Using Hierarchical Reinforcement Learning to Balance Conflicting Sub-Problems

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

September 26, 2005

Abstract

This paper describes the adaption and application of an algorithm called Feudal Reinforcement Learning to a complex gridworld navigation problem. The algorithm proved to be not easily adaptable and the results were unsatisfactory.

1 Introduction

This paper describes how Feudal Reinforcement Learning as described in the paper by [Dayan and Hinton, 1993] was applied to a complex gridworld navigation problem.

In [Dayan and Hinton, 1993] the algorithm of Feudal Reinforcement Learning was applied to a simple terminal gridworld problem in which an agent had to navigate past an obstacle in the gridworld to find a goal and complete the overall task. The state at any time could be represented by just the x and the y coordinates.

In the complex gridworld problem, the overall task is to stay healthy and keep hunger and thirst low. These are all conflicting sub-problems as they are all solved by a completely different series of actions. They all need to be balanced, as just keeping one of the needs satisfied is not sufficient. All three sub-problems need to be solved individually, balancing them so that each of them is in an optimally solved state simultaneously.

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. It is called 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 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.

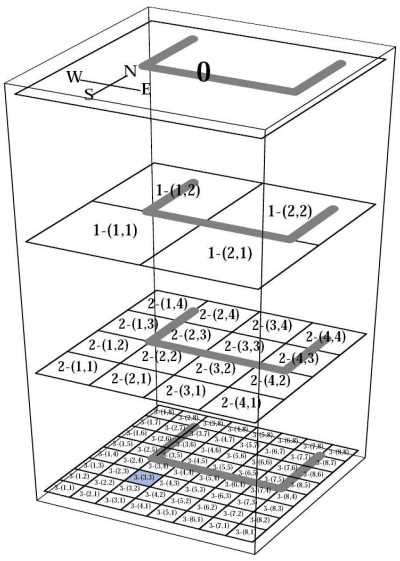

Figure 1: Hierarchical breakdown of a maze into sub-mazes from [Dayan and Hinton, 1993]

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.

3 The Complex Gridworld Navigation Problem

The complex gridworld navigation problem is made up of a 6x6 gridworld in which the agent can move around. The agent has five possible primitive actions, move one step left, move one step right, move one step up, move one step down and rest. There are five designated blocks on the maze with special properties. These are food, drink, shelter, a hazard, and wood and are represented by appropriate pictures as in figure 2. If the agent navigates onto a designated food block or drink block, its respective food or drink level is increased to maximum. If it navigates onto the designated hazardous block, its health is decreased to zero. If it navigates onto the wood block, it will be carrying wood from then on. If it navigates onto the shelter location while carrying wood, its shelter will be repaired and it will no longer be carrying wood. And finally, if it navigates onto a shelter in good condition, and then *also* rests on that block, health will be fully replenished. After every few moves, hunger, thirst, shel-

ter condition and health gradually decrease, but with health decreasing more slowly than the others.

The sub-problems are hence conflicting, since the agent can't try to solve one sub-problem without risking that it fails to solve another and each sub-problem is solved with a completely different series of actions. The sub-problems cannot be solved simultaneously, but rather be solved separately, in a balanced fashion, so that the overall task is optimally satisfied.

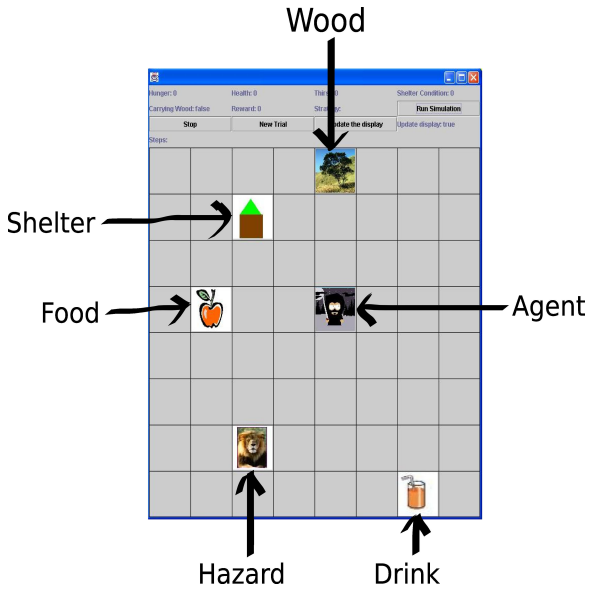

Figure 2: Complex Gridworld Navigation Problem Screenshot

4 Adapting Feudal Reinforcement Learning

4.1 Determining Abstract High Level Actions

The complex gridworld problem is more complex than the maze problem in [Dayan and Hinton, 1993] and hence Feudal Reinforcement learning needs to be adapted.

In the complex gridworld problem, high level states are described in a similar way to how they are defined in the simple maze problem. Each high level state variable is divided into two equal groups, and then each high level state is defined as a different permutation of high level state variable groups. There are hence 2^N different high level states, where N is the number of state variables. Therefore there are $2^7 = 128$ high level states in the complex gridworld problem, compared to $2^2 = 4$ high level states in the simple maze problem.

In the simple maze problem, because the state space is defined by just two variables x and y, which by definition of the primitive actions cannot increment or decrement simultaneously, high level actions can be, and are defined as being exactly the same as the four primitive actions ie. move left, right, up or down. A high level move would just be a directional transition from one high level grid of blocks to another. In the complex problem, state variables can change simultaneously, ie. shelter could deteriorate to a different level, hunger could be satisfied and the agent could move into a different high level grid of blocks all in one move. For this reason, high level actions are harder to define. As a result, high level actions are defined as being the desired high level state that the agent is trying to get into at any time. In this way all possible high level actions are included, but because not all high level states are reachable from a given high level state, this causes there to be a number of meaningless high level actions for every high level state, which is undesirable, but unavoidable.

4.2 Terminal and Non-Terminal Managers and Timeouts

In the simple maze problem, the task was terminal, and when the agent found the goal block, the task was completed. The complex gridworld problem, however is not terminal.

Although the complex gridworld problem is not terminal, sub-managers are, because of the fact that they are called on to perform an action until completion, when a different sub-manager is called upon. If an agent is already in a very desireable state, it may wish to stay in that state, and hence a high level action of the current state being the desired state was included. A timeout of 50 steps was also included, in order to give the sub-agent the appropriate reward when it has been trying to stay in the same state and hence no change of high level state has occured.

4.3 Real and Pseudo Rewards

In [Dayan and Hinton, 1993] it is described how the highest level manager is rewarded according to the "real" low level rewards, while the low level agents are given pseudo rewards according to whether or not they satisfy what the high level managers tell them to do. The pseudo rewards were quite easily implemented as just a reward of 1 if the sub-manager performs the task required by the super-manager, and a reward of 0 if it doesn't.

Rewarding the highest level manager according to the "real" rewards in the non terminal complex gridworld problem proved to be much less trivial than what it was for the simple terminal maze problem. For one, because the problem is not terminal, it is not clear when to issue these rewards. An average previous reward is associated to each high level state block by keeping track of rewards as they happen, and then issuing the reward the next time the high level manager chooses the state as its "action", or desired state as previously described. All possible transitions between states are also kept track of, and a reward is only issued if the transition was previously successful, hence we don't reward the super-manager for choosing an impossible action. However this may also cause inefficiency because some possible transitions may only rarely occur, and a reward can only ever be given for a transition once it has previously occurred.

The real low level reward function was described as follows: $(F+D+2H)^3$, where F is the food level, D the drink level, and H the health level. The health level was multiplied by 2 to make health twice as favourable for the agent to have. The function was raised to the power of 3 in order to give a much higher reward for having all three of F, D, and H high, rather than just one of them. This prevents the agent from getting stuck in a local maximum when just satisfying one of the sub-goals continually and also mimics reality more closely.

5 The Modified Feudal Reinforcement Learning Algorithm

When implementing the feudal reinforcement learning algorithm, because of the adjustments made to high level actions, and because of the complexity of the problem, memory started becoming a problem. Therefore the algorithm was limited to just two levels ie. a super-manager with multiple sub-managers. The levels of hunger, thirst,

health, and shelter condition were also divided into fewer discrete levels, in order to decrease the total number of possible states. The following is a summary of the steps of the adapted algorithm:

- nextState = "The current state from Maze object" at each level
- nextAction = "The action for nextState" at each level
- Loop:
  - state = nextState, for both levels
  - action = nextAction, for both levels
  - get low level action from super-manager
  - perform low level action
  - give super-manager low level reward
  - call learn for super-manager, which will in turn call learn for sub-managers
- Goto top of loop

SARSA with eligibility traces, as described in [Sutton and Barto, 1998], was used as the reinforcement learning algorithm at both levels.

6 Results

The implemented system was tested, and although it worked as desired, its performance was not optimal, as can be seen in figure 3, where it was tested against a flat reinforcement learning approach. The thick line is the flat reinforcement learning

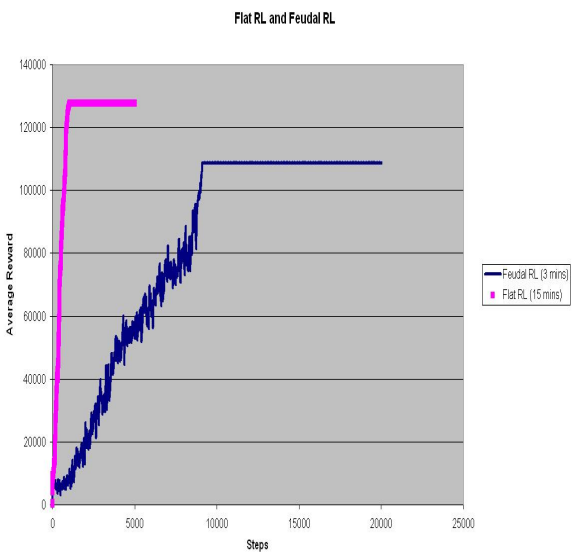

Figure 3: Results of a Test Run

algorithm which clearly reaches its maximum long before the adapted Feudal Reinforcement Learning approach, which is represented by the thin line. But, although the Feudal agent took more steps to learn, it took 3 minutes to run, compared to 15 minutes, which the flat reinforcement agent took. This is probably due to the reduction of size of the state-action tables in the Feudal Reinforcement Learning algorithm. Despite this speed increase, the algorithm does not perform satisfactorily well, and this is probably due to the modifications made to the original algorithm.

7 Conclusion

The Feudal Reinforcement Learning algorithm worked for the simple maze problem, but was not easily extensible to the complex gridworld navigation problem, and in the end proved unsatisfactory. A better approach might be one where there is more previous information given to the agent, such as

definition of sub-goals as described in [Dietterich, 1999] or where the sub-goals are first automatically discovered such as in [Bakker and Schmidhuber, 2004], [Goel and Huber 2003] and [McGovern and Barto 2001].

8 References

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.

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

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

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