sample of human options and the automatically extracted options is shown in Figure 3. The black dots show the locations identified by the algorithm as target states.

The automated algorithms generate 9 options. They are three more than the number base actions available. Four for going to each of the passenger locations and performing a pickup actions, another four to dropping off the passenger at one of the other four destinations and perform a dropoff. It is important to note here two characteristics of the Options

Taxi Domain	
Button name	Percentage of participants who gave this button
Go to the passenger and pickup	60%
Go to the destination and dropoff	60%
Go to the passenger	40%
Go to the destination	40%
Pickup/Dropoff	40%
Move away from the obstacles	20%
Pac-Man	
Go to the closest food	100%
Avoid Ghost	100%
Go to the nearest power pellet	100%
Eat the ghost	40%

Table 1: Human-defined Options for the two domains

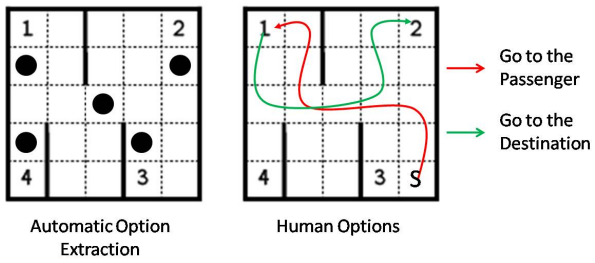

Figure 3: Diagram showing Automatic Sub-goals on the left and Human Options on the Right

[1] Available open source at the California Berkeley website

that humans provided. The first being that some buttons like "Avoid Ghost" do not have a clear end state. It is not clear how to construct an option for this button. Secondly the buttons given by the humans depend on specific features. The button "go to passenger" can be considered as a generalized option that depends on the passenger location.

After having defined the buttons in the two domains, we wanted to assess their usability and whether they help in a performance gain over learning the flat MDP. In order to do that, we conducted the next experiment where we learnt the options that humans provided and use them to plan in the MDP.

Learning Options from Humans

Having learned that human's can decompose the task domain in a way that looks compatible with Options, the next step is to learn the given Options. In this section we first describe our approach to soliciting examples of each Option from the human. Then we describe how we learn the components of an option using these examples. Finally, we describe an experiment that evaluates these learned Options compared to learning without Options and learning with automatically generated Options.

Interaction Mechanism: Soliciting Option Examples

We assume that the human is familiar with the optimal policy (using primitive actions) for the domain under consideration.

The human interacts with the system by providing option-feature dependencies and example trajectories. For each option to be learned, the domain is required to be setup such that only the option-relevant features are activated. This is necessary because in adversarial domains, learning an option might be affected by the presence of other features. For example, in the game of Mario, learning the option "go to coin" is independent of the position of the turtle or distance to the nearest pit. However having the turtle and pit present in the game reduces the ability to focus on the coin feature and directly affects learning of the required option. Therefore we suppress irrelevant features from being represented in the domain during the "go to coin" option learning. In order to extract such option-relevant features, it is necessary for them to be represented in a manner that is verbalizable to the human. Following this, through a series of *Yes* or *No* questions, as shown below, we can setup the domain for the option Learning.

Does the "go to coin" option depend on "position of mario"? - Yes

Does the "go to coin" option depend on "position of coin"? - Yes

Does the "go to coin" option depend on "position of turtle"? - No

With only the option-specific features activated, the humans are now required to provide an example of the expected function of the option. Given a sequence of start states, they show sample trajectories in the form of state, action and reward pairs, $s_0a_0r_0, s_1a_1r_1, ...s_na_nr_n$ from start to end, of the expected policy of the option. The state are represented as a vector of features $s = (f_1, f_2, ..., f_n)$. The human uses the primitive actions when providing trajectories. The need for different start states and the number of sample trajectories required are computed by the learning algorithm detailed in the next section. This interactive process is repeated for all the human-options required to be learned.

Learning Algorithm

In order to learn each option, we are required to learn its policy as well its applicability to different parts of the state space. To facilitate this, the sample trajectories acquired from the human are fed into a multi-class decision tree learner to learn a model of the options policy, π and a model of the initiation set *I*. We can use the members of the initiation set to populate the termination set β. From the human trajectories, we use the visited states as part of the initiation set, the actions taken in these states to represent the options policy and the reward acquired to calculate the average return of executing the option.

Algorithm 1 Human Option Learning

for each option O_i to be learned **do**
  Obtain the set of state features, $f_1, f_3, ..., f_m$ relevant to learning the option
  Let the human teacher have policy π_h for teaching the option
  Present the domain with only the option-relevant features activated
  Initialize average return = 0
  while average return below threshold and cross-validation error above threshold **do**
    Select start state that promotes exploration
    Human provides a sample trajectory, $s_0a_0r_0, s_1a_1r_1, ...s_na_nr_n$
    Compute average return, R_i
    Build decision tree model for the policy, π_o and the initiation set I_i.
    Compute average return from learned model and the cross validation error
  end while
  Use remaining states as the Termination set, β_o
end for

The input to the decision tree learner is a sequence of states (represented by a vector of features) followed by either the action taken (label) or a binary label indicating its membership to the initiation set. Given a set of N such examples for an option, the policy and initiation set model learned from the decision tree are tested to assess their respective optimality. The policy model learned is used to generate sample trajectories. The average return from these sample trajectories is compared to the average return acquired form the human demonstration and this threshold is indicative of the need of further examples from the human. The cross validation error of the decision tree model for

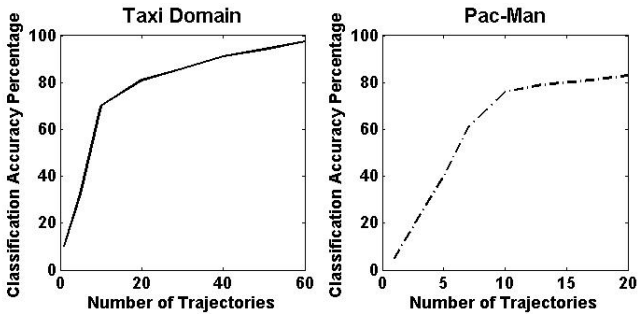

Figure 4: Diagram showing Classification Accuracy with increasing number of trajectories

Model	Computation Time (seconds)	Average Reward of Computed Policy
Taxi Domain		
MDP	17.45	-2.294
SMDP (Human Options)	10.45	-4.294
SMDP (Automated Options)	25.75	-4.311
Pac-Man		
MDP	-	-
SMDP (Human Options)	60.45	-1.094
SMDP (Automated Options)	120.47	-7.212

the initiation set represents how good the sample set (visited states) is. A higher value indicates that more of the state space is required to be explored.

In this manner, we continue to acquire trajectories from different start states from the human until both the criteria for policy and the initiation set are satisfied. This process is repeated for all the options. This method of option learning allows us to create a generalized option that can be utilized across the state space. Algorithm 1 provides a high-level overview of the important segments of human-option learning -

Experimental Design

After having acquired the names of all the buttons that people wanted to add in the first experiment. In this second experiment, the participants were then given the opportunity to teach the function of a subset of them. The buttons to be taught were chosen from Table as the ones with the highest percentage. For the Taxi domain, the top 2 buttons were used and for Pac-Man, the top 3 buttons were used. As described in Section , the participants were asked to show sample trajectories, from start to end, of the expected functioning of these buttons and a decision tree was used to learn the Option policy.

Each person was asked to demonstrate 2 or 3 buttons depending on the assigned domain. The required number of trajectories for each of these buttons was dependent on the cross-validation of the decision tree and the average reward acquired from the computed model. In an iterative manner, we learn the components of the individual human options. The details on this portion has been discussed in Section .

In order to evaluate the learned option model in each of the two domains, we compared the learned model to the optimal policy. We combined all the options learned from one person and then performed a classification test to understand how much of the state space is covered by the options. Figure 4 shows the effect of the number of human trajectories on the classification accuracy. We see that the accuracy increases with larger amounts of training data, however the accuracy does not reach 100% more so in the Pac-Man domain. This can be explained by several reasons - persistent error propagation, sub-optimal human trajectories and most importantly incomplete options. In the Pac-Man domain, the options we used did not cover parts of the state space where the agent could eat the ghost. Therefore the learned model performed poorly in these parts of the state space. This notion of incomplete option will be discussed further in the next section. To reach the maximum number of trajectories, the participants, on average, took no more than 15 mins in Taxi and 20 mins in Pac-Man.

Evaluation of the Policy Obtained

To evaluate the performance of the human options, we compared the performance of SMDP (using Options) value function to the MDP value function in terms of computation and in terms of efficiency.

Our hypothesis is that the time taken to learn the policy for Taxi and for Pac-Man using the human options will be significantly faster than planning with the base actions and also faster than planning with automatically extracted options. To make this comparison, we extract the reward model and the transition model as shown in Section and used Synchronous VI as described in (Precup 2000) to compute the optimal value function for the Options. We sampled the policy acquired and computed the average reward acquired. We observe that the Options selected by humans were more efficient and than the ones obtained automatically. The smaller number of Options was a result of the decision tree representation of the Option policy. This factored significantly into the speed up achieved during value function computation.

Table summarizes the computation time and optimality of each of the different inputs to Value Iteration. It is clear that there is significant speed-up in convergence however the computed value is never as good as when using primitive actions. This can be explained by the use of a sub-optimal model for Option during planning. The Automated Options do not do clearly as well as Human Options but they are faster when compared to the primitive actions. This is attributed to sub-optimal selection of bottleneck states. Figure 5 shows the learning curve when planning with Q-learning. The graph shows the speed-up in learning that can be achieved when using human options.

Discussion

From our experiments there were several insights that we gained about learning from human options. The first was that

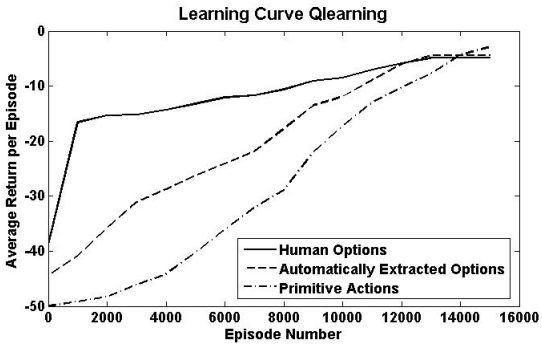

Figure 5: Average Return with Q-learning on primitive actions and human options.

the options given by humans do not necessarily form a complete set in that it is not always possible to learn a policy for the entire domain using only the human options as we noticed in the Pac-Man domain. This incompleteness of the human option set is indicative of the reward function used by the humans to complete the task. The inclusion of an option by the human indicates that it is relevant in the human's reward function. It is important to note here that the human's reward function can be different from the reward function defined for the task.

Secondly, in our experiments we allowed each human to devise the set of options without receiving any feedback about the options' performance. This leads to a question of whether humans might be inclined to come up with new options or maybe refine their current ones given feedback. There is also a question of how well humans can represent their options. In a few cases from our experiments, we noticed that the human trajectories were sub-optimal and therefore a number of samples were required in order to learn a good policy for the option.

In the domains that we tested the algorithm, we noticed quite often that the learned Options were required to be interrupted (terminated before completion). For example, in the Pac-Man domain, the Options of "go to the nearest food pellet" and "move away from ghost" are often never run till completion (Refer to Table). At time step t, a decision needs to be made to switch from one Option to the other in order to stay alive. There have been methods developed to approach this problem - a heuristic based on the value function or on the number of interruptions(Precup 2000), a parametric form of the termination condition (Comanici and Precup 2010) and policy iteration (Precup 2000). Each of these methods takes into account certain assumptions about the domain and it is not clear whether there is an optimal mechanism for when to interrupt an Option for the general case. This is especially true for Options that have not clear termination state - "move away from ghost".

Here we propose a different way of thinking of interrupting Options. Consider Options to be individual modules that are running simultaneously throughout the problem. In this setting, it would become necessary to make a decision on which module to be executed at time step t. Learning with such competing modules can be viewed as a Modular Reinforcement Learning problem (Samejima, Doya, and Kawato 2003). Given the detailed study in this field, we could utilize the existing framework to approach the problem of Option Interruption. Another approach here would be to utilize the help of a human to learn the necessary heuristic for Interruption.

Conclusions

We learnt how humans decompose problems and found that they resemble the framework of options to a great degree. We were able to learn options for two domains using the help of human given examples. With the help of a decision tree, we were able to learn the components of the option. We were able to efficiently generalize options to other parts of the space and we obtained significant speed-up in planning time and were able to perform optimally. Learning options in this manner gives the user and designer a significant advantage in that these models needn't be give apriori and they can be built for most real world domains. We also introduced the problem of interrupting options from a new perspective and we are interesting in pursuing that for future work.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *ICML*.

Argall, B.; Chernova, S.; Veloso, M. M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Comanici, G., and Precup, D. 2010. Optimal policy switching algorithms for reinforcement learning. In *AAMAS*, 709–714.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res. (JAIR)* 13:227–303.

Jong, N. K.; Hester, T.; and Stone, P. 2008. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, 299–306.

Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: the tamer framework. In *K-CAP*, 9–16.

Kolter, J. Z.; Abbeel, P.; and Ng, A. Y. 2007. Hierarchical apprenticeship learning with application to quadruped locomotion. In *NIPS*.

Konidaris, G., and Barto, A. G. 2009. Efficient skill learning using abstraction selection. In *IJCAI*, 1107–1112.

Konidaris, G.; Kuindersma, S.; Barto, A. G.; and Grupen, R. 2010. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *NIPS*.

Mannor, S.; Menache, I.; Hoze, A.; and Klein, U. 2004. Dynamic abstraction in reinforcement learning via clustering. In *ICML*.

McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*, 361–368.

Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *In Proc. ICML*, 663–670.

Precup, D. 2000. *Temporal abstraction in reinforcement learning*. Ph.D. Dissertation, UMass Amherst.

Ryan, M. R. K., and Reid, M. D. 2000. Learning to fly: An application of hierarchical reinforcement learning. In *ICML*, 807–814.

Samejima, K.; Doya, K.; and Kawato, M. 2003. Inter-module credit assignment in modular reinforcement learning. *Neural Networks* 16(7):985–994.

Simsek, Ö., and Barto, A. G. 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML*.

Stolle, M., and Precup, D. 2002. Learning options in reinforcement learning. In *Lecture Notes in Computer Science*, 212–223.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112(1-2):181–211.

Thomaz, A. L., and Breazeal, C. 2006. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*.

Wang, G., and Mahadevan, S. 1999. Hierarchical optimization of policy-coupled semi-markov decision processes. In *ICML*, 464–473.

Zang, P.; Zhou, P.; Minnen, D.; and Jr., C. L. I. 2009. Discovering options from example trajectories. In *ICML*, 153.