Feudal $\mathcal{Q}$-Learning

Peter Dayan[1]

Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Ontario M5S 1A4
Canada

September 25, 1995

Running head: Feudal $\mathcal{Q}$-Learning

Key words: reinforcement learning, dynamic programming, $\mathcal{Q}$-learning hierarchies

Abstract

One popular way of exorcising the dæmon of dimensionality in dynamic programming is to consider spatial and temporal hierarchies for representing the value functions and policies. This paper develops a hierarchical method for $\mathcal{Q}$-learning which is based on the familiar notion of a recursive feudal serfdom, with managers setting tasks and giving rewards and punishments to their juniors and in their turn receiving tasks and rewards and punishments from their superiors. We show how one such system performs in a navigation task, based on a manual division of state-space at successively coarser resolutions. Links with other hierarchical systems are discussed.

1 Introduction

Many tasks for real and artificial systems can naturally be cast in hierarchical terms. Division for conquest is a common metaphor, and it is certainly a conventional way for human designers to cope with task complexity in everything from the organisation of large corporations to chip design. Biological control systems, particularly in invertebrates, also employ it.

[1] Telephone number: (416) 978 6277 email: dayan@cs.toronto.edu

Adopting a hierarchy has two main advantages: it can reduce search (or learning) complexity for tasks, and it is a natural way to specify solutions in which there is shared structure – information about one part of the task can be used directly for another part. These benefits are only won at a cost: specifying and learning the hierarchy may be more expensive in terms of space or time or actions; the ultimate solution of the task can be suboptimal if the hierarchy is not well tailored to the environment; and, most worryingly, there may be no solution at all to a task based on a given hierarchical division.

Hierarchies and the abstraction of tasks and actions have a long and distinguished career in cybernetics, control theory and artificial intelligence – Polya (1945) is often cited as an early modern source, and Watkins (1989) has exerted an important influence on the work in reinforcement learning. In this paper we consider the issue from the perspective of adaptive optimising control in simple Markov decision problems. We focus on learning, both within and between levels of the hierarchy. Conventional reinforcement learning algorithms are often accused of being painfully slow, and so there is definite scope for improvement.

The next section discusses some general issues for hierarchical reinforcement learning, and section 3 presents and gives an example of the Feudal $\mathcal{Q}$-learning system (first discussed in Dayan & Hinton, 1993), which picks some from amongst the possible options for such learning. At about the same time, a number of groups (including Jameson, 1992; Kaelbling, 1993; Lin, 1993; Singh, 1992a;b;c) independently presented hierarchical reinforcement learning systems that are quite similar to each other and to the Feudal system. Section 4 discusses aspects of them, and other, less closely related, systems (Ring, 1993).

2 Hierarchical Reinforcement Learning

We are interested in the application of hierarchies to the reinforcement learning (RL) solution of Markov decision problems (MDPs). These are controlled Markov processes with numerical rewards or punishments according to states traversed and actions executed. The job for the agent is to maximise its short or long term average or discounted return. The set of RL methods we cover are asynchronous ways of doing dynamic programming (DP, Bellman, 1957; Barto, Sutton & Watkins, 1990; Barto, Bradtke & Singh, 1991), which is a method of optimal control from engineering. We use the $\mathcal{Q}$-learning algorithm (Watkins, 1989) which is designed to learn the optimal value of doing each action at each state from experience of rewards and transitions.

Much of the discussion will be focused on maze tasks, although the lessons are more general. For fixedness, the agent can move North, South, East or West (NSEW) in a two-dimensional grid, and has either to reach a single goal (see the lowest level in figure 1), or to navigate between two arbitrary points (pairwise navigation). In both cases it should use paths that are as short as possible. The agent can fully observe its state – however, there are edges to the grid and there may be barriers which it can only infer by observing that some actions have no effect. The representations of neighbouring states may have nothing in common, so even the notion of locality may have to be acquired.

We employ hierarchies that involve the division of space or time at multiple levels, the assignment of controllers to each division, and control structures in which higher level and lower level controllers interact in some way to specify what ultimate action the agent takes. The hierarchy is a way of storing, in a flexible manner, information about solving a problem. The division will be appropriate for a task if the knowledge acquired at the lower levels can be shared across different high level control objectives. The pairwise navigation task is an example of this – getting to any one of the states in the North-East corner of a maze might require almost all the same initial moves if the agent starts far away. We give an example like this in section 3.

This section fleshes out the description and discusses some of the options for forms of abstraction and mechanisms of interaction for control.

2.1 Spatial and Temporal Abstraction

Hierarchies can reduce the complexity of searching for solutions to MDPs using reinforcement learning (RL) because they introduce temporal and/or spatial (*ie* statewise) abstraction. Indeed, multi-resolution or state-aggregation methods are a mainstay for speeding up conventional dynamic programming (see Morin, 1978, for numerous examples and references). Standard methods focus on having a single task, and using coarser and finer representations of the value function. This is not hierarchical, in the terms of this paper, since there are not different but interacting controllers at the different levels. The advantage of standard multi-grid methods is that, given an accurate model of the task and the rewards, it is possible to guarantee the quality of the approximation at different resolutions (analogously to the multi-grid methods for solving partial differential equations). These guarantees do not apply in a straightforward manner to systems that lack such models or ones with interacting controllers, as described here.

Abstraction in a learning system is useful since it allows there to be fewer states at higher levels, which generally makes the search problem easier there. Also, if different higher level tasks can be satisfied by the same lower level actions, then these actions need not be relearnt. Problems can also arise. Since top level commands are typically insensitive to the exact low level state of the agent, the paths can be suboptimal. Worse, use of hierarchies can be disastrous if the different low-level states within a high-level state require inconsistent commands. Such problems reflect the fact that, from the perspective of the high levels, the problem becomes only partially observable (Lovejoy, 1990), because of the state abstraction.

As an example of state abstraction, in the case of navigating to a goal in the maze, Lin (1993) considered tessellating the space into a set of regions according to proximity to a set of selected points (landmarks or abstract states) in the space. The high level actions specified that the agent should get from an arbitrary point to its nearest landmark, and from this landmark to ones successively closer to the goal (which was also made into a landmark). Lin (1993) showed that if the complexity of finding (learning) an optimal solution to an MDP is $\mathcal{O}(n^2)$, where n is the number of states, then inventing k suitably spaced landmarks, each of whose domain of knowledge of optimal trajectories contains at most m points and whose maximal separation is l steps, reduces the overall complexity to $\mathcal{O}(km^2 + lk^2)$ The abstraction process can be recursed. Of course, the ultimate paths can be suboptimal – there would in general be no reason for the best path between two points to go through the landmarks.

Watkins (1989) described a different scheme for spatial abstraction in which some aspects of the state spaces of higher and lower levels were the same, but others were different. Also the actions available at each level were disparate. He discussed the example of a navigator giving orders to a helmsman on a sailing ship in which both could see which way the ship is moving. The navigator could also see higher level information about such things as the desired itinerary of the ship and the helmsman could also see lower level information about the direction of the wind and the force of the currents.

Note that it is possible to abstract over space without using hierarchies. Moore (1991;1994) considered splitting up the state space at variable resolutions, choosing finer divisions in one case according to divergence from a learned model along paths to the goal and in the other if it is required to prevent the possibility of being trapped by a currently infelicitous division. On each step, Moore (1991) performed full dynamic programming under the (temporary) assumption that the value of a state (*eg* its distance from the goal) is constant in each region. This nullifies the curse of dimensionality without resorting to multiple levels of division of space.

In some problems, abstraction over time is as important as abstraction over space. Jameson (1992) considered a version of the problem of balancing an up-ended pole in the case that the base of the pole is not allowed to stray too far from the start point. He noted that there are inherently two timescales in the problem – a fast one from the immediate instability, and a slow one from the fact that the pole can have a net drift in one direction (if it is balanced on average at a small non-zero angle to the vertical). The top level of his hierarchy averaged over a longer timescale than the bottom one (they were actually clocked at different rates), and was therefore in a good position to learn to extract the drift and command its removal. All the abstraction was temporal – high and low level controllers saw the state space at the same resolution.

For most hierarchies (including the Feudal system), however, abstraction over space is coupled with abstraction over time. If the states at high levels are viewed at a coarser resolution than those at the low levels, then the system will tend to dwell longer in the former than the latter. Just because the state does not change at the high level does not mean that the action at the high level cannot itself change (*eg* Singh, 1992a). Indeed, there are circumstances in which it is essential that the managerial command can change without the managerial state changing. An example is if the partial observability of the high level manager's MDP might allow the agent to become permanently trapped.

2.2 Hierarchical Choices

Given that some sort of hierarchy is to be adopted, it remains to choose the levels and the nature of the (presumably increasingly coarse) abstraction, and the jobs for the various elements. The whole process of specifying and using the hierarchy should ideally be recursive, so that there is nothing special about any of the levels, bar possibly the highest and the lowest.

High level controllers (we will also call them managers) have two basic prerogatives: selecting between low level controllers (sub-managers) and setting them tasks. We will not discuss the case in which more than one sub-manager can be active simultaneously with some form of direct, intra-level, contention for control – either there will be just one possible sub-manager at any time, or the manager will be responsible for selecting exactly one. Managers can also attempt to influence sub-managers by providing them with reinforcement signals indicating how well or poorly they are doing (instead of, or in addition to, any direct reinforcement they might acquire from the environment).

If the managers explicitly set tasks to their sub-managers, there might appear to be a problem of semantics – the sub-managers have to understand, or at least be capable of learning, what their managers mean. The language can either be specified by the designer at the outset, or sub-managers can learn the meaning on the basis of reinforcement from the managers when they successfully (or unsuccessfully) complete a task. For instance, in the Feudal system described below, a manager might specify to a sub-manager that it wishes the next transition it will observe (at its level of state abstraction) to be to the North. The sub-manager receives this command as an additional part of its state, and learns just from the reinforcement provided by the manager what it has to do to achieve this goal. It is clear that if a manager specifies a task to a sub-manager, then the manager has to know the satisfaction conditions of that task so that it can reward or punish its sub-managers correctly, or learn itself to choose better ones. Recursion is easy, so that sub-managers can have sub-sub-managers in their turn, provided that managerial reinforcement looks just like ordinary reinforcement to the sub-managers.

This still leaves a large number of possible options for the interaction. At a Darwinian extreme, one could use *selection,* in which there would be a large population of non-adaptive sub-managers and a high-level manager which learns to choose from them the one that works the best. At the other, *instructive*, extreme (closer to the Feudal system), only one sub-manager would be available at any time, and the high-level manager would set it tasks, and would have to cajole it into doing its bidding by providing reinforcement appropriately.[2] Managers may either influence only their immediate sub-managers, or could additionally affect the choices at even lower levels in the hierarchy.

Another choice which must be made to specify a system is the passage of control on descending a hierarchy. Watkins (1989) makes the distinction between *delegatory* and *supervisory* control. If a manager supervises its sub-managers, then, as in (*eg* Singh, 1992a), it does not relinquish control to them but rather re-selects them or re-directs them either every time-step, or at some intermediate times. For the hierarchy to be efficient, however, managers should lack the resolution in state-space to monitor the exact behaviour of their underlings, so raising the question as to what information they might use to make the reselection. One thing they could use is the amount of reinforcement the sub-managers are getting from the world. If, like Watkins' navigator, they have some components of their state-space in common with their sub-managers, they could use this too.

If a manager delegates to its sub-managers, then it will normally only regain control once they terminate. This would usually be only once the state changes at the manager's own level. If sub-managers can be completely incompetent, potentially allowing the system to get stuck in a state, then some form of 'escape' clause will be required so that the manager can regain control. There is a strong analogy with an economic arrangement with contracts, costs and penalty clauses.

[2] The Feudal system uses a table-lookup representational scheme in which no information is shared between the different tasks of a single sub-manager. For such a scheme, the choice of a task for a single sub-manager could equally be characterised as a choice between sub-managers, each of which is dedicated to learning to satisfy one of the manager's tasks on the basis of the manager's instructional signals.

3 Feudal $\mathcal{Q}$-Learning

Abstract Description

The Feudal system (Dayan & Hinton, 1993) is the result of picking a set of these choices, inspired by a mediæval feudal fiefdom. We will describe it in the context of a two-dimensional maze. The state space of the maze is divided manually at successive levels; a single manager, which is a straightforward $\mathcal{Q}$-learning entity, is appointed to each division of the state space at each level (except for the finest level at which this is trivial), and is awarded total control over the sub-managers in the level directly below it. Figure 1 shows an example of this division, where each manager is given as 'level-(x,y)' on the diagram, and has exactly four sub-managers. Only one of these sub-managers is relevant at any one time, since the agent only occupies one state at each level, but the manager does not know which one of the sub-managers is active, since its spatial resolution is too coarse. However, it can specify different commands to the collection of sub-managers – and these become part of their state space (as in Watkins, 1989) so that they learn how to satisfy them. Part of the reinforcement to a sub-manager comes from the world, to provide all levels with information about the lengths of paths, and part from its manager, to encourage it to accomplish the set tasks.

Almost absolute control is transferred from a manager to its sub-managers – having specified the command, it would normally only provide reinforcement when it detects a state change at its own level. However, for the reasons discussed above, it does have an escape clause if the same command has proved ineffectual for too many steps. Managers specify actions to their sub-managers in terms of the transitions they would like to observe at their own level of spatial abstraction, and reward them accordingly. The collection of sub-managers can therefore face a temporal credit assignment problem: reward or punishment only comes from their manager when the state changes at the managerial level – even though the state can change at the sub-managerial level without this happening.

Two principles characterise this strict hierarchy:

information hiding managers have no information about the means by which sub-managers carry out their tasks – they even lack the representational resources to appreciate what is going on. They do get to see the amount of reinforcement accrued from the world in the course of execution of their commands (they need this to help work out if the set task was appropriate). Sub-managers also do not get any information about the tasks that their managers were set by their own managers (the super-managers). Although both of these restrictions can lead to sub-optimal ultimate paths, that is the price to be paid for the abstraction benefits of the hierarchy.

reward hiding managers should reward their sub-managers for carrying out their bidding *whether or not* the managers were themselves rewarded by the super-managers. It is important that sub-managers become perfect slaves – they should leave it to their managers to decide what tasks to set and they have no right of appeal to the super-managers. By this means, in the early stages of learning, even if the highest level tasks have never been achieved, lower level sub-managers can have come to learn how to obey their managers.

Further, only the fruits of honest labour should be rewarded – if a super managerial task is achieved by a sub-managerial action, but that action violates a managerial command, then neither the manager nor the sub-manager should be rewarded. The sub-manager failed to obey its manager, and the manager failed to choose the right task. It was just happenchance that this confusion led to appropriate behaviour from the perspective of the super-manager.

3.1 Concrete Description

The particular task on which we tested one implementation of the Feudal system is shown in the lowest level of figure 1. This is a simple 8×8 two-dimensional maze with a goal, although we used grids of sizes up to 64×64 for the experiments. The agent had to solve a pairwise navigation task, in which the goal was signalled, but could be anywhere on the grid. Except where indicated, there was no barrier. As will become clear, the task is almost ideally suited to the Feudal system, and indeed was chosen for this reason.

As for a standard pairwise navigation task, the agent knows its own location, and also the location of the goal, and it has to get to the goal in as few steps as possible. When it reaches the goal, the current trial ends, the goal moves randomly somewhere else in the grid, and the agent has again to find it. The agent does not have a model of the world and does not know where the boundaries of the maze are. If it tries to cross them, it stays where it is. Except where mentioned, each of the lowest level actions has probability $1 - \pi = 0.9$ of being executed correctly (if this is possible) and probability $\pi = 0.1$ of being discarded and (without the agent knowing this) replaced by a random other action.

We are going to contrast the Feudal system with a straightforward $\mathcal{Q}$-learning system for this task. The $\mathcal{Q}$ function is indexed by the state of the agent in the world, which includes the current location of the goal (so if the maze is 32×32 then the agent really faces 1024 separate $\mathcal{Q}$-learning tasks, and the $\mathcal{Q}$ function has $32 \times 32 \times 32 \times 32 \times 4 = 4.2$

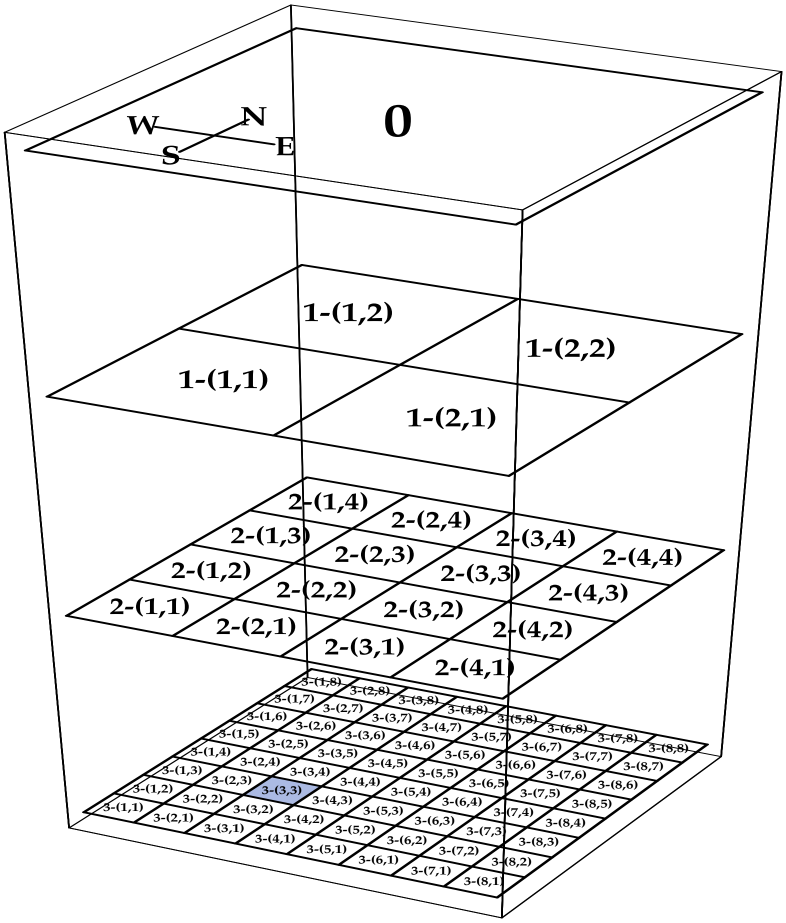

Figure 1: The Grid Task. This shows how the maze is divided up at different levels in the hierarchy. The shaded square is the goal (at 3-(3,3) in this case). Each high level state is divided into four low level ones at every step.

---

million entries). The appendix gives details for both standard and Feudal $\mathcal{Q}$-learning of the learning rates and how the trade-off between exploration and exploitation was arranged.

For the Feudal system, the state space is divided by a factor of 2 in each dimension at each level going down the hierarchy. Each state at each level has a manager which is responsible for setting tasks and awarding reinforcement to the four sub-managers within its spatial domain at the layer below, so, for instance, the manager 1-(1,1) controls the sub-managers 2-(1,1), 2-(2,1), 2-(1,2) and 2-(2,2). This division of space was designed to respect locality, clearly an important property for this task.

Exactly one manager is active at any time at each level, and so there is never any contention for control between different managers at the same level. A manager at each level emits one of five commands, the four 'geographical' moves, NSE and W, and a special action $*$, which, as described below, gets translated by the sub-managers into one of four actions $*_{SW}$, $*_{NW}$, $*_{SE}$ and $*_{NE}$. A geographical move is satisfied at a given level in the hierarchy if the next time the state changes at that level, the agent has moved in the given direction. Some can clearly never be satisfied, *eg* the agent cannot move North from 1-(1,2).

Given a managerial task, the collection of sub-managers solves the simple RL problem of trying to minimize the number of steps it takes to satisfy the manager's command whilst avoiding having the managerial state change incorrectly. In the latter event, the sub-manager just active is awarded a reinforcement penalty (the equivalent of a somewhat arbitrary 10 extra steps) by the manager (and also has to pay the previous estimate of the cost of satisfying the manager's command before the state changed). No explicit reward is given for satisfying the command, so sub-managerial $\mathcal{Q}$-values reflect the veridical (average) costs of satisfying their managers. As an example, if the agent currently occupies 3-(6,6), and the instruction from the level 1 manager is to move South, then the 2-(3,3) manager draws upon its experience in the temporal credit assignment problem that it and 2-(3,4), 2-(4,3) and 2-(4,4) have shared in trying to get to 2-(3,2) or 2-(4,2). The action the 2-(3,3) manager decides upon is then fed down one level, in this case maybe causing the agent to move Southh to 3-(6,5).

The special command $*$ is not required for conventional $\mathcal{Q}$-learning. It is the way a manager tells its sub-managers to search for the goal within its domain – a penalty will be awarded if it sees the state change at its own level before the goal is found. For each trial, the agent knows the coordinates of the goal – the intent is that during learning it works out how to use this information to navigate to it. The location of the goal is also represented in a hierarchical manner, taking advantage of the way that space is split. If (and only if) the goal lies within the domain at a given level, then the manager's action at that level will just be $*$. Consider the four sub-managers at the next level, indexed by their geographical relationship to the manager (South-West (SW), North-West (NW), South-East (SE) and North-East (NE)). They work out which of them contains the goal, and translate the $*$ action into one of $*_{SW}$, $*_{NW}$, $*_{SE}$ or $*_{NE}$ accordingly. This labelling of the sub-managers is arbitrary – the sub-managers that do not contain the goal have to learn which actions to demand of their sub-sub-managers to get to the domain of the one that does contain the goal.

An example should make this clearer. If the goal is at 3-(3,3), as in figure 1, and the agent is at 3-(2,2) (and therefore 1-(1,1) and 2-(1,1)), then the actions specified at the various levels are and their translations are:

Level	State	Action	Translation
0	0	$*$	$*_{SW}$
1	1-(1,1)	$*$	$*_{NE}$
2	2-(1,1)	?	
3	3-(2,2)	?	

where the question-marks indicate that the choice of actions for sub-managers whose domains do not include the goal is a function of their $\mathcal{Q}$-values, and is determined over the course of learning. These actions will not be $*$, and therefore need no translation. Clearly only the $*$ action is available at level 0, where there is just one state. The $*$ action is not available at the lowest level (level 3 in the diagram), since the agent is either already on the goal or just moves geographically.

Under normal circumstances, control is completely passed from one level to the next lower one – only returning if the state changes at the high level. However, an escape clause is required in case the agent is searching for the goal where it is not to be found or if it is trying to execute an impossible geographical action. This is implemented using a timeout – the agent tries to satisfy a manager's command for no more steps than twice the total number of lowest-level states within that manager's purview. This limit is important for the performance of the system – if it is too low then the agent might be too skittish and fail to find the goal. If it is too high, then the agent might waste substantial effort. Figure 3 shows this. Since all the managers are executing $\mathcal{Q}$-learning, the trade-off between exploration and exploitation is a function of all their choices. We used the conventional Gibbs sampling technique (Watkins, 1989), picking actions with probabilities governed by the exponential of their $\mathcal{Q}$-values. The appendix gives the parameters for learning and action selection and details of penalties.

Figure 2 shows how the Feudal system performs compared with standard, one-step, $\mathcal{Q}$-learning. Following the conventions in Barto, Bradtke & Singh (1991), an epoch is defined as twenty consecutive trials, and the curves shown are averages (and standard errors, shown as 'haloes') of the number of steps per single trial over 50 separate samples, each with a different random seed. The initial starting location of the agent was random and the goal is set to a new random position for the next trial when the agent finds it. The upper graphs labelled 'Feudal' are for the feudal system for the four sizes of maze shown. The inset has a linear scale for the y-axis. The lower graphs are for the standard $\mathcal{Q}$-learning system. The 64×64 maze requires too much memory to store the $\mathcal{Q}$-values in a lookup table and so it was not possible to test it. The inset graph is the same data with a linear scale for the y-axis (but a different maximum from the linear graph for the feudal system).

Feudal $\mathcal{Q}$-learning has one more parameter than standard $\mathcal{Q}$-learning which governs the number of steps the agent is allowed to use in attempting an action at a level

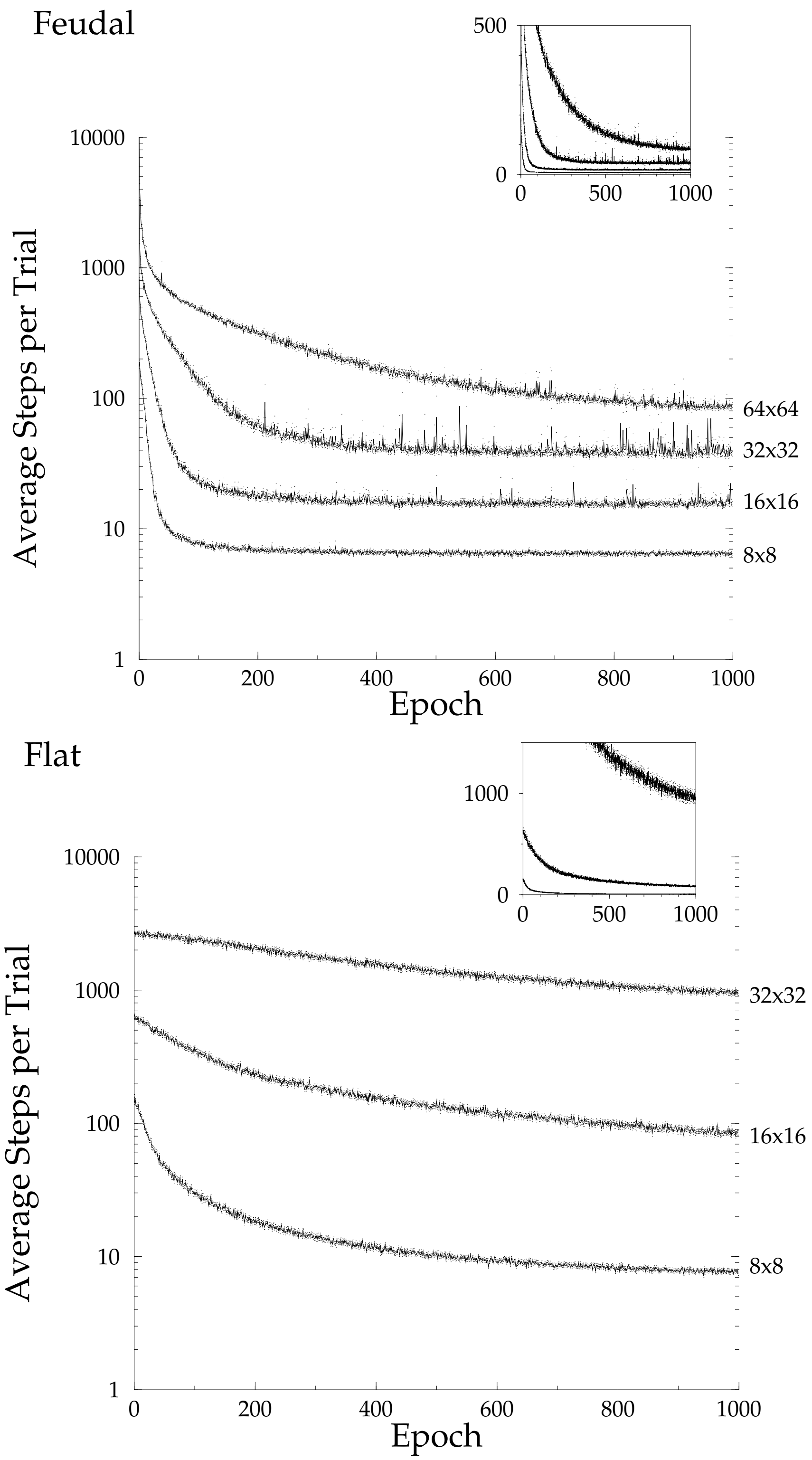

Figure 2: Comparative Performance. For description see text.

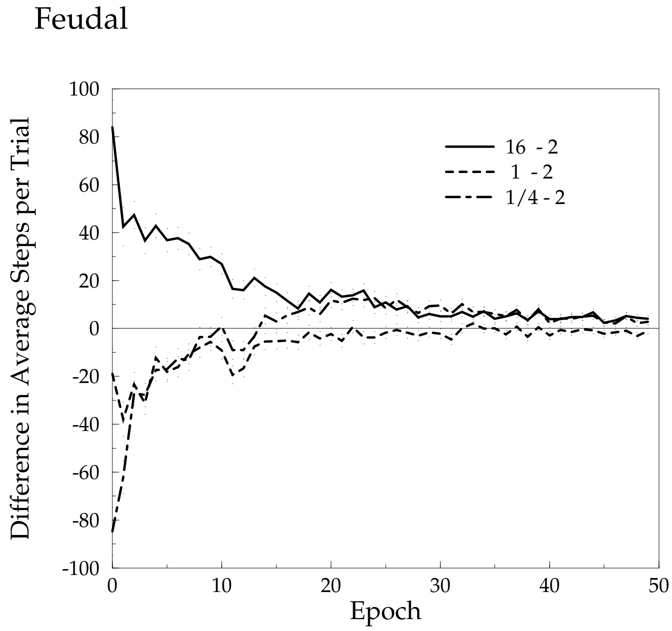

Figure 3: The effects of different timeouts on feudal $\mathcal{Q}$-learning. The curves show how much greater the average number of steps per trial is when using timeouts of 16, 1 and 1/4 times the number of lowest-level states covered by a manager compared to using the default of 2 times the number of states.

before the action times out. Figure 3 shows the early part of the learning curve in a 16×16 maze where the timeout is varied from 1/4 of the number of states at a level to 16 times the number of states. The curves shown are the differences in the average number of steps per trial for using the given timeouts from the average number of steps for using the default timeout, which is just twice the number of states. Making the timeout too big harms performance, but by less than 10% of the total number of steps per trial in the range shown. Reducing the factor below 1/4 is fatal – the system too rarely gets the chance to complete managerial actions successfully or unsuccessfully, and so fails to learn. The fact that the maze has no internal barriers expands the region of stability in this parameter.[3]

In the standard $\mathcal{Q}$-learning system there is no sharing of information between different goals. Therefore, the overall learning curve in figure 2 is a weighted sum of separate learning curves for getting to each possible goal in the maze, where the weights are

[3]Some insight into this is provided by the fact that if the agent were to execute a random walk on a two-dimensional grid starting in the centre of a square, then the expected number of steps it would take it to get to the edge of the square is roughly 35% of the number of grid points inside the square (see McCrea and Whipple, 1940).

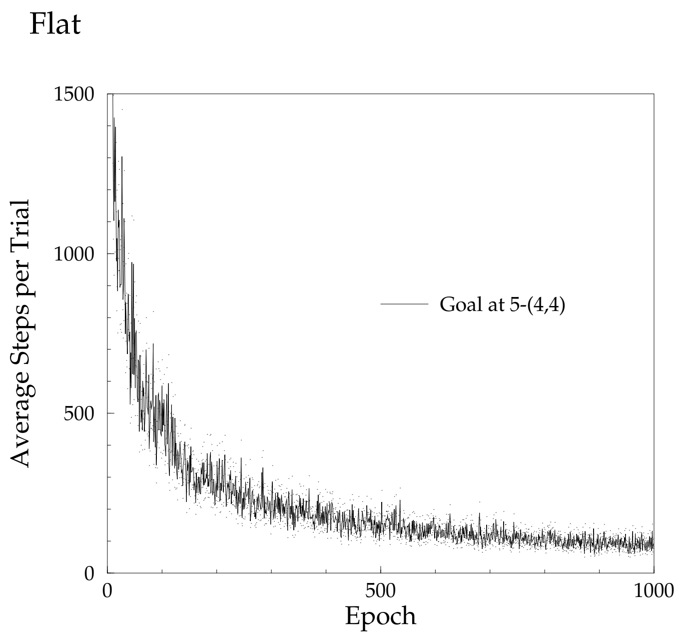

Figure 4: The course of learning for a single goal in the 32×32 maze for standard, flat, $\mathcal{Q}$-learning.

given by the number of times that those goals have previously been selected. Figure 4 shows part of the learning curve for a single goal (at 5-(4,4) in the 32×32 maze).

It is apparent that the Feudal system is dramatically better than conventional $\mathcal{Q}$-learning. This is not surprising – the hierarchical decomposition of the task suits the hierarchical decomposition of space exactly, and there is the potential for almost perfect sharing of information between the different tasks (*ie* the different goals). The advantage increases as the size of the example increases (in fact we were unable even to run conventional $\mathcal{Q}$-learning for the 64×64 maze). These results are only for the set of parameters given in the appendix, and therefore should be interpreted with some caution. In particular, there is little reason to think that the same parameters should be optimal for mazes of very different sizes. The Feudal system is slightly more expensive than the conventional system in the first few epochs (5 for 8×8, 2 for 16×16, 1 for 32×32), presumably because exploration is happening at all levels. Lower level managers are learning to satisfy higher level ones even though the higher level ones do not have enough information about the task to specify them correctly. However, this exploration is clearly put to good use, since the Feudal system is so much faster in subsequent epochs. Some aspects of the exploration are still based on a random walk, which is notoriously slow (Whitehead, 1991).

Figure 5 shows a different view of the progress of learning. The stacked graphs show the probabilities of choosing different actions at different levels after various numbers of epochs, for the case in which the goal is at 3-(4,4) (in an 8×8 maze) and random action choice probability $\pi = 0$. Throughout these epochs, the agent has never had 3-(4,4) as a goal, and therefore these graphs show the transfer of learning from the other goals. Various effects are apparent in these plots. Most dramatic is the extent of transfer. The only thing the agent will ultimately have to learn if it is set 3-(4,4) as a goal is how to get there from 3-(3,4), 3-(3,3) and 3-(4,3). Wherever else the agent starts, high and low level managers are competent to get it to somewhere within 2-(2,2). Learning actually happens both top-down (which is evident on the figures) and bottom-up (which is not). Top-down learning relies on the small number of states at high levels – control at level 1 is quite good even after 10 epochs. Bottom-up learning is not immediately evident, since the figures show the overall probability of selecting the actions rather than the probabilities of selecting actions conditional on a particular managerial command. However, bottom-up learning is clearly required to get top-down learning started.

The graphs also show some potential problems. The system has some difficulty in learning at high levels not to select actions that are unsatisfiable (*eg* going South at 1-(2,1). This is because their unsatisfiability has to be learnt using time-outs. No learning happens at level 1 if the level 1-(2,1) manager specifies action South, and the agent moves to 1-(1,1) or 1-(2,2). One could imagine penalising a manager directly for selecting an action that its subordinates never seem to be able to achieve, but that might hinder them from being made to try hard enough.

Note also that the ultimate action choice in 2-(1,1) (the bottom-left square at the level with 4×4 states) seems to be North. One of the differences between $\mathcal{Q}$-learning and the actor-critic architecture (Barto, Sutton & Anderson, 1983) is that under $\mathcal{Q}$-learning, actions that have equal costs should eventually be selected with equal probabilities, and therefore one would expect North and East to be equally favoured. Ultimately this would actually be the case, given the learning and exploration choices made by the agent (Barto, Bradtke & Singh, 1991). However, the learning rates specified in the appendix vary with the reciprocal of the number of epochs, and therefore it can take a while for any substantial differences in the $\mathcal{Q}$-values for North and East to be erased when the agent generally follows an optimal alternative in any case.

Figure 6 shows the effect of introducing more or less stochasticity into the way the agent moves in the 16×16 version of the maze. With probabilities π, shown on the graph, the moves the agent selected were discarded, and random move were substituted. The runs for the previous graphs were done with $\pi = 0.1$. Although making the problem less deterministic does make the performance of the system worse (as it is bound to), the deterioration is not really significant until each move has only

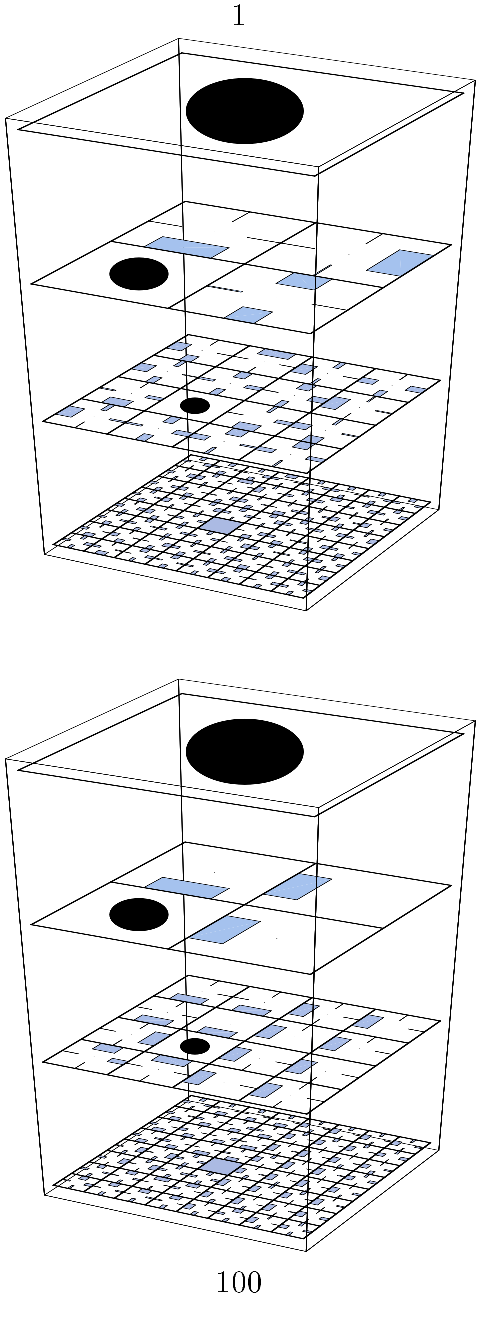

Figure 5: The overall probabilities of taking different actions at the different levels after 1, 10, 1000 and 5000 epochs for the (never-selected) goal at 3-(4,4) and action choice temperature $T = 1.0$ (see appendix). The area of the rectangles in each square is proportional to the overall probability of making a move in that direction. The black circles indicate $*$ actions. See text for discussion.

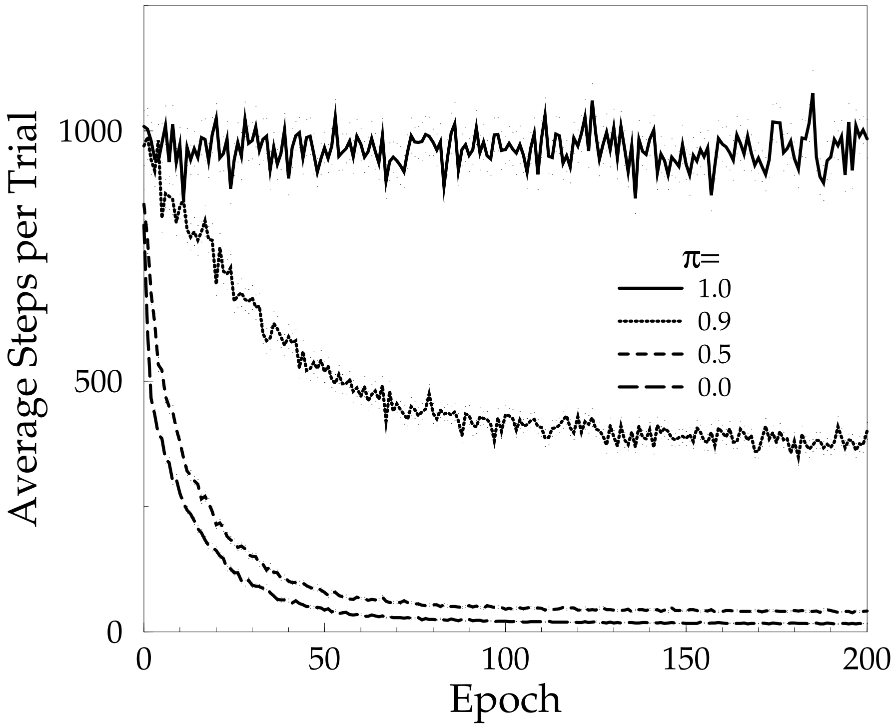

Figure 6: The effect of stochasticity. With probability π, the move selected by the agent was discarded and a random one substituted. The graph shows how this affects the learning performance of the system. Not until $\pi \sim 0.5$ is much difference apparent. The lines are averages over 50 runs with different random seeds in a 16×16 maze, and the haloes of points show the standard errors.

---

a 50% chance of being what the agent selected and 50% chance of being a random move. Since the agent is moving on a Manhattan maze, there are often two equally good moves, which reduces the effect of the uncertainty. If $\pi = 1$, the agent can neither learn nor express the effect of its learning – this is why the top line is flat.

If a barrier is introduced into the maze, then one of three things happens. If the barrier respects the hierarchical division of space then learning proceeds essentially as before, except that some of the paths are longer. If the barrier does not respect the division of space, then, as discussed above and below, the partial observability of the Markov chain can require different actions to be specified within the same high-level state. If the action selection system is ultimately deterministic (or nearly so, as the parameter governing the trade-off between exploration and exploitation comes to favour the latter), then this can cause severe problems when getting to the goal requires traversal of these inconsistent states (Singh, Jaakkola & Jordan, 1994; Jaakkola, Singh & Jordan, 1994). However, figure 7 confirms that if there is some residual randomness in the choice of actions (as in the case that there is some probability that a completely random action is chosen on each step) then performance is restored. The correct alternative, as in Moore (1994) is further to divide the space at higher levels, removing the requirement for inconsistencies in action selection.

4 Discussion

4.1 The Feudal System

The experiments here show the utility of the Feudal system, given that the state space is divided up in a generally sensible way, but without predigested knowledge about how levels in the hierarchy should interact. Although the resulting paths can be suboptimal if there is a barrier, as is almost inevitable using a hierarchical division of space, acquisition is far faster, since the Feudal structure learns and uses richer information about how to move around the space than the flat system. Acquiring this information takes time (*ie* steps), and storing it takes memory – so there are costs as well as benefits. This message is hardly surprising – hierarchical organisations such as certain corporations or governments embody significant rigidities and inefficiencies. The trouble is that efficiently controlling structures that are more heterarchical is even less a solved problem.

Our earlier work on the Feudal system (Dayan & Hinton, 1993) considered a navigation task that was less well suited to a control hierarchy (there was a one goal for 1000 iterations, followed by a different one, and no indication that the goal had changed or where it was, was provided to the agent). For this, the benefits of using a hierarchy only accrue during exploration (at the beginning and subsequently when the goal is moved). For a fixed goal, the high level actions will eventually become constant, and therefore there is no need for low level tasks to be shared for different high level goals. Nevertheless, even in this case, in a 32×32 maze, the system was faster than non-hierarchical $\mathcal{Q}$-learning to learn where the goal was, and much faster to learn to where it had been moved after 1000 iterations. Those results were also based on not necessarily optimal parameters.

The most serious flaw with the Feudal system is that the non-observability of the Markov process from the perspective of the higher-level managers may require them to produce inconsistent commands to their subordinates. Figure 8 shows an example of this – even if the state-spaces of the sub-managers of manager β correctly carve up the region according to the barriers, β has to issue three different commands according

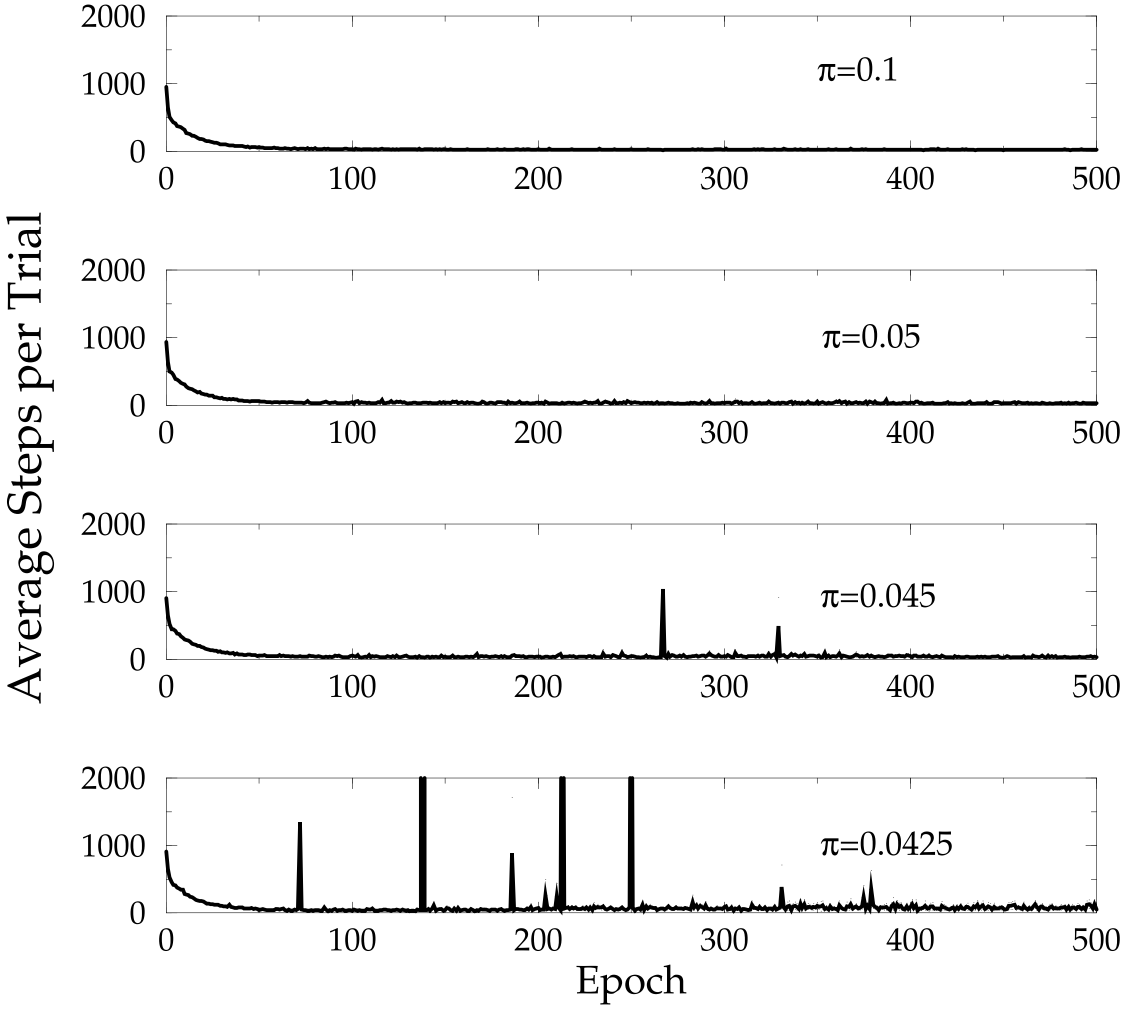

Figure 7: The effects of a barrier. A U-shaped barrier was placed into the 16×16 maze isolating the rectangle of states with corners 4-(3,5), 4-(3,7), 4-(7,7) and 4-(7,5) from all but Northwards transitions to the rest of the maze (but leaving intact all transitions inside this rectangle). The graphs show the average number of steps per trial over 500 epochs for different values of π averaged over 40 runs with standard error haloes. If $\pi = 0.1$, the barrier has little effect (it increases the ultimate average number of steps from 16 to 20), but for smaller values of π, learning becomes unstable. For $\pi = 0.0425$, the average for iteration 138 was 74820 steps. The agent failed to complete 40 runs without getting stuck for $\pi = 0.04$ or lower.

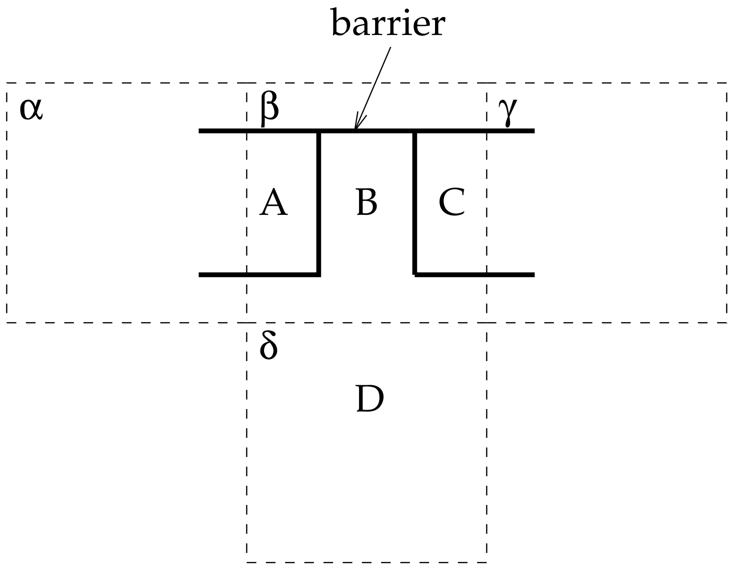

Figure 8: Counter-example for Partial Observability. Managers (α, β, γ and δ) control the areas shown. If the goal is at D and the agent cannot cross the barrier, manager β has to issue three different commands for the three different starting points A, B and C.

---

to where the agent actually is. This is impossible since it fails to see the space with the required resolution. RL solutions to partially observable Markov problems are currently the focus of substantial investigation (Cassandra, Kaelbling & Littmann, 1994; Jaakkola, Singh & Jordan, 1994; Singh, Jaakkola & Jordan, 1994), but having to solve an artificially introduced and complicated problem is hardly computationally attractive.

Moore's (1994) Parti-Games algorithm faces a similar concern, since it seeks to divide state-space up at as coarse a resolution as possible. It uses games-theoretic control theory – all the experience of the agent is stored, and if, just on the basis of this information, there appears to be a way in which the agent can fail to reach the goal, then the space is further sub-divided until this is no longer true. The same information about the inability of the agent to solve the problem is available in the sub-managerial $\mathcal{Q}$-values (subject to some assumptions about maximal lengths of paths within a managerial area), and could be used to split the domains of existing managers. To do this efficiently, however, would probably require the agent to build a model of the world.

There is one easily identifiable inefficiency in the existing system. At the end point of learning, managerial $\mathcal{Q}$-values should be averages over sub-managerial $\mathcal{Q}$-values, where averages are weighted according to the frequency of occurrence of the sub-managerial states and proper account is taken of the extra managerial reinforcements that induce the sub-managers to perform correctly. This should be used at least as a statistical check as learning proceeds. This requirement also points out a further possible problem with the hierarchical system, also stemming from the partial unobservability at high levels. It could well be that the task is presented in such a way that these averages are unstable over time (*eg* a non-stationary distribution of start points is used). Conventional $\mathcal{Q}$-learning will converge on an optimal policy irrespective of this, whereas the high levels of the Feudal system could be forced to track the averages as they change. There will be cases in which this leads to suboptimal behaviour. Singh (1992c) describes a hierarchical scheme (discussed in more detail below) in which the only role of higher levels is to speed the training of the value function for the lowest level. His scheme operates in a domain in which this averaging is unnecessary.

Performance was evaluated in deterministic and stochastic environments. The Feudal system is not particularly sensitive to stochasticity in the final actions in the world. However, it would be vulnerable to a different sort of stochasticity – if a sub-manager and a manager can disagree about the task the latter has set the former. This would affect both learning and use.

4.2 Related Work

There are innumerable suggestions and implementations of hierarchical systems even just in the narrow field of connectionism (Jacobs, Jordan, Nowlan & Hinton, 1990) and reinforcement learning, let alone in artificial intelligence (to name just a few: Sacerdoti's (1973) ABSTRIPS, Korf's (1985) macro-operators, and the subsumption architecture, Brooks, 1989; Mahadevan & Connell, 1991), cybernetics (*eg* Powers, 1973), and conventional control theory, and it would be impossible to do them all justice. The themes are constant – trying to get faster search, more compact controllers, or to adapt correctly to a hierarchical environment. Each discipline brings its own assumptions and prejudices. A number of RL groups almost simultaneously published systems that bear some similarity with the Feudal architecture, and we will discuss them to highlight some of the alternative options. We are making no claim that they are all the same or that one subsumes the others. Indeed this is false.

Singh (1992a;b;c) discussed two aspects of hierarchical RL systems. His first paper (1992a) introduced the compositional $\mathcal{Q}$-learning architecture (C$\mathcal{Q}$-L) in which there is a collection of $\mathcal{Q}$-modules and a gating network (Jacobs, Jordan, Nowlan & Hinton, 1990) that selects between them. The system was designed for compositional tasks ('get to A', 'get to A and then B'), and it was intended that a single $\mathcal{Q}$-module be responsible for just one of the elemental tasks ('get to A'), and the gating network should choose the $\mathcal{Q}$-modules in the right order to achieve the overall goal. Kaelbling (1993) pointed out the charitable nature of this class of tasks – there need be no penalty for using a hierarchical control structure.[4] Single $\mathcal{Q}$-modules just received as input the state of the agent in the world; the gating network received both a unique identifier for the overall task, and also 'completion bits' which reported that the agent had completed sub-goals (*eg* that it had already been through A), but not the state in the world. The gating module learnt to choose $\mathcal{Q}$-modules on the basis of their predictive accuracy (essentially minimal temporal difference error), and an extra bias module learnt the difference between the value of a state within an elemental task and its value within an overall composed task. This amounts to adding contributions from the remaining elemental tasks in the sequence. During training, both elemental and composite tasks were specified, and this allowed to system to learn how to separate out the task components and to sequence them appropriately.

It is possible to make an analogy between C$\mathcal{Q}$-L and the Feudal system in which the manager is like the gating module, the sub-managers, the $\mathcal{Q}$-modules, the state abstraction is as just described, and the bias module plays the role of the managerial $\mathcal{Q}$-values.[5] The random choice of tasks for the whole architecture is the equivalent of the managerial choice of tasks for sub-managers. Although the gating network can choose a different $\mathcal{Q}$-module at every time step, its own input does not change until an elemental task is completed, and it has no direct means of monitoring the progress of the elemental task controllers. It does have an indirect means, namely the magnitude of the temporal difference error. This makes for an interesting difference from the Feudal system – even the manager does not know the satisfaction conditions of the actions it sets, instead the elemental $\mathcal{Q}$-modules have to learn them themselves. The training methodology aids this by intermixing the composite tasks with the separate, low level, ones of which they are built.

Singh (1992b;c) proved that a controller can use hierarchical methods based on elemental controllers, like the ones learnt in Singh (1992a), to lengthen its span of temporal credit assignment. Given the existence of competent elemental controllers, all that matters is how expensive their controls are, not what they are. Given a set of competent sub-managers, a Feudal manager learns to operate in almost exactly this manner. A key difference in Singh (1992c) is that the high-level controllers were only used to control the learning of the value function (using 'imagined' backups, as in Sutton's (1990) DYNA scheme), and actions were always selected by the controller at the lowest level. That this is possible relies on the compositional nature of the tasks the agent faces.

[4]The same is true of the pairwise Manhattan navigation task in the absence of barriers.

[5]Note that the manager chooses sub-managers rather than setting them tasks, as in the Feudal system. However, as discussed above, this is equivalent under a table-lookup scheme for representing the $\mathcal{Q}$-values.

Aspects of other navigation systems were also mentioned above. Kaelbling (1993) looked at a pairwise navigation task in a stochastic environment. She used a spatial hierarchy with a network of selected landmarks at the upper level. At the lower level, there were two controllers. First, her system employed a $\mathcal{Q}$-learning system that learnt from the environment the shortest expected paths from every point to each of the landmarks which are *neighbours* of that point's own landmark. Second, it had pairwise navigation controllers which operated between all points closest to the same landmark. Given a start state and a goal, the higher level control system would report on the landmark nearest to the start state, and the next landmark in the shortest landmark-landmark route going from this one to the one nearest the goal. The low level control system would then direct the agent towards this neighbouring landmark. As the state of the agent changed (including a stochastic component), this process was repeated – so the agent did not need actually to go through the landmarks – it was always being routed towards the next one in sequence. When the agent was in the domain of the landmark closest to the goal, the low level used the pairwise navigation system to get there. Kaelbling (1993) used a manually designed decomposition of space, but now (personal communication) has a system that learns a decomposition. The whole process can be recursed to generate a multi-level system.

If one imagined that each of the divisions of space in the feudal system had an associated landmark, then the Feudal pairwise navigation system can be seen as instantiating something like Kaelbling's system, although the learning is different. If there are only two levels and the goal does not lie within the current state, then the choice of action at the top level amounts to a directive to head for the top level state which the agent believes to be next on the route to the goal. As soon as the agent leaves the domain of the first high level state, a new action is selected at the high level, and the agent heads for the succeeding high level area – the continual routing towards the subsequent landmark. Learning around the goal is different in the Feudal system, as is the hierarchical decomposition of the representation of the goal.

Lin (1993) performed complexity analysis for goal finding in a deterministic single-goal navigation system, again using landmarks (which he calls abstract states). He considered the regime in which it is possible explicitly to train the low level controllers initially (this requires a local 'reset' operator which takes the agent from a landmark to a random state within its domain), and then learns how to sequence them together. Unlike Kaelbling's (1993) system, the ultimate control requires the system explicitly to go through each of the landmarks on the route to a goal (which in general would lead to longer paths), however, this is a function of the instantiation rather than the architecture. Lin showed how adopting a hierarchy can dramatically lessen the complexity of finding paths to the goal. Unfortunately, although deeper hierarchies lead to reduced complexity (at least up to a point), they also tend to lead to worse paths.

Jameson's (1992) system, described briefly above, solved the temporal credit assignment problem that the pole-balancer faces, which arises from the possibility that the cart to which the pole is attached can drift slowly towards end-stops. No explicit spatial abstraction was employed, rather a high-level controller (*ie* the manager) was clocked at an explicitly slower rate than the low-level controller (the sub-manager), forcing its commands to be based on different information. Only the sub-manager actually controlled the pole; the manager provided direct task input to the sub-manager through a component of its state, in a similar manner to the Feudal system. His systems used the actor-critic architecture of Barto, Sutton & Anderson (1983) with a multi-layer perceptron representation (Anderson, 1986), and he also looked at the utility of learning world models.

Jameson tried two versions. In the first, all the reinforcement for the sub-manager came from the manager - and was explicitly the squared difference of the angle of the pole from a reference signal (*ie* command or task) provided by the manager. The sub-manager thereby had no temporal credit assignment problem because it received a direct error signal – although it did have to learn the relationship between the task input and the reinforcement provided. This error signal does not completely respect the original hierarchical intent, which might be to have the sub-manager worry about the pole falling over in the short term and the manager worry about the slow drift. Rather, if the pole starts to fall over, the sub-manager must take an unrecoverable reinforcement penalty during the time that it must deliberately move the pole further away from the managerial reference signal such that it can ultimately return it there.

Jameson's second system, which he called response induction learning, changes this by providing the sub-manager with direct reinforcement from the world rather than reinforcement from the manager. If it nevertheless still received a task input from the manager, it would have no incentive to do anything with it, since its reinforcement did not depend on it. Jameson therefore explicitly encouraged hierarchical behaviour by explicitly encouraging the sub-manager's critic (the state value function) to depend on the manager's task request.

Lin (1993) raised the important question of high level actions that (ideally) should never terminate – such as keeping the pole in balance. Jameson's (1992) system shows that this should not be a particular problem for hierarchical control systems.

There are other hierarchical architectures for RL that are further removed from the Feudal system. Ring (1993) describes one such system which implements a bottom-up construction of a hierarchy. High-level actions are constructed by a process of disambiguating cases in which particular lower level actions apparently sometimes should and sometimes should not follow each other, on the basis of the reinforcement of their consequences (*cf* chunking, Rosenbloom, Laird & Newell, 1989). Ring tried his algorithm in partially observable Markov chains, for which a main task for the agent is (implicitly) working out where it is, on the basis of the recent history of perceptual and motor actions. The agent does not have to distinguish states except in as much as they have different requirements for action (Whitehead & Ballard, 1990), and the result of the learning process is something like the Update Graph of Rivest and Schapire (1987; 1989), except that states are not disambiguated unless there is a requirement to do so which comes from the need to maximise reinforcement. Ring's system would not build a hierarchy in fully observable deterministic Markov chains, since the lack of ambiguity obviates the need for disambiguation. From the perspective of this paper, the lack of top-down control would limit the use for things like changed tasks in the same domain.

Watkins (1989) discussed the case of supervisory control. Managers shared some elements of their state-space with sub-managers, and could continually issue commands. For this sort of control, the sub-managerial decision problems can become non-Markovian too. This happens because the manager's command is incorporated as part of the state-space of the sub-managers – and it can change according to information to which the sub-managers are not privy. Watkins discussed ways in which the problem this raises can be mitigated – including having the sub-manager build a model of its manager. These have yet to be tested.

Using hierarchies is one way of addressing some of the problems of search and representational complexity. Although we find it natural to solve tasks in a hierarchical way, the resulting sub-optimalities can be significant – for instance, one might expect few evolved systems to respect any strong hierarchical constraints. This paper has explored a reinforcement learning system inspired by feudal notions of control. It has shown its utility and discussed some problems in the context of a maze task, and related it to other recent hierarchical reinforcement learning systems. All of these together barely scratch the surface of the space of layered control systems.

Acknowledgements

This work started as a collaboration with Geoff Hinton (Dayan & Hinton, 1993). I am very grateful to him and Leslie Pack Kaelbling, Andrew Moore, Jürgen Schmidhuber, Chris Watkins and Ron Williams for discussions and Sageev Oore, Mark Ring and Lisa White for extensive comments on earlier drafts of this paper. Support was from the Canadian NSERC.

Appendix

This appendix describes Feudal learning explicitly and gives the values used for the various parameters.

Following Barto, Bradtke & Singh (1991), for a state i (at one level) and managerial command z (we will refer to the state as i, z) after k iterations, define

$$f_k(i,z) = \min_{u \in U(i,z)} \mathcal{Q}_k(i,z,u)$$

where $U(i,z)$ is the set of possible actions at state i, z (N,S,E or W unless the goal lies within the domain of the state in which case the only possible action is $*$) and $\mathcal{Q}_k(i,z,u)$ is the $\mathcal{Q}$-value (Watkins, 1989).

Let $0 < \alpha_k(i,z,u) < 1$ be the learning rate for action u at state i, z after k iterations. If this action is selected at this state for iteration $k+1$, then

$$\mathcal{Q}_{k+1}(i,z,u) = [1 - \alpha_k(i,z,u)]\mathcal{Q}_k(i,z,u) + \alpha_k(i,z,u)[c + f_k(\mathsf{hsuc}(i,z,u))] \quad (1)$$

where c is the immediate cost of doing action u and hsuc is generally the state the controlled Markov process enters after doing this action. Both hsuc and c are more complicated for the Feudal architecture depending on whether and why there is a state transition observed at a given level. Specifically, if j is the next state of the chain observed at a given level, and $c_{i,z}$ is the total number of steps observed at that level since the last state change, and @ is a special state for which (abusing notation somewhat) $f_k(@, ?) = 0$, then:

Condition	c	hsuc
agent gets to the goal	0	@
z was successfully executed	$c_{i,z}$	@
z was unsuccessfully executed	$10+c_{i,z}$	i
managerial state is unchanged	$c_{i,z}$	j
time-out at the level of i	$c_{i,z}$	i

Action $u \in U(i,z)$ is selected with Boltzmann probability

$$p_k(u) = \frac{e^{-\mathcal{Q}_k(i,z,u)/T_l(g)}}{\sum_{v \in U(i,z)} e^{-\mathcal{Q}_k(i,z,v)/T_l(g)}}$$

where $T_l(g)$ is a temperature parameter indexed by the current goal g and l counts the overall number of steps the agent has used in getting to that goal over the whole course of learning. T updates according to:

$$\begin{aligned} T_0(g) &= T_{\text{MAX}} \\ T_{l+1}(g) &= T_{\text{MIN}} + \beta(T_l(g) - T_{\text{MIN}}) \end{aligned}$$

where exactly the same parameters as in Barto, Bradtke & Singh (1991) were used – $\beta = 0.992$, $T_{\text{MAX}} = 75$ and $T_{\text{MIN}} = 0.5$. Making T a function of the goal is sub-optimal, since, as seen in figure 5, the agent learns a substantial amount about how to get to one goal from its experience navigating to another. Provided that all settings of T allow the agent to try all actions at all states infinitely often, its choice does not affect the convergence theory for $\mathcal{Q}$-learning (Watkins, 1989; Watkins & Dayan, 1992; Tsitsiklis, 1993; Jaakkola, Jordan & Singh, 1994).

The learning rates α_k were taken to be global functions of the iteration:

$$\alpha_k = \frac{\alpha_0 \tau}{\tau + n}$$

where n counts the number of epochs (sets of 20 trials), and $\alpha_0 = 0.5$ and $\tau = 50$.

For flat $\mathcal{Q}$-learning, as in Barto, Bradtke & Singh (1991), each state-action pair had its own learning rate, with $\alpha_0 = 0.5$ and $\tau = 300$. The Feudal systems used a lower τ since n counts whole epochs, which might potentially involve thousands of state-action pairs.

References

[1] Anderson, CW (1986). *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD Thesis, University of Massachusetts, Amherst, MA.

[2] Barto, AG, Bradtke, SJ & Singh, SP (1991). *Real-Time Learning and Control using Asynchronous Dynamic Programming.* TR 91-57, Department of Computer Science, University of Amherst, MA.

[3] Barto, AG, Sutton, RS & Anderson, CW (1983). Neuronlike elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man, and Cybernetics,* **13**, pp 834-846.

[4] Barto, AG, Sutton, RS & Watkins, CJCH (1990). Learning and sequential decision making. In M Gabriel & J Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks.* Cambridge, MA: MIT Press, Bradford Books.

[5] Bellman, RE (1957). *Dynamic Programming.* Princeton, NJ: Princeton University Press.

[6] Brooks, R (1989). A robot that walks. Emergent behaviors from a carefully evolved network. *Neural Computation,* **1**, 253-262.

[7] Cassandra AR, Kaelbling, LP & Littmann, ML (1994). Acting optimally in partially observable stochastic domains. *American Association for Artificial Intelligence.*

[8] Dayan, P & Hinton, GE (1993). Feudal reinforcement learning. In CL Giles, SJ Hanson & JD Cowan, editors, *Advances in Neural Information Processing Systems, 5,* 271-278. San Mateo: CA: Morgan Kaufmann.

[9] Jaakkola, T, Jordan, MI & Singh SP (to appear). Stochastic convergence of iterative DP algorithms. *Neural Computation.*

[10] Jaakkola, T, Singh, SP & Jordan, MI (1994). Monte-Carlo reinforcement learning in non-Markovian decision problems. In *Proceedings of the Seventh ACM Conference on Computational Learning Theory.*

[11] Jacobs, RA, Jordan, MI, Nowlan, SJ & Hinton, GE (1990). Adaptive mixtures of local experts. *Neural Computation,* **3**, pp 79-87.

[12] Jameson, JW (1992). Reinforcement control with hierarchical backpropagated adaptive critics. Submitted to *Neural Networks.*

[13] Kaelbling, LP (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth Machine Learning Workshop.* San Mateo, CA: Morgan Kaufmann.

[14] Korf, RE (1985). Macro-operators: A weak method for learning. *Artificial Intelligence,* **26**, 35-77.

[15] Lin, L-J (1993). *Reinforcement Learning for Robots using Neural Networks.* PhD Thesis, School of Computer Science, Carnegie Mellon University.

[16] Lovejoy, WS (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research,* **28**, 47-66.

[17] Mahadevan, S & Connell, J (1991). Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eighth International Workshop on Machine Learning,* 328-332.

[18] McCrea, WH & Whipple, FJW (1940). Random paths in two and three dimensions. *Proceedings of the Royal Society of Edinburgh,* **60**, 281-298.

[19] Moore, AW (1991). Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state spaces. In *Proceedings of the Eighth Machine Learning Workshop.* San Mateo, CA: Morgan Kaufmann.

[20] Moore, AW (1994). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In JD Cowan, G Tesauro & J Alspector, editors, *Advances in Neural Information Processing Systems 6.* San Mateo, CA: Morgan Kaufmann.

[21] Morin, TL (1978). Computational advances in dynamic programming. In ML Puterman, editor, *Dynamic Programming and its Applications.* New York: Academic Press.

[22] Polya, G (1945). *How to Solve It.* Princeton, NJ: Princeton University Press.

[23] Powers, WT (1973). *Behavior: The Control of Perception.* New York, NY: Aldine.

[24] Ring, MB (1993). Two methods for hierarchy learning in reinforcement environments. In JA Meyer, H Roitblat & S Wilson, editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior"*, 148–155. Cambridge, MA: MIT Press.

[25] Rivest, RL & Schapire, RE (1987). A new approach to unsupervised learning in deterministic environments. In P Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning,* 364-375.

[26] Rivest, RL & Schapire, RE (1989). Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing,* 411-420.

[27] Rosenbloom, PS, Laird, JE & Newell, A (1989). The chunking of skill and knowledge. In BAG Elsendoorn & H Bouma, editors, *Working Models of Human Perception.* London, England: Academic Press.

[28] Sacerdoti, ED (1973). Planning in a hierarchy of abstraction spaces. In *Advance Papers of the Third International Joint Conference on Artificial Intelligence,* 412-422.

[29] Singh, SP (1992a). Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning,* **8**, pp 323-340.

[30] Singh, SP (1992b). Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the Eleventh Machine Learning Conference.*

[31] Singh, SP (1992c). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence,* 202-207.

[32] Singh, SP, Jaakkola, T & Jordan, MI (1994). Model-free reinforcement learning for non-Markovian decision problems. In *Proceedings of the Eleventh International Conference on Machine Learning.*

[33] Sutton, RS (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning,* 216-224.

[34] Tsitsiklis, J (to appear). Asynchronous stochastic approximation and $\mathcal{Q}$-learning. *Machine Learning.*

[35] Watkins, CJCH (1989). *Learning from Delayed Rewards.* PhD Thesis. University of Cambridge, England.

[36] Watkins, CJCH & Dayan, P (1992). $\mathcal{Q}$-learning. *Machine Learning,* **8**, 279-292.

[37] Whitehead, SD (1991). Complexity and cooperation in $\mathcal{Q}$-learning. In *Proceedings of the Eighth International Conference on Machine Learning (ML91),* 363-367.

[38] Whitehead, SD & Ballard, DH (1990). Active perception and reinforcement learning. *Neural Computation,* **2**, pp 409-419.