The Thirty-Third International
FLAIRS Conference (FLAIRS-33)

Discovering Hierarchy for Reinforcement Learning Using Data Mining

Dave Mobley
Department of Computer Science
University of Kentucky
Lexington, KY 40508
dave.mobley@uky.edu

Brent Harrison
Department of Computer Science
University of Kentucky
Lexington, KY 40508
harrison@cs.uky.edu

Judy Goldsmith
Department of Computer Science
University of Kentucky
Lexington, KY 40508
goldsmit@cs.uky.edu

Abstract

Reinforcement Learning has the limitation that problems become too large very quickly. Dividing the problem into a hierarchy of subtasks allows for a strategy of divide and conquer, which is what makes Hierarchical Reinforcement Learning (HRL) algorithms often more efficient at finding solutions quicker than more naïve approaches. One of the biggest challenges with HRL is the construction of a hierarchy to be used by the algorithm. Hierarchies are often designed by a human author using their own knowledge of the problem. We propose a method for automatically discovering task hierarchies based on a data mining technique, Association Rule Learning (ARL). These hierarchies can then be applied to Semi-Markov Decision Process (SMDP) problems using the options technique.

Introduction

Reinforcement learning (Sutton 1988) is a common approach to solving sequential decision problems where agents learn through trial-and-error interaction with their environment. Guidance is given to the agent through an environmental *reward* signal, which is a numeric value that represents how good or bad a state is. As the space of possible environmental states becomes larger, and rewards become sparse, reinforcement learning agents can struggle to learn optimal behavior. In these situations, providing a structured representation of the task to be completed in the form of a task hierarchy can speed up learning. This technique of creating a set of tasks for the agent to take is called Hierarchical Reinforcement Learning (HRL).

In HRL, the agent is given access to prior knowledge about the structure of the task to be completed. Often, these structures are hand-crafted, and can be quite complex. Learning these hierarchies automatically can be a difficult task, but has the potential to greatly reduce the burden on the author to create high quality hierarchies for learning. In this paper, we introduce a method for discovering task hierarchies using a popular data mining technique, association rule learning (ARL).

Association Rule Learning looks at sets of items and finds commonality between them based on their item composition. A specific metric that measures how often an item or itemset exists in the collection of items is called *Support*. By applying this technique to an RL trace, an agent can look at trace sets as collections of items and find subsets of states that were traversed in those trace sets. By using Support, if a state or set of states appears in all successful traces, then the state is likely to be required. These states that are likely required are known as *chokepoints*.

There is no guarantee that the state sets found by Support are actual chokepoints. It is possible that an example might exist where a specific chokepoint set of states is not needed. As more traces are sampled without a counterexample, this becomes less likely, but still exists as a possibility.

ARL chokepoints can be used and ordered as waypoints. A *waypoint* is a set states that will be visited by the agent at some point during the trace. A hierarchy can be created with those waypoints, where subtasks are the steps to transition from one waypoint to the next. This collection of subtasks defines a hierarchy that was successful in the sample set of traces and can be used to learn a policy for each subtask. One of the key motivators for discovering hierarchy automatically is that it frees up humans from defining the hierarchy.

Background

There are a number of algorithms that currently utilize hierarchies to improve the learning speed of an RL problem. Examples are options (Sutton, Precup, and Singh 1999), MAXQ (Dietterich 2000), and abstract machines (Parr and Russell 1998). Each of these improve on naive Q-learning, but require the development of a hierarchy of subtasks beforehand. There has been work to automate the discovery of hierarchies through randomly generating options (Stolle and Precup 2002) and using prior knowledge to find candidate states for starting and terminal states of options (McGovern and Barto 2001), or by trying to locate chokepoints, required pathways, or by using reachability (Şimşek and Barto 2004; Şimşek, Wolfe, and Barto 2005; Dai, Strehl, and Goldsmith 2008). Although these approaches are each suitable under their own conditions, we propose a general technique to use prior knowledge from past successful attempts and apply ARL to build a hierarchy of options.

To do this, the problem at hand is assumed to be an MDP. A *Markov Decision Process (MDP)* is a tuple (S, A_s, R_a, T_a), where S is the set of states in the problem, A_s is the set of actions available to be taken in a given state $s \in S$, R_a is the reward that will be received immediately when transitioning to next state, s' because of action $a \in A_s$. For a given *episode*, a series of states traversed over discrete time steps until a terminal state is reached in the problem, we write s_t for the current state at time step t of the episode. Markov Decision Processes are stochastic in nature, so there is a transition probability T_a such that at time t given state s_t and action a_t there is a probability distribution over next states. This can be written as $T_a(s_t, s_{t+1}) = Probability(s_{t+1} = s' | s_t = s, a_t = a)$.

With an MDP, the *options* technique of hierarchical reinforcement learning can be applied (Sutton, Precup, and Singh 1999). An *option* has three parts: (i) a set of starting states, I, (ii) a policy of actions π, (iii) a termination probability β. To complete some subtask, once an agent enters a starting state, it must execute until it terminates stochastically as per β. To consider using options where they can call other options, the MDP framework must consider variable time length actions, i.e., a Semi-Markov Decision Process (SMDP), so that rewards over a time interval can be applied. States mapped to an option with a single time step are called *primitive* options. Because options can call options, this allows the creation of hierarchy.

Our technique creates options based on prior successful traces, determining chokepoints, and creating a set of options to be used to learn a global policy for the problem.

Methodology

We use successful traces as prior knowledge about the problem to infer an initial hierarchy using ARL to be applied to an options-based HRL algorithm. First we consider ARL and then provide extensions to ARL that allow us to discover chokepoints in traces. Our ARL algorithm discovers states that recur during successful completion of a learning task. It then uses this state set as the starting point in constructing the hierarchy, splitting traces into two halves, those states visited before the chokepoint, and those visited after. It recursively calculates a new set of chokepoints for the prior and post sets of traces. It does this until no more chokepoints are found. After ordering this set of chokepoints, we convert them into options that can be used by an HRL algorithm to learn a policy for the problem.

Association Rule Learning

As mentioned, ARL is a useful statistical tool that was originally used to discover relationships between elements in itemsets. This data mining technique derives rules that suggest some item or items in a collection of transaction data are likely to be present given the presence of other items (Agrawal, Imieliński, and Swami 1993).

An *itemset* is a subset of items I containing one or more items of interest. *Support* is the frequency an itemset appears in a set of transactions. If the support for an itemset has a value of 1.0 this means that the itemset exists in every transaction in the set of all Transactions. Because it exists in every transaction in the database, it is likely, though not guaranteed, to be required.

Applying Association Rule Learning for Automatic Hierarchy Discovery

To use ARL for discovering chokepoints, we generate successful traces by random walking the environment. By formulating a trace as a transaction, or collection of itemsets, it is possible for us to use support to identify chokepoints in a transaction. Itemsets (states visited) are more likely to be chokepoints if they have high support. Once chokepoints have been determined, they can be used as a set of waypoints. Itemsets with smaller numbers of states are preferred over those with larger sets because they are simpler and make a more abstract rule.

The algorithm divides the traces, separating them at the chokepoint. Those before the chokepoint and those after the chokepoint each form two new sets. The subtraces can now be mapped into new sets of transactions, forming a database based on subtraces before the chokepoint and those after the chokepoint. Each new database of subtasks is recursively operated on in the same manner. The traces are tested for a chokepoint. The chokepoint is removed and the pre- and post-chokepoint traces again form new databases. Once the recursion finishes, a tree of chokepoints is created which can be converted to a sequential list.

An extension of the support idea is to consider if multiple itemsets together form a support = 1. A two-tuple is considered a *weaker* itemset than a single element itemset in that it requires more states to meet the burden of support. It is weaker in the sense that it forms a weaker constraint on the agent.

If a set is a chokepoint, then any superset is also a chokepoint, but because the superset is weaker, it should not be considered as it produces a more complex hierarchy and is thus less precise. A balance should be considered as to how many states are used to compose chokepoints. This will have a direct impact on the effort necessary to calculate a hierarchy. The factors that have a large impact on the scalability of this algorithm are the number of states s and the maximum size of the itemset of a chokepoint to be considered, k. The algorithm will need to compute from $\binom{s}{1}$ up to $\binom{s}{k}$ until it finds a chokepoint or exhausts all possibilities. For the ARL algorithm for hierarchy generation, this would be $\sum_{a=1}^{k} \binom{s}{a}$. The complexity of this sum is the largest term in the summation. The assumption is that the state space is very large and that the number of states that represent a chokepoint is small, i.e., $k < s - k$. This gives a complexity on the order of $\mathcal{O}(s^k)$. The complexity grows exponentially as the size of the itemset for chokepoints grows, therefore in the ARL experiments itemset size is limited to at most three states.

On some problems, like the Rooms problem presented by (Sutton, Precup, and Singh 1999), the size of the chokepoint set k may grow very large very quickly. In these scenarios, a modified version of the algorithm is used. To find reasonable chokepoints, though not necessarily required chokepoints, a threshold is set and the Support of a given itemset need only exceed this threshold to be counted. Thus if a set of states appears in only 80% of traces, if the Support threshold is only .75, then the itemset is considered.

Experiments

To evaluate the quality of our approach, we compared agents trained using our hierarchies discovered with ARL against hand-crafted hierarchies in two benchmark RL problems: the taxi problem from (Dietterich 2000) and the rooms problem from (Sutton, Precup, and Singh 1999). We also compared against a variety of hierarchical and non-hierarchical baselines.

Taxi Experiment

The Taxi Problem is a grid-based environment where the agent controls a taxi. The grid itself is a 5x5 grid with some barriers inside the grid. It also contains four sites for pickup and dropoff of a passenger. Figure 1 shows an example problem. The agent must perform the task of taking a passenger to their desired destination. In this problem, movement from location to location is stochastic where there is an 80% probability of going in the selected direction and 10% probably for moving to each of the orthogonal directions.

In the first experiment, a set of 1000 successful traces for the domain space were used. Association Rule Learning was applied to the collection of traces, sorting them first into their distinct variants of the problem. With the passenger possibly picked up at one of the 4 locations and put down at one of the 4 locations, there could be 16 variations of the problem. The goal was to see if specific states would show up in itemsets 100% of the time. Itemsets of at most 3 elements were considered.

To gather initial traces, an agent randomly navigates the environment, limited to 128 moves, where successful traces were saved. A program takes the trace file and performs the ARL calculation on it to generate a hierarchy. There are 16 subproblems, so there are 16 collections that are to be processed and 16 hierarchies will be created, one for each subproblem. The experiment compares the results of ARL with those of other techniques, including simple Q-learning and randomly selected options.

A second experiment was used to determine how many traces were typically needed for the ARL agent to accurately form a hierarchy of chokepoints. This would give an indication of how much foreknowledge is required on this specific problem to develop a good hierarchy. This was done by limiting the number of traces that were available to the ARL agent and comparing that to the set of chokepoints that a person recognized as being required in the problem.

Rooms Experiment

The Rooms Problem is also a grid-based environment where the agent must navigate from its randomly chosen starting point to a randomly chosen destination. Unlike the taxi problem, this introduced two key aspects to challenge the ARL algorithm, specifically the problem is much bigger so that there are many more than 16 basic types of problems, and

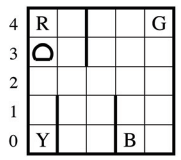


Figure 1: An example of the Taxi Problem. A 5x5 grid with the agent at location (0,3).

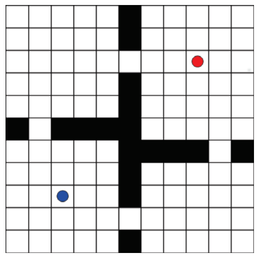


Figure 2: An example of the Rooms Problem. An 11x11 grid with the agent at location (2,2) in blue and the destination at (8,8) in red.

also there are examples of the problem where chokepoints might require itemsets larger than size 3. An example of the Rooms grid is in Figure 2.

There are 10,816 variations in this problem. If you imagine the start location and the end location are both in one of the same rooms, such as start at (6,5) and destination at (8,8), many states would need to be grouped to form an itemset. Because testing itemsets grows exponentially, we focus on smaller sets for chokepoints, and thus still limit them to groups of at most 3 states, but ease up the support requirements from being always required, to being *mostly* required. A threshold of .75 was used in this particular problem, so any itemset of states that appeared in a given problem at least 75% of the time was used as a chokepoint in training.

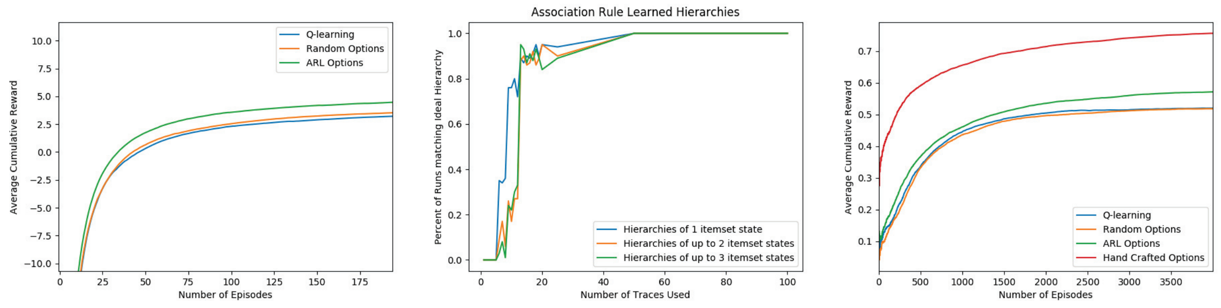
To compare the ARL algorithm with other solutions, Q-learning, random options, and a very good hand-crafted set of options all from the original paper were used as baselines.

Results

The first experiment for the Taxi problem was a comparison of the ARL algorithm against Q-learning and random options. All algorithms have all primitive options available to them. They each will construct an equivalent number of their own options which are available to the agent as well.

After running the agent for 200,000 episodes under each of the algorithms, the Q-learning algorithm creates a baseline for reference in Figure 3a. As shown by (Stolle and Precup 2002), random options outperforms the baseline Q-learning. Using ARL, it can be seen that ARL quickly jumps ahead, performing as well as the more basic algorithms in less than half the episodes, achieving the same results in as little as 75,000 episodes. The results are based on averages for all 16 problems over the 200,000 episodes.

For the Taxi Problem, an analysis was done testing the ARL algorithm such that it chose a random chokepoint as a



(a) Random Options, Q-learning Options, and ARL Options on the Taxi Problem.

(b) Traces required to learn ideal hierarchy for the Taxi Problem.

(c) Hand crafted, Random, Q-learning, and ARL Options on the Rooms Problem.

Figure 3: Results for ARL Options.

root of the hierarchy, and then built a hierarchy around that random chokepoint. If the hierarchy devised was the same as a logically ideal hierarchy, then it was said to pass. A logically ideal hiearchy in this case is a hierarchy where each of the chokepoints is logically deduced by a person for the given subproblem. The algorithm was run on a varying number of traces and sampled for 100 random starts. Figure 3b shows the results of 1-state, 2-state, and 3-state chokepoints. For this particular problem it took between 25 and 50 traces for the ARL algorithm to correctly create an ideal hierarchy every time in 100 starts.

Similar to the Taxi problem, the Rooms problem was also compared against Q-Learning and random options. The hand-crafted algorithm from the original paper was also used to compare against a human created solution. Figure 3c shows the initial results after running each algorithm for 4000 episodes. We perform the experiment 20 times on each algorithm and average the results. In this particular scenario, random options did not perform as well because we are limiting it to the same number of options as the other algorithms. The random options do not provide enough extra information and appear to cause the agent to wander before catching up to Q-learning. It can be seen that as before, even without Support of 100%, but rather only 75%, the chokepoints determined were still sufficient to allow it to learn much faster than the baser algorithms. ARL performed almost as well as the baseline algorithms after only 1000 episodes compared to 4000 episodes for the others.

Hand-crafted options performed significantly better because the human policy is more abstract, creating two options for each state, and the terminating states for those two policies are the two exits in the corresponding room. This abstraction is very effective because the agent has a choice to exit the room at each step and learn a way to get to the destination that much faster.

Conclusion

In this paper we've discussed the idea of using prior knowledge through successful past traces as a mechanism to develop an algorithm that can create a hierarchy which can be used to train a reinforcement learning agent. As shown in the Taxi and Rooms problems, this approach does outperform more traditional algorithms while providing the benefit that a human does not have to create a hierarchy for the agent. Future work in this area suggests that coming up with more focused options, like those that humans create, could still dramatically improve the performance of the agent. This could entail the need for incorporating additional chokepoint identification mechanisms as well as developing better heuristics to group chokepoints together to overcome the exponential limitations that result from testing all combinations of chokepoints to determine a good itemset.

References

Agrawal, R.; Imieliński, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, 207–216. ACM.

Dai, P.; Strehl, A. L.; and Goldsmith, J. 2008. Expediting rl by using graphical structures. In *Proc. AAMAS*, 1325–1328.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. *University of Massachusetts Amherst Computer Science Department Faculty Publication Series*.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, 1043–1049.

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

Şimşek, Ö.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proc. ICML*.

Stolle, M., and Precup, D. 2002. Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, 212–223. Springer.

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

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.