Literature Review

Stephen Robertson
Department of Computer Science
Rhodes University
Grahamstown, 6140
g02r3566@campus.ru.ac.za

May 30, 2005

Abstract

Reinforcement learning is an attractive method of machine learning. However, as the state space of a given problem increases, reinforcement learning becomes increasingly inefficient. Hierarchical reinforcement learning is one method of increasing the efficiency of reinforcement learning. It involves breaking the overall goal of a problem into a hierarchy subgoals, and then attempting to achieve each subgoal separately. This paper discusses various approaches at hierarchical reinforcement learning, and of automatic subgoal discovery by agents.

1 Introduction

1.1 Hierarchical Reinforcement Learning

Reinforcement learning has been applied to various problems with great success. However, because of the table structure of state action pairs, with large state spaces, reinforcement learning becomes increasingly inefficient. A method for dealing with this is that of hierarchical reinforcement learning. Hierarchical reinforcement learning is an approach to reinforcement learning which splits the global goals of a reinforcement learning agent up into smaller subgoals, and then attempts to tackle each subgoal separately. By doing this the state space is decreased and therefore the efficiency increased. In [Kaelbling, Littman and Moore, 1996], hierarchical reinforcement learning is mentioned as a good way of increasing efficiency in reinforcement learning.

In hierarchical reinforcement learning, conventional methods of reinforcement learning are used for learning at each level of the hierarchy, and therefore hierarchical reinforcement learning does not lose the desirable qualities of reinforcement learning. Reinforcement learning is very good at dealing with stochastic errors which might creep into the description of the state. A hierarchical reinforcement learning approach to navigating a gridworld was

used in [Bakker and Schmidhuber, 2004], and to test its ability to handle stochastic errors, the latter were introduced into the description of the state. The algorithm dealt with them very well, and was almost as efficient as the errorless example.

For reinforcement learning, the current choices of actions by an agent for its different states is known as its policy. In hierarchical reinforcement learning, with the breaking up of an overall goal into subgoals, comes the introduction of subpolicies. These subpolicies are a common aspect in literature on hierarchical reinforcement learning and are given different names by various researchers, such as actions, options, skills, behaviours and modes [Barto and Mahadevan, 2003].

Various methods for hierarchical reinforcement learning exist, and these will be discussed in greater detail in the rest of this paper.

1.2 An Application of Hierarchical Reinforcement Learning

Hierarchical reinforcement learning has been applied to some complex problems with great success. In [Pfeiffer, 2004] a hierarchical reinforcement learning approach was used to create a program which plays a board game called The Settlers of Catan, which is a popular modern board game in the German-speaking area. It is a very complex board game and therefore a flat reinforcement learning approach would have been inefficient. In their approach they use hierarchical reinforcement learning and model trees for value function approximation. Both Q-learning and SARSA are used as the conventional reinforcement learning algorithms at different stages of the learning process. Self-play is used for the actual training process. Self-play is a method where an agent is played against a copy of itself. Self-play is used for learning strong policies in adverserial domains. The major drawback of self-play though is that without sufficient exploration, agents only learn to play against a very small set of policies.

1.3 Value Function Approximation

Another way of dealing with very large state spaces is to use value function approximators, which try to map the state-action pairs to some form of function rather than storing all state-action pairs in a table, as described in [Sutton and Barto, 1998]. One of the most impressive applications of reinforcement learning to date is that of TD-Gammon by [Tesauro, 1995], which made use of value function approximation. TD-Gammon is a backgammon playing program that used the reinforcement learning algorithm TD(λ), and nonlinear function approximation using a multilayer neural network trained by backpropagating TD errors. This program was very successful and by the third version was playing at the level of the best human players in the world. The program played the opening moves differently from the conventional way of playing them, to great success, and the best players in the world have now altered their way of playing to be like that of TD-Gammon.

There are various methods of value function approximation and not all of them are effective for all situations. [Sutton and Barto, 1998] describes how for bootstrapping methods of reinforcement learning, value function approximation only works over a rather narrow

range of conditions. Even with function approximation, with increasingly complex problems, reinforcement learning gets increasingly inefficient.

Value function approximation is an alternative approach to reducing the state space, and can also be used in conjunction with hierarchical reinforcement learning, as in [Pfeiffer, 2004]. However, for this paper, I will not discuss value functions any further, but see [Sutton and Barto, 1998].

2 Different Approaches to Hierarchical Reinforcement Learning

As [Bakker and Schmidhuber, 2004] highlights, most studies on hierarchical reinforcement learning assume that the actual hierarchical structure is given by the designer, and the agent is allowed to learn within this concrete structure. For most problems, where the hierarchical structure can easily be extracted, this is sufficient. However, in some cases, it might be desirable for the agent itself to be able to identify subgoals, as this minimises the designers role in the system. It also makes the learning process more flexible, as an agent can automatically learn a new hierarchical structure, if the hierarchical structure is suddenly changed. In this section I will give a brief overview of some commom algorithms of hierarchical reinforcement learning, starting with those in which the hierarchical structure is supplied by the designer, and then those in which the agent attempts to uncover the hierarchical structure for itself.

2.1 Navigating Continuous Spaces using Hierarchical Reinforcement Learning

[Borga, 1993] describes his algorithm for hierarchical reinforcement learning in which two different hierarchical levels are given by the designer. The lower level is made up of certain actions, and the higher level is a set of strategies, each strategy consisting of a series of actions. The algorithm he describes is specifically for navigating continuous environments and hence the possible actions are a set of vectors, specifying in which direction to move. He gives an example problem, to which his algorithm could be applied, of trying to walk around an obstacle. In this case, an action would be a step in a certain direction and two different strategies would be to either choose a path to the left of the obstacle, or a path to the right of the obstacle.

2.2 Feudal Reinforcement Learning

[Dayan and Hinton, 1993] give an approach to hierarchical reinforcement learning, in which the hierarchical structure is given by the designer. They call it Feudal reinforcement learning and in it, they apply their algorithm to a task in which an agent has to navigate a gridworld maze which contains in it a barrier. The hierarchical structure is designed by the

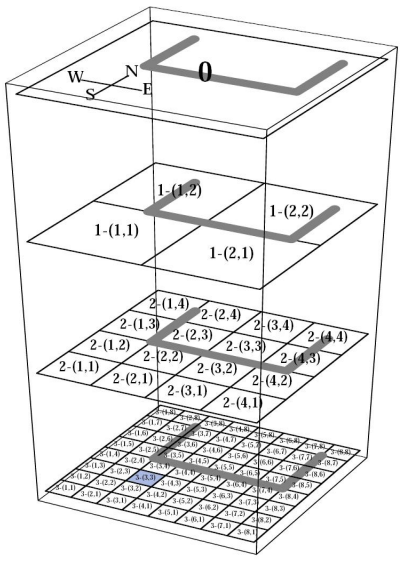

Figure 1: Hierarchical breakdown of a maze into sub-mazes from [Dayan and Hinton, 1993]

designer to consist of multiple levels. At each level of the hierarchy the gridworld blocks are grouped into increasingly large blocks, as in figure 1. Managers are assigned at each level of the hierarchy. Managers are in charge of choosing which sub-manager to assign next to complete a desired goal. Therefore each manager will have a super-manager, except for the highest level manager. In the same way, every manager will have a number of different sub-managers which it will have control over, and only the lowest level managers are allowed to perform actions inside the gridworld. There are certain rules that need to be applied when applying this method. Managers must reward sub-managers for achieving the manager's desired task, even if this task is not desired by the super-manager. Similarly, sub-managers do not get rewarded even if they perform the desired task of the super-manager, unless they satisfy what the manager has told them to do. Therefore rewards are strictly only given one level down the hierarchy. Managers also only need to know the state of the system at the granularity of their own choice of tasks. In testing the performance of this system, the designers tested it on a given gridworld and compared the results to that of a flat reinforcement learning agent. For the first few iterations, the flat reinforcement learning agent outperformed the Feudal reinforcement learning agent, but the Feudal reinforcement learning agent quickly overtook the performance of the flat reinforcement learning agent and after about 500 iterations was by far outperforming it.

2.3 MAXQ Value Function Decomposition

Another method of hierarchical reinforcement learning, called MAXQ value function decomposition, is proposed by [Dieterich, 1999]. It is an algorithm which expects the hierarchical structure to be supplied by the designer. It suggests breaking the main problem's

value function up into an additive combination of smaller value functions, each associated with a smaller problem. As an example the author gives a problem where a taxi agent needs to move around a gridworld picking up and dropping off passengers at there desired locations. The author breaks up the problem into a hierarchy of problems, represented by figure 2. The MAXQ algorithm is tested against flat Q-Learning and significantly outperforms it.

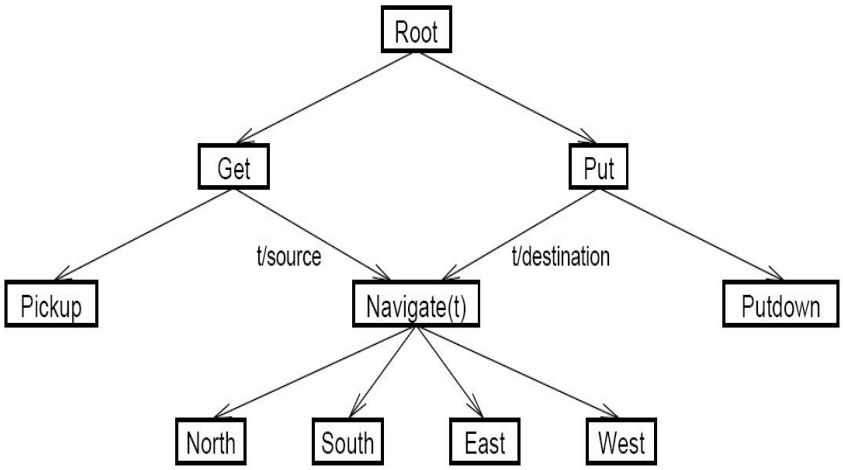

Figure 2: Hierarchy of subproblems for the taxi problem from [Dieterich, 1999]

2.4 Reducing the State Space in Hierarchical Reinforcment Learning

[Asadi and Huber, 2004] give a method for dividing up the global state space of a problem into smaller state spaces. Their method is an extension of a method called ϵ-learning. It describes how a problem's state space can be divided up into a series of intervals if the states in the same interval block have the same properties in terms of transitions and rewards. This type of division can be used to reduce the overall state space of the given problem. This method is not directly related to my work. However, the different divisions of the state space could be thought of as subgoals, and hence we could say that this method fits in with the idea of automatically discovering subgoals.

This method was tested on a gridworld problem to see its ability to reduce state space. The algorithm worked well, and reduced the state space to the full extent which the restrictions of the method allowed.

2.5 The HASSLE Algorithm

In [Bakker and Schmidhuber, 2004] the HASSLE algorithm is described in more mathematical detail than the algorithm on Feudal reinforcement learning in [Dayan and Hinton, 1993]. It follows the same general idea as Feudal reinforcement learning, but is more advanced and has methods to automatically discover subgoals and extract high level observations from the given information, without the intervention of a designer. For simplification, it considers only two levels of a control hierarchy, but the control hierarchy will be the same at all levels. At the higher level there is a high level policy π^H and at the lower level there are a number of low level policies π^L. The action of a high level policy is to request a low level policy to perform a specific subgoal. The subgoals are a set of high level observations o^H which is completely different from and smaller than the low level observations o^L. π^H takes as its input o_s^H, the current observation, and its action is another high level observation o_g^H which is the next desired observation. There are a limited set of low level policies π_i^L, none of them are initially associated with a specific subgoal o_g^H or any of the possible o_s^H, the associations are learned over time. Every π_i^L contains a table of so-called C-values of (o_s^H,o_g^H) pairs each which represents the capability of π_i^L to reach subgoal o_s^H from the high level observation o_s^H, the input observation. For the current (o_s^H, o_g^H) pair, a low level policy is selected based on its capability, C(o_s^H,o_g^H), to get from the current observation to the desired observation. If a low level policy gets to the desired observation o_g^H, its capability, C(o_s^H,o_g^H), is increased, otherwise its capability is decreased. The low-level policy also receives a positive reward if it reaches the subgoal, and zero reward if it doesn't, so that it becomes better at reaching it. Hence different low-level policies will have different capabilities of reaching different observations o_g^H, and therefore different low-level policies learn to specialise at the tasks that they are the most capable of. To arrive at high level observations, the HASSLE algorithm is not constrained to any particular method. In the example given, a method called ARAVQ (Adaptive Resource Allocation Vector Quantization) was used.

The HASSLE algorithm was tested against various reinforcement learning agents, some using linear function approximation and others using multilayer feedforward neural networks for value function approximation. HASSLE outperformed all the flat reinforcement learning algorithms.

2.6 Automatic Discovery of Subgoals in Hierarchical Reinforcement Learning Using Diverse Density

[McGovern and Barto, 2001] propose another algorithm for the automatic discovery of subgoals. Their theory is that subgoals often occur at bottlenecks in the state space. As an example they describe a gridworld in which there are different rooms and connecting the rooms are doorways. If an agent starts inside one room, and the goal is in another room, getting through the door to the other room could be thought of as a subgoal, because it needs to be accomplished before the final goal is reached. The doorway can also be thought of as a bottleneck in the state space, and hence the correlation between bottlenecks and

subgoals. Bottlenecks are not confined to navigation tasks though, and the concept of bottlenecks can be applied to a range of different state spaces. As a general definition of bottlenecks the authors talk about an agents path through the state space as its trajectory, and say that a bottleneck is a region which the agent experiences somewhere in the state space on every successful trajectory, but not at all on an unsuccessful trajectory. A method of finding bottlenecks for general state spaces, and hence for finding subgoals is then given, which uses the concept of diverse density.

These types of bottleneck subgoals show that the reason hierarchical reinforcement learning would be more efficient than flat reinforcement learning isn't only because breaking goals up into subgoals significantly decreases the state space, but also because it enables more efficient exploration. In flat reinforcement learning, if an agent starts in a room, but has to get through a door to get to the goal, it might spend far too much time exploring the first room, because the chances of it finding its way through the door while still exploring are quite slim.

The algorithm was tested in a gridworld navigation task, and the agent correctly identified a doorway as a subgoal.

2.7 Automatic Discovery of Subgoals Using Learned Policies

[Goel and Huber, 2003] offer another method of automatic subgoal discovery. The authors agree with [McGovern and Barto, 2001] that a good example of a subgoal is that of a doorway in a gridworld navigation task. They describe a subgoal as a state with the following structural property: the state space trajectories originating from a significantly larger than expected number of states lead to the subgoal state while its successor state does not have this property. This property is due to the fact that states that are not a subgoal have a much higher connectivity than states that are a subgoal. This can be understood quite easily by considering the gridworld navigation task. The doorways have much less connectivity with other states than the open spaces in the rooms, and therefore would be identified as subgoals by this algorithm, which is what is desired.

This algorithm was also tested on a gridworld navigation task, where there were multiple doorways, and the algorithm recognised all doorways as subgoals, except one, which it failed to recognise as a subgoal. The authors explained that the reason for this one failure was that the size of the room with the undiscovered doorway subgoal was significantly smaller than the other rooms. Although this did not seem like a significant drawback, it suggests that this algorithm may have limitations.

3 Conclusion

Because of the advantages of automatic subgoal discovery, current research in hierarchical reinforcement learning seems to be focused mainly on getting agents to automatically discover subgoals for themselves. These are very useful, because the more the agent can figure out on its own, the more flexible and environment independent it will be.

Various algorithms have been formed for the application of hierarchical reinforcement learning, but because this is a recent area of study, theories are quickly building on each other and changing. For an actual implementation of hierarchical reinforcement learning to a problem a specific solution may have to be tailor made, and various algorithms may need to be mixed in order to achieve the desired result.

4 References

Asadi, M and Huber, M (2004). *State Space Reduction for Hierarchical Reinforcement Learning*. In Proceedings of the 17th International FLAIRS Conference, pp. 509 - 514, Miami Beach, FL. 2004 AAAI

Bakker, B and Schmidhuber, J (2004). *Hierarchical Reinforcement Learning Based on Subgoal Discovery and Subpolicy Specialization*. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, and B. Krse (Eds.), Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8, Amsterdam, The Netherlands, p. 438-445.

Barto, A and Mahadevan, S (2003). *Recent advances in hierarchical reinforcement learning*. DiscreteEvent Systems journal, 2003.

Borga, M (1993). *Hierarchical Reinforcement Learning*. S. Gielen and B. Kappen, ICANN'93, Springer-Verlag, Amsterdam, p. 513

Dayan, P and Hinton, GE (1993). *Feudal Reinforcement Learning*. In Advances in Neural Information Processing Systems 5, 1993. Morgan Kaufmann, San Mateo, CA

Dietterich, TG (1999). *Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition.* In Fifteenth International Conference on Machine Learning.

Goel, S and Huber, M (2003). *Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies*, In Proceedings of the 16th International FLAIRS Conference, pp. 346-350, St. Augustine, FL. 2003 AAAI

Kaelbling LP, Littman ML and Moore AW, (1996). *Reinforcement Learning: A Survey.* Journal of Artifcial Intelligence Research, 4:237-285, 1996.

McGovern, A and Barto, A (2001). *Automatic discovery of subgoals in reinforcement learning using diverse density*. Proc. 18th International Conf. on Machine Learning, 2001

Pfeiffer, M (2004). *Reinforcement Learning of Strategies for Settlers of Catan.* Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education, Reading, UK. November 2004

Sutton, RS and Barto, AG (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Tesauro, GJ (1995). *Temporal Difference Learning and TD-Gammon*. Communications of the ACM