A Symbol's Role In Learning Low Level Control Functions

Chris Drummond

A Thesis
Submitted to the School Graduate Studies and Research
of the University of Ottawa
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in Computer Science*

School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario

*The Doctoral Program in Computer Science is a joint program with Carleton University, administered by the Ottawa Carleton Institute for Computer Science

Abstract

This thesis demonstrates how the power of symbolic processing can be exploited in the learning of low level control functions. It proposes a novel hybrid architecture with a tight coupling between a variant of symbolic planning and reinforcement learning. This architecture combines the strengths of the function approximation of subsymbolic learning with the more abstract compositional nature of symbolic learning. The former is able to represent mappings of world states to actions in an accurate way. The latter allows a more rapid solution to problems by exploiting structure within the domain. A control function is learnt over time through interaction with the world. Symbols are attached to features in the functions. The symbolic attachments act as anchor points used to transform the function of a previously learnt task to that of a new task. The solution of more complex tasks is achieved through composing simpler functions, using the symbolic attachments to determine the composition. The result is used as the initial control function of the new task and then modified through further learning. This is shown to produce a significant speed up over basic reinforcement learning.

Acknowledgments

This thesis is the culmination of a task long but made less arduous by the support of others. It would be hard to envisage its completion without the guidance of my supervisor, Dr. R. Holte. I thank him for the many long hours of discussion that distilled out the ideas presented in this thesis. My thanks also go to my colleagues in the "Knowledge Acquisition and Machine Learning" group, and particularly Dr. S. Matwin, not only for many useful suggestions but also for giving me a chance to be part of such a stimulating research community. I would also like to thank the Natural Sciences and Engineering Research Council of Canada and the Ontario Government for the scholarships that made this thesis possible.

Contents

1 Introduction **1**

1.1 Intuitions Motivating this Research . . . 2

1.2 System Architecture . . . 4

1.3 Identifying and Composing Subtasks . . . 7

1.4 Research Contributions . . . 12

1.5 Thesis Outline . . . 13

2 Related Research **14**

2.1 Justification . . . 14

2.2 Positioning . . . 17

2.2.1 Symbolic Systems . . . 17

2.2.2 Subsymbolic Systems . . . 20

2.2.3 Hybrid Systems . . . 23

2.3 Novelty . . . 25

2.3.1 Transfer . . . 26

2.3.2 Abstraction . . . 29

3 The Control Function **32**

3.1 Spline Function Approximation . . . 35

3.1.1 B-splines . . . 36

3.1.2 Accuracy of Fit . . . 39

3.2 Features in the Control Function . . . 43

3.2.1 Changing the Goal Position . . . 43

3.2.2 Detecting Features Early . . . 45

3.2.3 Features in Different Domains . . . 47
3.2.3.1 A Robot Arm . . . 48
3.2.3.2 A Car on a Hill . . . 50
3.3 In Summary . . . 52

4 Subsymbolic Learning **55**
4.1 Reinforcement Learning . . . 57
4.1.1 Function Approximation . . . 61
4.2 Other Low-Level Learning Methods . . . 63
4.3 Using Reinforcement Learning with Splines . . . 64
4.3.1 An Example of Divergence Using Splines . . . 65
4.3.2 Using Smoothing Splines to Eliminate Overshoot . . . 68
4.3.3 Finding the Right Smoothness Constraint . . . 70
4.3.4 Preventing Divergence . . . 75
4.4 In Summary . . . 77

5 Task Partitioning **78**
5.1 The Snake Approach . . . 80
5.2 Three Extensions to the Snake Approach . . . 80
5.3 Details of the Snake Approach . . . 88
5.3.1 Internal Forces . . . 88
5.3.2 External Forces . . . 90
5.3.3 The Snake Dynamics . . . 91
5.3.4 Scaling the Function . . . 93
5.4 Shape Extraction . . . 95
5.4.1 Polygonal Shapes . . . 96
5.4.2 More General Shapes . . . 97
5.4.3 Snake in Higher Dimensions . . . 98
5.5 In Summary . . . 104

6 The Symbolic Level **107**
6.1 Creating Graphs . . . 108

6.1.1 Creating Subgraphs 109
6.1.2 Merging Subgraphs 111
6.2 Symbolic Planning 113
6.2.1 Decision Boundaries 116
6.2.2 The Matching Process 119
6.2.2.1 Selecting Similar Shaped Subgraphs 119
6.2.2.2 Determining the Transformation Error 124
6.2.2.3 Transforming the Function 128
6.2.3 Composing Search Functions 129
6.3 In Summary 130

7 Experiments 133
7.1 Experimental Set up 133
7.2 Experimental Results 136
7.2.1 Robot Navigation, Goal Relocation 136
7.2.2 Robot Arm, Goal Relocation 139
7.2.3 Robot Navigation, New Environment 141
7.2.4 Robot Arm, New Environment 143
7.3 Analysis of Results 144
7.3.1 Limitations of the Experimental Set Up 144
7.3.2 Limitations with Doorway Detection 149
7.3.3 Performance Variation with Room Configuration 153
7.4 In Summary 157

8 Philosophical Implications 158
8.1 Symbolism and its Critics 159
8.2 Relationship to the Language of Thought 160
8.2.1 Arguments for a Language of Thought 160
8.2.2 Symbols and Propositional Attitudes 162
8.2.3 From Attitudes to Actions 163
8.3 A Layered Approach 164
8.3.1 Levels of Representation 165

8.3.2 Coupling the Layers 165
8.4 Defining A Position 168
8.4.1 Grounding in the World 170
8.4.2 Grounding in the Physical Substrate 172
8.4.3 Representations 173
8.4.4 Intelligence Tests 175
8.5 In Summary 177

9 Conclusions **179**
9.1 Limitations 181
9.1.1 Limitations in the Approach 181
9.1.2 Limitations in the Implementation 183
9.2 Future Work 185
9.2.1 Iterative Updating 185
9.2.2 Representing the Control Function 187
9.3 A Few Final Words 189

Bibliography **190**

List of Figures

1.1 System Architecture . . . 5
1.2 Robot Navigating Through a Series of Rooms . . . 8
1.3 The Reinforcement Learning Function . . . 9
1.4 Functions for Subtasks . . . 9
1.5 Adding in the First Function . . . 10
1.6 Adding in the Second Function . . . 11
1.7 The Function Produced by Composition . . . 12

3.1 The Control Function . . . 32
3.2 Combining Functions for Each Action . . . 34
3.3 The Family of B-splines . . . 36
3.4 Function Approximation with B-splines . . . 37
3.5 A Two Dimensional Cubic B-spline Basis . . . 38
3.6 Functions to be Approximated . . . 40
3.7 Fitting to an Exponential . . . 40
3.8 Fitting to an Exponential with One Step . . . 42
3.9 Fitting to an Exponential with Two Steps . . . 42
3.10 The Features Generated by Two Different Tasks . . . 44
3.11 The Old Task . . . 45
3.12 The New Task . . . 45
3.13 Start Function . . . 46
3.14 No Walls Function . . . 46
3.15 Early Function . . . 47
3.16 Early Gradient . . . 47

3.17 New Task Function 47
3.18 New Task Gradient 47
3.19 Work Space 49
3.20 Configuration Space 49
3.21 The Reinforcement Learning Function 50
3.22 The Gradient 51
3.23 The Gradient after 4000 steps 51
3.24 The Car on the Hill 52
3.25 Car State Space 53
3.26 Reinforcement Learning Function 54
3.27 The Gradient 54

4.1 Subsymbolic Layer 55
4.2 A Discrete State Space 57
4.3 Propagating Back Values 58
4.4 Discrete Q-learning 60
4.5 Fitting with Linear Splines 64
4.6 Overshoot using Cubic B-splines 65
4.7 Overshoot along a Discontinuity 66
4.8 Divergent Approximation 66
4.9 Preventing Overshoot 69
4.10 The Kernel 71
4.11 Smoothing Followed by Least Squares Fit 72
4.12 Reflecting the Function 73
4.13 Convergent Approximation: Cubic Splines 76
4.14 Convergent Approximation: Linear Splines 76

5.1 Task Decomposition 78
5.2 A Complete Partition 79
5.3 Fitting the Snake 81
5.4 Controlling Forces on The Snake 82
5.5 Using the Ballooning Force 83

5.6 Thresholded Features . . . 85
5.7 Smoothed Function . . . 85
5.8 Mercury Flow . . . 86
5.9 Arm "Room" . . . 87
5.10 A Smoothed Function . . . 94
5.11 Scaling for Distance from Goal . . . 94
5.12 Partition for the Robot Arm . . . 95
5.13 A Cross Shaped Room and its Partition . . . 97
5.14 A Z-Shaped Room and its Partition . . . 98
5.15 Car on the Hill: Auto Scaling . . . 99
5.16 Car on the Hill: Lower Scaling . . . 99
5.17 Adding a Z-Dimension . . . 100
5.18 Delimiting the First Room . . . 100
5.19 Delimiting the Second Room . . . 101
5.20 The Complete 3D Partition . . . 101
5.21 The Complexity of the Snake Algorithm . . . 103
5.22 The Partitioning Algorithm . . . 106

6.1 Symbolic Level . . . 107
6.2 From a Function to a Graph . . . 108
6.3 Extracting a Case . . . 109
6.4 Direction of the Gradient at a Doorway . . . 111
6.5 Merging the Graphs . . . 112
6.6 Merging Nodes . . . 113
6.7 Using Dijkstra's Algorithm . . . 114
6.8 Transformation . . . 115
6.9 Concatenating Functions . . . 115
6.10 Multiple Paths to Goal . . . 117
6.11 Combining Two Functions . . . 117
6.12 Decision Functions . . . 118
6.13 Combining Decision Functions . . . 119
6.14 Isomorphic Mapping . . . 120

6.15 Weighting Graph Nodes 123
6.16 Symmetric Transforms 125
6.17 Affine Transforms 125
6.18 Affine Transforms on a Unit Circle 127
6.19 Interpolation and Extrapolation 129
6.20 A Search Function 130
6.21 The Planning Algorithm 132

7.1 Steps in the Experiment 134
7.2 The Different Suites of Rooms 137
7.3 Learning Curves: Robot Navigation, Goal Relocation 138
7.4 The Robot Arm Obstacle and Goal Positions 140
7.5 Learning Curves: Robot Arm, Goal Relocation 140
7.6 The Single Rooms 141
7.7 Learning Curves: Robot Navigation, New Environment 142
7.8 The Different Obstacle Positions 143
7.9 Learning Curves: Robot Arm, New Environment 143
7.10 Learning Curves in a Partial Observable Domain 146
7.11 Value "Leak Through" 147
7.12 A Wrongly Positioned Doorway 150
7.13 Effect of Doorway Positioning on Learning Curves 150
7.14 False Doorways 151
7.15 Success in Robot Navigation Moving Goal 154
7.16 Failure in Robot Navigation Moving Goal 155
7.17 Success in Robot Navigation New Environment 156

8.1 Coupling Representations 166
8.2 Coupling Layers 167
8.3 Grounding 169
8.4 Internal Representations 174
8.5 Important Behaviours 176

9.1 The High-level Algorithm 180

9.2 Looking For Shortcuts 186

Chapter 1

Introduction

This research focuses on control tasks where the system must learn the best action to take in a particular world state. A series of such actions describes a trajectory through state space which brings the system closer to some optimal state. The aim of this thesis is to justify a role for symbolic and subsymbolic processing and thus be able to exploit the advantages of both. In the view of this thesis it is in such control tasks that the function approximation of subsymbolic learning must be combined with the more abstract compositional nature of symbolic learning. The former is able to represent mappings of world states to actions in an accurate way. The latter allows a more rapid solution to problems by exploiting structure within the domain.

The role of symbolic processing in intelligence is the subject of much debate. There are many differing opinions held by researchers in Artificial Intelligence and related fields. The opinions range from the view that symbolic processes are all that is needed except for the direct connection to the world to the view that symbolic processes are not needed at all except for language faculties. This thesis takes a position midway between these two extremes. It proposes an approach where the tight coupling of subsymbolic and symbolic levels promotes the rapid learning of control tasks.

This research explores the use of levels of abstraction. The lower levels are represented by multi-dimensional functions, the higher levels by symbolic graphs. A system for learning control tasks must be able to operate in continuous domains, a common occurrence in real world situations such as process control or robot navigation. Here, by representing lower levels as functions, a direct relationship between continuous inputs and outputs is established. By grouping easily recognisable and significant features in these functions, a partitioning of the tasks into subtasks is produced. By associating symbols with regions of the partitions and combining them into graphs, abstractions of the functions are formed. Thus the symbolic level captures important structural information about the tasks. This facilitates a controlled exploration of the environment and the transfer of knowledge between tasks.

1.1 Intuitions Motivating this Research

This research is based on a number of intuitions which are discussed in the following paragraphs. They will be illustrated by example cases outlining scenarios and how the system should perform in such a situation. These examples are used both as an argument for the approach and to clarify the goals of this research.

The first intuition is that symbol manipulation confers some distinct advantages to a system. Central to the symbolic processing view is the idea of compositionality, that symbols can be combined syntactically to create more complex ones. We might equate this notion to the generation of trajectories through some state space in a control system. Suppose that a robot arm moves to some point, picks up an object and places it at some other point. The complete task is, of course, composed of distinct subtasks: that of moving the arm to the first point, that of grasping the object, that of moving the arm to the second point and that of releasing the object. Thus if we consider the solution of each subtask as a movement then one might postulate a language of movements where more complex movements are constructed through composition of simpler ones.

The second intuition is that the symbolic view does not account for everything. The movement of the arm to be in a position to grasp an object may best be described by a smooth trajectory that has no obvious decomposition. The mapping from sensor states to control output seems best learnt in toto, directly from the world. In principle such a mapping might be represented by symbolic rules. But the very notion of symbolism suggests a clear distinction between individual symbols. At such a level any such distinction would be arbitrary. Finding the correct mapping is a task of function approximation rather that syntactic composition and thus is better suited to subsymbolic processing. This mapping is a form of associationism roundly criticised by the exponents of the classical model of symbolic processing. The position taken by this thesis is that, at the lower levels, there are processes which do not have the essential properties of symbolic processing, but that it is exactly these properties that are important and exist in processes at higher levels.

The third intuition is that knowledge should be transferred from the solutions to old tasks to new related tasks. Assuming that the system has a sufficient low-level competence, the solution to any task however complex could be learnt over time through interaction with the world. This tabula rasa learning has the advantage of not requiring knowledge about the environment to learn to successfully carry out a task. Such learning lessens the potential brittleness of a system. Of course, this advantage is somewhat offset by the time taken to learn the task. If the results of prior learning can be exploited then this learning time can be substantially reduced.

If transfer occurs at the level of the whole task, the likelihood of previous learning being relevant is small. If transfer occurs at the level of constituent subtasks, this likelihood will be considerably increased. The ability to syntactically compose complex structures from primitives is the property of symbolic systems which this research aims to exploit. It is useful when the solution of a new task consists, at least approximately, of the composition of solutions of several already learnt subtasks. An example, given earlier in this section, had a robot arm moving an object from one position to another. If we imagine that solutions to the subtasks of moving the arm and picking up and releasing the object had been learnt independently then they can be composed to produce a solution to the composite task.

The fourth intuition is that there are often features in an agent's interaction with its environment which are stable and easy to recognise accurately. The features determine the form of this interaction at some qualitative abstraction. If the features differ by a small amount one would expect the interaction to differ by a small amount. The features are not "in the world" but are dependent on the agent's interaction with the world. This correlates with research into human learning. Piaget's research (Piaget & Inhelder, 1967) suggests spatial learning in children comes from "acting in the world". Grouping these features and attaching them to symbols couples the subsymbolic and symbolic levels. This allows the symbolic level to identify when a new task is the same or similar to a previously solved task and to partition a complex task into subtasks.

The fifth intuition is that the meaning of a complex symbol can evolve over time. Initially the complex symbol is exactly the syntactic composition of its constituent symbols. Assuming the task associated with the complex symbol is carried out many times, refinement of the solution would cause the constituent parts to coalesce. Thus the parts can no longer be considered symbols (within this task anyway) but the whole movement might be considered one.

The strongest criticisms of symbolism stress exactly these cases where the relationship among parts is not the simple syntactic one. It is the view of this thesis that many of these result from the refinement of instances of syntactic composition. Once refined, they form new symbols which can, themselves, be composed in a syntactic manner. The learning of complex movements might parallel the evolution of a phrase. For instance an idiomatic phrase like "the die is cast", has moved from being a parsable expression meaning a gambling die has been thrown, to being a lexical unit essentially a "word" in the language, meaning some decision is irrevocable. In this form it no longer has symbolic parts but might itself be regarded as a symbol. Of course, the evolution of a phrase occurs over a long period of time within a society of individuals. The evolution of a "movement phrase" occurs within an individual on a much shorter time scale and is likely dependent on the frequency of practice. Infrequently used "movement phrases" are likely to remain syntactically distinct.

1.2 System Architecture

The intuitions discussed in the previous section led to the system architecture shown in figure 1.1. The system learns the solution to a control task by taking actions in the world and observing the change in state. This interaction is determined by a control function, a mapping from world states to actions. The function is not necessarily optimal, being constantly revised as the world is explored.

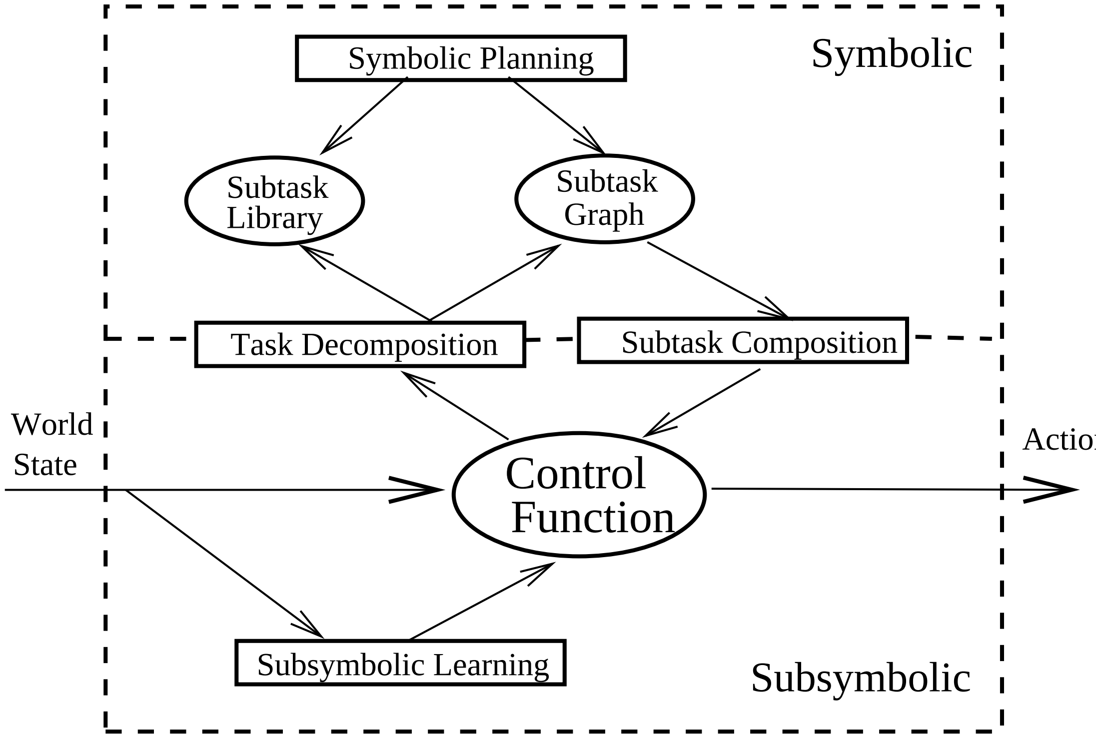

Figure 1.1: System Architecture

Subsymbolic learning updates this function by watching how actions affect the world state. The broad approach discussed in this thesis is not intended to be strongly committed to how subsymbolic learning is carried out, but the method used is a type of reinforcement learning algorithm (Sutton & Barto, 1998). One of the attractive properties of such an algorithm is that it does not require an internal model to learn the best control function. A standard reinforcement learning algorithm, applied to a series of related tasks, could learn each new task independently. It only requires knowledge of its present state and infrequent numerical rewards to learn the actions necessary to bring a system to some desired goal state. But this very paucity of knowledge results in a slow learning rate. This thesis shows how to exploit the results of prior learning to speed up the process while maintaining the robustness of the general learning method.

To speed up learning the task is decomposed into subtasks. The subtasks are identified on the basis of strong features in the control function. Here “strong” means that the features are stable (i.e. relatively insensitive to variations in the low level learning process) and easy to recognise and locate accurately early in the learning process. The features partition the function into regions, each representing knowledge about how to solve a particular subtask. The boundary of each region is a low order polygon demarcating an individual subtask.

The symbolic layer controls the exploration of the environment and the transfer of knowledge between tasks. It converts the individual polygons into graphs. The graphs are used in two ways. Firstly they are added to a library, along with the associated region of the function. Secondly they are merged to form a graph describing the complete task. The symbolic layer uses this graph in a syntactic method of composition much like in symbolic planning, but the novelty arises in that the parts being composed are multi-dimensional real-valued functions. The functions, generated when solving previous tasks, are taken from the subtask library. They were learnt using subsymbolic learning either individually or as part of more complex functions associated with compound tasks. The efficacy of this approach is due to the composition occurring at a sufficiently abstract level where much of the uncertainty has been removed. Each function acts much like a funnel operator (Christiansen, 1992), so although individual actions may be highly uncertain the overall result is largely predictable.

A new control function is composed out of solutions to subtasks using the generated plans. This replaces the old control function. New tasks may not consist of precisely these subtasks and there may be some degree of context sensitivity, the solution to one subtask is partially dependent on the solution to another. Composed solutions are therefore likely to be only approximately correct, but an exact solution is not necessary. It is sufficient that the solution is close enough to the final solution often enough to produce an average speed up. Subsymbolic learning will further refine the function and quickly remove any error. The refinement of the solution also means that it is no longer an exact composition of the set of subtasks. The subtasks become more interdependent producing a more synergistic solution. This solution can then be used if exactly the same task or very similar one is required to be solved. The process need not stop there. As subsymbolic learning refines the function, new features may emerge and the symbolic representation change. Such changes may trigger further symbolic processes which will modify the function, with the cycle repeating as many times as necessary.

1.3 Identifying and Composing Subtasks

This section begins with a very high level introduction to reinforcement learning and the function it produces. It shows how features are used to partition this function and how the pieces produced are composed to form a solution to a new task.

One of the experimental testbeds used in this thesis is a simulated robot environment of different configurations of interconnected rooms. The robot must learn to navigate efficiently through these rooms to reach a specified goal from any start location. Figure 1.2 shows one example with 5 rooms and the goal in the top right corner. The robot's actions are small steps in any of eight directions. Here the location, or state, is simply the robot's x and y coordinates. The thin lines of figure 1.2 are the walls of the rooms, the thick lines the boundary of the state space.

If each action is independent of preceding actions, the task becomes one of learning the best action in any state. The best overall action would be one that takes the robot immediately to the goal. But this is only possible in states close to the goal. Suppose the robot is in a particular state and that the number of steps to goal from each of its neighbouring states is known, indicated by the numbers in figure 1.2. Then a one step look ahead procedure would consider each step and select the one that reaches the neighbouring state with the shortest distance to goal. In figure 1.2 the robot would move to the state 10 steps from goal. If this process is repeated the robot will take the shortest path to goal. In practice we must, of course, learn such values. This can be done using some type of reinforcement learning (Watkins & Dayan, 1992; Sutton, 1990) which progressively improves estimates of the distance to goal from each state until they converge to the correct values.

The function shown in figure 1.3 is the result of reinforcement learning on the problem of figure 1.2. But instead of it representing the actual distance to goal, it represents essentially an exponential decay with distance to goal. The reasons for this

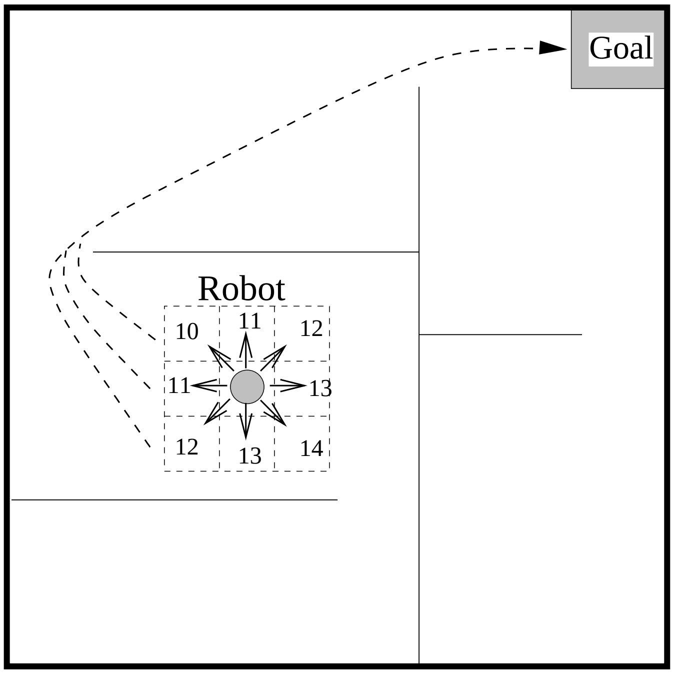

Figure 1.2: Robot Navigating Through a Series of Rooms

will be made clear in section 4.1. The shaded areas represent large gradients in the learnt function. Comparing this to the environment shown in figure 1.2 it is apparent that these correspond to the walls of the various rooms. These are the *strong features* discussed in the previous section. They exist because of the extra distance for the robot to travel around the wall to reach the inside of the next room on the path to the goal. These features are visually readily apparent to a human, so it seems natural to use vision processing techniques to locate them.

A popular technique in object recognition, the snake (Suetens, Fua, & Hanson, 1992) introduced by Kass, Witkin and Terzopolous (1987), is used to partition the function. In object recognition, the snake produces a closed curve that lies along the boundary of an object as defined by edges in an image. In this application the snake groups together sets of features to define a region of the function. Each region represents the solution to a particular subtask, allowing the function to be decomposed as shown in figure 1.4. Most of these pieces determine how the robot should move to get from somewhere in a particular room to reach a door. The piece

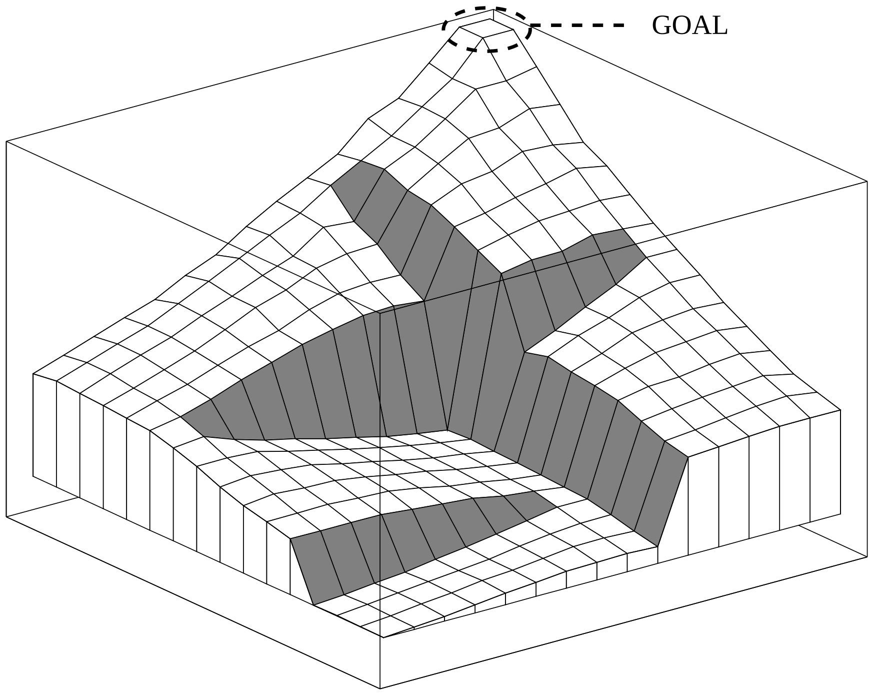

Figure 1.3: The Reinforcement Learning Function

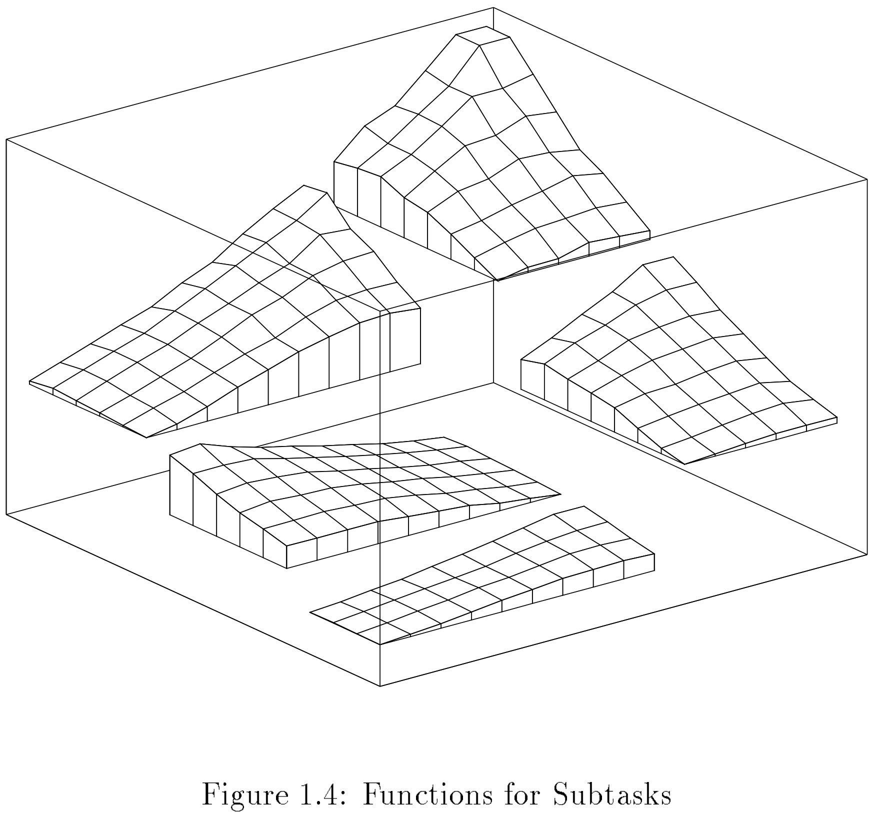

Figure 1.4: Functions for Subtasks

in the top right hand corner determines how the robot should reach the goal, once in the room containing the goal.

Now suppose the goal is moved from the top right corner to the top left corner of the state space as shown at the base of figure 1.5. Reinforcement learning in its most basic form would be required to learn the new function from scratch. In the work presented in this thesis if the goal is moved, once the new goal position is known, functions associated with the original task or some other already learnt task can be composed to form an approximate solution to the new task.

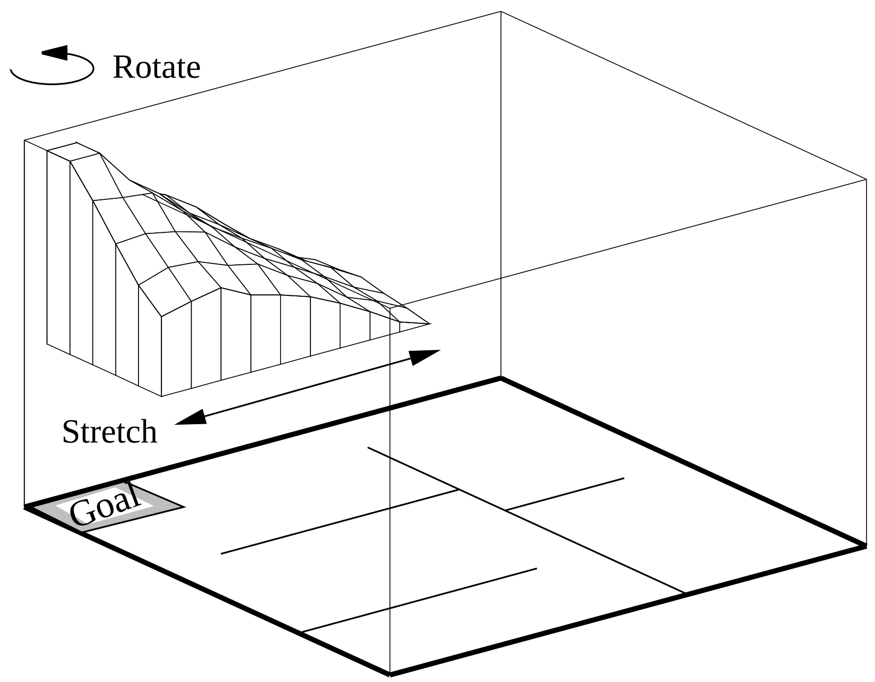

Figure 1.5: Adding in the First Function

The navigation problem has not significantly changed for three of the rooms. The two that have changed are the one originally containing the goal and the one now containing the goal. It is certainly possible to exchange these two, using an appropriate transform. But other previously learnt functions may better match the new task. The best match for the room containing the new goal is, in this example, the function for the goal in the original problem. To fit this to the new task the function is rotated and stretched slightly by changing the coordinates of its nodes, see figure 1.5.

For the room containing the original goal a function obtained when solving another task and held in the subtask library is a better match. This is transformed appropriately and added to the function, see figure 1.6. The other three rooms use the functions from the original problem, since changing the goal position has little effect on the actions taken. In fact only the height of the functions must be changed. This is simply a multiplication by a value representing the distance to goal from the doorway closest to the goal. By adding these three remaining functions a composite function representing a solution to the new task is produced, shown in figure 1.7. Because the transform may produce some error the resulting function may not be exact. But as the experiments will demonstrate, the function is often very close and further reinforcement learning will quickly correct any error.

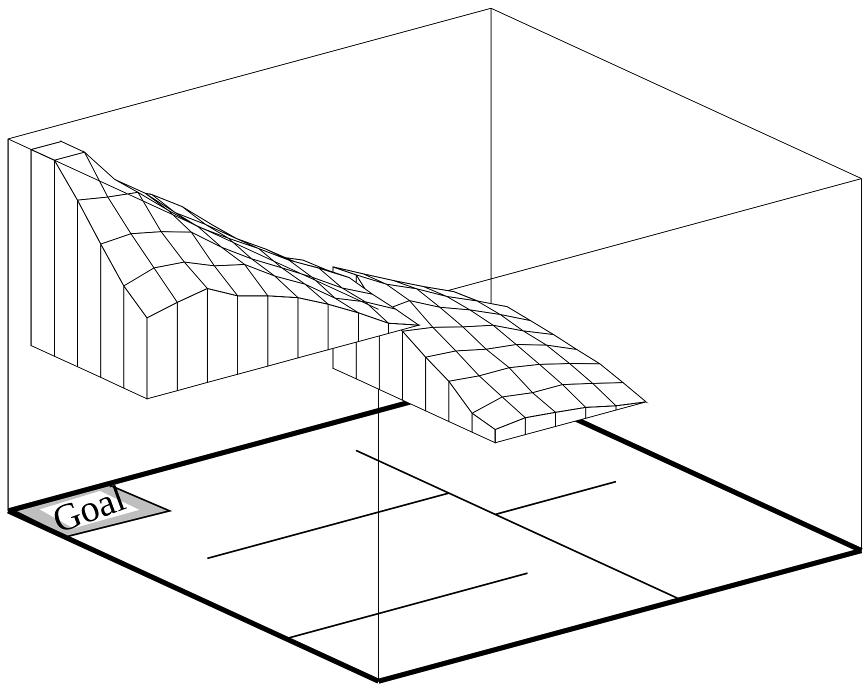

Figure 1.6: Adding in the Second Function

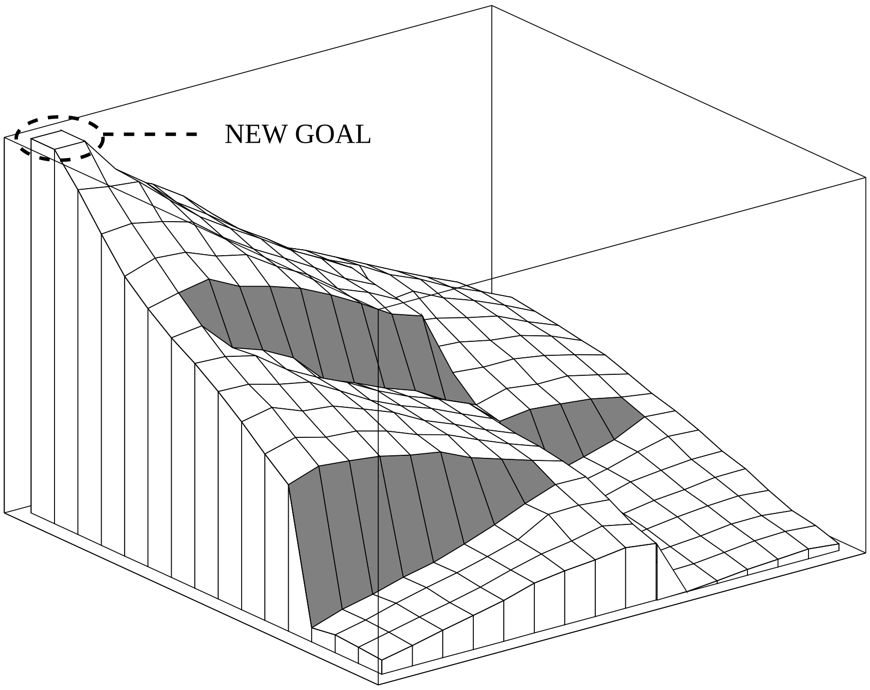

Figure 1.7: The Function Produced by Composition

1.4 Research Contributions

The work discussed in this thesis has impact in three Artificial Intelligence research areas: symbolic processing, hybrid architectures and reinforcement learning. It is a contribution to symbolic processing research by demonstrating an important role for symbols in the learning of control tasks, often thought to be the realm of subsymbolic processing. It is a contribution to hybrid architecture research as a novel approach to combining symbolic and subsymbolic processes. It is a contribution to reinforcement learning research by showing how to partition the reinforcement learning function into subtasks and use this to speed up learning on a new task. This research has developed a system implementing the architecture presented in section 1.2 to explore the intuitions discussed in section 1.1. The system has been validated empirically to demonstrate its effectiveness. Two papers have been published (Drummond, 1997,

1998) discussing this work.

The specific contributions of this thesis are:

- A spline function approximation scheme for reinforcement learning without divergence.
- A way of partitioning the function into subtasks.
  - The identification of features in the function delimiting subtasks.
  - The extension of the snake representation to robustly locate features.
- A method of composition to build a new solution from subtasks.
  - The definition of transformations to fit old subtask solutions to a new task.
- A demonstration of significant speed up over basic reinforcement learning.
  - In the learning of new tasks.
  - When dealing with a changing world.

1.5 Thesis Outline

The rest of this thesis is organised as follows. Chapter 2 presents related work. It particularly focuses on this work's positioning with respect to other research and what aspects of it are novel. Chapter 3, 4, 5 and 6 give a description of this approach at a more detailed level stressing how the parts discussed in section 1.2 are realised in practice. Chapter 7 describes experimental results that empirically validate the approach. Chapter 8 investigates the philosophical implications of this work. Chapter 9 discusses limitations and future work and presents some conclusions.

Chapter 2

Related Research

This chapter addresses the justification, positioning and novelty of this research. The justification motivates this research by fleshing out the ideas behind the intuitions presented in the introduction. The positioning takes a broad view of the Artificial Intelligence field comparing and contrasting this research to other work involving symbolic, subsymbolic and hybrid models. The novelty takes a narrower view discussing more closely related work particularly emphasising where it differs from this research and why those differences are important.

2.1 Justification

This section takes the intuitions of section 1.1 and offers some additional support for their being important in a system for learning control tasks.

The first intuition is that symbol manipulation confers some distinct advantages to a system. Fodor and Pylyshyn (1988) have argued that the structure sensitivity of symbolism makes it a good model of human thought. This property is particularly useful in planning, where solutions to complex problems are composed of solutions to simpler ones. Planning has long been a mainstay of symbolic Artificial Intelligence. It has proven successful not only in addressing simple synthetic problems but also more practical industrial problems. It has been used, for instance, in determining the steps to produce machined parts (Britanik & Marefat, 1995) and for finding ways to improve their manufacturability (Nau, Gupta, & Regli, 1995). One central feature of planning in that it provides some sort of look-ahead, an important human cognitive faculty (Dennett, 1995; Holland, 1992). There is also some evidence that other animals do this. Vervet monkeys have been observed to choose the shortest route in retrieving food caches (Cramer & Gallistel, 1997). Rats learn more than just the task in hand (Tolman, 1973). They also learn some sort of map which accelerates learning on different tasks. With look-ahead, a system can decide which paths look the most promising by "mentally" simulating the expected trajectory. As a famous quote of Karl Popper suggests this look-ahead "permits our hypotheses to die in our stead" (Dennett, 1994).

The second intuition is that the symbolic view does not account for everything. One criticism of symbolic approaches is that they are disconnected from the world. Traditional planning systems require accurate hand-crafted models of the world from which to build plans. An alternative approach better connected to the world and not requiring a model is reinforcement learning (Sutton, 1988; Watkins & Dayan, 1992). Sutton and Barto (1990) discuss the close connection of this type of learning with Pavlovian reinforcement. Reinforcement learning has been successfully applied to many problems such as game playing (Samuels, 1959; Tesauro, 1988) and industrial strength problems such as job-shop scheduling (Zhang & Dietterich, 1995) and elevator control (Crites & Barto, 1995).

The third intuition is that knowledge should be transferred from the solutions to old tasks to new related tasks. Although low-level learning approaches such as reinforcement learning can learn without a model of the world, learning is slow. Transfer of learning from previous tasks should go a long way to mitigate this effect. It has been for some time, and still is, extremely controversial as to the task specificity of transfer in humans. People are poor at the transfer of very high level abstract structural properties and are often misled by the surface similarities of two problems (Gentner, 1990). Nevertheless ample evidence exists for transfer (Cormier & Hagman, 1987; Singley & Anderson, 1989). Some degree of task specificity may increase the confidence in the usefulness of prior learning and thus increase the probability that the effect is beneficial rather than deleterious. Many in the machine learning community now see this as an important research area. Some have looked at transfer in neural networks (Pratt, 1994). More recent research (Baxter, 1997; Caruana, 1997; Ring, 1997; Schmidhuber, Zhao, & Wiering, 1997) has investigated transfer in a broader range of machine learning algorithms. The work in this thesis effects transfer through a case-base of solutions (Aamodt & Plaza, 1994) to previously learnt subtasks. The index is symbolic. This allows a process much like symbolic planning to construct the solution to a new task from previously learnt subtasks.

The fourth intuition is that there are often features in an agent's interaction with its environment which are stable and easy to recognise accurately. They can be used to determine if the same or similar problem has be encountered previously. They can be used to divide a complex task into subtasks. The features in the reinforcement learning function are akin to edges in an image. They are located by finding the zero crossing point of the Laplacian (Marr, 1982). Herve, Sharma and Cucka (1991) used the singularities in a robot arm's work space as identifying features. In this thesis, as in Herve et al's research, the features need not only be easily recognisable. They must also be sufficient to define the function at some qualitatively accurate level. Mallat and Zhong (1992) have shown that a function can be accurately reconstructed from a record of its steep slopes. In this thesis, cases are regions of the reinforcement learning function delineated by features. Grouping of the features forms the symbolic index.

The fifth intuition is that the meaning of a complex symbol can evolve over time. One commonly used psychological model of motor control learning (Anderson, 1982), is the idea of compilation from the cognitive to the motor level. A high level declarative form of information is compiled into a low-level procedural one. Such rules become more operationally effective by being faster yet more task specific (Russell, 1989). This is not the end of the process: at the resultant procedural level, termed the autonomous level by Fitts and Posner (1969), further refinement takes place. In the approach discussed in this thesis a plan is constructed at the symbolic level. Then steps in the plan are replaced by functions in a process akin to compilation. This is used to initialise the low-level learning algorithm, but further learning takes place refining the solution. Shavlik (1994) has used a similar idea to initialise neural networks with a domain theory. In Shavlik's words "The domain theory produces a useful inductive bias .....". The importance of bias in learning has long been an important topic in the machine learning community. Successful learning is often seen as exploiting the right bias (Utgoff & Mitchell, 1982; Mitchell, 1990). In this thesis, initialisation of the low-level learning algorithm with a good approximation to the solution of the new task produces the right bias.

2.2 Positioning

The primary aim of this research is justifying roles for both symbolic and sub-symbolic processing and establishing a clearer division and connection between them. It can be contrasted with approaches using only one form of processing and compared to those having a similar division. The following sections compare this thesis's approach to those addressing the problem at a purely symbolic level, a purely subsymbolic level or some hybrid of the two.

2.2.1 Symbolic Systems

The research presented in this thesis is primarily interested in autonomous systems. An important activity of such a system is to determine a series of actions that achieve some goal. One approach to solving this problem is planning. This section argues that planning has a long research history and is a valuable problem solving technique. Nevertheless traditional planning has made certain simplifying assumptions. This thesis shows that combining planning with subsymbolic learning is a useful way of removing some of these assumptions.

Planning has long been a significant part of symbolic Artificial Intelligence. It has a research history of some 40 years (Hendler, Tate, & Drummond, 1990) growing out of GPS (Newell & Simon, 1963) and question answering systems (Green, Wolf, Chomsky, & Laughery, 1963). Typically a planning system, such as STRIPS (Fikes & Nilsson, 1971), finds a series of operators that transform an initial state to a goal state, via a series of intermediate states. States are essentially nodes in a graph and the operators edges. Thus planning can be recouched as search. In fact, a central idea in symbolic Artificial Intelligence is that problem solving generally can be recouched as search. This is motivated by a long held view that "Heuristic search ...... is in fact the principal engine for human problem solving ......"(Simon, 1957).

Planning systems, until recently, have been based on a number of simplifying assumptions (Weld, 1994). It is assumed that each action in the plan will always work and work correctly in the sequence generated. It is assumed that the planner has complete knowledge of the world. It is assumed that there is sufficient time to construct a complete plan. This does not allow for the actions to be stochastic. This does not allow for a world changing independently of the system. This does not allow for actions to be taken before planning is complete and planning can be slow, in the worst case it is P-space complete (Bylander, 1991).

Research has addressed the speed limitations of planning either by restricting search to a short distance ahead of the present state or by speeding up the planning process. In reactive planning, a plan net describes the state-action space of the system (Drummond, 1989). Default actions can be taken while in parallel the net is searched for critical choice points determining whether or not the goal can be achieved. New control rules are then produced which tell the system what to do at these points. A similar approach called reaction-first search (Drummond, Swanson, Bresina, & Levinson, 1993) searches a short distance ahead and tries to avoid dead-ends. Korf (1990) proposes a limited horizon look-ahead variant of the well known heuristic search algorithm A* (Hart, Nilsson, B., & Raphael, 1968), to reduce how far ahead planning searches. The distance of look-ahead can be increased by planning at an abstract level (Sacerdoti, 1974). In addition, only the immediately applicable low level plan steps are required, the rest of the plan need only be completed at an abstract level. Generally abstraction speeds up planning. Another way to speed up planning is to modify an old plan from a case base (Hammond, 1990; Veloso, Munoz-Avila, & Bergmann., 1996) rather than planning from scratch. The work presented in this thesis has much in common with abstract and case-based planning. It plans at an abstract level where the search space is small and a solution found quickly. It transforms solutions from a case base to instantiate the abstract plan steps.

One way to deal with an uncertain world is to model the uncertainty explicitly. But this requires a more detailed model and makes planning even slower. Collins and

Pryor (1995) use predicates to represent uncertainty, the planner producing results for all possible outcomes. Unfortunately as uncertainty increases the complexity of planning for all possible outcomes rises dramatically. Koenig and Simmons (1995) deal with an uncertain world by treating it as a two person game. By searching only a single step ahead of the current state, speed problems are avoided. If the uncertainty is due to the outcome of the actions being stochastic and the transition probabilities are known, stochastic planning can be used. But again this is slow. The speed can be improved by restricting the search to an envelope of states, those likely to be encountered when carrying out a task (Dean, Kaelbling, Kirman, & Nicholson, 1995). If the system has enough time, essentially an anytime algorithm (Boddy & Dean, 1989), it expands the envelope. An alternative solution is to aggregate states and treat each collection of states as an abstract state (Dean & Lin, 1995; Boutilier & Dearden, 1994). This produces good but not necessarily optimal plans. Identifying a structural breakdown of a problem can accelerate convergence to an optimal solution (Boutilier, Dearden, & Goldszmidt, 1995). The work presented in this thesis is also concerned with identifying structure. But this is used to speed up learning rather than stochastic planning. Structure is identified through a system's interaction with the world rather than the exploration of a model. Planning occurs at a level where much of the uncertainty has been removed. By coupling planning to a low-level learning algorithm which naturally deals with stochasticity any remaining uncertainty is easily dealt with.

Planning is difficult if the world is not completely and accurately modelled. Dejong and Bennett (1995) advocate adjusting the bias of the planner to choose plans better suited to the actual world. Etzioni et al (1993) assume information about the world is correct but incomplete. They propose a planning language, extended to include actions that only sense the world and information gathering goals. Knoblock (1995) also includes sensing operators and if an operator fails, domain specific heuristics are used to repair the plan. Stenz (1995) addresses the problem of replanning, using a variant of A*, when new sensor information updates the costs of edges in the graph. Gervasio and Dejong (1994) propose completeable plans which defer decisions until information becomes available, completing plans at run time. The work presented in this thesis does not assume a model of the world. Structure identified while learning the solution to a task is used to form a model at a high level of abstraction. This allows planning to compose solutions to a new task out of solutions to previously learnt subtasks.

In summary, traditional planners adopted the strongly simplifying assumption of having a complete, accurate and deterministic internal model of the world. But more recently many researchers have worked on extending planning to deal with the vagaries of interacting directly with the world without throwing away the idea of look ahead. Exploiting many of the ideas generated in the planning community seems essential in dealing with complex problems. The research, discussed in this thesis, adopts the position that planning is a useful approach but only at sufficiently abstract levels where the uncertainty has been minimised. This is achieved by using operators, similar in principal to funnel operators (Christiansen, 1992), that reduce uncertainty. These operators are learnt by a subsymbolic learning algorithm which readily accommodates the uncertainty of low-level actions.

2.2.2 Subsymbolic Systems

There are many alternatives to symbolic planning. Some seem particularly suitable for control tasks. The one of primary interest for this thesis is reinforcement learning. The main limitation of many of these approaches is the time taken to learn. This thesis shows that combining symbolic planning with subsymbolic learning significantly improves the learning rate.

There are a number of approaches that aim to do away with symbolic representation altogether. Brooks proposes, as an alternative, the subsumption architecture. This consists of weakly coupled horizontal layers: taking in sensor data and controlling output behaviour (Brooks, 1986). Should a higher level fail, perhaps due to insufficient input data, lower levels take over control, thus improving robustness. Using this approach, a six-legged robot learnt the tripod gait (Maes & Brooks, 1990). Each leg had a number of associated behaviours which learnt independently what conditions were critical for their activation. To include goal directed behaviour, both Maes (1990) and Mataric (1990) have investigated the idea of networks of subgoals whose selection is achieved by spreading activation. The work presented here does use a layered architecture, but one important difference is the strength of coupling between layers. A deliberate aim in Brooks's (1997) research has been to minimise the internal interaction between layers, which layer is active being largely determined externally by the state of the world. This thesis proposes a much stronger coupling of layers to promote the rapid learning of new tasks.

Many connectionist researchers also do not see any need for symbolic processes (Brooks, 1986; Rumelhart & McClelland, 1986; Smolensky, 1987). Certainly connectionist systems have been successful in learning many things. One well known example is Nettalk (Sejnowski & Rosenberg, 1987), a connectionist network which successfully learnt the pronunciation of English words from a small window of text. Such networks have also been used in classifying encephalograms (Tsoi, So, & Sergejew, 1994) and learning robot arm movements (van der Smagt & Krose, 1991). Graf (1989) proposes a neuroplanner for robot arm control. A neural map is learnt that correlates sensor information about the world with the arm's joint angles. Neurons correlated with the goal state are activated and those correlated with states that the sensors determine are blocked are inhibited. The system then finds the shortest path to the goal by spreading activation. One criticism of connectionist approaches is that they are not structure sensitive, although this has become the subject of much debate which will be discussed in detail in chapter 8. The work presented in this thesis aims to exploit the structure sensitivity of the symbolic approach to speed up learning at the subsymbolic level.

Without examples provided by a teacher, learning requires some sort of feedback from the environment to determine success or failure. Often feedback is infrequent and delayed. Learning in this situation Sutton (1988) termed temporal difference learning. An early example is Holland's classifier system (Booker, Goldberg, & Holland, 1989). Here a classifier, a simple low-level rule, that produces desirable behaviour is rewarded. As the system iterates, classifiers that initiate the desirable classifier are given part of its reward. This process, the bucket brigade algorithm (de Jong, 1988), continues passing back the reward to classifiers earlier and earlier in the process. In Sutton's (1988) own work a neural network is trained not by waiting until the final outcome of the experiment but rather by correlating it to subsequent predictions by the network. Thus if the network were designed to predict the outcome of a game of chess, if the subsequent position predicts a win the present position is biased towards predicting a win.

Temporal difference learning forms part of a broader set of learning approaches called reinforcement learning (Sutton & Barto, 1998), which has become a significant research field in its own right. A key ingredient of reinforcement learning is a value function, which indicates the long term utility of a particular state. A very closely connected idea is dynamic programming (Bellman, 1957). If a model of the environment is known, dynamic programming can be applied directly to the model to produce the value function. In one popular reinforcement learning algorithm Q-learning (Watkins & Dayan, 1992), the model forms an implicit part of the state/action mapping, while in another, Sutton's Dyna Architecture (Sutton, 1990), the model and the mapping are learnt simultaneously. The aim of this thesis is to exploit the robustness of low level learning approaches such as reinforcement learning while adding the speed and structure sensitivity of symbolic methods.

In summary, many of the systems discussed in this section have deliberately avoided use of symbolic processing. They have achieved impressive performance on practical problems. An important lesson learnt from such systems, is that classical planning is a questionable way to control low-level behaviour. Notably, however, many approaches incorporating goal directed behaviour required searching a graph representing the world. Spreading activation was used to search the graph as opposed to the classical planning approach of successive operator application. In the view of this thesis, however, this is not a distinction of great consequence. In some cases these graphs represented abstract properties of the world such as topological relationships. This thesis also proposes search at an abstract level but the level is symbolic. In fact, the use of a topological representation has some of the essential properties of classical symbolic systems. For instance connectivity is a symmetric relationship that defines the relationship between two places or control states. Thus this would seem to have the systematic and compositional properties that were part of symbolic systems. It also seems to strongly relate to the idea of plan nets (Drummond, 1989) discussed in section 2.2.1. It seems apparent that the two fields, in dealing with their own shortcomings, have moved closer together. This thesis takes the position that this closing of the gap is indicative that the two approaches should be combined to best exploit their advantages, while minimising their disadvantages.

2.2.3 Hybrid Systems

The previous two sections dealt with symbolic and sub-symbolic approaches. This thesis has argued for a hybrid approach. This section reviews alternative hybrid models. The following paragraphs move across the gamut of essentially connectionist systems, with symbolic processes used to help understand them, to essentially symbolic systems with connectionist systems used to connect them to the world.

A significant drawback with neural networks is their "impenetrability", it is difficult for a human to determine exactly what has been learnt. One way to remedy this problem is cluster analysis (Sejnowski & Rosenberg, 1987) which can be used to better understand and thus improve the way a network learns. An alternative approach is to extract rules from a network (Gallant, 1993; Thrun, 1994). Rules not only help to understand the network but can also be used to explain a decision to the systems users, possibly domain experts (Setioni & Liu, 1995).

In this context a rule is an explanatory tool, but this need not be its only purpose. A critical problem identified in the 1980's is the "knowledge acquisition bottleneck" (Buchanan et al., 1983). If a neural network learns directly from an environment the rules extracted can be added to a knowledge base. Research has addressed combining rules from experts with those extracted from a neural network to produce medical expert systems (Herrmann, 1995; Ma & Harrison, 1995). These approaches are based on the belief that neural networks are better at learning than symbolic methods. Indeed, some results suggest that neural networks generalise better (Shavlik, Towell, & Noordewier, 1991). Quinlan (1994) suggests, however, that which type of system performs better is very much problem dependent. It seems critical therefore to include different learning algorithms in a single system allocating each a well defined role (Aamodt & Plaza, 1994).

Symbolic and neural processes can also work in parallel connected externally through the environment. A symbolic process can both control the system while learning and supervise the training of a neural network (Handelman, Lane, & Gelfand, 1989). An alternative approach, as advocated in this thesis, is a closer coupling of the systems. Shavlik (1994) proposes knowledge based neural networks. Information in the form of rules can be used to define the structure of the neural network and the initial allocation of weights. Some random links can be added to allow the network to learn new associations and the network trained. The network can then be analysed and new rules produced that abstract the information in the network. The view of this thesis is that this close coupling of systems is needed to maximally exploit the advantages of both.

There are many interesting ways neural networks can interrelated to symbolic processes. The feedforward networks can be interrelated to Bayesian networks (Schwalb, 1993; Tchoumatchenko & Ganscia, 1994), decision trees (Ivanova & Kubat, 1995) or fuzzy logic (Herrmann, 1995). Recurrent networks can be interrelated to discrete finite state automata (Omlin & Giles, 1994; Fransconi, Gori, Maggini, & Soda, 1996). Such representations are often already a part of symbolic systems. These examples demonstrate the relative ease of producing a two-way coupling of symbolic and subsymbolic processes.

An alternative is to to use a neural network to generate symbols. Dyer (1994) addressed the connection between scenes and the language describing them. Harnad (1994) has investigated categorical perception. A neural network not only connects the symbolic representation to the outside world but also accentuates the intercategory differences and intracategory similarities. The idea of using subsymbolic methods for connection to the world is one this thesis aims to exploit. But in this thesis, the connection is not only to the symbolic level it is also from perception to action.

In summary, this section has dealt with the continuum of essentially subsymbolic to essentially symbolic. Typically the subsymbolic is a neural network and the symbolic level is a rule based system. An interesting research area is how to combine these processes in a neural architecture. Some work has looked at adding inference mechanisms (Hall & Romaniuk, 1990; Sun, 1992; Lacher, 1992), other work at combining qualitative symbolic associations with more quantitative subsymbolic associations

(Sun, 1992; Almor, 1992). Still, the division between symbolic and subsymbolic processing is not always clear. Recurrent neural networks have been shown to be Turing equivalent. So some researchers feel the issue is largely one of searching the "design space" thus finding the most effective architecture (Honavar & Uhr, 1995). But the view of this thesis is that the division should not be taken as synonymous with the division between neural networks and production systems. This thesis takes the properties discussed by Fodor and Pylyshyn (1988) to define this division. Subsymbolic systems are those that learn associations, they are typically function approximators for a continuous state space. Symbolic systems are those that manipulate atomic structures through syntactic composition and are thus discrete.

2.3 Novelty

This thesis advocates a hybrid model of symbolic and subsymbolic layers, combining the strengths of symbolic planning with those of reinforcement learning. Symbolic planning has one principal advantage: being able to compose solutions to many different tasks from a single set of primitives. Its main weaknesses are the requirement for an accurate model of the world and the assumption of determinism. Approaches such as reinforcement learning avoid these problems but at the cost of taking a long time to learn. The research, discussed in this thesis, investigates the transfer of prior knowledge to accelerate such learning.

Prior knowledge is in the form of functions, extracted by decomposing the task into subtasks. If such a decomposition is known in advance, particular rewards can be associated with solving particular subtasks resulting in a speed up in learning. Mahadevan and Connell (1992) use reinforcement learning in behaviour based robot control, where the individual behaviours are essentially solutions to subtasks and are rewarded independently. Mataric's (1994) research also addresses behaviour based robot control. Again intermediate rewards are used but it is the triggering conditions for the behaviours that are learnt, rather than the behaviour itself. The approach in this thesis does not require a previously defined decomposition. It identifies the decomposition from features in the reinforcement learning function. Symbolic planning uses the results of previous decompositions to compose solutions to new tasks, thus avoiding the necessity of intermediate rewards.

This research aims to define a role for both symbolic and subsymbolic processing in the learning of control tasks. But rather than viewing their roles as distinct, one novelty of this research is that it combines them together in a closely coupled abstraction hierarchy. In some ways this work might be seen as the merging of an abstraction hierarchy with an analogical, or case-based reasoning system. Branting and Aha (1995) have also investigated this combination, with solutions to problems at different abstract levels being saved in a case base. In their work the abstraction relationship is between cases of essentially the same type. This thesis exploits a characteristic of case bases to define an abstraction relationship between functions and symbols. Case-based reasoning systems can be viewed as having two distinct spaces: the index space which can be readily mapped to the problem and the solution space. This is effectively a two level abstraction where one level is not simply an approximation of another level but is of a different kind. The view of this thesis in that the symbolic level is an abstraction of the subsymbolic level but of a different kind.

The following sections discuss research closely related to that of this thesis. They address in turn the two main aspects of this research: transfer and abstraction.

2.3.1 Transfer

One aim of this research is to maximise transfer to speed up low level learning. To this end it is important to identify and exploit any structure in the world. In multi-task learning (Caruana, 1996), the learning of a particular task is accelerated by learning strongly related tasks concurrently. The common structure of the solution to multiple tasks can be used to bias the learner. This can radically reduce the number of examples needed to learn a concept (Baxter, 1995).

The essence of multi-task learning is to develop a set of features that represent the common properties of the different tasks. In many ways such a representation is very similar to a parameterisation. The normal input dimensions are projected down to a smaller set of parameters. The parameter space is a much smaller space than the original one and can be searched much faster. The danger of such a bias is that it may be inappropriate so that the best solution found in this reduced space is far from optimal. In multi-task learning the set of features must express all the essential properties of different tasks. If they do not and there are insufficient extra degrees of freedom, the performance on individual tasks may suffer (Caruana, 1996). A similar effect has been observed when using reinforcement learning and the concurrent tasks are different goal positions in the same environment. Kaelbling (1993) uses a graph of landmarks to break up the space into subtasks. The goal of each subtask is getting to a nearby landmark and a complete task entails following a series of landmarks to the goal. This initially speeds up learning but degrades performance later on, as the subtask decomposition is not optimal for all possible paths.

The errors that can arise from a mismatch between the parameters and the problem are well-known in statistical regression. Traditionally domain knowledge was used to construct a parametric representation. But often extra parameters were needed to fit the data well, making the identification of the important underlying structure difficult. To prevent this problem non-parameterised approximations were used (Lancaster & Salkauskas, 1986; Eubank, 1988). Non-parametrised representations need a lot of data or they may fail to capture the important structure. So some researchers have proposed new ways that domain knowledge can be used to restrict the functional form (Ramsay & Heckman, 1996). This dichotomy is often termed the bias/variance problem (Geman, Bienenstock, & Doursat, 1992). It is noteworthy that in reinforcement learning system the major successes are exactly those where some parameterisation of the space has occurred (Samuels, 1959; Tesauro, 1988).

The approach used in this thesis is similar to Kaelbling's (1993) in that the solution to a task is represented as the combination of solutions to subtasks. One significant difference, however, is that the subtasks are identified during learning based on the natural structure of the task. This allows previous experience with similar problems to be applied to the new task. Furthermore it allows the solution to such problems to be modified to better fit the new task. This, of course, is the principle behind analogical reasoning (Veloso & Carbonell, 1993), or the closely related idea of case-based reasoning (Aamodt & Plaza, 1994). In this thesis this experience is contained in functions stored in a case base, representing solutions to subtasks. The aim of this research is to use these solutions as qualitative information to bias search for the correct function to the most likely part of the search space. The bias is produced by composing the solutions to produce a close approximation to the solution of a new task. This is used to initialise the low-level learning algorithm, but further reinforcement learning can refine the function to whatever degree necessary. This should accelerate search but not prevent the learner from being able to reach other parts of the search space where the actual solution may lie.

There is other research that uses instance based learning (Aha, Kibler, & Albert, 1991), a technique closely associated with case-base learning, in conjunction with reinforcement learning. In this other research the transfer is not between tasks but between the stages of learning a single task. So rather than keeping the solution to a subtask, the case base contains individual states and their utilities. Peng's (1995) motivation is to economically represent the state space, only those states actually visited need be represented. Tadepalli and Ok (1996) go further maintaining only a subset of states. If the utility value is within a threshold of that predicted by neighbouring states using linear regression the sample is discarded. Another reason to keep certain samples is in dealing with hidden state. McCallum (1995b) expands the representation of particular states to include prior states removing ambiguity due to hidden states. In further work, he uses a single representation to address both the hidden state problem and the economical representation of the state space by using a case base of state sequences associated with various trajectories (McCallum, 1995a). A third reason to create a case base of samples is to improve the learning rate. Moore and Atkeson (1993) save predecessor states, those most significantly affected by new updates in the value function.

In the thesis the case is not an example of the utility of an individual state. Instead, it is the result of a complete learning episode, so the method should be complementary to these other approaches. More importantly its motivation is quite different. The use of functions from the case base aims to speed up reinforcement learning on a new task, by the transfer from other related tasks.

2.3.2 Abstraction

This thesis has proposed an approach where the symbolic levels form an abstract representation of the subsymbolic levels. Abstract problem solving embodies the intuitive notion that search time can be reduced dramatically by first considering the problem at an abstract level (Sacerdoti, 1974). This allows the determination of areas in the search space that are most likely to contain solutions. Abstraction hierarchies have typically been constructed by the removal of detail (Holte & Hernadvolgyi, 1999), by dropping predicates (Prieditis, 1993) or by weakening constraints (Ellman, 1993). Each more abstract level is of essentially the same form as the level below but faster to search. Under certain theoretical conditions using abstraction hierarchies can reduce the search space from exponential to linear (Knoblock, 1991). Abstraction is not always be beneficial (Backstrom & Jonsson, 1995), but there is much empirical evidence of its effectiveness (Knoblock, 1991; Holte, Drummond, Perez, Zimmer, & MacDonald, 1994).

Abstraction has also become an important research topic in reinforcement learning. Sutton (1995) has shown how models of the world at different levels of temporal abstraction can be produced by reinforcement learning. These models represent abstract prediction tasks, such as the probability of hitting a wall. Precup, Sutton and Singh (1997, 1998) extend this work to include higher level actions (or macro actions) that apply to more extended periods of time. They propose a possible semantics for macro actions within the framework of normal reinforcement learning. Macro actions can be intermixed with normal primitive actions, much as with the macro-operators in traditional planning. There other more strict hierarchical schemes where the solutions to subtasks are combined at a more abstract level. For instance, Singh (1992) uses policies learnt to solve low level problems as primitives for reinforcement learning at a higher level. In feudal reinforcement learning (Dayan & Hinton, 1993), states are progressively aggregated to form a hierarchy. Upper levels in the hierarchy issue commands to lower levels to solve a particular task. When the lower level successfully solves the task it is rewarded and its solution forms part of the abstract solution. Unfortunately not all levels maintain the Markov property making some tasks insoluble. Dietterich (1998) extends the approach removing such limitations and developing a general formalism for hierarchical reinforcement learning. All these systems require programmer input to aid the decomposition of the task into subtasks. The work presented in this thesis gives one way that this can be determined directly from the system's interaction with its environment.

Thrun and Schwartz (1994) propose a method that does automatically identify subtasks and their solutions, by finding commonalities in multiple tasks. But unlike the research presented here, no mapping of such solutions to new tasks is proposed. In addition this thesis shows that it often unnecessary to consider multiple tasks, structure can be identified in a single task. Hauskrecht et al. (1998) discuss various methods of generating macro actions, by identifying exit states for predefined regions of the state space. Parr (1998) develops algorithms to control the caching of policies that can be used in multiple tasks. But in both cases a partitioning over the state space must be given. It is the automatic generation of just such a partition that is the focus of much of the work presented in this thesis. It may well be that this approach to generating partitions will prove useful to this other research.

One of the main differences between the research outlined above and that discussed in this thesis is that in the latter the abstract levels are different kinds of representation. The symbolic/subsymbolic distinction made in this thesis is also essentially a qualitative/quantitative one. Taking quantitative information and converting it to a qualitative form, although it throws away information, makes high-level relationships clearer. This has often entailed converting metrical information into a topological form: to aid the visualisation of data (Blackmore & Miikkulainen, 1995), to determine the relationships of objects in geographical data bases (Grigni, Papadias, & Papadimitriou, 1995). A qualitative level not only aids human understanding it also aids machine reasoning (Forbus, Nielsen, & Faltings, 1987; Kuipers, 1993).

In fact this thesis is close in spirit to Kuipers's work in robotics (Kuipers & Levitt, 1988; Kuipers, 1996). Kuipers introduces the notion of a "spatial semantic hierarchy", with different representations and different problem solving methods at different abstract levels. Sensorimotor and control levels combine to guide the robot between distinctive states. Higher topological and metrical levels plan the movements between distinctive states. This thesis also investigates the idea of a qualitative hierarchy with a combination of discrete high level symbolic representations with lower-level continuous ones, although not with such fine granularity of levels as in Kuipers's work. There are many differences however. At a concrete level these concern how the subtasks are learnt, how features are identified and mapped to the more abstract level and how this is used to solve tasks. But perhaps more importantly, at a conceptual level the focus of this thesis is how to exploit the compositional properties of symbolic representation to facilitate the transfer of the results of learning from one task to another. It addresses how the subtasks are indentified, transformed and composed to form solutions to new task and then further refined to produce more synergistic solutions.

Chapter 3

The Control Function

This chapter addresses the control function that determines the action to take in each world state when solving a particular task. The control function forms the hub of the architecture discussed in section 1.2. Both the symbolic and subsymbolic levels affect the form of this function. Subsymbolic learning interacts with it directly, the symbolic layer through task decomposition and subtask composition, as shown in figure 3.1. This chapter will show how an important family of function approximators, namely B-splines, can be used to represent this function accurately. The main thesis contribution discussed in this chapter is the identification of features within this function which delimit the solutions to subtasks that combine to solve the overall task.

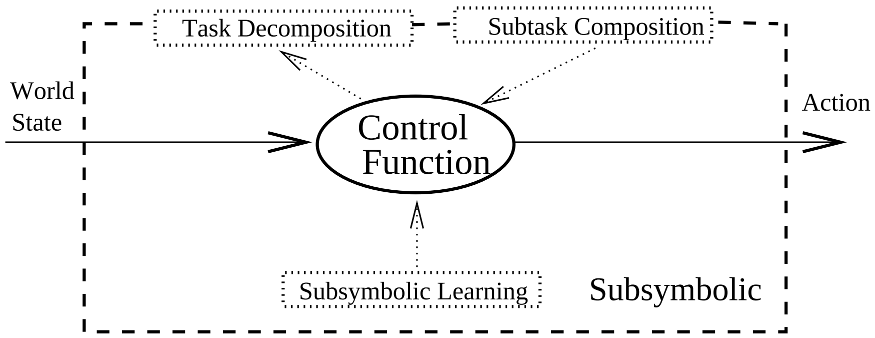

Figure 3.1: The Control Function

A control function is a mapping from world states to actions. Section 1.3 showed how this might be realised to find the shortest path to a goal in a robot navigation problem. The action taken in each state was decided by a one step look ahead procedure applied to a function representing distance to goal. This will work when the state reached after taking an action is known. This is not always the case. For instance, if actions have a stochastic outcome, the state reached may vary each time the action is taken. An alternative representation is to store a value for every action in every state. Then the shortest path to a goal is followed by selecting the action with the best value in each state reached. This representation not only includes information on the best action in a state but also on all other actions. Small changes in the world can cause dramatic changes in the best action but are much less likely to cause dramatic changes in these values.

This thesis investigates continuous state spaces so some sort of function approximation is needed to represent these values. If the action space is also continuous, a single function would represent the whole state/action space. In this thesis, discrete actions have been used and there is a separate function for each action. Figure 3.2 shows these functions for four of the actions in the robot navigation problem. The term control function will be used to refer to this collection of functions. The function discussed in section 1.3 representing an exponential decay with distance to goal is called the value function in reinforcement learning (Sutton & Barto, 1998). It is produced by taking the maximum value of these functions across the whole state space. In the rest of the thesis, for simplicity of presentation, diagrams will show just this value function.

Subsymbolic learning adjusts parameters that control the shape of these functions. Discretising the state space results in parameters representing the average value for a specific action taken anywhere within a rectangular area. More generally function approximators use a combination of basis functions and subsymbolic learning adjusts the height of these bases. The symbolic level interacts with the control function by partitioning it into regions delineated by features. A new control function is produced by combining functions representing solutions to subtasks. The following sections will first introduce spline function approximation and the coefficients that subsymbolic

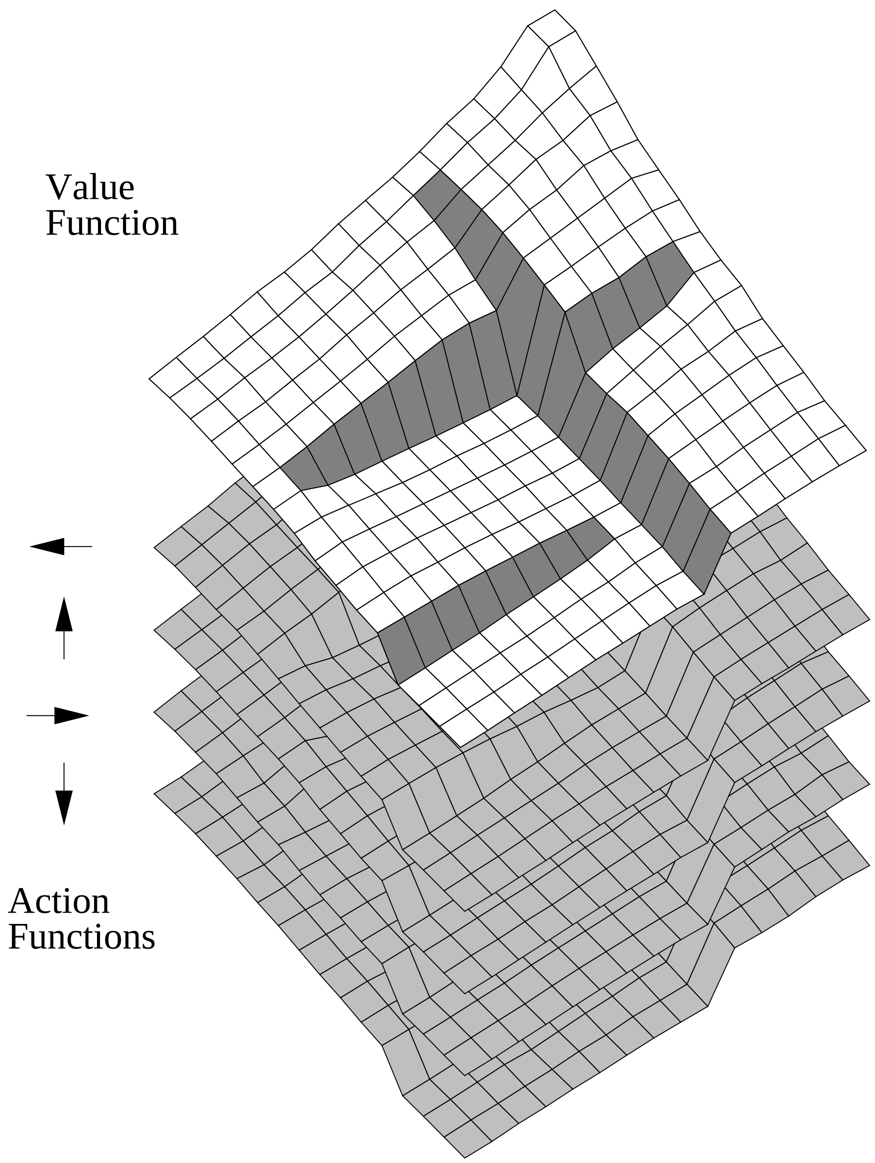

Figure 3.2: Combining Functions for Each Action

learning adjusts. It will then discuss the features that emerge in the control function which can be used to connect to the symbolic level.

3.1 Spline Function Approximation

The approach used in this thesis is to represent the control function using multidimensional splines. The following discussion will be restricted to one or two dimensions for simplicity, but the approach is readily extended to any number of dimensions (Lancaster & Salkauskas, 1986; Eubank, 1988). A spline approximates a function by a curve consisting of piecewise polynomials. Additional flexibility is achieved over a single polynomial by allowing discontinuities in certain derivatives at the point two neighbours join, called a knot. This is important as polynomial approximation can produce large changes far away from the small perturbations that cause them. Splines, however are much more local and do not produce this effect. In addition, polynomial approximation may fail to converge under fairly benign conditions (Esch, 1983) whereas spline approximation converges.

A spline is typically constructed from a series of basis functions which are translates of a single canonical function. The spline function $\hat{f}(x)$ is the linear sum of the product of each basis function $\phi_j(x)$ with its coefficient c_j, as shown in equation 3.1. The coefficients are chosen so as to minimise some error of fit to the input function.

$$\hat{f}(x) = \sum_j c_j \phi_j(x) \tag{3.1}$$

A commonly used measure of error is the L_2 norm. Equation 3.2 shows the norm for a continuous function, for discrete data the integral is replaced by a summation. Finding the coefficients that minimise the functional $D(\hat{f})$ results in the least squared error fit of the spline to the input function $f(x)$. Strictly speaking the L_2 norm should be the square root of this function but minimising either measure produces the same result.

$$D(\hat{f}) = \int \left(\hat{f}(x) - f(x)\right)^2 dx \tag{3.2}$$

This measure is quadratic and has the intuitive appeal of penalising larger errors proportionally more than smaller ones. Perhaps more importantly it is a convex function with a single minimum and thus mathematically easy to minimise. There are many ways to find the minimum value. It can be easily differentiated with respect to the coefficients resulting in a set of simultaneous linear equations, that can be solved in matrix form. Alternatively an iterative algorithm can be used such the Widrow-Hoff rule (Gallant, 1993). Such algorithms are typically different ways of hill climbing in the error space. Because the error function is quadratic, hill climbing is guaranteed to find the global minimum.

3.1.1 B-splines

One important family of function approximators is the B-splines which have many desirable properties (Unser, 1997, 1998; Aumann, 1997). One particular property of this family of splines is that they have small support. The support of a basis function relates to its width, the interval over which it is non-zero. This determines the number of coefficients that are affected by each data point. Small support results in sparse matrix representations which can be solved rapidly. Iterative algorithms are also faster needing only to update a few coefficients each time a new data point is added.

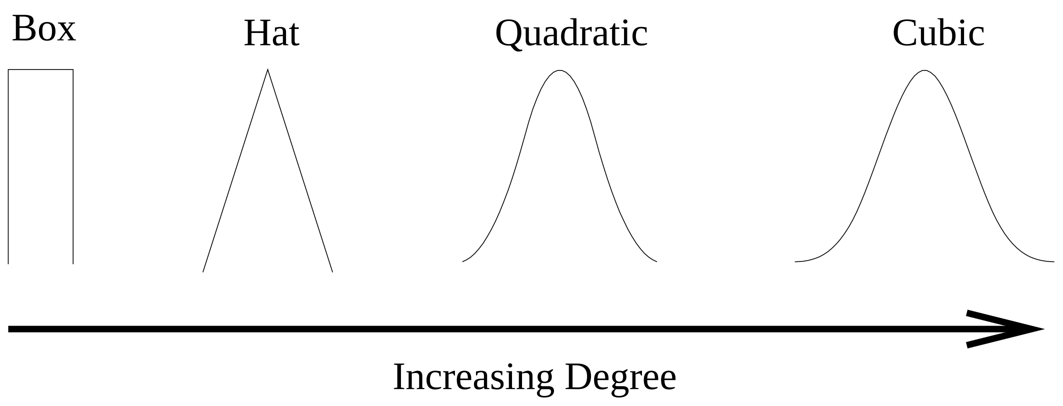

Figure 3.3: The Family of B-splines

Figure 3.3 shows the first four members of the B-spline family ranked from left to right by increasing degree. The degree zero B-spline basis is a box function, producing a piecewise constant approximation. The degree one is a hat function, producing a piecewise linear approximation. The degree two is a quadratic function, producing a piecewise quadratic approximation. The degree three is a cubic function, producing a piecewise cubic approximation.

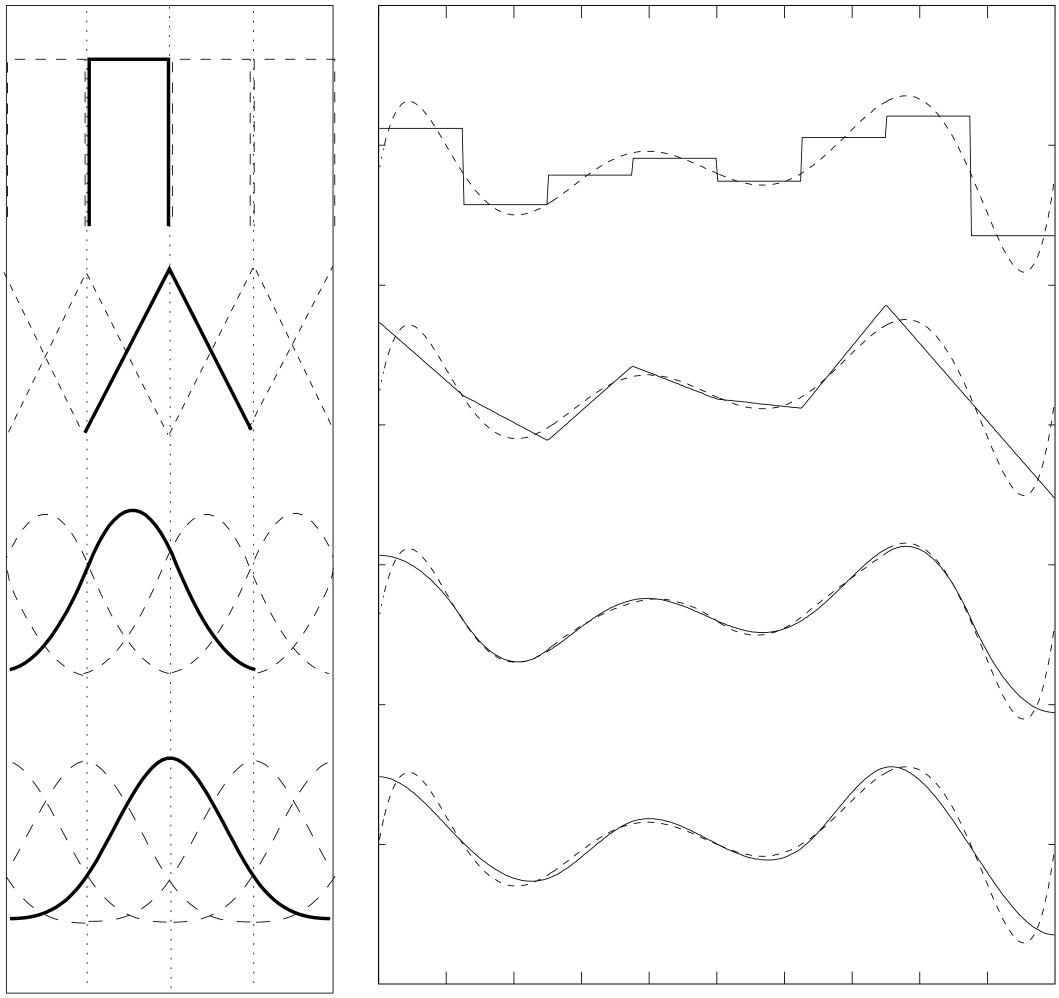

Figure 3.4: Function Approximation with B-splines

The left hand side of figure 3.4 shows the support of each function. The bold curve is the basis function and the number of intervals it straddles is indicated by the vertical dotted lines. The dashed lines are neighbouring basis functions. The right hand side of figure 3.4 shows the least squares fit to an eight degree polynomial. The splines are represented by solid lines, the polynomial by the dashed lines. The box spline has no continuity across intervals and one coefficient determines the form of the function in an interval. The coefficient in this instance represents the average value of the input function, or set of data points, in the interval. The hat spline is continuous but the first derivative is not. Two functions overlap, so two coefficients determine the function in an interval. For the quadratic spline both the function and the first derivative are continuous and three coefficients determine the function. For the cubic spline up to the second derivative is continuous and four coefficients determine the function. Thus increasing the degree of the basis function increases the degree of the polynomial pieces, the highest derivative that is continuous and the size of the support. As can be seen in figure 3.4 when fitting smooth functions like polynomials, smoother bases like the cubic are the better approximators.

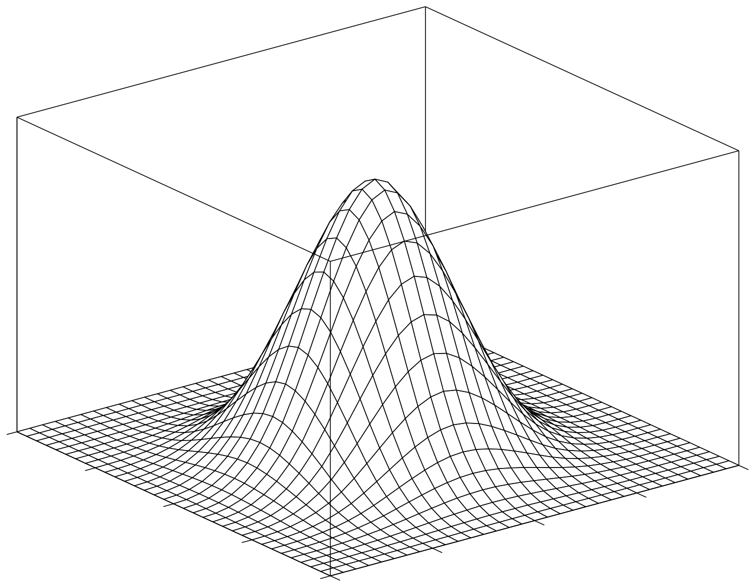

Figure 3.5: A Two Dimensional Cubic B-spline Basis

Typically higher dimension basis functions are produced by a simple product, the basis function value at a point is just the product of the values of one dimensional basis functions at their respective coordinates, as shown in figure 3.5. To completely define the spline approximation scheme it is also necessary to define what happens at the boundary of the function. A common choice is periodic boundaries, but an alternative approach more appropriate here is that the boundaries are mirror reflecting (Gortler, 1995). Intuitively one can imagine that each basis function is built up by repeatedly folding over its tail each time it crosses the boundary. The only real restriction this puts on the representation is that the differential is zero at the boundary.

3.1.2 Accuracy of Fit

The motivation for using spline function approximators is the intuition that the control function is largely smooth, nearby states have similar values. But the very features that this thesis uses to delineate the subtasks occur where this does not hold. The ideal function approximator would economically represent the smooth regions without smoothing out the features. This section compares the accuracy of fit of the four types of B-splines discussed in the previous section.

The three input functions shown in figure 3.6 were used to evaluate the accuracy of fit. The functions are intended to roughly emulate the shape of a typical control function. They are similar to the distance to goal function used in the robot navigation problem but in one dimension. All the functions are flat at the far left of the x-axis representing the goal. All the functions decay exponentially towards the right, with distance from the goal. The difference between the functions is the number of steps introduced. In figure 3.6 the function represented by the solid line has none. The function represented by the dotted line has one step and the function represented by the dashed line has two.

The subsequent three graphs show the fit of each of the various splines to the three input functions, shown as a solid line. Values for the error measured by the L_2 norm (including the square root) are shown in the top right hand corner of each graph. Figure 3.7 shows the fit when there is no step. The accuracy of fit is almost indistinguishable for splines of degree greater than one and the fit to the input function is almost exact. A small amount of difference occurs at either end of the x-axis. The best fit is the quadratic spline. The worst fit is the box spline. From the numerical value of error it can be seen that the quadratic spline has an error approximately a tenth of that of the box spline.

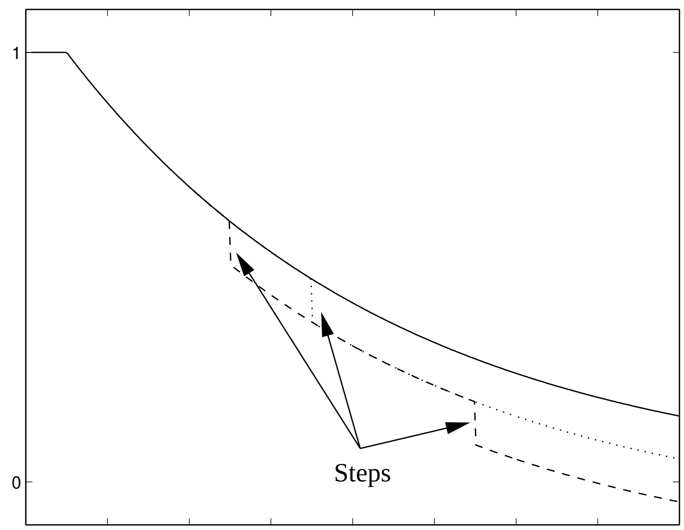

Figure 3.6: Functions to be Approximated

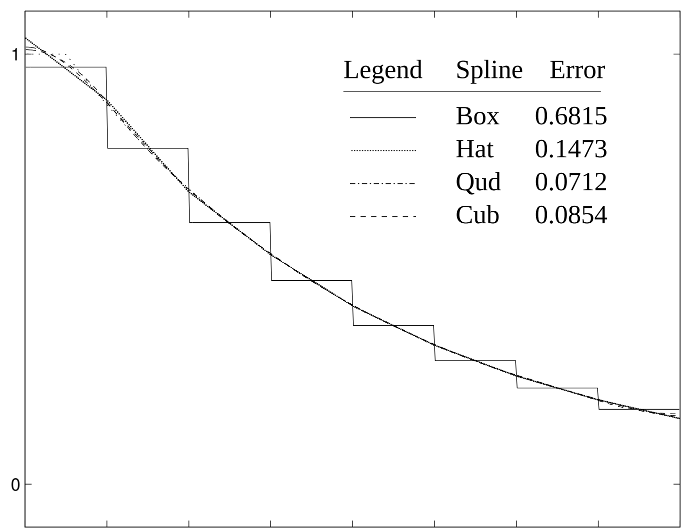

Figure 3.7: Fitting to an Exponential

When a step is added, the error for all splines increases but that of the box spline increases the least. Again there is little difference between the high order splines. In figure 3.8 the ratio of error between the best fit, the cubic spline, and the worst, the box spline, is about one quarter. This is further reduced when a second step is added to slightly larger than a third, as shown in figure 3.9. As might be expected, as the smooth regions become smaller the benefit of using a smoother spline is reduced. But even if seven steps are added the error for the cubic spline (0.5478) is still only a little less than a third of that of the box spline (1.4999). In all these examples the steps are positioned midway between two knots. If the steps are positioned coincident with the knots, the box function will have the same error as when fitting the original exponential whereas the cubic spline's error will be larger. In fact for seven steps the error ratio is only slightly less than one, the errors being 0.5612 and 0.6815 for the cubic and box splines respectively.

It is not surprising that the higher order splines better fit exponential functions, as they can represent more low order terms of the Taylor expansion. The continuity constraints do however worsen the fit to less smooth functions. But as the results have shown unless the steps actually occur at the knots, an unlikely event in practical situations, the fit is still better than the piecewise constant representation of the box splines. As was shown in section 1.3 the loss of smoothness may arise due to walls or perhaps other obstacles in the domain. If there are no obstacles at all then higher order splines will fit accurately. As the space becomes more cluttered with obstacles, the more interesting and common type of problem, the benefit decreases but is still significant. Of course, one may have very complex problems such as mazes. But in such cases it is probably important to have a fine granularity of representation to be able to accurately solve the problem. It should also be remembered that although higher order splines require fewer basis functions, the support is larger than for box functions and more coefficients need updating for each new data point. There is therefore is a trade-off of accuracy of representation and processing time. As there is little difference in fit between the higher order splines, linear splines having the smallest support might represent the best compromise.

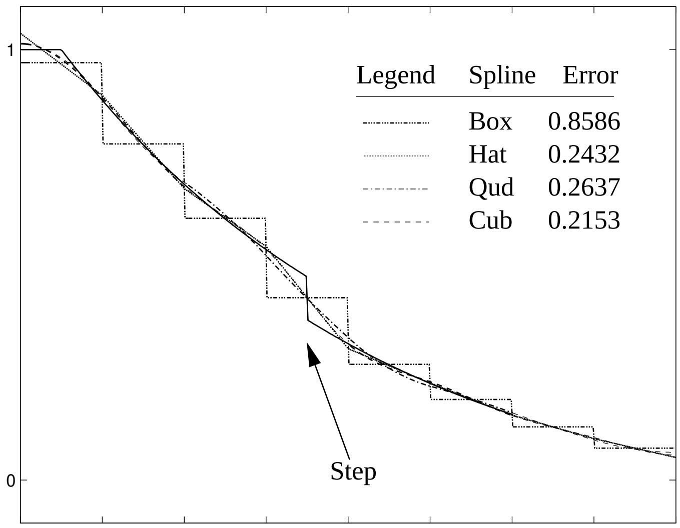

Figure 3.8: Fitting to an Exponential with One Step

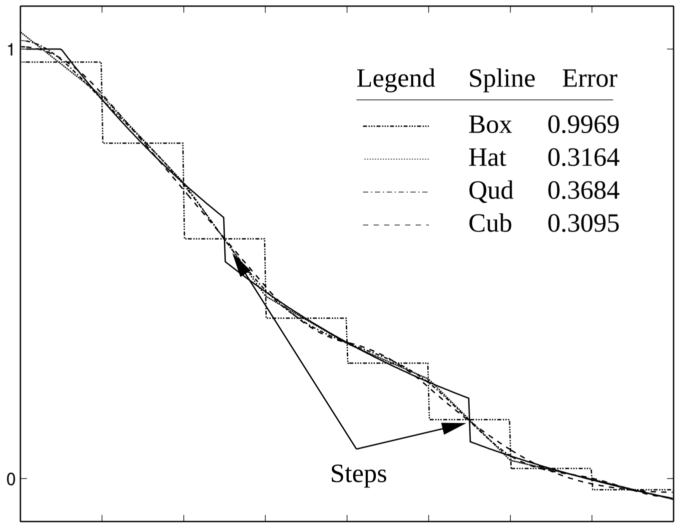

Figure 3.9: Fitting to an Exponential with Two Steps

3.2 Features in the Control Function

The previous section dealt with a B-spline representation of the control function, whose coefficients are determined by subsymbolic learning. To connect to the symbolic level, features are exploited that naturally arise in the control function. The following sections demonstrate in turn: that these features are stable when the goal is moved; that these features are easy to detect early in the learning process; that these features exist in multiple domains.

3.2.1 Changing the Goal Position

An important research area in machine learning, and particularly reinforcement learning (Sutton, 1990), is being able to deal with a changing world. This issue will be addressed in greater detail in section 9.2.1. The focus of the discussion in section 1.3 was the change of goal position. As was suggested there, if the goal is moved basic reinforcement learning will have to learn the new function from scratch. In fact, unless the algorithm includes the facility to detect that the goal has moved it will continue using the existing function. The algorithm will then largely have to "unlearn" this function before it can successfully learn the new function. In chapter 7 experiments demonstrate that this increases the time to learn the new function substantially. In the approach taken in this thesis, once the new goal is located, the function can be initialised to a close approximation to the solution of the new task. Any errors will be quickly removed by further reinforcement learning.

The success of transfer of the solution of subtasks between tasks is dependent on the stability of the features used to facilitate the transfer. Let us look at the features for two different goal positions in the robot navigation problem introduced in section 1.3. The top left of figure 3.10 shows the gradient of the function for the goal in its original position. The shaded box below the gradient function represents the goal, the straight lines the walls of the rooms. The valleys in the gradient function correspond to the rooms, the ridges to the walls. The ridges decrease towards a minimum corresponding to a doorway. Looking down on the gradient function and tracing out the base of the ridges produces the dashed lines shown at the top right of

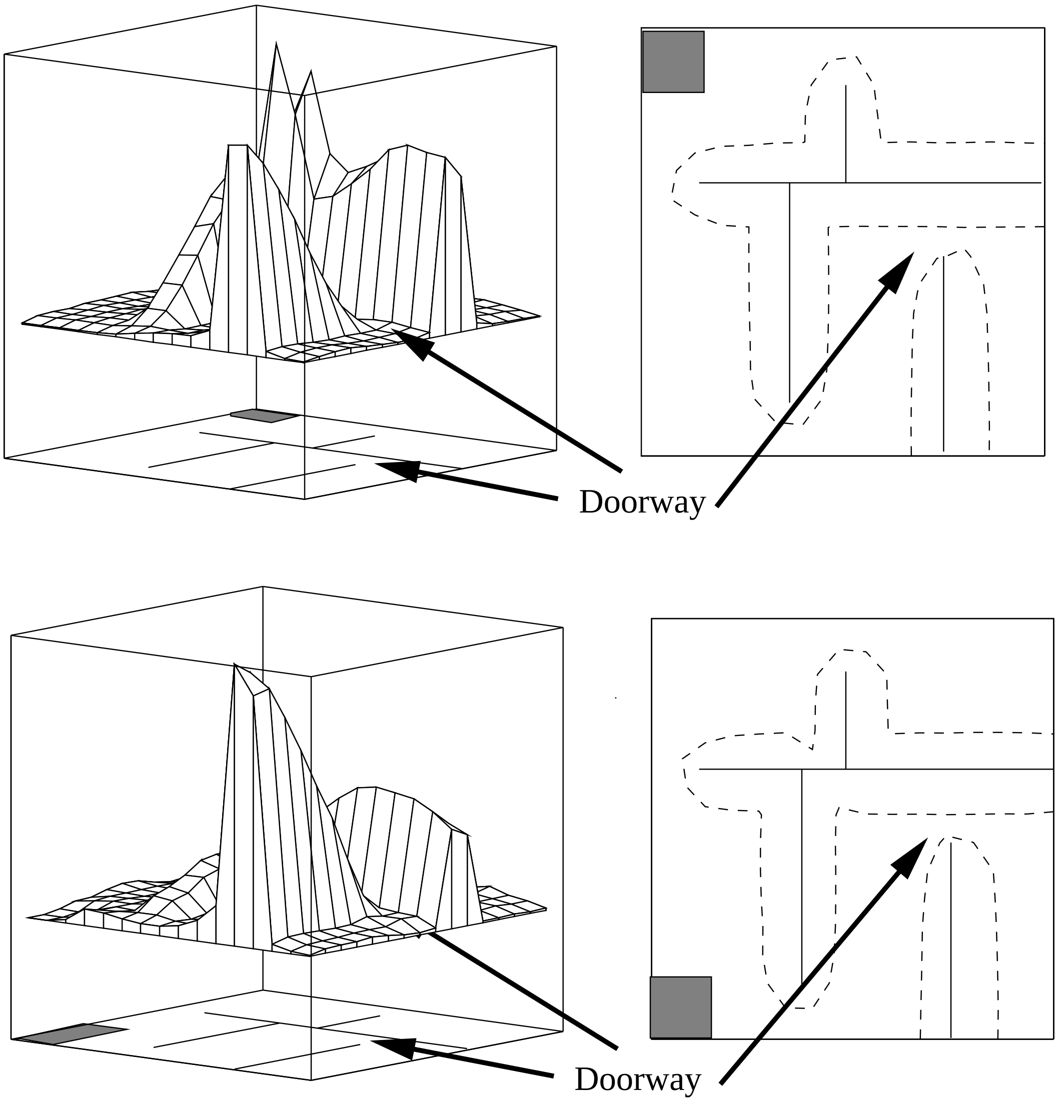

Figure 3.10: The Features Generated by Two Different Tasks

figure 3.10. To improve clarity this is again projected over the layout of the rooms. The bottom of the figure shows the gradient of the function learnt for the same suite of rooms but with the goal in its new position. Comparing the ridges, one can see that although the magnitude has changed the location of the features has not, nor has the position of the minima corresponding to the doorways. This consistency allows successful transfer between the two tasks.

3.2.2 Detecting Features Early

In the previous section the existing task and the new task were strongly related, the walls and doorways were fixed only the goal position was different. In this section no such a relationship is assumed. The robot is faced with a brand new task and must determine what, if any, relationship exists between the new task and any previous tasks.

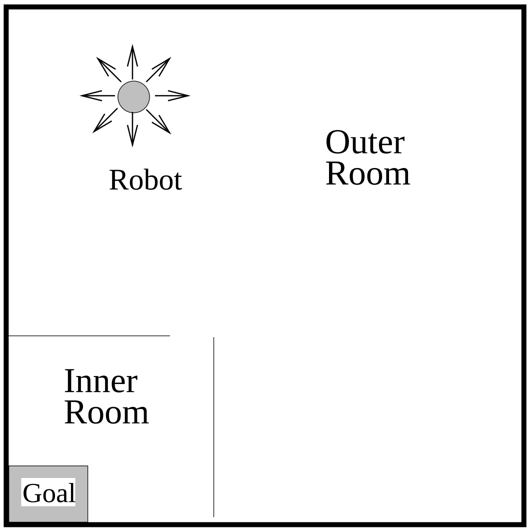

Figure 3.11: The Old Task

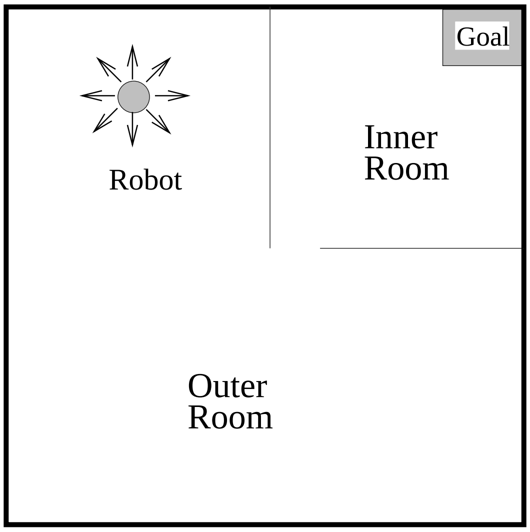

Figure 3.12: The New Task

The experimental testbed is again a simulated robot environment but this time the problem is simplified to just an inner rectangular room and an outer L-shaped room. Figures 3.11 and 3.12 show two possible room configurations. Again the thin lines are the walls of the room, the thick lines the boundary of the state space. Suppose the robot had already learnt a function for the "Old Task" of figure 3.11. We would hope that we could adapt the old solution to fit the closely related "New task" of figure 3.12.

As the learning process is started afresh there are no features and the system must wait until they emerge through the normal reinforcement learning process. In this example the first step in the process is to locate the goal. The function is initialised to a constant value (see figure 3.13), allowing some limited learning which encourages the system to move away from regions it has explored previously. This prevents a completely random walk through state space. Once the goal is located the learning algorithm is reinitialised with a function for the same goal position but no walls (see figure 3.14). If such a function has not been previously learnt any rough approximation could be used instead. The "no walls" function is not used exactly as it was learnt. The difference between the goal and the rest of the state space is reduced, by scaling the function then adding a constant. This reduces the "bias" of the function, allowing the learning algorithm to alter it relatively easily as new information becomes available.

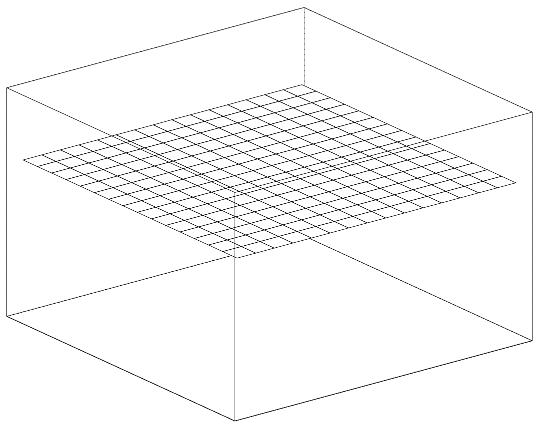

Figure 3.13: Start Function

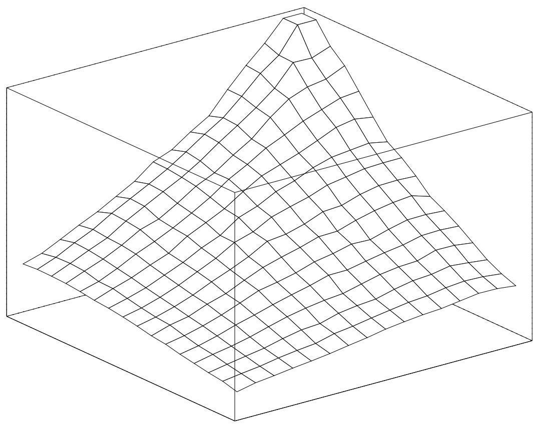

Figure 3.14: No Walls Function

Figure 3.15 shows the resultant function about 3000 exploratory steps from the beginning of the learning process. Again, the large gradients associated with the walls are readily apparent. These features become even clearer when looking at the gradient of this function as shown in figure 3.16. Figure 3.17 shows the reinforcement learning function for the new task if it had been allowed to converge to a good solution, figure

3.18 shows the gradient. Both gradients have roughly the same form even though the converged function took in excess of 200,000 steps.

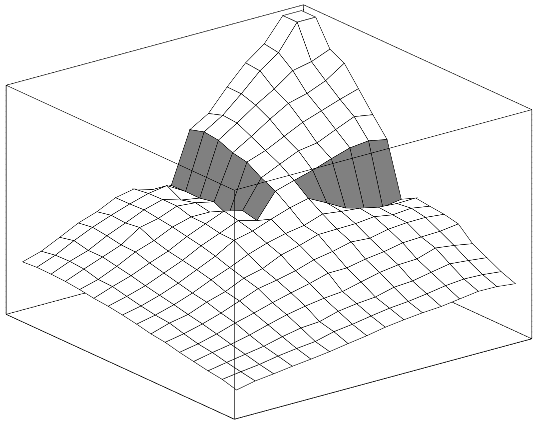

Figure 3.15: Early Function

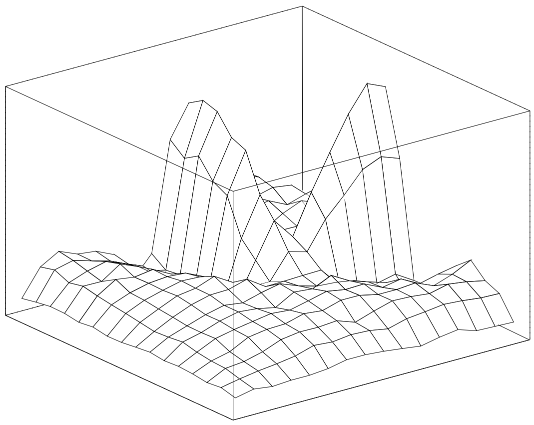

Figure 3.16: Early Gradient

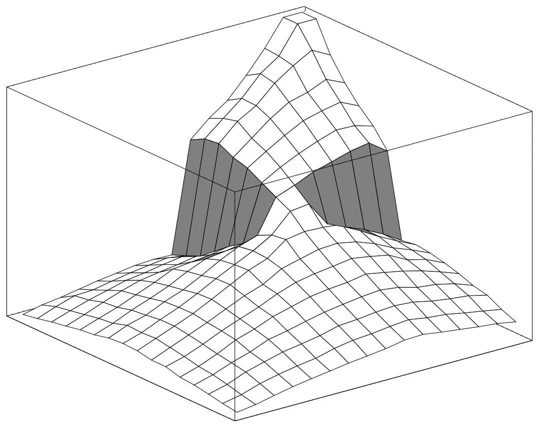

Figure 3.17: New Task Function

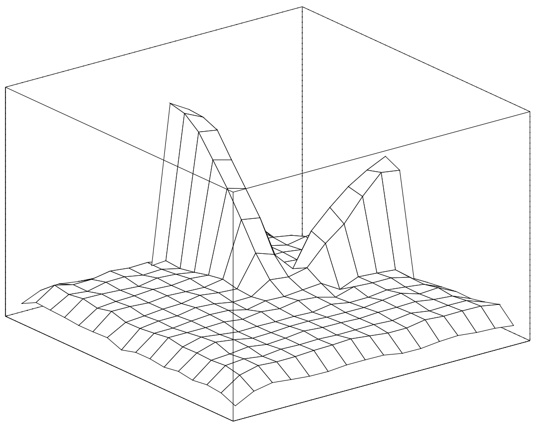

Figure 3.18: New Task Gradient

3.2.3 Features in Different Domains

The previous sections dealt with a simple robot navigation problem. This section demonstrates that these features also exist in two quite different domains.

3.2.3.1 A Robot Arm

The first domain is that of a two degrees of freedom robot arm, as shown in figure 3.19. The aim is to learn to move the arm efficiently from any initial position until the hand reaches the goal on the perimeter of the arm's work space. The shoulder joint can achieve any angle between $\pm\pi$ radians, the elbow joint any angle between $\pm\pi/2$ radians, zero is indicated by the arrows. If the arm is straight and the shoulder joint rotated the elbow joint will describe the inner dotted circle, the hand the outer dotted circle. There are eight actions, small rotations either clockwise or anti-clockwise for each joint separately or together.

The state space for the purposes of reinforcement learning is the configuration space for the arm, sometimes called the joint space (see figure 3.20). The x-axis is the angle of the shoulder joint, the y-axis the elbow joint. The eight actions when mapped to actions in the configuration space become much like the actions in the robot navigation problem, as shown by the shaded diamond, labelled "Arm" in figure 3.20. To map an obstacle in the work space to the configuration space, one must find all pairs of shoulder and elbow angles blocked by the obstacle. The obstacles in this space become elongated to form barriers much like the walls in the experiments of the previous sections. If this is not clear imagine straightening the arm in the work space and rotating it such that intersects one of the obstacles, the middle dotted line in figure 3.19. The arm can then be rotated at the shoulder joint with a roughly linearly proportional rotation in the elbow joint but in the opposite direction so as to keep it intersecting the obstacle. This produces the "wall" in the configuration space. This linearity holds only for small objects not too far from the perimeter of the work space. More complex larger objects would be result in more complex shapes in the configuration space.

The reinforcement learning function produced by this problem is shown in figure 3.21. As before the features are shaded for clarity. The large gradient for the obstacle on the left hand side of the configuration space can be clearly seen. The one for the obstacle on the right side is mostly obscured, part of it can be seen at the bottom right hand side of the figure. Again these features can be used to control the composition of functions if the goal is moved or for a different task in the same domain.

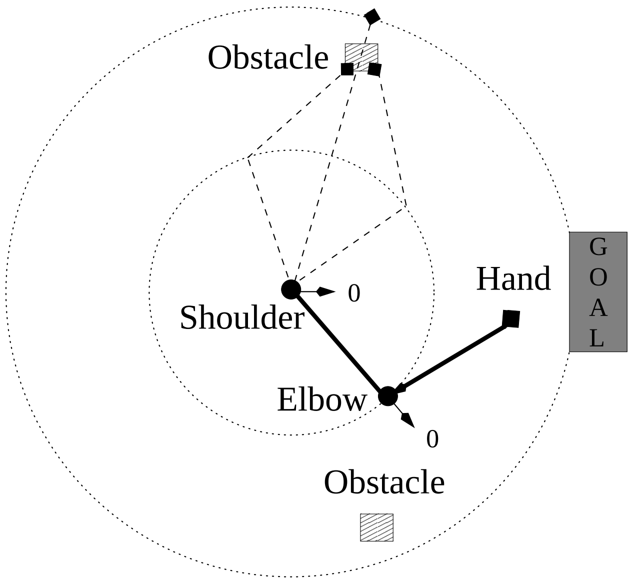

Figure 3.19: Work Space

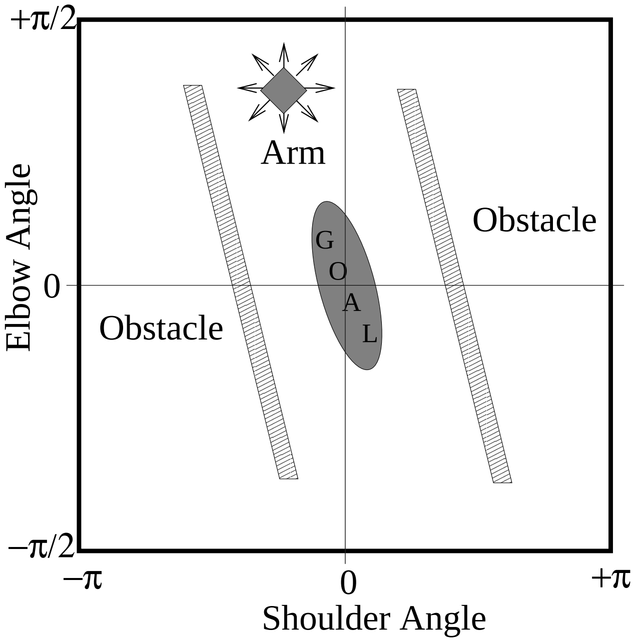

Figure 3.20: Configuration Space

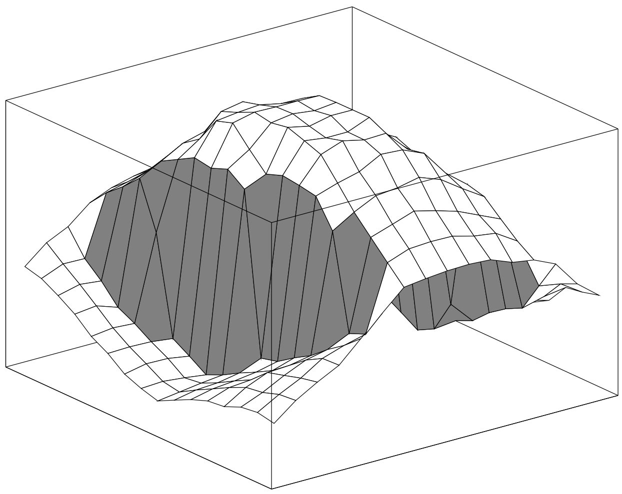

Figure 3.21: The Reinforcement Learning Function

Figure 3.22 shows the gradient of the function after about 200,000 learning steps. Figure 3.23 shows the gradient after 4000 learning steps. As in robot navigation domain discussed in section 3.2.2 these features emerge early in the learning process and their shape and position is stable.

3.2.3.2 A Car on a Hill

An alternative domain is that of the "car on the hill" (Moore, 1992) that has become a standard test bed for reinforcement learning algorithms. The task, simply stated, is to get a car up a steep hill, figure 3.24. If the car is stationary part way up the hill, in fact anywhere within the dotted line, then it has insufficient acceleration to make it to the top. So the car must reverse down the hill and then get up sufficient forward velocity, by accelerating down the other side, before accelerating up the hill.

The state space for the purposes of reinforcement learning is the position and velocity of the car, as shown in figure 3.25. The goal is to get up to the top of the hill with a small positive or negative velocity. In this domain there are two possible actions: accelerate forward, accelerate backwards. Unlike the previous domains there is no clear mapping of the actions onto the state space. The state achieved on applying an action is determined by Newton's laws of motion. As the car has insufficient

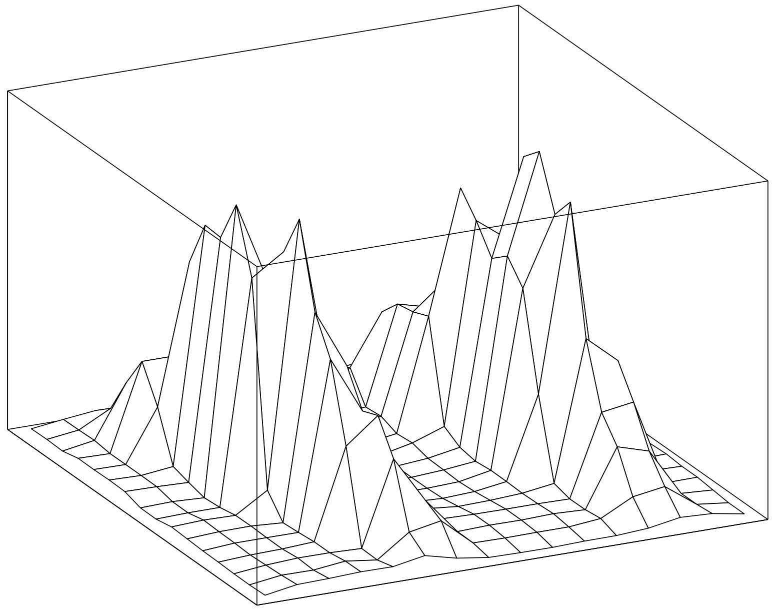

Figure 3.22: The Gradient

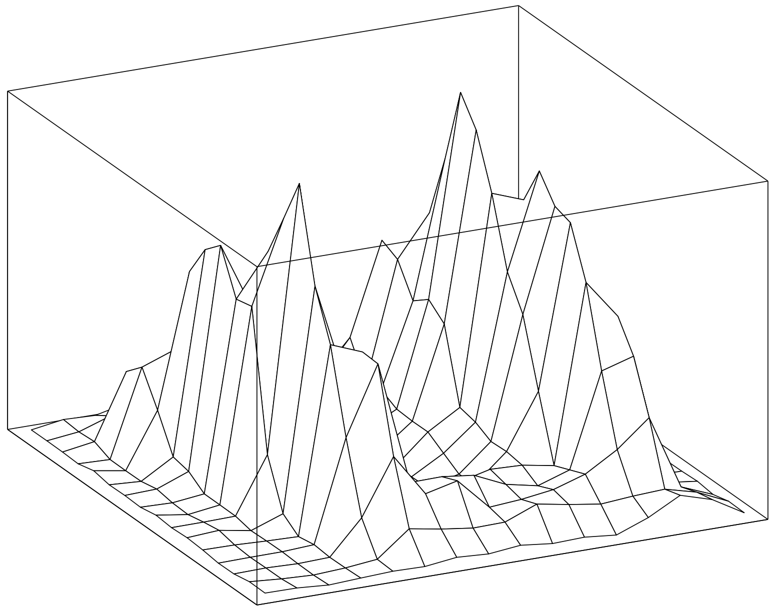

Figure 3.23: The Gradient after 4000 steps

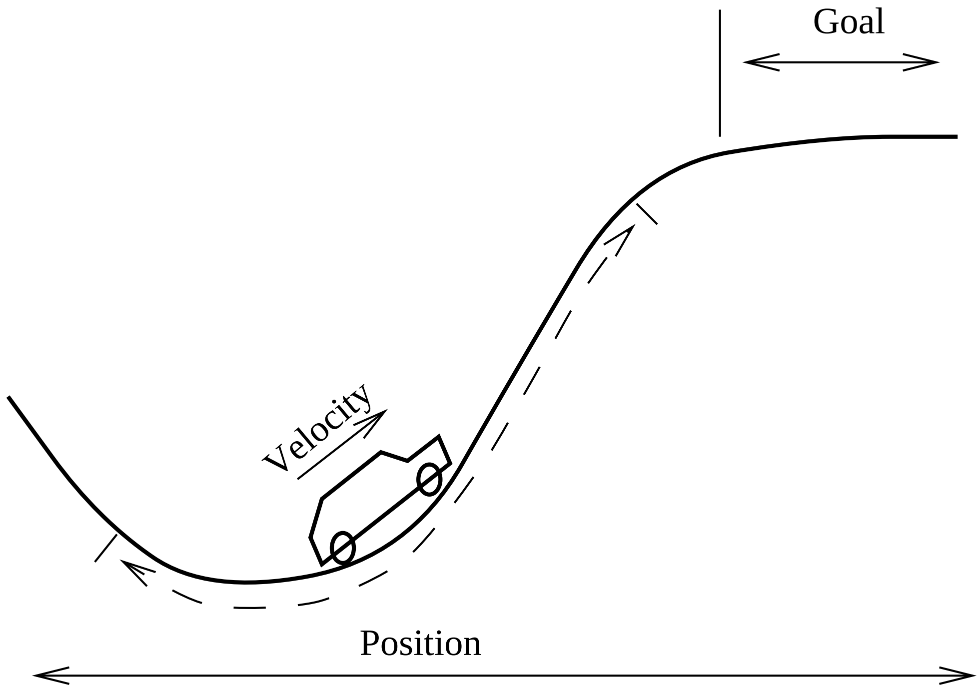

Figure 3.24: The Car on the Hill

acceleration to make it up the hill from everywhere in the state space, a "wall" is effectively introduced, the bold line in figure 3.25. To reach the top of the hill the car must follow a trajectory around this "wall" the dashed line in figure 3.25.

Figure 3.26 shows the reinforcement learning function. It exhibits the same steep gradient as the other domains as shown in figure 3.27. The important point to note is that unlike the other domains no physical object causes this gradient, it is implicit in the problem itself. Yet the features still exist.

3.3 In Summary

This chapter has discussed the control function, the central component in the system's architecture. Connection to subsymbolic learning is through the coefficients of the spline function approximator. The B-splines used have many advantages: offering many choices of smoothness and continuity, having small support and therefore efficient implementation. Connection to the symbolic layer is through features in

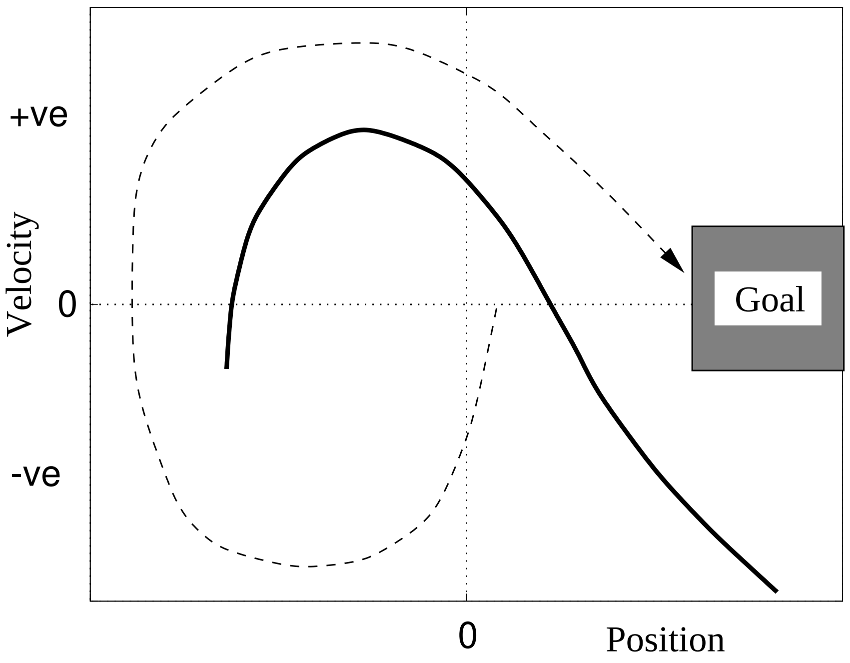

Figure 3.25: Car State Space

the control function. This chapter has demonstrated the stability of these features and that they become prominent early in the learning process. It has also shown that these features are not restricted to domains with walls like the robot navigation problem. They are also apparent in other domains. They may not even be caused by physical objects. In the "car on the hill" domain there are no explicit obstacles that prevent the car form reaching the top of the hill. The features that arise are completely implicit in the problem.

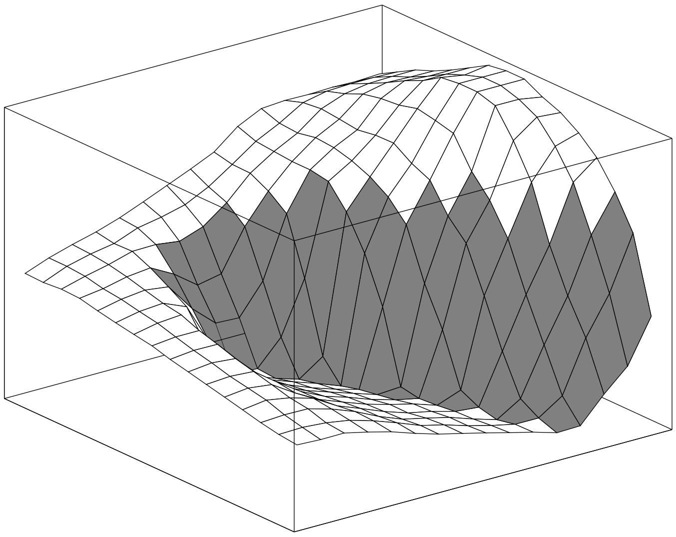

Figure 3.26: Reinforcement Learning Function

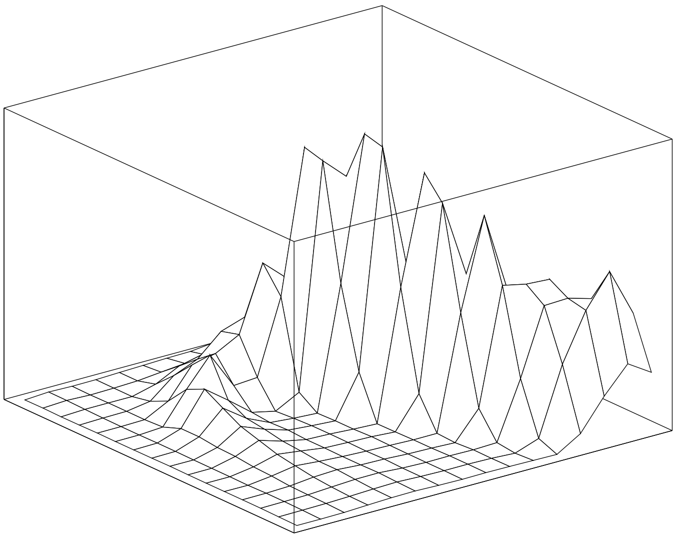

Figure 3.27: The Gradient

Chapter 4

Subsymbolic Learning

This chapter addresses how to learn the control function. In the context of the architecture discussed in section 1.2 this is subsymbolic learning as shown in figure 4.1. There are many algorithms that might be used to learn a control function, some of which will be discussed in section 4.2. Reinforcement learning algorithms have proved to be particularly useful in control tasks and are the focus of this work. To generalise reinforcement learning to continuous spaces some form of function approximation is required. Many forms of function approximation can cause reinforcement learning to diverge. The main thesis contribution discussed in this chapter is showing how the B-splines, discussed in chapter 3, can be controlled so as to prevent divergence.

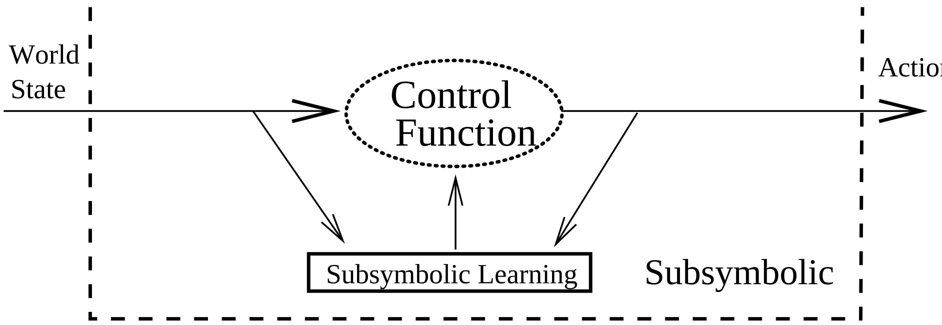

Figure 4.1: Subsymbolic Layer

A control function is a mapping from world states to actions. Learning such a function is a matter of determining the best action to take in each world state to solve a particular task. To this end, a real valued reinforcement signal is generated internally on achieving certain world states. If reinforcement is immediate it is only necessary to try each action, in a particular state, and select the best one. If reinforcement is infrequent actions cannot be evaluated individually. Yet evaluating all possible combinations of actions would be prohibitively expensive. Some way to explore this space more effectively is needed. One approach might be a genetic algorithm (Holland, 1975), where each gene represents an action in a particular state, the chromosome would represent the control function for the whole state space. Alternatively some form of temporal difference learning (Sutton, 1988) might be used, such as a classifier using the bucket brigade algorithm (Holland, 1986, 1976) or one of the reinforcement learning methods.

For the purposes of this thesis all low-level learning methods are viewed as associative learning: the association between actions that maximise rewards (or minimise costs) and the relevant states become strengthened over time. Various approaches store the strength of this association in various ways. In schemes like learning automata (Narenda & Thathachar, 1974) the strength is stored in a vector of probabilities of an action being taken in each state. In a genetic algorithm the strength is essentially the number of schemata for a particular state-action couple in the population. In a classifier system it is the relative strength of a classifier to those competing to service a message. In reinforcement learning systems it is the relative value of an action in a particular state with respect to other actions in that state.

Control problems have many inherent difficulties. Reinforcement may be stochastic: the best action is the one that results in the maximum expected reward. Reinforcement may be infrequent: the best action is the one that results in the maximum expected sum of rewards. The result of an action may be stochastic: taking the same action in the same state may result in moving the system to different states. The state-action space may be continuous: the system never takes exactly the same action from exactly the same state twice. Reinforcement learning research has to some extent investigated all these problems. But for about the last decade the problem of continuous spaces has been largely ignored. The main focus has been on using look up tables in discrete spaces (Sutton & Barto, 1998). In the last few years there has been a resurgence of interest in more general function approximators. Unfortunately many types of approximators diverge. This chapter discusses how an important type of function approximator, B-splines, can be modified to prevent divergence.

4.1 Reinforcement Learning

This section begins with a simple overview of reinforcement learning applied to the robot navigation example given in section 1.3. Figure 4.2 shows the state space divided up into discrete states. The robot has eight actions taking it to one of its eight neighbouring states.

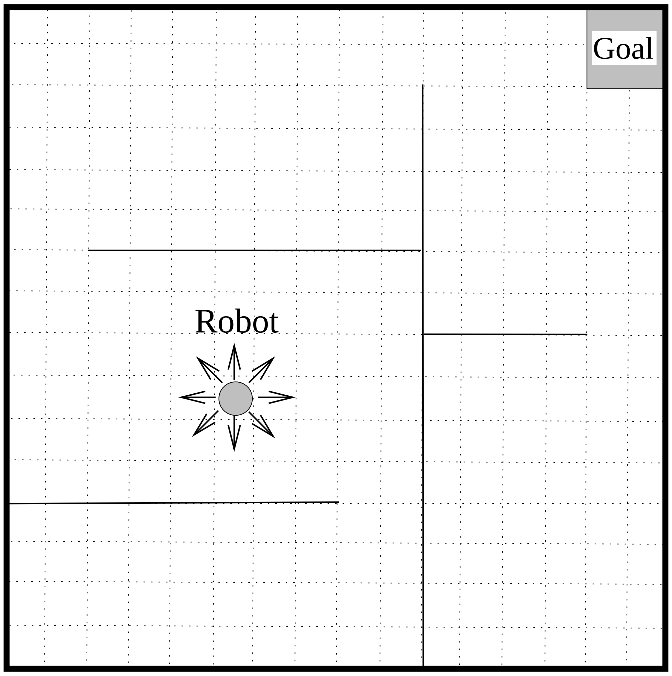

Figure 4.2: A Discrete State Space

Suppose the aim is to label each discrete state with the shortest distance to the goal. Section 1.3 discussed how this might be used to determine the optimum path to goal. One way to label the states would be to enumerate all possible routes from each start state and then take the minimum. An alternative is some form of reinforcement learning. At the beginning the robot only knows the shortest distance to the goal

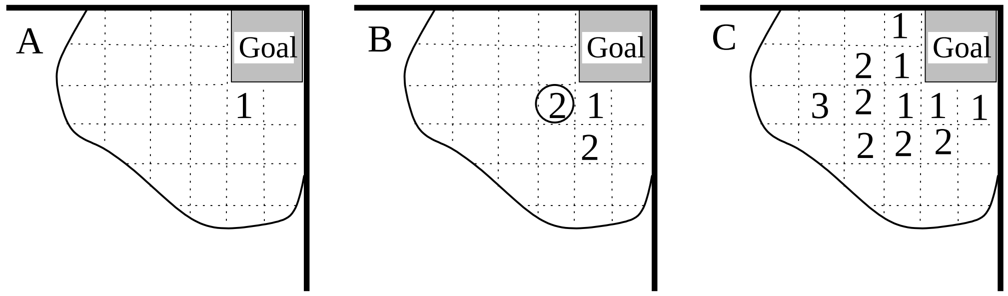

Figure 4.3: Propagating Back Values

at the goal itself. But on reaching the goal it can label the prior state as having a distance of one, see figure 4.3 A. Whenever it reaches this state again it can label its prior states as two steps from the goal, see figure 4.3 B. Strictly speaking this number is only an upper bound on the distance to goal. It is possible the goal can be reached in a single step. If it is on some subsequent occasion, the state can then be updated with the new information, e.g. the circled distance of 2 in figure 4.3 B is changed to 1 in figure 4.3 C. The result is the progressive flow of information backwards from the goal, with the values being successively updated until they represent the exact distances to goal from each state. This description assumes the world is deterministic, in a stochastic world the same effect can be achieved using expectations rather than actual values.

Iterating the value backwards from the goal means a linear number of calculations is needed rather than the exponential number needed for enumerating all paths. However in normal learning the robot moves forward towards the goal while the information flows backwards. This, combined with the size of the state action space, means that a large number of actions must be taken to produce an accurate solution.

This approach is effective as the optimal path consists of sub-paths that are optimal and these sub-paths are part of many optimal paths. This notion is apparent in the structure of the reward function typically used in reinforcement learning, the "discounted cumulative reward". This is based on the sum of all future rewards, but where the value of a reward is reduced progressively by the γ factor the farther into the future it occurs. The reward r_n is received after taking n actions, the first few rewards are shown in equation 4.1. The structure becomes more apparent when it is rewritten in the form of equation 4.2

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \tag{4.1}$$

$$r_0 + \gamma(r_1 + \gamma(r_2 + \gamma(r_3 + \ldots))) \tag{4.2}$$

The value V_s, or utility, of the state s is then the expected sum of future rewards, as shown in equation 4.3. This is equivalent to the sum of the expected values of immediate rewards. This gives rise to the famous Bellman equation (Bellman, 1957) and shown in equation 4.4. The optimal value function V_s^* for state s is the maximum of the expected immediate reward $R_{s,a}$ plus the optimal value $V_{s'}^*$ of the next state s', discounted by γ, of any action a in the current state. The optimal value of the next stateis just the expected sum of future rewards from that state onwards.

$$V_s = E[\sum_{i=0}^{\infty} \gamma^i r_i] = \sum_{i=0}^{\infty} \gamma^i E[r_i] \tag{4.3}$$

$$V_s^* = max_a(R_{s,a} + \gamma V_{s'}^*) \quad where \quad R_{s,a} = E[r_0] \tag{4.4}$$

This "embedding" notion is strongly tied to the notion of Markov processes. The Markov assumption normally used is that properties of the state are independent of how the process reached the present state. Thus all trajectories that arrive at that state have the same potential future. This simplification has been used extensively in modelling, and most reinforcement learning algorithms are dependent of this assumption.

The two most popular reinforcement learning algorithms are Q-learning, and TD(λ). These two algorithms are very similar in many respects and both have been proven convergent under certain constraints (Watkins & Dayan, 1992; Jaakkola, Jordan, & Singh, 1994). They differ in the fact that TD(λ) uses two distinct structures – one for the value of the state and one for the efficacy of individual actions – whereas Q-learning combines these into a single structure indexed by the state and action. TD(λ) also includes the λ factor, which ranges from 0 to 1. When this factor is zero only immediate rewards are used in learning, when it tends towards one actual rewards obtained including those farther in the future are used. This latter feature of TD(λ) has been added to Q-learning to produce Q(λ) (Peng & Williams, 1994) and seems to speed up learning.

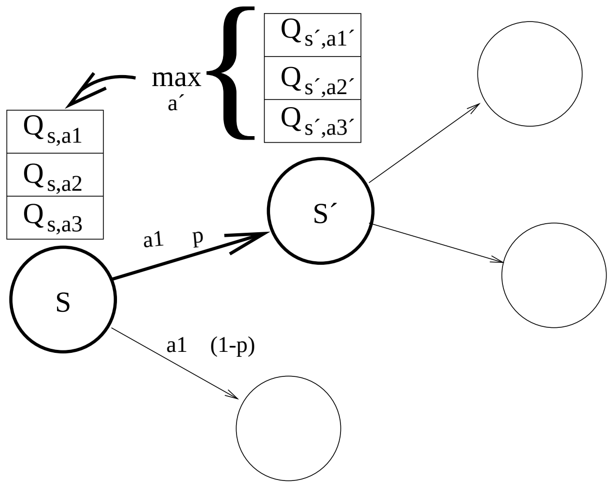

Figure 4.4: Discrete Q-learning

This research adopts the Q-learning algorithm, at present the one most commonly used by the reinforcement learning community. Associated with each state is a record of the present estimate of the expected reward for taking each action, figure 4.4. The collection of these records produces a look-up table for the whole state action space. As the process is stochastic, taking a particular action may move the system into any one of a number of states. In figure 4.4, taking action $a1$ in state s has a probability p of moving the system to state s' and the probability $(1-p)$ of moving it to some other state.

$$Q^{t+1}_{s,a} = (1-\alpha)Q^t_{s,a} + \alpha(r_{s,a} + \gamma max_{a'} Q^t_{s',a'}) \tag{4.5}$$

Let us assume that the system moves from state s to state s' as the result of taking action a. The system now backs-up the result, equation 4.5. It first finds the maximum value for any action a' in this new state, multiplies it by the discount factor γ and adds it to any reward $r_{s,a}$ received when taking the action from the old state. The value for the action taken from the old state is then updated, the amount depending on a learning rate α. The action selected in each state is usually the one with the highest value of $Q^t_{s,a}$. But to encourage exploration of the state space this paper uses an ϵ-greedy policy (Sutton, 1996) which chooses a random action a fraction ϵ of the time.

4.1.1 Function Approximation

The assumption that interacting with the world is a discrete Markov decision process is central to many reinforcement learning algorithms. Thus both the states and actions are discrete. This is obviously a strongly simplifying assumption. But to date this approach, allowing the representation to be a discrete look up table, is the only case where there are true convergence results, summarised in a paper by Barto, Bradtke and Singh (1995). Beer (1995a) discusses the world and the agent as being a dynamical system describable by the interaction of two differential equations. A better model of such an interaction is as a continuous Markov process. Such dynamical systems may be very complex and some approximation must be made. Modelling the world as a finite state automata (Basye, Dean, & Kaelbling, 1995), for instance, is intentionally an approximate model of the world. One can view such discretisation as just one choice of basis, i.e. piecewise constant functions. There are many other choices of basis functions, some having more inherent smoothness and are thus better able to represent smooth dynamics. This chapter discusses reinforcement learning using such smooth bases.

This thesis replaces the look-up table, described in the previous section, with splines. As the actions are discrete, each action is represented by its own spline. The back-up of values is similar to the discrete case, as shown in equation 4.6. The discrete Q-value is however replaced by the function approximator $\hat{Q}_a(x)$. Once an action a is taken and a new state x' reached the value for each action's spline at that point is determined and the maximum found. This value is reduced by the discount factor, but instead of using a constant value, γ is raised to the power of the size of the step taken, the distance $d(x, x')$ between x and x'. The result is then added to the immediate reward $r_a(x)$. Now rather than directly updating the old value, $D_a(x)$

becomes a new data point for the spline function approximation.

$$D_a(x) = r_a(x) + \gamma^{d(x,x')} max_{a'} \hat{Q}_{a'}(x') \tag{4.6}$$

This thesis will consider two ways of updating the spline function representing the Q-values: solving a matrix representation or using the Widrow-Hoff rule. The matrix representation (Lancaster & Salkauskas, 1986; Eubank, 1988) is shown in equation 4.7. The vector $\mathbf{D}$ contains the D values of all states visited for a particular action, the matrix $\mathbf{V}$ the values of the basis functions at the associated states. There are many ways to solve for the coefficients $\mathbf{C}$ such as LR decomposition and various iterative techniques well suited to sparse matrices (Dahlquist & Bjorck, 1969; Stewart, 1992). Multiplying these coefficients with their associated basis functions produces a new function approximator, $\hat{Q}_a(x)$.

$$\mathbf{V^T V C} = \mathbf{V^T D} \qquad where \qquad \mathbf{V}_{ij} = \phi_j(x_i) \tag{4.7}$$

It is not in fact necessary to keep all the D values obtained so far in one large matrix. Instead of updating $\mathbf{V}$ and $\mathbf{D}$, we can directly update two fixed size matrices $\mathbf{V^T V}$ and $\mathbf{V^T D}$ as each new point is added. Updating can be done in batch mode, say by collecting a fixed number of samples or all the samples for a complete trajectory. The influence of old values must be reduced as learning proceeds. This is achieved by multiplying the old $\mathbf{V^T V}$ and $\mathbf{V^T D}$ by $(1 - \alpha)$ and the new values by α as in the discrete case.

The Widrow-Hoff rule can be applied as each new state is reached. First the error is determined, the difference in the value of the approximator and the new data point, as shown in 4.8. This is multiplied by the contribution of the particular basis function ϕ_j and the learning rate α to produce the change to ϕ_j's coefficient c_j. This exploits one feature of B-splines. The small support of the basis functions reduces the number of coefficients that determine the Q-value in the new state and that need to be updated when a new point is added.

$$\Delta c_j = \alpha \phi_j(x) \left(D_a(x) - \hat{Q}_a(x) \right) \tag{4.8}$$

4.2 Other Low-Level Learning Methods

Although a reinforcement learning algorithm has been used in this thesis, the overall model is not intended to be tied to a particular form of low-level learning. In fact, Sutton and Barto (1998) suggest that reinforcement learning is less a matter of the algorithms used and more a matter of the types of problem being addressed. Certainly the control tasks discussed in this thesis are of this type. But one common feature of reinforcement learning algorithms of particular importance here is the value function representing the utility of states. It is the differences in the utility value that gives rise to the features discussed in section 3.2. Nevertheless quite different algorithms could be used. For instance, in a genetic algorithm (Holland, 1975) each gene might represent a coefficient of the splines representing the control function.

A difficulty with this approach is evaluating the fitness of each chromosome. If each chromosome represents a control function, ideally trajectories from each state to the goal would be needed to assess the fitness. In fact as the process is stochastic a single trajectory from each state is not really enough. Still a smaller number of trajectories might be used to get some reasonable assessment of the fitness of the chromosome. Moriarty and Mikkulainen (1996) detail a genetic algorithm that learns a neural network to solve the "pole and cart" problem. It is necessary to evaluate each candidate network to produce a fitness function. This approach does not exploit the embedding property of the problem, although the authors claim a speed-up in learning.

One way to combine these features is to use a classifier system. This already incorporates the notion of temporal difference learning typically through the bucket brigade algorithm. Much as with the reinforcement learning methods information is passed back from stages that are close to the source of the reward to those that are more distant. In this case it is not the discounted reward itself that is passed back. The reward is passed back but how much is dependent on the competition between classifiers. It is possible to modify the classifier system to essentially carry out the same procedure as the reinforcement learning algorithms (Mataric, 1991).

4.3 Using Reinforcement Learning with Splines

This section discusses combining reinforcement learning with a function approximator based on B-splines. As was suggested in section 3.1.2 the advantage of using higher order B-splines is getting a more accurate fit with same number of basis functions. Alternatively, one can get the same accuracy with fewer basis functions. Figure 4.5 shows the value function for the robot arm problem discussed in section 3.2.3.1, but instead of using 256 box functions 81 linear functions were used. The average distance to goal is almost identical (9.5 and 9.65 respectively), yet less than a third of the number of basis functions is used. Reducing the number box splines to 64 (odd ordered splines need one extra basis in each dimension) produces an average path length of 14.75. The fit is particularly poor close to some of the doorways, effectively missing certain paths to goal.

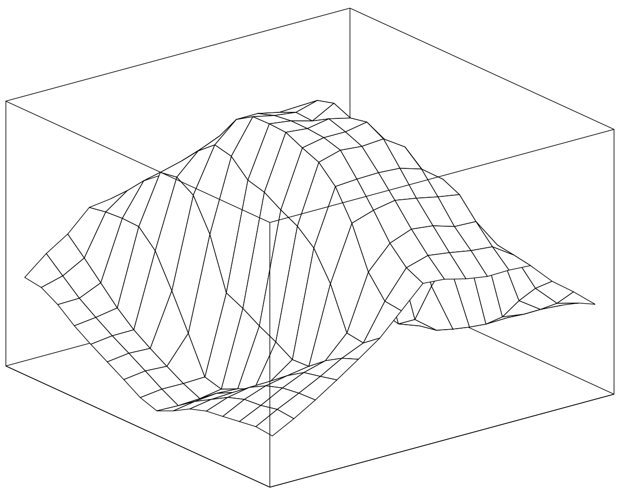

Figure 4.5: Fitting with Linear Splines

Unfortunately sometimes function approximation combined with reinforcement learning can diverge. Subsequent sections discuss an example where divergence was observed, the source of the problem and how it was solved.

4.3.1 An Example of Divergence Using Splines

Early experiments in the "car on the hill" domain, discussed in section 3.2.3.2, produced divergence when using a cubic B-spline function approximator. The problem was traced to overshoot of the spline when fitting a discontinuity in the value function. Overshoot occurs when the approximator tries to fit a function with a steep slope. Figure 4.6 shows the effect in one dimension when approximating a step function. The "least squared" curve is the best fit, in terms of minimising the squared error, using a cubic B-spline.

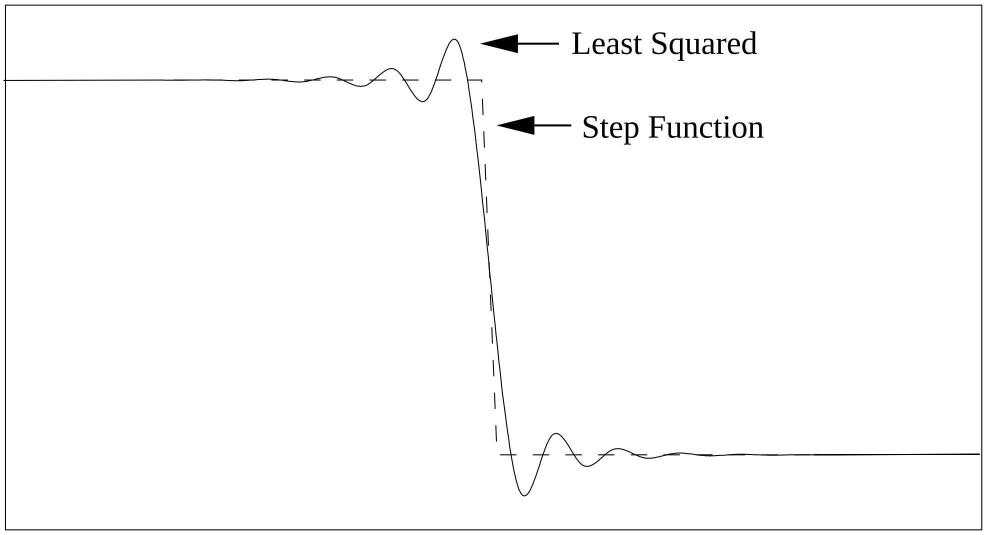

Figure 4.6: Overshoot using Cubic B-splines

The smooth function produced by the B-spline can accurately fit the flat regions on either side of the step the problem is the step itself. Reducing the error very close to the step increases the error further away. The minimisation process trades off errors in different regions of the function to obtain the best fit. This produces an oscillating function, as seen in figure 4.6, alternately producing too high and too low a value for some distance from the step.

Figures 4.7 and 4.8 show the value function for the "car on the hill", at two stages in the learning process. The function approximator is a two-dimensional cubic B-spline using a grid of 33 by 33 basis functions. As the discontinuity is approximated,

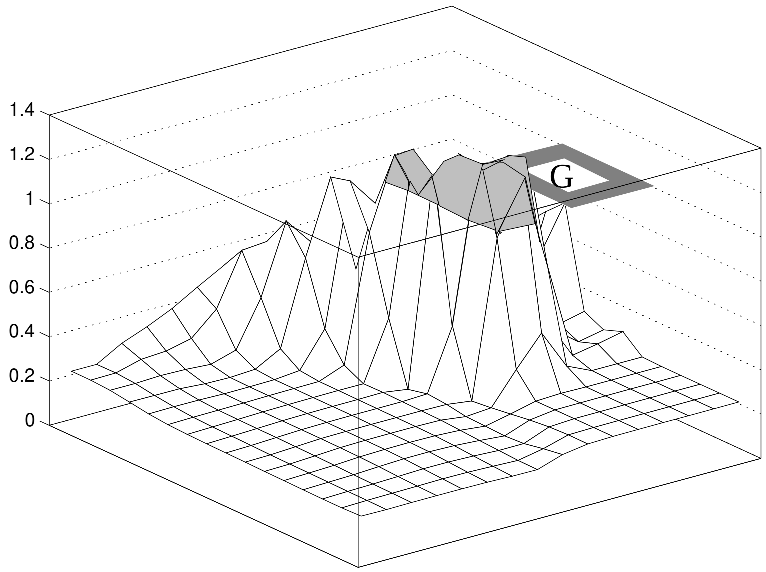

Figure 4.7: Overshoot along a Discontinuity

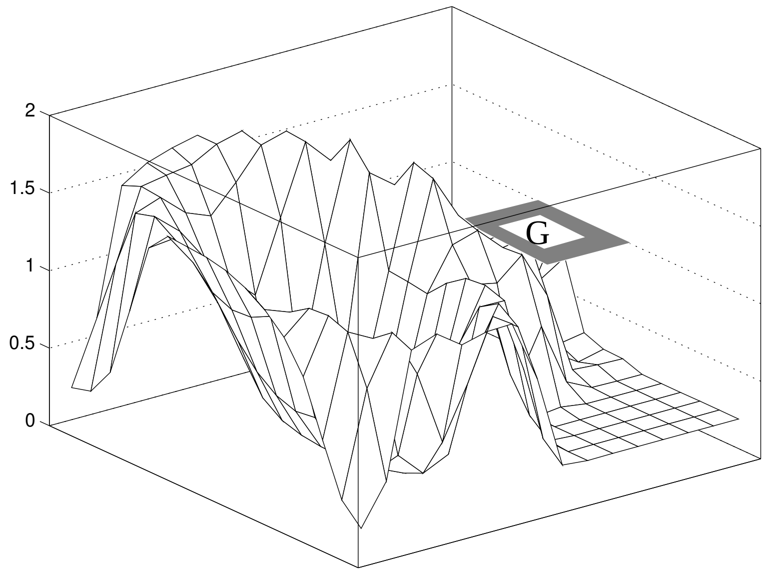

Figure 4.8: Divergent Approximation

overshoot occurs along the top of the resultant edge. This is principally evident just in front of the goal, indicated by the quadrilateral labelled 'G' in figure 4.7, where the steepest slope occurs. The overshoot has caused a region of the function, indicated by the shading, to have a value greater than that of the goal. This becomes the local best value and other states leading up to it become exaggerated. Eventually a ring of states is formed of values greater than the goal, as shown in figure 4.8. The average height of this collection of states then continues to increase, producing divergence.

The potential divergence of function approximators is a well known problem in reinforcement learning. It became apparent when many researchers started looking at more general representations than look-up tables. Although there have been a number of successes using neural networks to approximate the value function (Lin, 1993; Tesauro, 1988), it was quickly realised that many well known function approximators were not guaranteed to converge (Boyan & Moore, 1995). In dynamic programming, from which much of the work in reinforcement learning derives its inspiration, splines have been used for some time (Daniel, 1976; Birnbaum & Lapidus, 1978). But here splines had been used in limited horizon approximation which does not have the divergence problem of the infinite horizon approximation used in reinforcement learning. Recent research has been concerned with constructing special conditions under which some sort of function approximation will work. Gordon (1995b, 1995a), for instance, restricts the approximation to be an averager. This does not exaggerate the differences between points and thus prevents divergence. Baird (1995) combines global error correcting techniques with local ones to prevent divergence. In the research discussed in this chapter a stabiliser is used which further smooths the spline. Choosing the right stabiliser function should give the right conditions for the approximation to converge. This choice will be shown to be closely related to the averager discussed in both of Gordon's papers. The advantage of this approach is that it extends Gordon's result to an important family of function approximators.

Overshoot is one possible factor in producing divergence. Whether or not there is a single factor or at least one dominant factor has been the centre of some speculation. One suggestion is that divergence is primarily due to using off-line rather than online learning (Kaelbling, Littman, & Moore, 1996; Sutton, 1996; Tsitsiklis & Roy, 1997).

In online learning values are backed-up for the actual states visited when exploring the world. In off-line learning, unconstrained by real world experience, back-ups can occur in any order. Although Boyan and Moore (1995) had shown many cases of divergence, when some of these were repeated by Sutton (1996), where one principle difference was the use of online learning, the problems did not re-occur.

The original example of divergence discussed in this thesis did indeed occur using off-line learning. But the example given above used online learning, but with a reasonably high exploration rate. A ϵ-greedy policy was used where ϵ was set to 0.5. With only two possible actions, this results in the greedy action being taken 75% of the time and the non-greedy action the remaining 25% of the time. Higher values of ϵ produced functions that initially considerably exaggerated the value of the same set of states, but then decayed and convergence to a reasonable value function was achieved. This is more in agreement with recent speculation by Sutton and Barto (1998) that adhering closely to the actual policy learnt so far may prevent divergence in Q-learning. Even with the ϵ value used in the experiments it is possible that the process would not continue to diverge, but eventually settle down. However in the experiments this did not occur after observing the process for many millions of actions. Whether or not using an ϵ-greedy policy with a small ϵ eliminates the problem is an open question. The solution investigated in this thesis is to change the function approximation scheme so that it does not produce overshoot.

4.3.2 Using Smoothing Splines to Eliminate Overshoot

If a smoothness constraint could be added to the function approximation scheme this should remove overshoot. One extension to standard spline approximation is the idea of a smoothing spline. Instead of just minimising the least squares error, an additional functional term is added. This term, often called the stabiliser, penalises functions that are not sufficiently smooth (Terzopoulos, 1986). What is meant by smoothness dictates the type of stabiliser used. One commonly used example is the membrane spline which intuitively acts in two dimensions like an elastic sheet, in one dimension like an elastic band.

$$D(\hat{f}) = \int \left(\hat{f}(x) - f(x)\right)^2 dx + \omega \int \left(\frac{d\hat{f}(x)}{dx}\right)^2 dx \qquad (4.9)$$

The membrane spline discourages high gradients through the second term in equation 4.9. The constant ω, often called the tension, controls which term dominates the functional. If the tension is zero then the approximation is a least squares fit. As the tension is increased the effect of the stabiliser becomes larger, smoothing the function. As the tension tends to infinity the function tends towards the default form a constant value, the average of the input function.

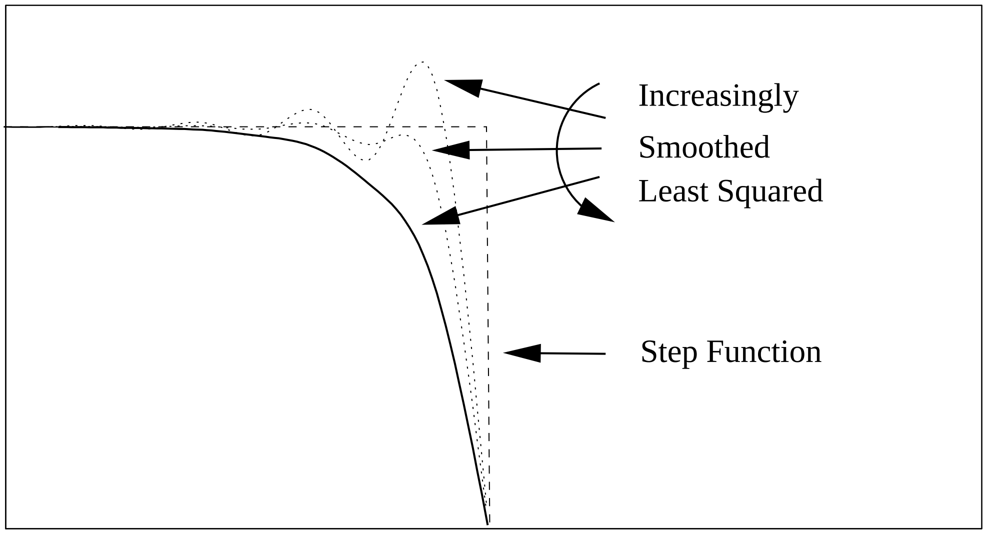

Figure 4.9: Preventing Overshoot

Figure 4.9 shows the result of progressively increasing the smoothing of the least square fit shown in figure 4.6. The bold continuous curve has just enough smoothing to eliminate overshoot, no point on the curve goes above the maximum value of the step function. This smoothing is the result of applying the membrane stabiliser with the "right" value of tension, i.e. the right value for ω in equation 4.9. The trade off necessary to produce this smoother function is to increase the square error of the fit of the spline to the input function.

Section 4.1.1 discussed a matrix method for calculating the least squares fit of the basic spline. The matrix equation 4.7 is produced by differentiating with respect to the coefficients of the basis functions and equating the results to zero. When a smoothing spline is used an additional matrix $\mathbf{H}$ is added representing the stabiliser function, as shown in equation 4.10 .

$$(\omega\mathbf{H} + \mathbf{V^T V})\mathbf{C} = \mathbf{V^T D} \quad where \quad \mathbf{H}_{ij} = \int \left(\frac{d\phi_i(x)}{dx}\frac{d\phi_j(x)}{dx} \right) dx \tag{4.10}$$

This thesis proposes the simple update rule of equation 4.11 to implement the smoothing term with iterative updating. The idea is to replace the exact solution of the integral with Monte Carlo integration (Dahlquist & Bjorck, 1969). Rather than just updating the coefficients due to the error of fit between the function and the data, a similar rule will account for the smoothing term. This rule adjusts the coefficients such as to minimise the size of the differential, as shown in equation 4.11. In this case the data point is selected from a uniform random distribution across the state space. This rule is applied concurrently with the Widrow-Hoff rule of equation 4.8.

$$\Delta c_j = \alpha\omega \frac{d\phi_j(x)}{dx} \left(-\frac{d\hat{Q}(x)}{dx} \right) \tag{4.11}$$

4.3.3 Finding the Right Smoothness Constraint

This section details the analysis that determines how to select the right value of tension for the membrane stabiliser. To this end, the process of finding the least squares solution for the smoothing spline is re-represented as the convolution of the input function with a kernel. The same analysis should apply to discrete data using a weighted average as opposed to an integral. The kernel integrates to one. But for small values of tension it will be sometimes negative, the solid line in figure 4.10. It is this property which causes the overshoot. By increasing the tension, a point can be found where this kernel is always non-negative, discussed further at the end of this section, and thus will not create overshoot, the dashed line in figure 4.10.

The kernel is constructed in a two stage process associated with the two terms of the smoothing spline of equation 4.9. Rather than finding the minimum by solving the matrix equation of section 4.3.2 an equivalent method is to first smooth the data and then carry out the least squares fit. Figure 4.11 shows the result of smoothing

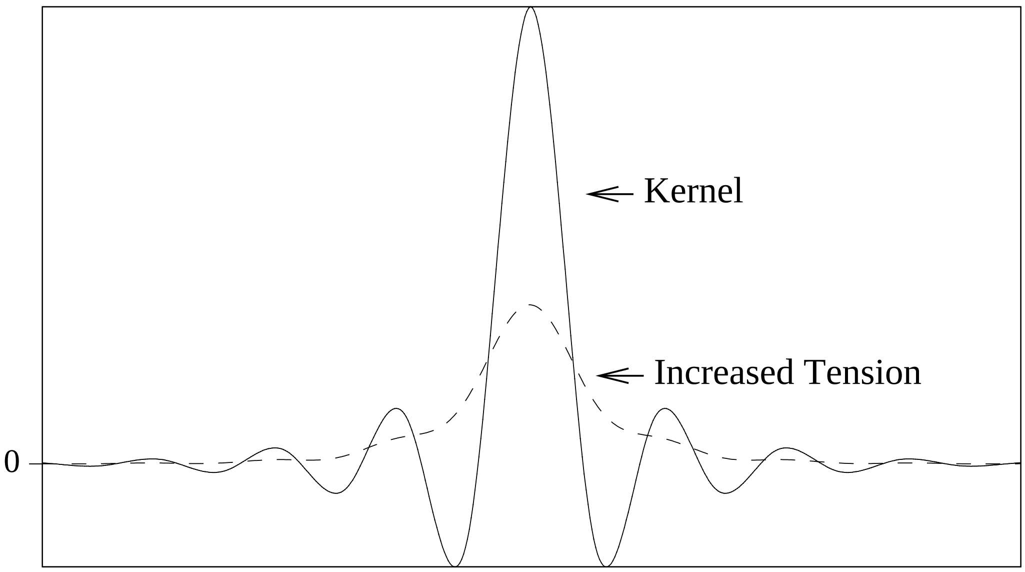

Figure 4.10: The Kernel

the step function then fitting a cubic B-spline. The fit to this smoothed function, using seventeen basis functions, is relatively tight. The shaded area in figure 4.11 has been expanded to highlight the error, the solid line being the spline approximation.

The first stage in the analysis is to determine the form of the function used to smooth the data. The analysis uses the connection between the smoothing spline and the solution of a partial differential equation. The membrane spline determines the least squares approximation of the solution $U(x)$ of an Euler equation, the differential equation 4.12. The constant k^2 equates to the reciprocal of ω used in the smoothing spline and $f(x)$ is called the forcing function, in this case the input function.

$$-\Delta U(x) + k^2 U(x) = f(x) \tag{4.12}$$

There are many ways of finding the exact solution of partial differential equations, the one of interest here is using Green's functions. A Green's function, $G(x)$ in equation 4.13, is an impulse response solution of the differential equation.

$$-\Delta G(x) + k^2 G(x) = \delta(x) \tag{4.13}$$

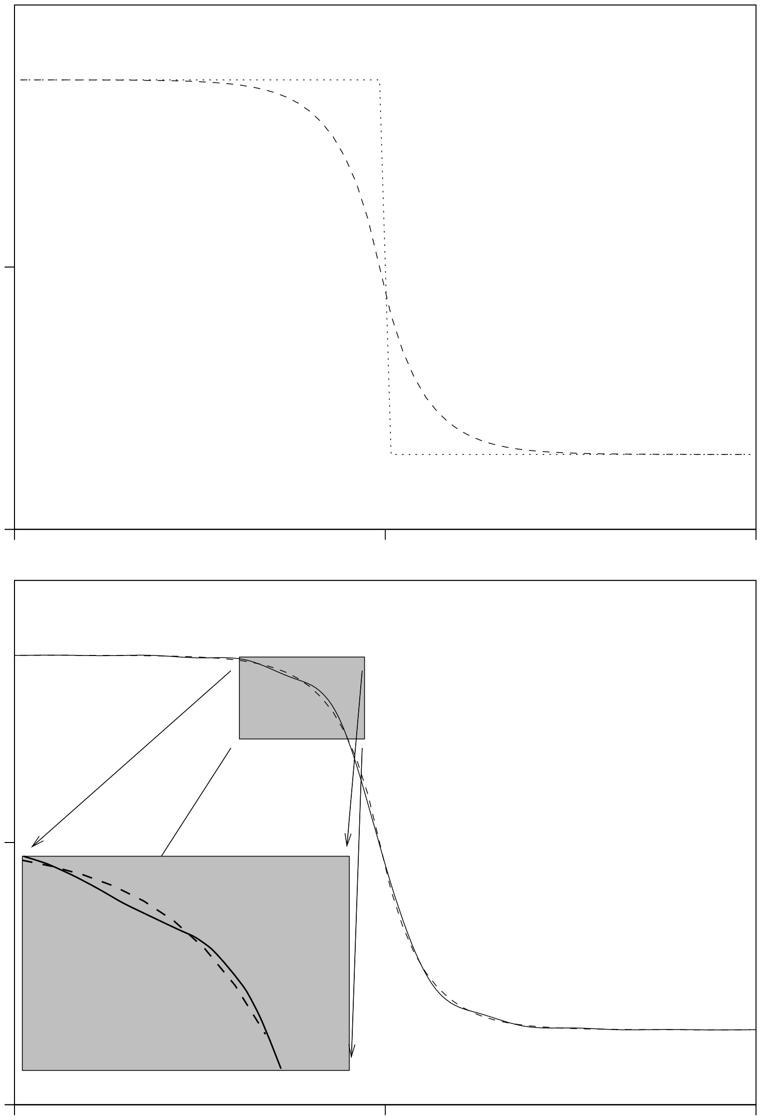

Figure 4.11: Smoothing Followed by Least Squares Fit

This is the needed smoothing function. To produce the membrane spline approximation, one convolves the input function with the appropriate Green's function, equation 4.14, thus solving the Euler equation, and then minimises the squares fit to the output.

$$U(x) = \int_{-\infty}^{\infty} G(x-\eta) f(\eta) d\eta \tag{4.14}$$

In partial differential equation problems, the boundary conditions must be considered. With the conditions discussed in section 3.1.1, one can imagine the function reflecting back and forth between a pair of mirrors situated at the edge of the state space. A virtual function is produced, stretching to infinity in both directions as shown in figure 4.12. This same effect is realised in higher dimensions with mirrors on the faces of a hypercube.

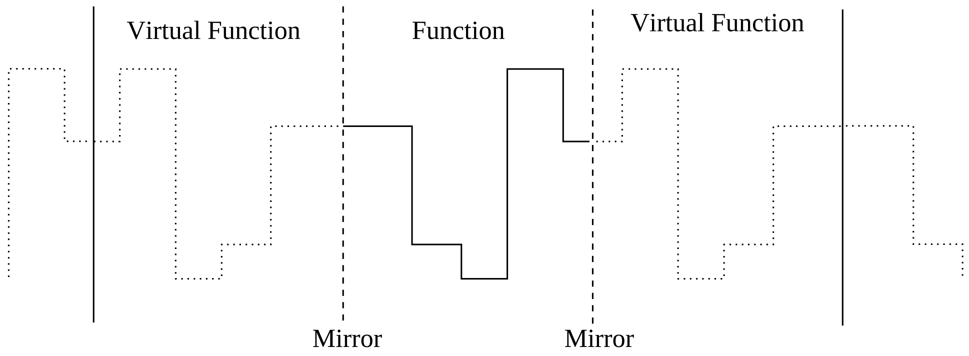

Figure 4.12: Reflecting the Function

For the purposes of this analysis we can consider smoothing this infinite function allowing the Green's functions for free space to be used. In actual use the smoothing function close to the boundary will have a different form, but will still exhibit the necessary characteristics to prevent overshoot. Equation 4.15 shows the Green's functions, for free space, for one and two dimensions (Stakgold, 1979), K_0 is the modified Bessel function of the second kind, order zero (Press, Teukolsky, Vetterling, & Flannery, 1993).

$$G(k,x) = \frac{e^{-k|x|}}{2k} \qquad G(k,x,y) = \frac{K_0(k\sqrt{(x^2+y^2)})}{2\pi} \tag{4.15}$$

There is also an alternative method for finding the least squares fit. Had the original bases been orthogonal, the coefficient of any one basis could have been found by integrating it with the input function. The same process can still be carried out with B-splines even though they are not orthogonal. Instead of integrating with the original basis, a special basis function called the dual basis is used. The dual basis for B-splines introduced by Chui (Chui & Wang, 1992; Chui, 1992; Sakakibara, 1994) spans the same space as the B-splines. So the dual basis itself can be represented as a sum of B-spline bases. Reconstructing the function within any region needs the sum of m^d bases, where m is the order of the B-spline and d the dimension. Equation 4.16 shows this for one dimension, ψ is the dual basis.

$$\hat{f}(\eta) = \sum_{i=1}^{m^1} \left(\int_{-\infty}^{\infty} \psi_i(x) f(x) dx \right) \phi_i(\eta) \tag{4.16}$$

The next stage is to combine equation 4.16 with 4.14, rearranging the terms, reordering the integrals, and expanding the dual basis in terms of the original basis.

$$\hat{f}(\eta) = \int_{-\infty}^{\infty} \underbrace{\left(G(k,x) * \left(\sum_{j=1}^{m} \sum_{l=1}^{n} a_{l-j} \phi_{l-j}(x) \phi_j(\eta) \right) \right)}_{Kern(\eta,k,x)} f(x) dx \tag{4.17}$$

The resulting kernel, shown in equation 4.17, is sometimes negative. All that is necessary now is to find the value of k that smooths the function sufficiently to make it always non-negative. Such a value must exist, at least in the limit. The unsmoothed kernel integrates to one and the Green's function tends to a straight line as k decreases. Thus eventually the convolution will produce a positive constant value. The dual basis is of infinite width, but decays exponentially and a truncated version is used for the analysis. It is possible that the true kernel would still have negative regions but these would be bounded by a small value. The convolution can be solved in the Fourier domain and followed by an inverse transform. This has been done for one dimension. Its complexity, however, makes it difficult to solve. Instead, at present, the correct value is found by using a discrete convolution and a binary search over the values of k. As the basis functions are positioned on a grid, the kernel depends on the relative position within a cell. So a search is also done over the values of η.

The right value of tension that prevents overshoot results in a kernel that is an averager. Gordon (1995b, 1995a) has proved that such approximators do not diverge in reinforcement learning. This work shows how the important family of B-splines can meet Gordon's criterion. As in Gordon's analysis there are a number of restrictions on the sample points. Although the analysis is based on a continuous function it should be appropriate for gridded data. It should also extend to randomly distributed data by carrying out a form of Monte Carlo integration, as in equation 4.8. At present this assumes a uniform distribution of data unlikely to be realised in practice. Additional splines of the same degree could be used to estimate the true distribution of the data. This would be used to normalise the data over the state space to prevent overshoot. Using random distributions of data would, unfortunately, break the connection to Gordon's proof and therefore non-divergence would not be guaranteed.

4.3.4 Preventing Divergence

The following experiment demonstrates the effectiveness of the stabiliser in preventing divergence. The experimental set up is the same as was used to produce the problem of divergence shown in section 4.3.1. The dimensions of the state space of the "car on the hill" problem are the position (range $\pm$ 1) and the velocity (range $\pm$ 2) of the car. A reward of one is received if the car gets sufficiently up the hill, beyond position 0.6, within a limited velocity ± 0.4. The discount value γ is set to 0.99. Only two actions are used, a forward and a reverse acceleration.

Figure 4.13 shows the value function produced by using a stabiliser with $k = 21.6$, the value determined by the analysis of section 4.3.3. The update procedure used the Widrow-Huff rule with $\alpha = 0.1$ applying smoothing using equation 4.8. As before a cubic B-spline with a grid of 33 by 33 basis functions was used. Although the use of the stabiliser prevents divergence, the resultant function is overly smooth. It is noteworthy however that even with this degree of smoothing the average distance to goal is only about ten percent longer than that using the box spline (33.3 steps). The

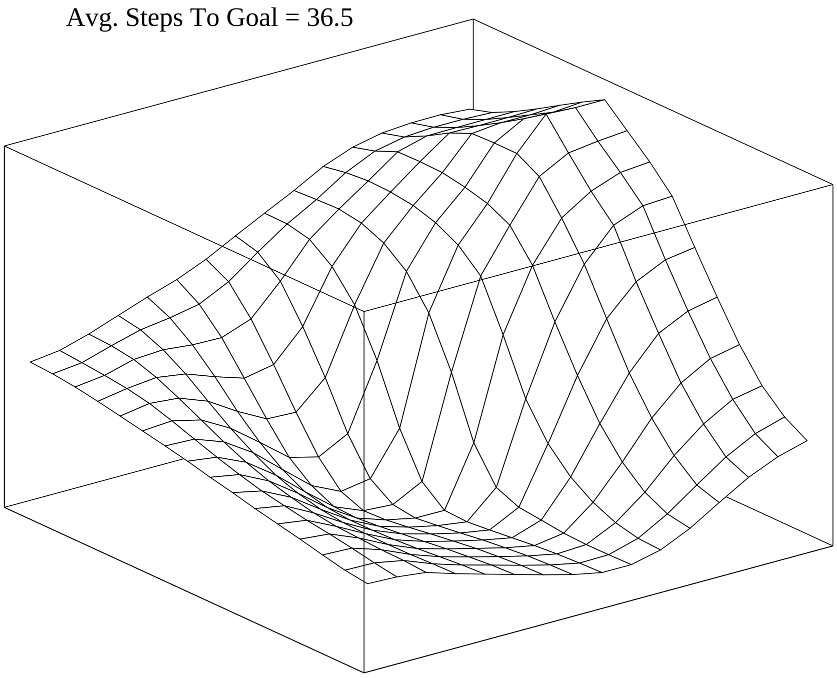

Figure 4.13: Convergent Approximation: Cubic Splines

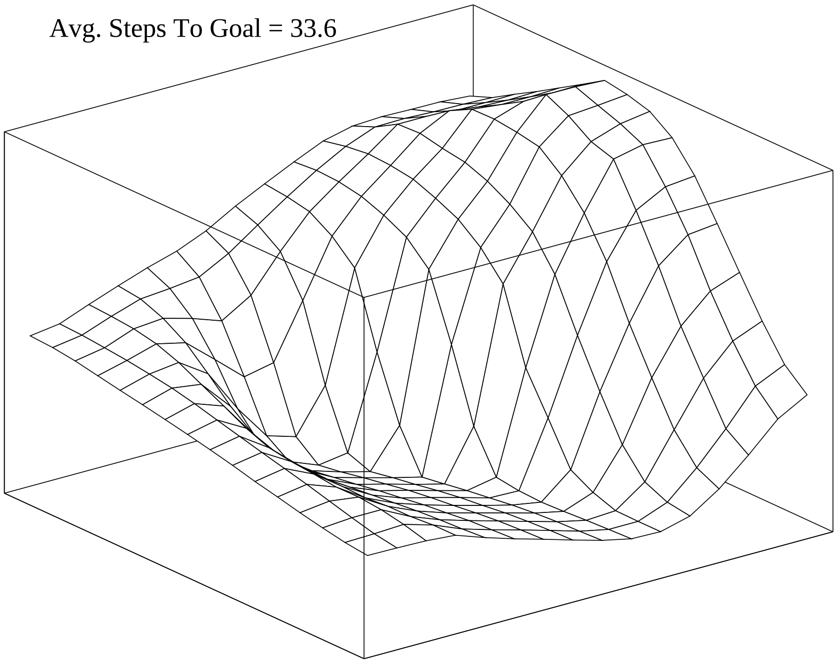

Figure 4.14: Convergent Approximation: Linear Splines

analysis of section 4.3.3 is by no means limited to cubic splines, it was also applied to linear splines. Figure 4.14 shows the value function produced using linear splines ($k = 36.6$). Here the fit is much better and the average distance to goal is almost identical (33.6 steps) to the box spline.

4.4 In Summary

This chapter has discussed reinforcement learning and other alternative low-level learning methods as ways of generating a control function. It has shown how using higher order B-splines can reduce the number of basis functions required to represent the control function while maintaining the same average number of steps to the goal. It has also shown how the addition of a smoothing term can prevent divergence. Unfortunately the addition of the smoothing term, removes the advantage of the better fit of higher order splines. It may be that further research will characterise better when divergence occurs. If, as Sutton and Barto (1998) suggest, using very greedy policies prevents divergence, smoothing may only be necessary when more exploration is needed. Much research has used "Softmax action selection" (Sutton & Barto, 1998) where the amount of exploration is progressively reduced over time. Smoothing could be used early on in the learning process and then reduced in line with amount of exploration. The work described in the rest of this thesis is somewhat orthogonal to this issue and the simple box splines are used.

Other work in reinforcement learning with function approximators might benefit from this analysis of smoothing splines. The ideas in this chapter are built around basis functions that are translates of a canonical basis and should be readily applicable to wavelets. Some research within the neural network community has investigated stabilisers (Bishop, 1992; Girosi, Jones, & Poggio, 1995; Poggio & Girosi, 1990, 1989). Certainly where such networks use bases that are translates of a canonical function, the analysis carried out in this chapter should be relevant. If, for instance, spline or wavelet networks are used as function approximators in reinforcement learning the same potential for divergence will exist and the same solution will be appropriate.

Chapter 5

Task Partitioning

This chapter addresses partitioning the value function produced by reinforcement learning into regions representing the solutions to subtasks. In the context of the architecture discussed in section 1.2 this is the transition from the subsymbolic level to the symbolic level as shown in figure 5.1. The main thesis contribution discussed in this chapter is to show how an edge extraction algorithm from vision processing has been adapted to generate the partition. The algorithm, called a snake, has been significantly extended to improve its robustness in this application. These extensions should also be of value in its more traditional vision processing role.

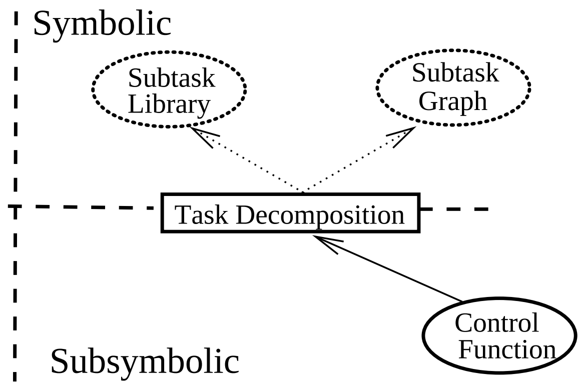

Figure 5.1: Task Decomposition

In vision processing, the snake determines the boundary of an object by locating its edges in an image. An edge is indicated by sharp changes in image intensity.

Typically due to less than ideal lighting conditions and image quality, the intensity change will vary along the boundary and perhaps in places be missing altogether. The snake, being a smooth closed curve, can span the gaps of little or no intensity change and thus approximate the boundary of the object.

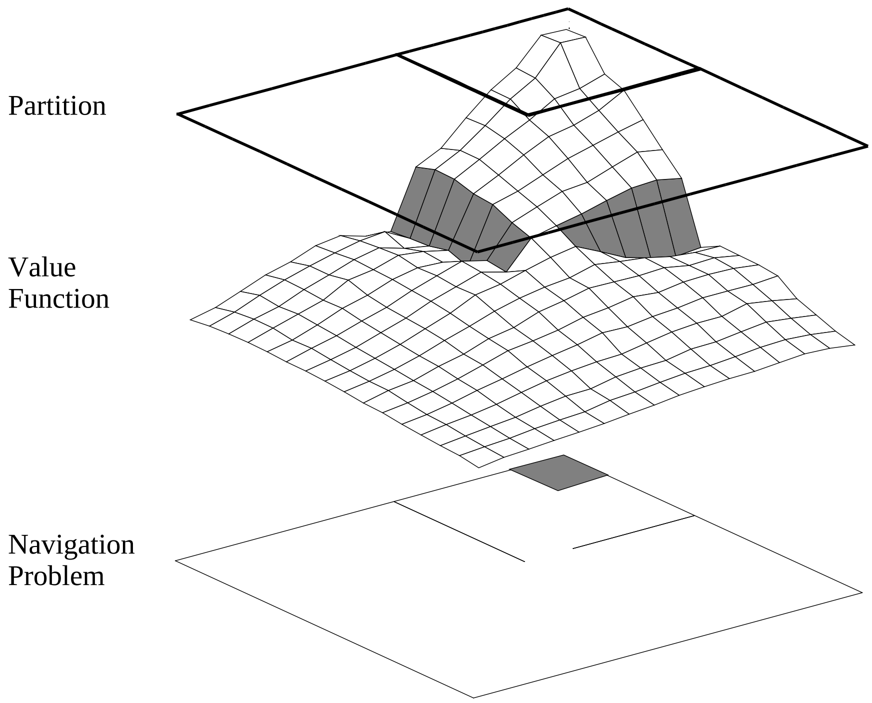

Figure 5.2: A Complete Partition

The bottom of figure 5.2 shows the robot navigation problem of section 3.2.2 and the value function produced after 3000 exploratory actions. In this thesis the sharp changes the snake locates are in the value function, shown as the shaded regions. As in vision processing the size of the change varies over the function disappearing altogether at the doorways. The top of figure 5.2 shows the partition produced by the snake. Each region represents a solution to a particular subtask: the small square the subtask of getting to the goal from within the inner room, the larger L-shape the subtask of getting to the door of the inner room.

5.1 The Snake Approach

This chapter begins with an intuitive discussion of the snake, a more detailed mathematical treatment will be given in section 5.3. The top part of figure 5.3 is the gradient of the value function shown in figure 5.2. The system has added a gradient around the border to represent the state space boundary. To locate the features a curve is found that lies along the ridge of the hills. On the bottom of figure 5.3 the dashed lines are contour lines for the small inner room as indicated.

The bold lines at the bottom of figure 5.3 are the snake at different stages of the process. The snake is first positioned approximately in the centre of the room, the innermost circle. It is then expanded until it abuts on the base of the hills. Now to simplify the exposition, we can imagine that the snake consists of a number of individual hill climbers spread out along the line representing the snake, indicated by the small white circles. But instead of being allowed to climb independently their movement relative to each other is constrained to maintain a smooth shape. In this thesis, when the snake reaches the top of the ridge, it is further constrained to be polygon - in this instance a quadrilateral - the outside dark line in figure 5.3. At this point it will tend to oscillate around an equilibrium position. By limiting the step size the process can be brought into a stationary state.

5.2 Three Extensions to the Snake Approach

Applying the snake to the problem of feature extraction brought to light a number of weaknesses in the basic approach. This section introduces three extensions that significantly improved the robustness of feature extraction. It is expected that these extensions should also prove useful when extracting the edges of an object in an image.

The first extension affects the direction the snake moves when hill climbing the gradient. In normal hill climbing, each step is taken in the direction of steepest ascent, the step size being determined by the size of the differential. Roughly this translates into forces at points along the body of the snake. Each force points in the direction of steepest ascent locally but interacts with other forces through the various shape

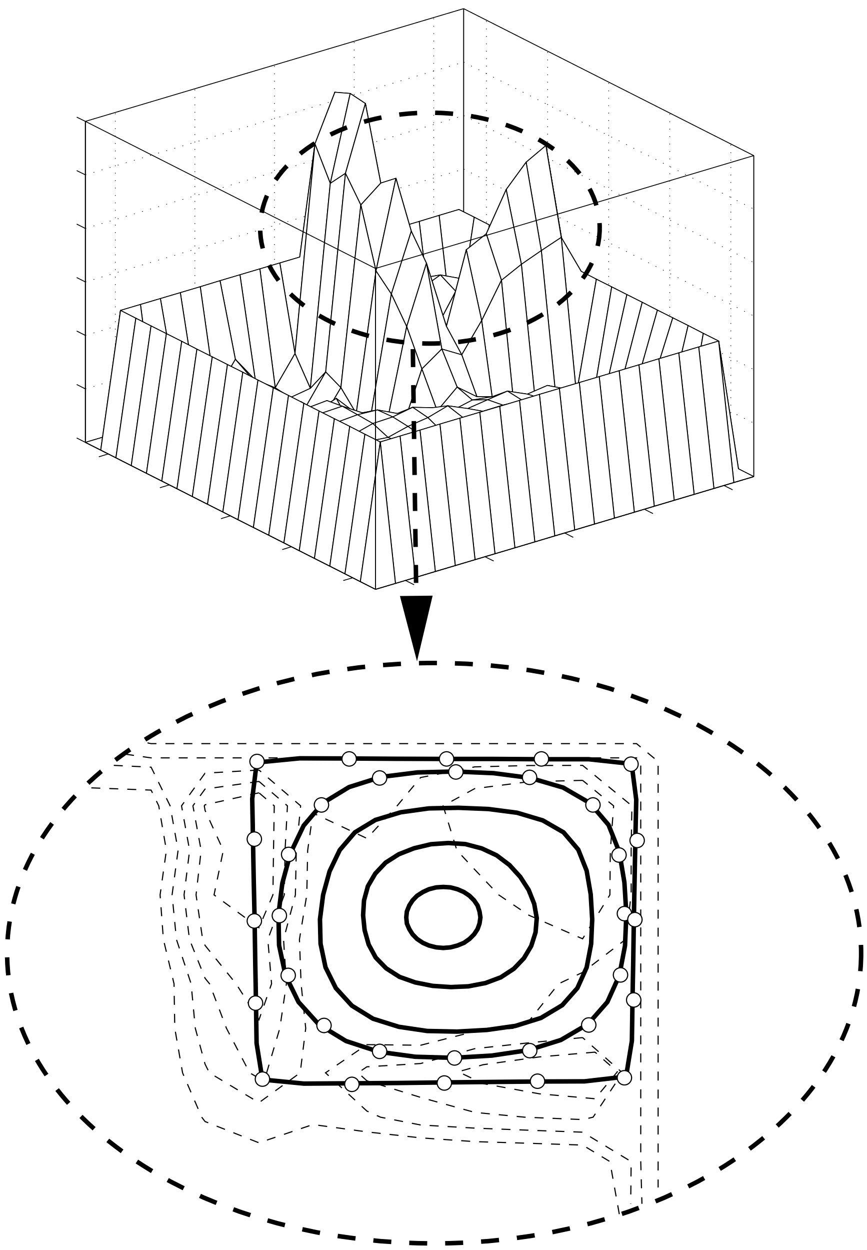

Figure 5.3: Fitting the Snake

constraints. Looking at the gradient function and contour lines of figure 5.3, there is a steep slope leading to the top of each ridge. But there is also a significant slope along each ridge away from the doorway towards the boundary of the state space. Thus the force on a single point on the body of the snake is not directly towards the top of the ridge but turned towards its apex, as indicated by the bold black arrow named "Steepest Ascent" on the left hand side of figure 5.4.

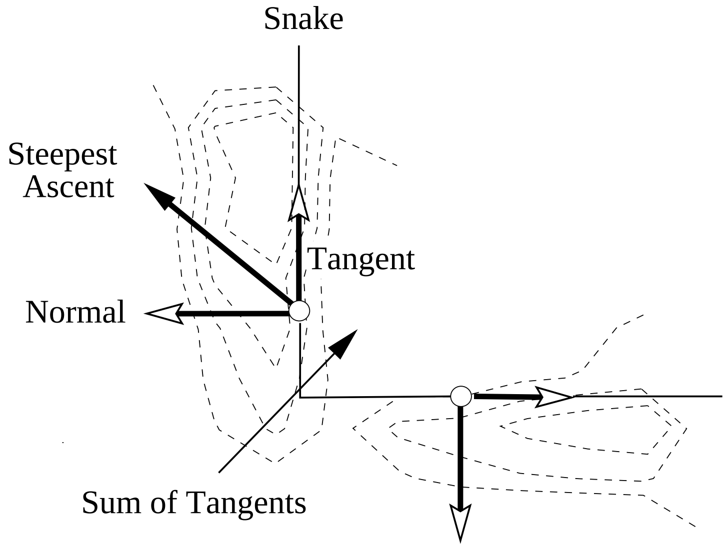

Figure 5.4: Controlling Forces on The Snake

This force can be broken into two components with respect to the snake: normal and tangential. The latter force acts along the body of the snake. Once the shape is constrained to be a quadrilateral this will cause the relevant side to shrink. This effect will be partially counteracted by the force towards the top of the ridge on the adjacent side of the quadrilateral. But the net result will be a shrinking of the two sides associated with the ridges inwards until the forces are balanced. This will push the corner of the quadrilateral near the doorway inwards. The exact direction is determined by the vector sum of the tangent forces as indicated by the thin black arrow in figure 5.4. In an extreme case this might cause the snake to collapse into something close to a triangle. But the more likely outcome will be just a degradation of the accuracy of registration of the ridges.

In earlier work (Drummond, 1998) this degradation of accuracy was prevented by restricting the snake to a rectangular shape. But with the weakening of this constraint to more general polygons this effect again became a problem. The problem is addressed by removing the component of the force tangential to the snake. Then hill climbing is always in the direction of the normal. This also proved to be advantageous at other stages in the snake's evolution. It does not significantly restrict the motion of the snake, all that is being removed is the component along its body. Thus it mainly prevents the stretching and shrinking of the snake due to the gradient.

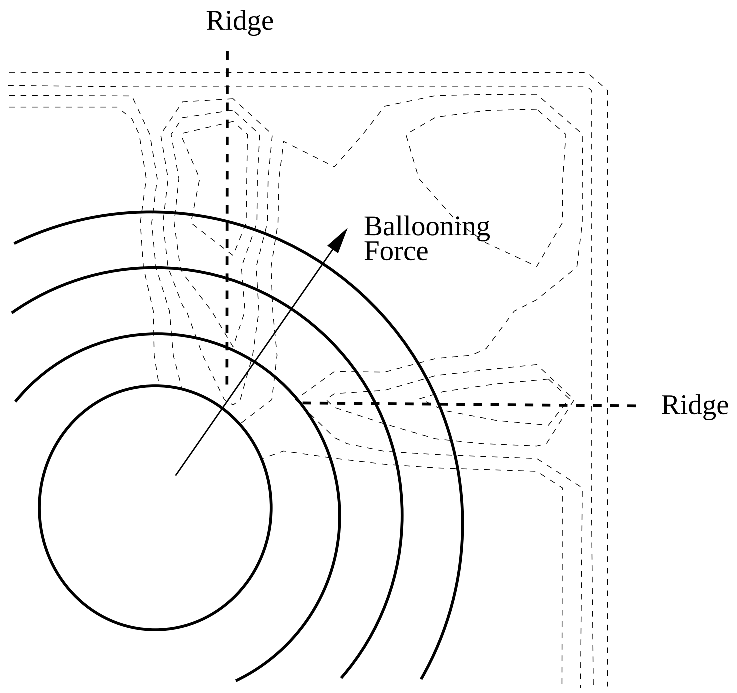

Figure 5.5: Using the Ballooning Force

The second extension controls the way the snake is expanded to reach the base of the hills. In earlier work (Drummond, 1998) a ballooning force was used as introduced by Cohen and Cohen (1993). But problems arose when extending the system to deal with more general shapes than rectangles, such as the outer L-shaped room shown in figure 5.2. The ballooning force expands the snake in directions normal to its body. One deleterious effect of this is if it contacts a sharp external corner such as that of the inner room, it tends to push the snake through the corner. This can be seen in figure 5.5, the bold solid lines are the snake, the bold dashed lines are the ridges. If we imagine starting off with a circular snake in the middle of the L-shaped outer room by the time it reaches the walls of the inner room the sides of the snake are roughly perpendicular to the ridges. Thus there is little to restrain the expansion of the snake and it passes completely through the walls of the inner room.

The approach adopted here is analogous to the flow of mercury. If we imagine starting somewhere in the middle of the L-shaped room and progressively adding mercury it would tend to fill up the lower regions of the valley first and reach the bases of the hills at roughly the same time. The analogy of mercury is used as it has a high surface tension preventing it from flowing through small gaps in the edges associated with doorways. The valleys formed by the gradient tend to be flat in the middle and only start to rise close to the walls of the room and the boundary. If the function was smoothed the width of the base of the hills would increase. Smooth it enough and the bases would meet, producing a bowl associated with each room. Then as the mercury is added it will tend to take up the approximate shape of the room a long time before it reaches the ridges of the hills.

Rather than using the gradient itself, a function including the second differential is used. This proved to be more stable than the actual gradient and will be discussed in detail in section 5.3. Figure 5.6 shows this function after thresholding. It is then smoothed with a kernel closely approximating a Gaussian, producing the bowls shown in figure 5.7. An advantage of using Gaussian smoothing is that it tends to fill the gaps in the features associated with doorways. In this example the smoothing has almost completely obscured the presence of the doorway, although this is generally not the case. But in all cases this reduces the pressure to flow through the doorways.

The mercury-like effect is achieved by varying the force normal to the body of the snake according to the height difference with the average height of the snake. Thus points along the snake which are higher than average (further up the side of the bowl) tend to get pushed inwards, those lower pushed outwards. The surface tension of the mercury is produced by various smoothing constraints on the snake. The process begins with the snake initialised as a small circle at the minimum of one of these bowls. This is shown as the circle in the middle of figure 5.8, where the dashed

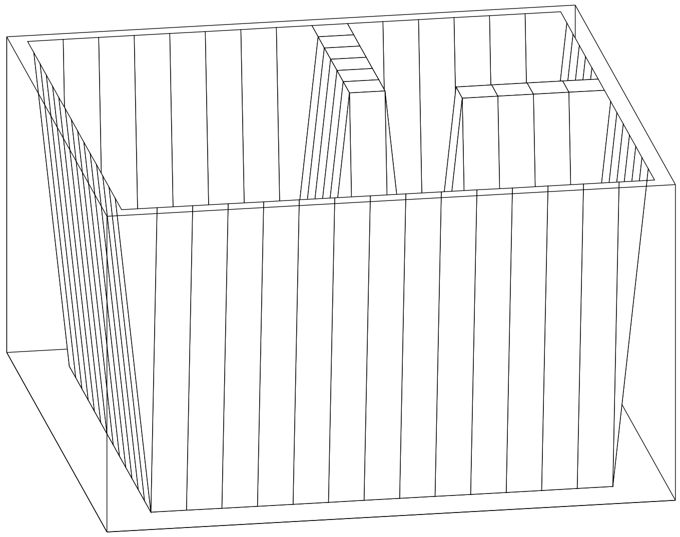

Figure 5.6: Thresholded Features

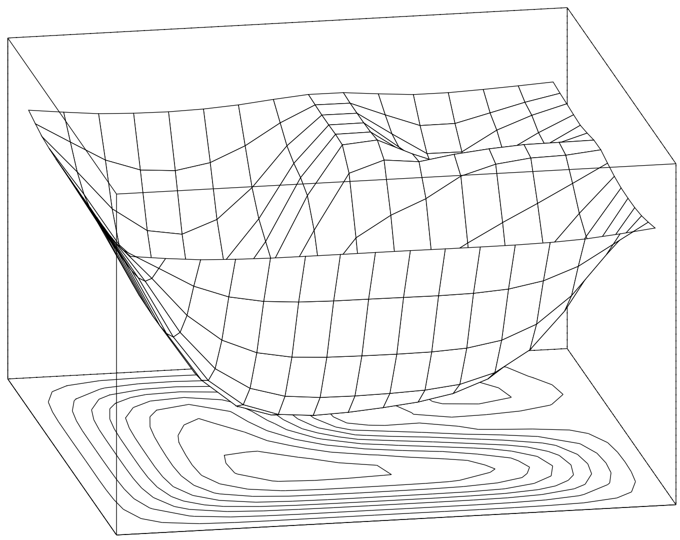

Figure 5.7: Smoothed Function

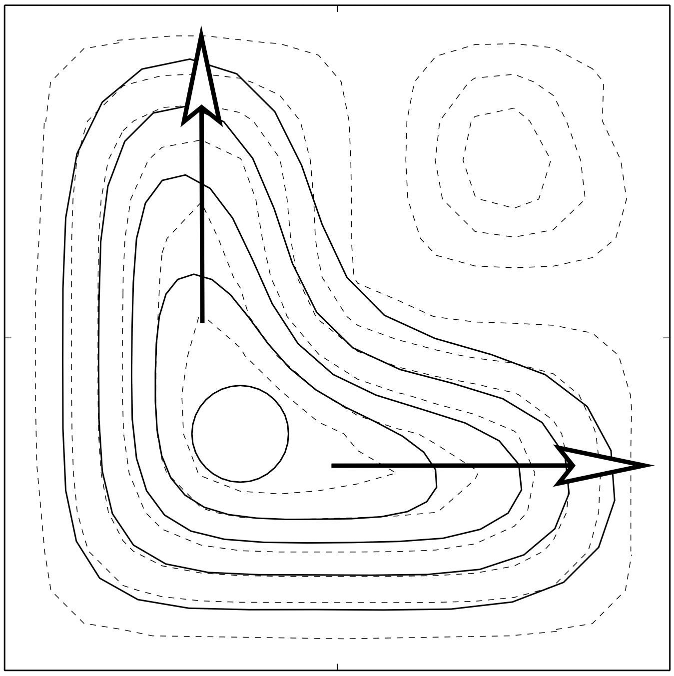

Figure 5.8: Mercury Flow

lines are the contour lines of the bowl. It then flows outwards so as to follow the contour lines. At the centre the contour lines are roughly triangular. Moving closer to the ridges the contour lines take on progressively more of an L-shape. The distance between contour lines along the arms of the L tends to be more widely separated. Thus the largest component of the flow is along these arms in the direction of the arrows in figure 5.8.

The third extension limits changes in the shape of the snake as it expands from its initial position to reach the base of the hills. The smoothness constraints on the snake that give the mercury-like properties prevent the snake flowing through the gaps associated with the doorways. But even this proved insufficient if the width of the rooms and the width of doorways were of similar sizes. In figure 5.9 looking at the "room" on the left hand side of the configuration space of the robot arm, the "doorway" and the "room" at the top are of similar width. Increasing the surface tension of the mercury sufficiently to prevent flow through the doorways also prevents the flow to the top of the room.

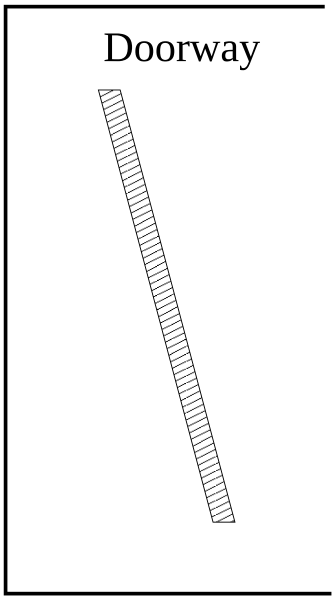

Figure 5.9: Arm "Room"

The solution is to limit the amount the snake can change its shape as it grows. This would be undesirable in the original balloon approach as the shape of the snake could not be determined until it reached the ridges. In the mercury approach the snake makes smaller incremental changes in shape, progressively better approximating the room as it moves closer to the ridges. Constraining the snake to a smooth closed curve penalises the snake the more it deviates from a circle. To produce the closed curve a parameterised form of function representation is used, $(x(s), y(s))$. A circle is produced by two components, $(sin(s), cos(s))$ and maximally smoothing these functions individually produces constant values. Thus overall the pressure is towards a circle of zero radius. The alternative used here is to reduce this smoothing constraint and add a second constraint that penalises large incremental shape changes. In this example the snake takes up the roughly triangular shape of the room quickly and the flow can continue to the top of the room as the general shape is maintained.

5.3 Details of the Snake Approach

This section gives a more detailed and mathematical treatment of the snake approach outlined in the previous sections. The basic snake model is taken from two papers (Cohen & Cohen, 1993; Leymarie & Levine, 1993). It has been modified to include the extensions discussed in section 5.2. Throughout this section numerical values for the various parameters will be given. These have been obtained empirically and have been reasonably robust across different tasks and different domains. Snake research is a very active area, it is probable that algorithms for the automatic setting of parameters will be forthcoming.

Rather than using the more usual discrete representation, in this thesis the snake is represented by cubic splines. A parameterised form is used. In two dimensions this is given by $\hat{f}(s) = (x(s), y(s))$ where $x(s)$ and $y(s)$ are individual splines giving the x and y coordinates associated with a variable s along the body of the snake. One way to visualise the snake is as the interaction between two sets of forces: internal and external. The internal forces are due to the smoothness constraints, the external forces to hill climbing and mercury flow. An imbalance of forces produces a potential energy. The dynamics of the snake are described by a partial differential equation minimising this energy and converging to a point where the forces balance.

5.3.1 Internal Forces

There are two internal forces: one trying to smooth the snake, the other trying to maintain its shape. Both are derived from a stabiliser function but instead of the membrane stabiliser discussed in chapter 4 an alternative function is used. This stabiliser produces the thin plate spline. In two dimensions it intuitively acts like a thin metal plate, in one dimension like a thin metal band. The one dimensional version is the mathematical equivalent of the wooden spline originally used as a draughting tool. Rather than penalising the first differential, the thin plate spline penalises the second differential as shown in equation 5.1. The factor ω_{tp} will determine the size of the penalty with respect to other terms on the left of the plus sign, for the thin plate spline this is typically the least squares error. For an open curve minimising this energy functional produces a straight line, for a closed curve, as used here, a circle of zero radius.

$$E_{tp}(\hat{f}) = \cdots + \omega_{tp} \int \left(\frac{d^2 \hat{f}(s)}{ds^2} \right)^2 ds \tag{5.1}$$

The coefficient ω_{tp} can vary over the length of the snake. This idea is used to constrain the snake to be a polygon when it has reached the ridges. The coefficient ω_{tp} is set to zero at the corners (where the normal passes through $\frac{(2n+1)\pi}{4}$ where $n = 0 \ldots 3$) and to 15 in the intervals separating them. Where the coefficient is zero the snake bends easily; where the value is higher the snake is rigid. This produces a polygon which is flexible at its vertices. To encourage the snake to take up this shape before changing the coefficient a force is added pushing knots towards intervals of higher than average curvature to allow more flexibility at these points.

The coefficient ω_{tp} can also vary as the snake grows. This is done prior to adding the polygonal constraints. Ideally the number knots of the snake should be in a fixed ratio with its length (Cohen & Cohen, 1993) and they should be evenly spaced. One way to achieve the former is to add more basis functions as the snake grows. For simplicity this is not done at the moment but the initial value $\omega_{tp} = 8.0$ is reduced proportionately to the snake's length. Reducing the influence of the stabiliser gives the spline more degrees of freedom (Ramsay & Heckman, 1996). A force is also added at each knot which encourages segments to expand which are shorter than average and to shrink if longer than average to maintain the necessary separation.

Equation 5.1 penalises the spline if the second differential is greater that zero. An alternative is to penalise the spline for the difference in second differential when compared to previous iterations. The constant $\omega_c = 512.0$ is used as shown in equation 5.2. The first differential determines the length of the snake. As the differential is a linear operator the second differential will increase with the length of the snake. To maintain the shape it is penalised for the difference to the previous iteration scaled by the ratio of their lengths, the factor R.

$$E_c(\hat{f}) = \cdots + \omega_c \int \left(\frac{d^2 \hat{f}^t(s)}{ds^2} - R * \frac{d^2 \hat{f}^{t-1}(s)}{ds^2} \right)^2 ds \tag{5.2}$$

5.3.2 External Forces

There are two external forces: one towards the ridges of the hills, the other to take up the approximate shape of the room. The ridges that the snake locates are strictly speaking the maxima of the magnitude of the gradient vector $|\nabla Q_{max}|^2$. The magnitude can be expanded into terms for each dimension, as shown in equation 5.3 and is just the sum of the squared values of the gradient for each dimension. Higher dimensional problems require additional terms at the right of the equation.

$$-\left|\nabla Q_{max}\right|^2 = -\left[\left(\frac{\partial Q_{max}}{\partial x}\right)^2 + \left(\frac{\partial Q_{max}}{\partial y}\right)^2 + \left(\frac{\partial Q_{max}}{\partial z}\right)^2 + \cdots\right] \tag{5.3}$$

The snake hill climbs this magnitude. Finding the maxima can be solved by an iterative process which treats each dimension separately (Cohen & Cohen, 1993). The step size is proportional to the differential of the magnitude of the gradient vector in each dimension. Thus if only two dimensions are considered the step for the x direction is given by equation 5.4. A similar equation is used for the y direction. Each additional dimension requires one extra equation and one extra term in each equation.

$$-\frac{\partial\left|\nabla Q_{max}\right|^2}{\partial x} = -2\left[\left(\frac{\partial Q_{max}}{\partial x}\right)\left(\frac{\partial^2 Q_{max}}{\partial x^2}\right) + \left(\frac{\partial Q_{max}}{\partial y}\right)\left(\frac{\partial^2 Q_{max}}{\partial x \partial y}\right)\right] \tag{5.4}$$

Equation 5.4 contains both first and second differential terms. If the reinforcement learning function is discrete a finite difference approximation could be used. But even when using finite differences there is merit in interpolating between the discrete values (Cohen, 1991). This prevents the snake oscillating between two adjacent discrete values. The approach taken here is to represent the Q_{max} function by a spline that is an order at least quadratic. Thus the first and second differentials can be measured directly. If learning takes place using a spline of lower order then a quadratic spline is fitted to the value at the centre of each basis function. To prevent the problem of overshoot which can produce false edges in the function, the membrane spline of chapter 4 is used.

In the basic snake model there is a single external force due to the gradient vector as shown in equation 5.3. Here, as in Cohen and Cohen (1993), to produce the external force $F(\hat{f})$ another factor is added, the second term on the right hand side of equation 5.5. But instead of it being a constant, it is a variable dependent on the height of the snake at a different points along its body. Both terms on the right hand side of equation 5.5 are applied in a direction normal to the body of the snake, as indicated by the term $\overrightarrow{n}(s)$.

$$F(\hat{f}) = \left[\nabla\left(-\left|\nabla Q_{max}(\hat{f})\right|^2\right) + M(\hat{f})\right]\overrightarrow{n}(s) \tag{5.5}$$

The function $M(\hat{f})$ is obtained from a quadratic spline producing the bowls associated with the rooms as discussed in section 5.2. The Euclidean sum of the force shown in equation 5.4 and the same for the y direction is determined on a uniform grid across the state space, thresholded and then the quadratic spline fitted. This spline uses the thin plate stabiliser with an ω_{tp} of 0.5 giving roughly Gaussian smoothing (Poggio, Voorhees, & Yuille, 1985). The values used to produce this function are weighted. Values close to one are given weights of 500, lower values a weight of 1. This prevents the sides of the bowls collapsing under smoothing. The resultant function is sampled at points along the body of the snake. The force $M(\hat{f})$ in equation 5.5 is proportional to the difference in height of the bowl at each of these sample points from the average.

To maintain a flow of the snake, its length should grow in roughly equal steps. This is achieved by adding a step to the average height to produce an $M(\hat{f})$ that tends to expand the snake. The step size is initially zero to allow the snake to take up the shape of the bowl. The step size is then increased by a constant plus a variable dependent of the difference in length of the snake with respect to the last iteration. Thus if the snake starts to increase in length too rapidly the step size decreases, if increases too slowly or even shrinks the step size is increased.

5.3.3 The Snake Dynamics

The general model of the dynamics of the snake used in this thesis is based on the Euler-Lagrange equations given in the two papers (Cohen & Cohen, 1993; Leymarie & Levine, 1993). The two additional terms, not addressed so far, in equation 5.6

control the momentum and drag on the snake, using constants $\mu = 96.0$ and $\sigma = 96.0$ respectively. The main value of these parameters is that they smooth the trajectory of the snake and make the matrices better conditioned. The rest of the terms on the left hand side of equation 5.6 represent the internal forces, the terms on the right hand side the external forces.

$$\underbrace{\mu\frac{\partial^2 \hat{f}}{\partial t^2}}_{momentum} + \underbrace{\sigma\frac{\partial \hat{f}}{\partial t}}_{drag} + \underbrace{\frac{\partial}{\partial t}\left(\frac{\partial}{\partial s^2}\left(\omega_c(s)\frac{\partial^2 \hat{f}}{\partial s^2}\right)\right) + \frac{\partial}{\partial s^2}\left(\omega_{tp}(s)\frac{\partial^2 \hat{f}}{\partial s^2}\right)}_{internal forces}$$

$$= \underbrace{\left[\nabla\left(-\left|\nabla Q_{max}(\hat{f})\right|^2\right) + M(\hat{f})\right]\overrightarrow{n}(s)}_{external forces} \qquad (5.6)$$

To produce the complete partition, all the local minima in the coefficients of the spline defining the bowl function are located. Due to the "variation diminishing property" (Aumann, 1997) there are guaranteed to be more such local minima than in the spline itself. The one with the smallest value is selected. Due to the greater distance to the walls, the height at a minimum of the bowl function is less for larger rooms. To minimise the number of iterations, the snake is initialised to a circle of radius determined by this height ($(3.0 - height) * 0.075$) and thus starts larger in larger rooms. The snake evolves according to a discrete time approximation of equation 5.6. As the process iterates the snake progressively adopts a more complex shape under the external forces. Initially both internal forces act in concert to maintain the circular form. But once a reasonable approximation to the room is achieved, one will minimise changes in either direction, while the other smoothes the function. Once the ridges are reached only the smoothness function implementing the polygonal constraints is used as discussed in the first paragraph of section 5.3.1. On reaching an approximately stable position subject only to small oscillations, the momentum and drag are increased until the snake effectively comes to rest. To fit the next room all local minima contained within the polygon representing the first room are discarded. Of the remaining minima the smallest is again chosen and the process repeated. This continues until there are no more local minima.

5.3.4 Scaling the Function

The success of the snake depends on its ability to discriminate signal from noise. Here, the signal is due to the features that develop in the reinforcement learning function. The noise arises from variations in the low level learning process and the stochastic nature of the task. Noise is differentiated from the signal by two properties: firstly the amplitude of the noise should be much smaller than that of the signal, secondly uncorrelated noise is very unlikely to produce a continuous differential in the function. Correctly scaled, small individual noise spikes will not be sufficient to stop the flow of the snake, but the relatively large continuous edge of the signal will.

The correct scaling is a balance between the size of the signal and the size of the noise. Both of these can vary over the state space and change with learning. Fortunately in the two main circumstances where scaling is necessary the sizes are roughly correlated. Firstly the size of the noise and signal are dependent on the distance to the goal. This is due to the reinforcement learning function being an exponential in this distance. The size of the features and the size of local fluctuations will be roughly proportional to the height of the function. Secondly, early in the learning process both the signal and the noise tend to be small but get larger as learning continues. If noise could be measured locally both these these problems could be addressed together. Wavelet research offers one possible solution. Some research (Chang & Vetterli, 1997; Chang, Yu, & Vetterli, 1998) has looked at spatially adaptive wavelet thresholding for images, extending ideas on denoising signals by suppressing small wavelet coefficients (Donoho & Johnstone, 1993; Dononho & Johnstone, 1994). This approach would deal with both situations, but at present the problem of distance to goal is dealt with by using a function to locally scale the differential.

The dashed line of figure 5.10 is a cross section of the reinforcement learning function for the robot navigation problem of five rooms. The steep slopes are the features associated with two of the walls. The solid line is a smoothed version of the function decaying from right to left. The dashed line of figure 5.11 is the gradient, the hills are the features that the snake locates. The one on the right is larger than that on the left . By scaling the differential, dividing it by the value of the smoothed function in figure 5.10 the difference is significantly reduced. There is less variation

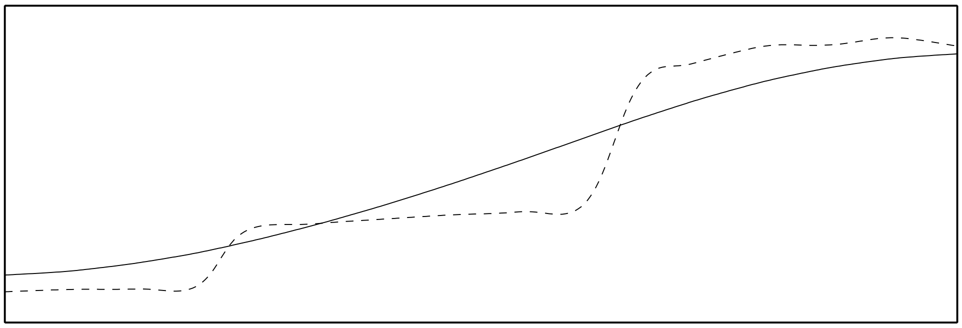

Figure 5.10: A Smoothed Function

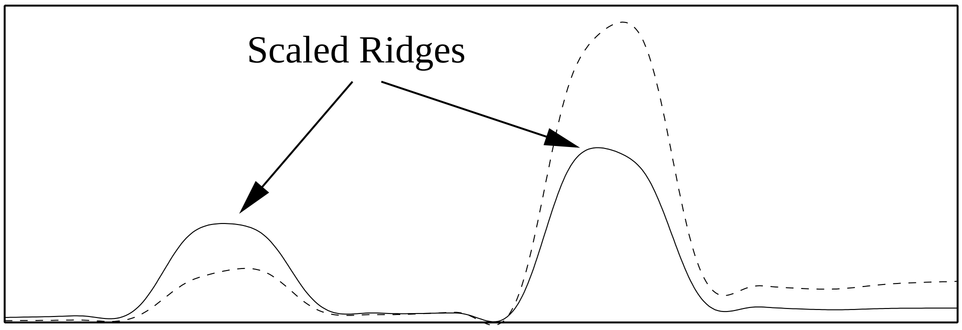

Figure 5.11: Scaling for Distance from Goal

in the size of the features, making location easier. Due to the correlation between noise and signal, the noise is also made more uniform.

To determine the scale at different times during learning, the function shown in equation 5.4 is uniformly sampled. Pairs of values for the x and y directions are combined in a Euclidean Norm. The median of their values is used as a measure of the noise. The assumption, much as in wavelet denoising (Donoho & Johnstone, 1993), is that most of the signal is in only a few terms. This is intended to ignore the size of the features and measure the noise of the regions in between. This produces a single value that is used to scale the differential, again affecting signal and noise equally, and thus facilitating the extraction of features by the snake.

5.4 Shape Extraction

The aim of this section is to demonstrate the versatility of the snake in producing partitions of different shapes. The examples given also represent the limits of shape extraction in the present implementation. The examples will demonstrate in turn: more complex polygons, removing the polygonal constraint and partitions in higher dimensions. Along with each example there will be a brief discussion of what happens when this limit is exceeded and what possible future changes might remove the limitation.

One aim of the extensions to the snake presented in section 5.2 is to increase the variety of shapes the snake can extract. The addition of a constraint to maintain the shape of the snake as it grows was motivated by experience in the robot arm domain. Figure 5.12 shows the partition produced by the snake for one problem from that domain.

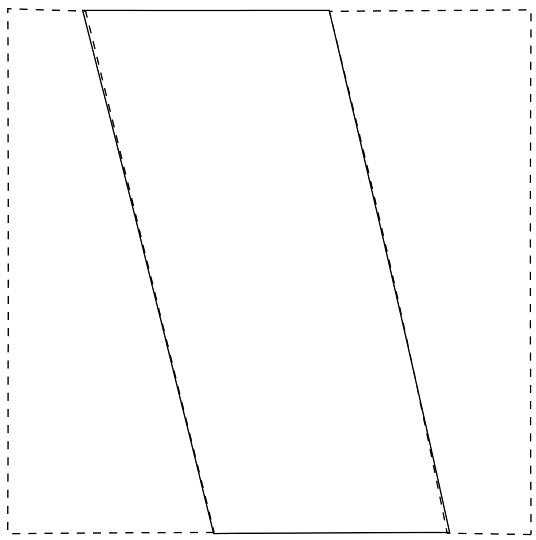

Figure 5.12: Partition for the Robot Arm

Figure 5.12 shows that three snakes have successfully partitioned the space for the robot arm problem. Considering the "room" on the left hand side, there are two factors that make this problem difficult. The first is the rapid narrowing of the "room" with the bottom being about three times as wide as the top. The second is the size of the room at the top which is very close to the size of a "doorway". Decreasing the width at the top of the room or increasing the width at the bottom will prevent the snake from reaching the top. This limitation may be addressed by making sure there is sufficient flexibility, i.e. number of knots, at the top of the snake as it flows upwards towards the top of the room. The problem for the snake is also exacerbated by using a discretised surface to represent the reinforcement learning function. Rather than the "wall" being smooth its has steps in it which can impede the snake. Using smoother basis functions as proposed earlier in this thesis should allow the capture of more extreme shapes.

5.4.1 Polygonal Shapes

This sections shows two examples of more complex rectilinear shapes. Figure 5.13 shows a cross shaped room and the resulting partition. An experiment was also tried where the arms of the cross were narrower. The result was appreciable error at the end of two of the arms. This was due to the method of enforcing the polygonal constraint. When the constraint is added there are only a few knots at the end of the arms giving insufficient room to add a rigid piece. As in the robot arm example additional knots at the end of the arm may be the answer. A change in how the polygonal constraint is added would also remove the problem. In section 5.3 a way of increasing the number of knots at higher than average curvature points was discussed. It may be possible to use this method to obviate the need for adding the polygonal constraints.

Figure 5.14 shows a Z-shaped room and its partition. The snake is initialised as a circle in the lower knee of the Z at one minima of the bowl function, as indicated by the dotted lines on the right hand side of figure 5.14. As it starts to grow, the solid lines, it begins to take up roughly an L-shape. It then moves towards a second minima in the bowl. This causes a reduction in the average height of the snake and a squeezing of the middle of the snake. The top of the snake then flows outwards to capture the top "bar" of the Z. In a second experiment where the centre hallway is narrower, the snake takes up the initial L-shape and stays this way failing to capture the Z-shape. To overcome this effect the force that encourages the snake to expand

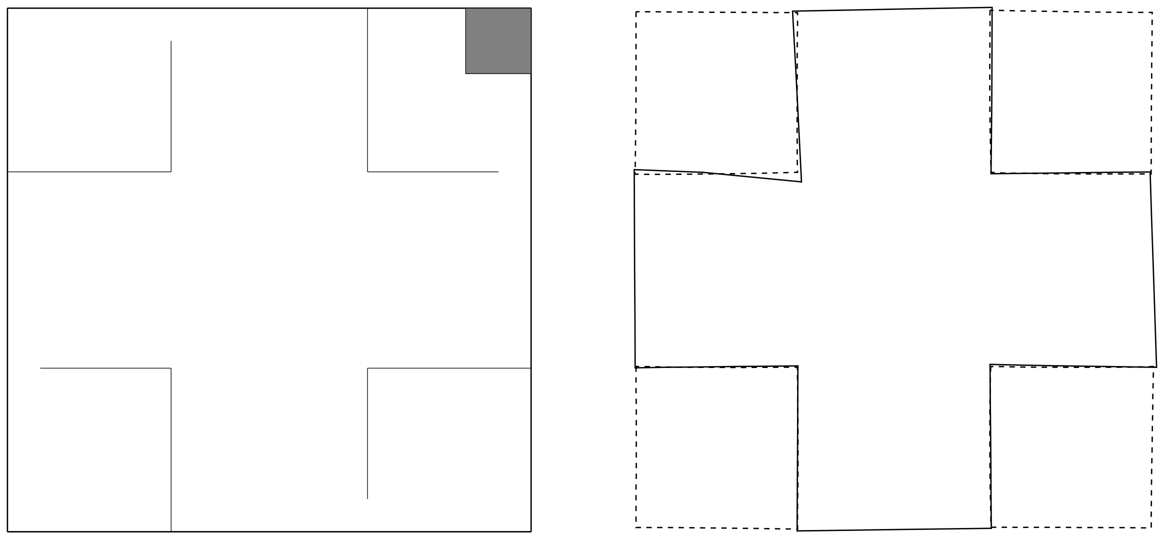

Figure 5.13: A Cross Shaped Room and its Partition

may have to be weakened to allow the snake to better adjust when additional shape information becomes available.

5.4.2 More General Shapes

The previous sections dealt with extracting low order polygonal regions. This section look at more general shapes. Figures 5.15 and 5.16 shows the partition produced when applying the snake to the "car on the hill" domain. The main differences from the previous examples is that the polygonal constraint has not been used. When the snake initially comes to rest the mercury force is turned off and then the snake is allowed to find the minimum energy state. Figure 5.15 is using scaling generated by the estimated noise value. One snake (the solid line) fits the edge well. The other (the dashed line) produces a poor fit on the lower right. This is caused by too low a threshold producing the bowl function. This results in a thickening of the sides of the bowl, preventing the snake being driven all the way to the edge. This problem is easily remedied by reducing the scaling by about a factor of three quarters, as shown in figure 5.16.

The fit around the top left corner of the second snake(the dashed line) also has some problems: the snake is growing very slowly downwards and is at present only

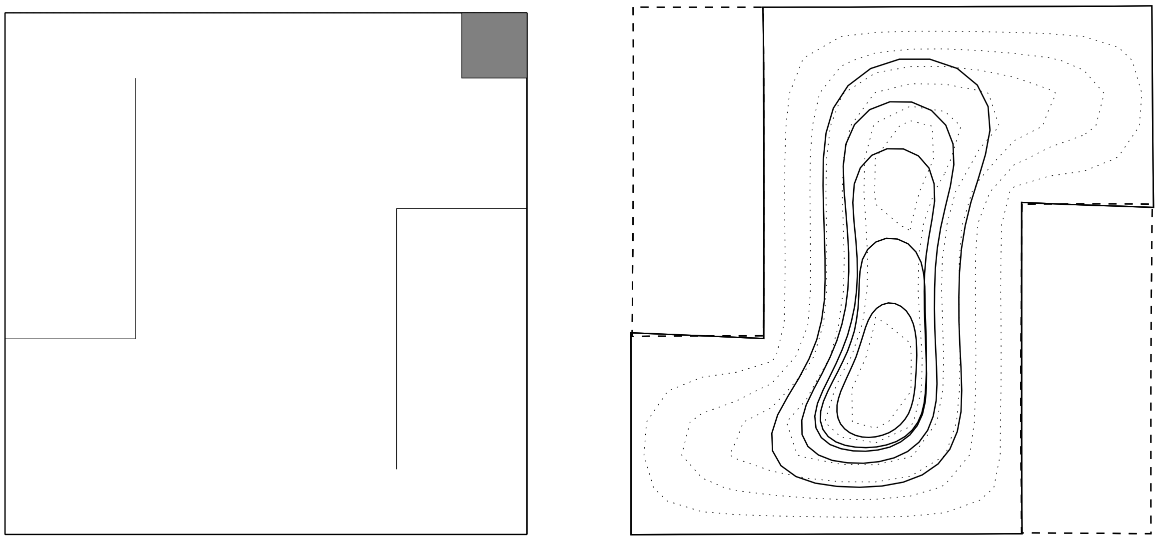

Figure 5.14: A Z-Shaped Room and its Partition

stopped because it has reached the maximum number of iterations allowed. One difficulty in this example is that there is not such clear delimitation of the upper and lower regions at the end of the feature. Future work will investigate altering the stopping condition to eliminate this problem. Alternatively the snakes may be "zipped up" along there common edge so that they form a complete non-overlapping partition of the state space.

5.4.3 Snake in Higher Dimensions

The work in this thesis has focused on domains of two dimensions. But the ideas presented here are more general. The implementation of the snake itself has been updated to work in higher dimensions. The bold lines at the top of figure 5.17 are one of the simpler problems from the robot navigation domain. For this example the problem has been extended in the Z-dimension. Here the snake in each room starts out as a sphere and then expands outwards until it fills the room. As in the "car on the hill" example the polygonal constraint has not been used but everything else remains the same. Figure 5.18 shows the result of fitting the first room, figure 5.19 the second. Figure 5.20 shows the complete partition of the problem.

The present implementation is intended to work with higher than three dimensions

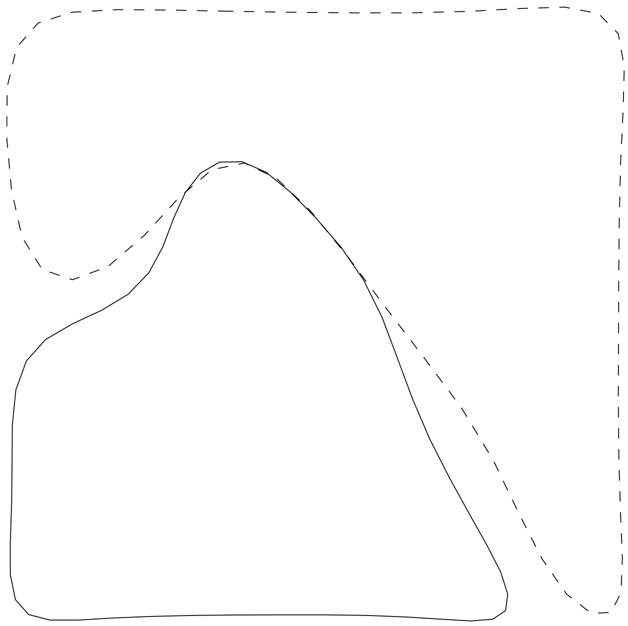

Figure 5.15: Car on the Hill: Auto Scaling

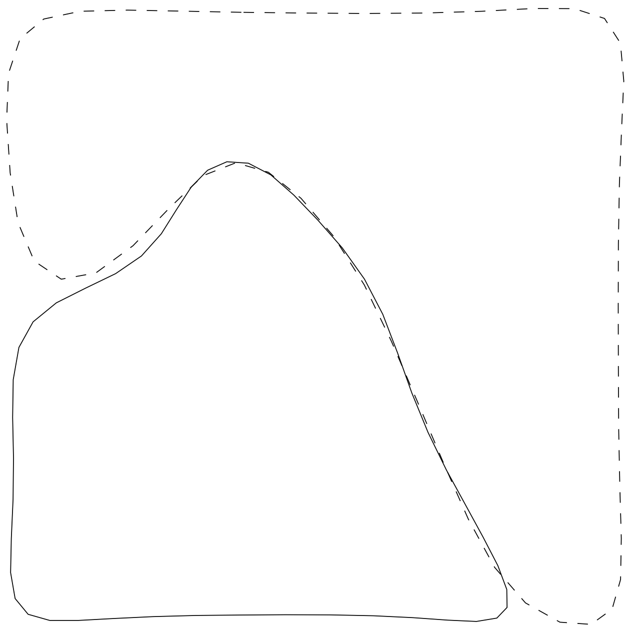

Figure 5.16: Car on the Hill: Lower Scaling

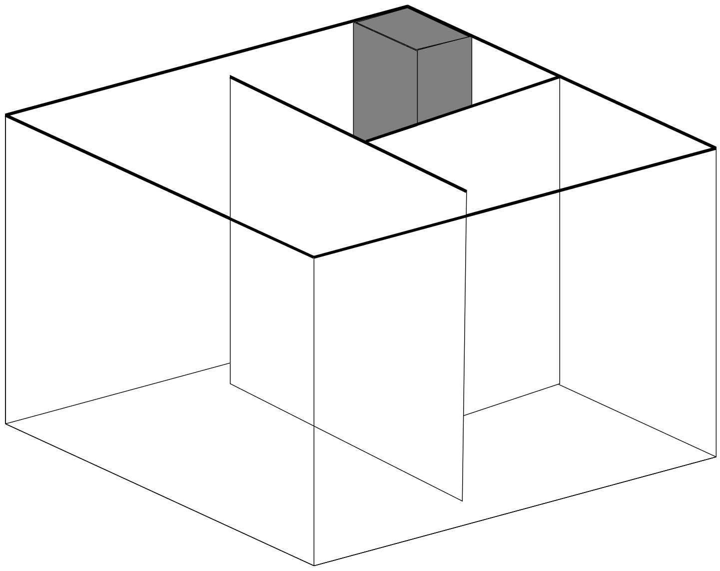

Figure 5.17: Adding a Z-Dimension

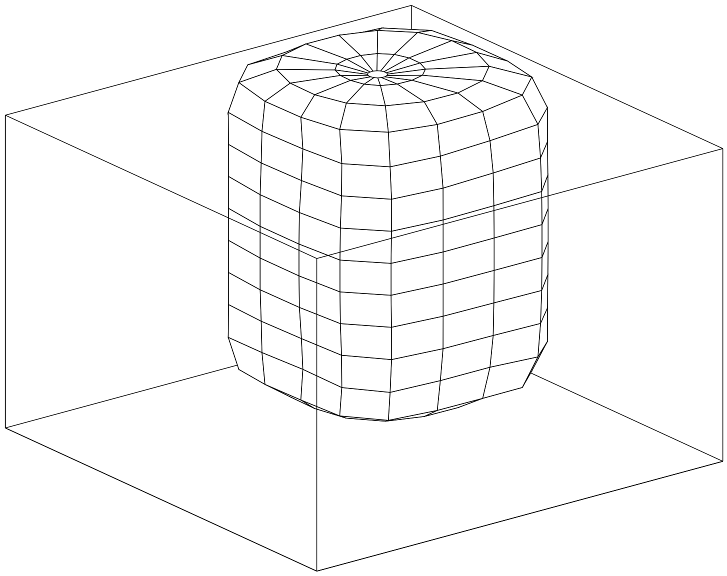

Figure 5.18: Delimiting the First Room

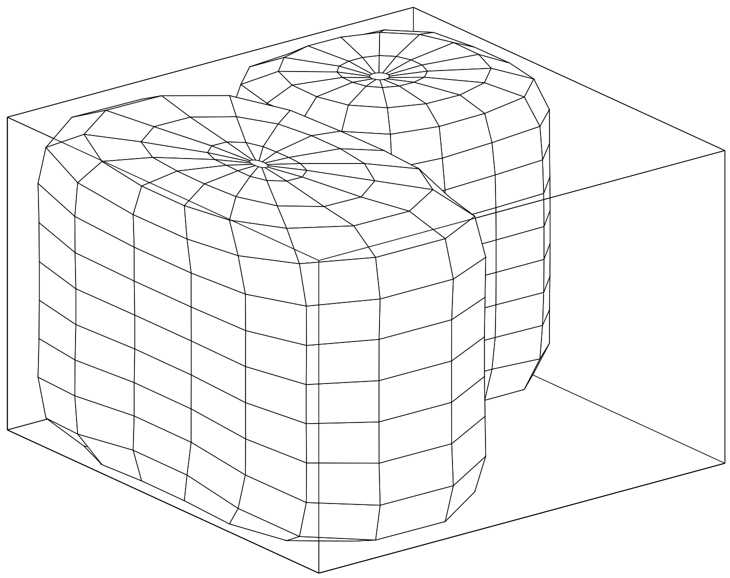

Figure 5.19: Delimiting the Second Room

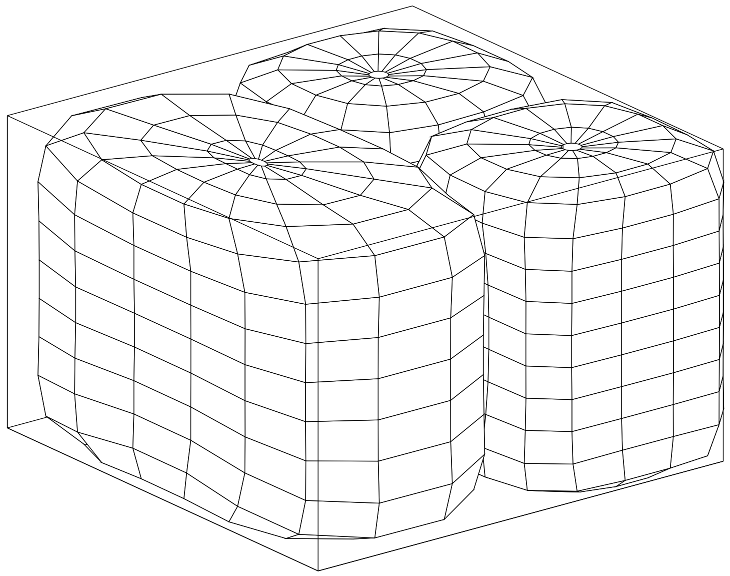

Figure 5.20: The Complete 3D Partition

although this has not been tried. One concern is the time taken by the snake to fit the rooms and how this scales with the number of dimensions. A variational approach is used to solve the differential equation as discussed in section 5.3.3. Each iteration of the snake requires finding the spline coefficients by matrix factorisation. At present sparse Choleski factorisation (Stewart, 1992; Dahlquist & Bjorck, 1969) is used but iterative algorithms may be faster. Generally, due to the sparsity of the matrices, finding the coefficients should be $O(nm)$ where n is the number of basis functions and m the number of neighbouring functions that overlap. The existing snake has a fixed number of knots, and little attempt has been made to reduce the number of iterations, the main focus being on the accuracy of fit. The process is therefore relatively slow being of order $O(inm)$ where i is the number of iterations.

There are many possible ways to speed up the process. The number of iterations and the number of basis functions required should reflect the complexity of the representation of the reinforcement learning function. Further, instead of using a fixed number of basis functions the snake might be reparameterised as it grows. The hope would be to achieve an average complexity of $O(rm)$ for the whole process, where r is the number of basis functions representing the reinforcement learning function. To illustrate how this might be achieved the top of figure 5.21 shows the snake in the shape of a square with the centres of the basis function represented by black circles. The grid represents discrete regions in the reinforcement learning function. Ideally each iteration (or a small constant number of iterations) would move the snake to intersect a new set of cells. The snake would then be reparameterised to have a basis function per cell. Of course this assumes a completely open space, with no walls. The worst case occurs when it is divided into a series of long rooms as shown at the bottom of figure 5.21. The snakes can only grow in one direction, the lower left hand corner shows three iterations, having 4,6 and 8 basis functions respectively. The total number of calculations will be proportional to the total of the basis functions at each iteration, so the overall complexity is $O(r^2m)$. It is noteworthy however that this is a much more difficult problem for normal reinforcement learning. The complexity of the snake algorithm should roughly mirror the difficulty of the problem.

One major advantage of using B-splines is that m is small, the small support

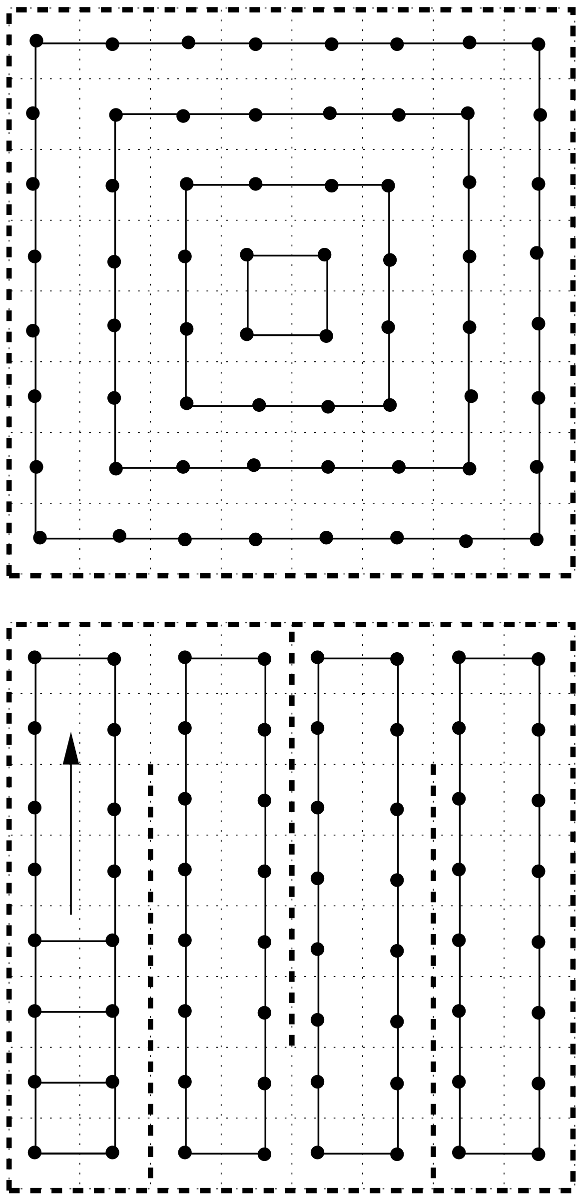

Figure 5.21: The Complexity of the Snake Algorithm

means minimal overlap with other functions. But although m is small in any one dimension (3 for linear, 5 for quadratic and 7 for cubic) it will be raised to the number of dimensions. It may be preferable to use discrete representations and finite difference methods if the dimensionality is high. It will still be necessary to determine the partial differentials, or their finite difference equivalents, as discussed in section 5.3.2. This would require replacing m with a 3^d where d is the dimensionality of the problem.

The research discussed in this thesis has aimed to exploit the results of research in many areas. There is certainly a large amount of active research into many aspects of the snake which might be exploited in the future. Schnabel (1997) investigates various algorithms used to fit the snake. An alternative to the variational approach is dynamic programming (Amini, Weymouth, & Jain, 1990) which uses a discrete representation and has a complexity of $O(sc^3)$. A more greedy approach (Williams & Shah, 1992) has a complexity of $O(sc)$. In both algorithms s is the number of sample points, roughly equivalent to the number of basis functions when using splines. The factor c is the number of candidate positions for an initial sample point, in other words all the discrete states that the sample point might eventually reach. The lower complexity of the latter algorithm comes from using a greedy rather exhaustive approach. This does not guarantee a global minimum in energy over the states searched but typically finds good solutions. In applying the snake to task partitioning, finding the global minimum is not critical. It is sufficient to find a good approximation to the region representing a subtask. Alternative ways to speed up the fitting process are also under investigation. One example is hierarchical methods (Schnabel, 1997; Leroy, Herlin, & Cohen, 1996) which find solutions for the snake at progressively finer and finer resolution scales.

5.5 In Summary

This chapter has presented a way of partitioning the reinforcement learning function into regions associated with solutions of subtasks. The boundaries of the regions are determined by an edge extraction technique called a snake. The algorithm shown in figure 5.22 summarises the steps required to produce the partition. The numbers at the end of many of the lines indicate the sections where a more detailed explanation of the related steps can be found.

This chapter has detailed extensions to the basic snake approach that make it more effective in producing a partition. They should also be generally useful to the more traditional application of snakes to edge location in image processing. The numeric values for the various parameters given in this chapter were established empirically from hand crafted examples. The experiments of chapter 7 will show that these parameters are effective on randomly generated examples. This chapter has shown that the snake can extract a large variety of shapes in two dimensions and also works in three dimensions. Future work will address extending the variety of shapes the snake can extract and looking at higher dimensions.

The overall approach borrows much from object recognition in vision research (Suetens et al., 1992; Chin & Dyer, 1986). Object recognition is potentially a fertile source of algorithms to speed up reinforcement learning. Partitioning is typically a matter of determining which states belong together, one of the problems that vision research has been addressing for a number of years. Snakes are one very active area of research. Concerns such as the automatic setting of parameters and improving the speed of convergence are being addressed. The results of such research should be of benefit to the partitioning of subtasks.

1. **Scale the value function according to the amount of noise and the distance from the goal (Sec 5.3.4).**
2. **Construct the bowl function by applying Gaussian smoothing to thresholded samples of the differentials of the value function (Sec 5.2, 5.3.2)**
3. **Put the coordinates of the local minima of the bowl function on an ordered list called MINIMA with the deepest values first (Sec 5.3.3).**
4. **If MINIMA is empty, exit (partitioning completed).**
5. **Remove coordinates from MINIMA and initialise the snake as a circle centred on coordinates.**
6. **Repeatedly calculate and apply forces to the snake until length stabilises (Sec 5.3.3).**
7. **Add polygonal constraints to the snake (Sec 5.3.1).**
8. **For a fixed number of steps calculate and apply forces to the snake, increasing momentum and drag (Sec 5.3.3).**
9. **Add region delineated by snake to list PARTITION.**
10. **Remove all coordinates in MINIMA covered by region.**
11. **Go to step 4.**

Figure 5.22: The Partitioning Algorithm

Chapter 6

The Symbolic Level

This chapter shows how the partition produced by the snake can be converted into a graph. The graph allows a form of symbolic planning that composes subtask solutions to form a solution to a new task. The planning exploits individual subgraphs associated with regions delimited by the partition. The subgraphs are used to index a case base of subtask solutions. In the context of the architecture discussed in section 1.2 this is the symbolic level shown in figure 6.1. The main contributions discussed in this chapter are the method of composition that builds a solution to a new task and the means of transforming subtask solutions to fit it.

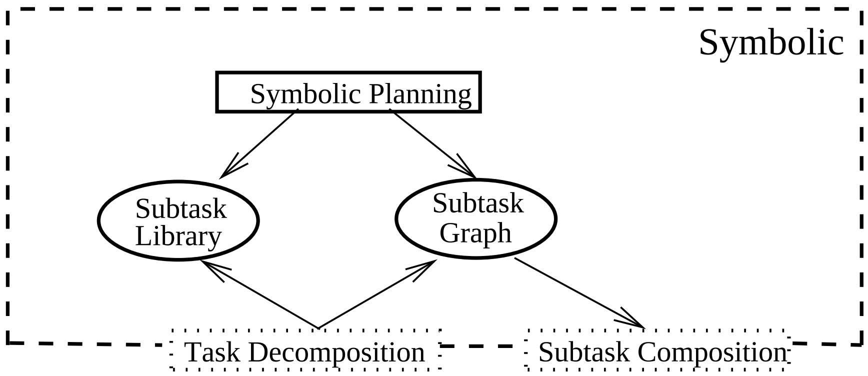

Figure 6.1: Symbolic Level

6.1 Creating Graphs

Figure 6.2 shows the different levels and types of representation for the robot navigation problem of five rooms. At the bottom is the reinforcement learning function, immediately above it its gradient. The snake hill-climbs the gradient generating the next level, the partition. From this a graph, the topmost level, is produced. The nodes represent corners of the regions (white ellipses), doorways (shaded ellipses) and the goal (striped ellipse). The edges (dashed lines) connect the doorways and the goal, their direction indicating the paths from the rooms to the goal.

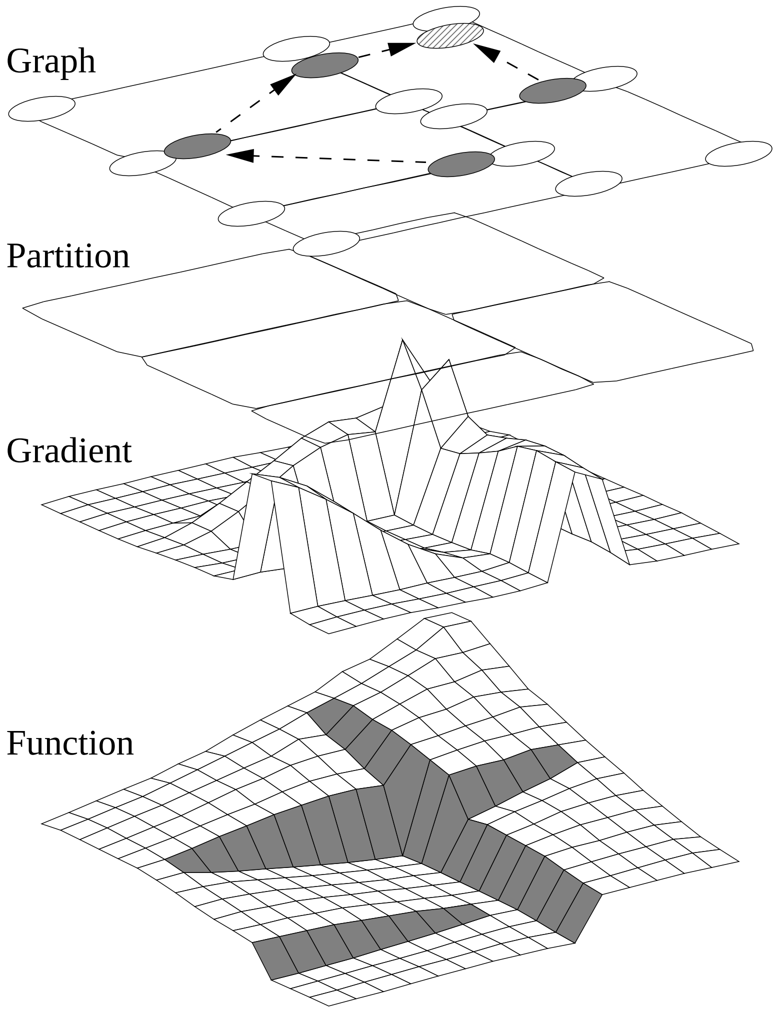

Figure 6.2: From a Function to a Graph

6.1.1 Creating Subgraphs

The top level graph is an abstract representation of the solution to the complete task. It is built up by constructing subgraphs for each particular subtask and then merging them. Figure 6.3 shows just the levels of representation associated with a single room. The snake produces a polygon, one piece of the partition shown in figure 6.2. In this example the polygon is a rectangle. It delimits a region of the reinforcement learning function, its gradient and forms the basis of the subgraph.

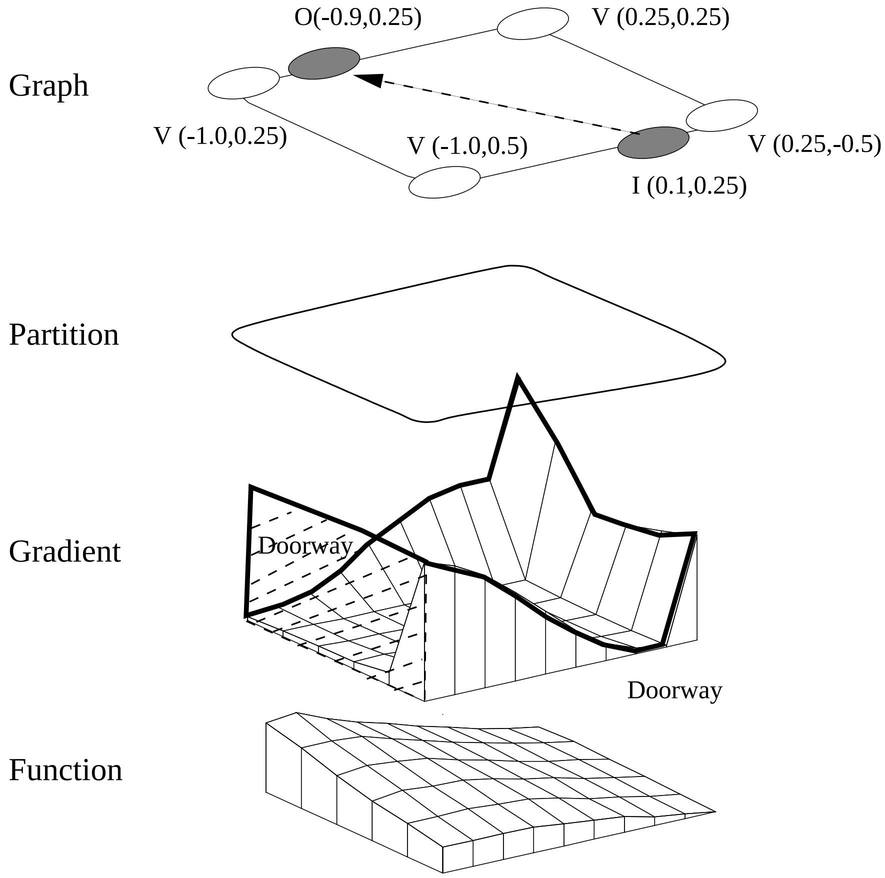

Figure 6.3: Extracting a Case

This information is stored in a case base. Each subgraph is an index and the corresponding part of the control function is the case. Thus learning at the symbolic level is a form of instance based learning. At present all learnt functions are kept, though it might only be necessary to keep prototypical examples as in the work of

Aha, Kibler and Albert (1991).

The subgraph is constructed using the position of the corners of the region, the doorways and the goal if appropriate. It is a plane graph with an (x, y) coordinate for each node (subsequent figures will not show the coordinates). The position of the goal is determined by finding the smallest bounding rectangle that includes all points in state space where a reward of one was received. A node for the goal is included in the subgraph, positioned at the centre of this rectangle, if it lies within the region. The position of the corners are simply the vertices of the polygon, producing nodes labelled "V" in figure 6.3. The positions of the doorways are minima in the gradient along the sides of the polygon. These local minima are located by steepest descent on the function shown as the bold line in figure 6.3, the gradient at the state space boundary is indicated by the shaded region. The direction of the gradient with respect to the appropriate side of the polygon determines if it is an entrance or an exit, producing the nodes labelled "I"and "O" for in and out respectively.

The gradient contains many local minima not associated with doorways. These arise either from the inherent noise in the process or from errors of fit in the snake. The aim is to remove the ones not associated with doorways by smoothing and thresholding. This is achieved by first sampling the gradient at points along the snake. The values are then normalised to lie between zero and one. To establish a lower limit, the minimum value across all samples is found. To establish an upper limit, an average value is used. Different upper limits are used for different portions of boundary of the region. The average value of gradient can vary considerably particularly between the different sides of the polygon. So the upper limit is determined by finding large jumps in the sampled values and then averaging the intervals between the jumps. The values are thresholded at the average value and renormalised to lie between the upper and lower limits.

These values are approximated by a one dimensional cubic spline with ω_m of 0.15, divided by the length of the boundary of the region to maintain roughly the same number of degrees of freedom for equal distances. Here a weighted least mean squared fit is used. The weighting function is the inverse square of the values, preventing the spline being overwhelmed by large values. Starting points for steepest descent are changes in the sign of the coefficients of the gradient of the spline. The initial step size is set to slightly larger than a knot spacing and then decreased over time. When a local minimum is found if the value exceeds a threshold (of 0.5) it is rejected.

Even this is not always sufficient to reject false doorways. When the function has converged the doorway will be top of a slope, flowing into the room if it is an "in" doorway and out if it is an "out" doorway. The direction of flow can be determined by looking at the gradient close to the candidate doorway. If it is real doorway it will point towards the door, if it is not it will point in some other direction along or away form the wall, as shown in figure 6.4. So if the gradient a short distance from the doorway points roughly towards the door ($\pm\pi/4$) it is kept otherwise it is rejected.

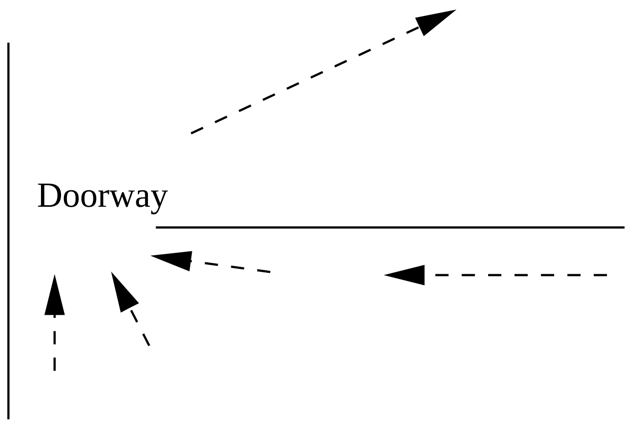

Figure 6.4: Direction of the Gradient at a Doorway

6.1.2 Merging Subgraphs

The left hand side of Figure 6.5 shows plane graphs for all the rooms. The node representing the goal is labelled "G". Associated with the edges between doorways and the goal is a number representing the distance between the nodes. This is determined from the value of the function at the points of the doorways, assuming exponential decay with a fixed exponent.

Each individual subgraph is then merged with its neighbour to produce a graph for the whole problem, the right hand side of Figure 6.5. The subgraphs on left

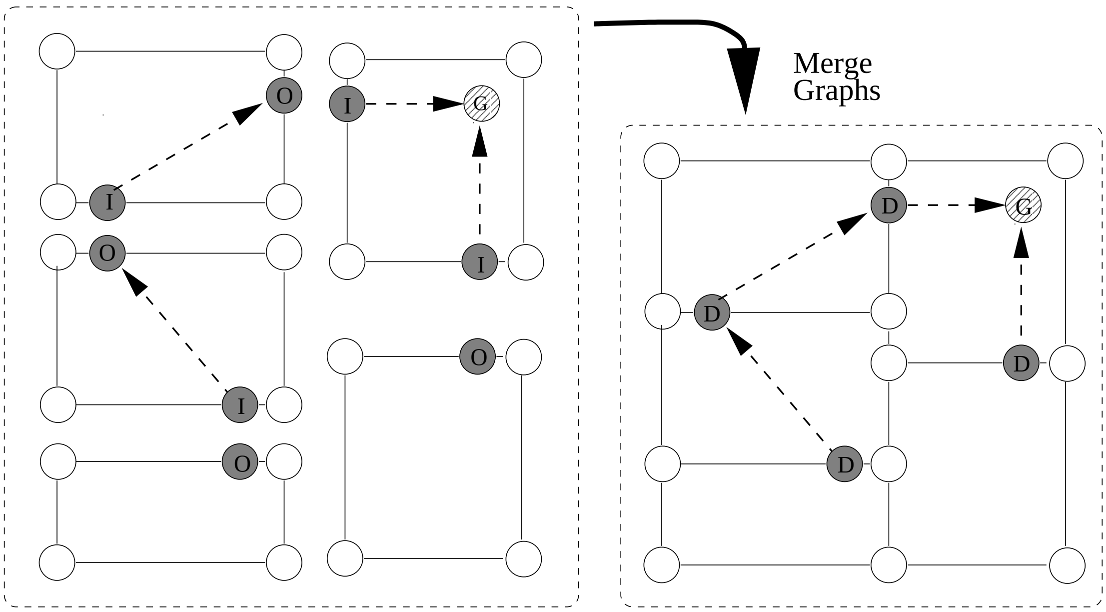

Figure 6.5: Merging the Graphs

hand side of the figure have been moved apart for clarity. In practice many of the nodes overlap. The merging process determines the overlapping nodes and replaces them with a single node reconnecting the edges appropriately. For instance, every doorway node should overlap with another of opposite direction, i.e. "in" with "out". Failure to do so is an indicator of problems with the partition. This is used in the experiments of chapter 7 to filter out underdeveloped features. Once all the doorway nodes are merged, a path is established from each subgraph to the goal along the edges connecting them, the dashed lines in figure 6.5.

Overlapping corner nodes are also replaced by a single node, and the graph is reconnected to remove duplicate edges and position the unpaired nodes correctly, as shown in figure 6.6. The process of merging maintains copies of the original subgraphs and constructs a mapping of nodes and edges of the composite graph to the subgraphs. This allows an easy reversal of the process which is used in the symbolic planning discussed in the next section.

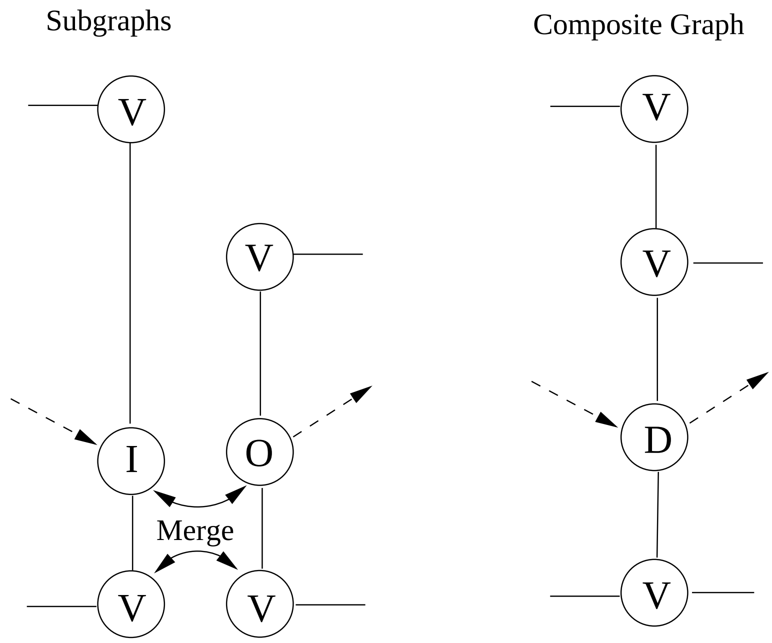

Figure 6.6: Merging Nodes

6.2 Symbolic Planning

This section describes how a method similar to symbolic planning uses subgraphs extracted from the composite graph to produce the solution to a new task. The left hand side of Figure 6.7 shows the composite graph after moving the goal in the robot navigation problem of five rooms. The edges connecting the doorways and the goal were changed to account for the new goal position. To produce a new function, the idea is to regress backwards from the goal along these edges. The right hand side of Figure 6.7 shows the graph controlling this regression. For each edge, the small subgraph containing the edge is extracted. This is just a reversal of the merging process as outlined at the end of the previous section. The extracted subgraphs are used to index the case base of functions. The retrieved functions are transformed and added to the appropriate regions of the state space to form the new function. Regression uses a slightly modified form of Dijkstra's algorithm (Dijkstra, 1959) to traverse the edges between doorway nodes.

To begin the process, the subgraph which contains the goal is extracted and the

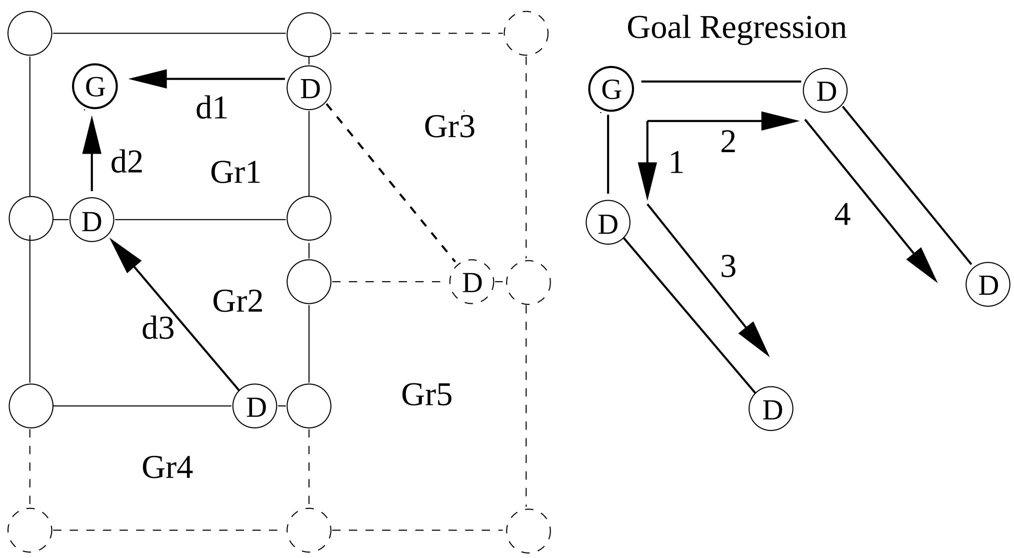

Figure 6.7: Using Dijkstra's Algorithm

best matching isomorphic subgraph is found. The edge lengths in the composite graph are then updated using the scaled length of the corresponding edge in the matching isomorphic subgraph, producing the new distances d1 and d2 in figure 6.7. As d2 is less than d1, the next subgraph extracted, Gr2, is the one sharing the doorway node with the edge of length d2. The best matching isomorphic subgraph is found and the edge length d3 updated. The shortest path is again determined, as d1 is less than d2 + d3 subgraph Gr3 is extracted. The process is repeated until all subgraphs have been updated.

The transformations applied to the subgraphs during the matching process are then applied to their corresponding functions from the case base. Figure 6.8 shows how the function for the room containing the old goal is rotated, translated and stretched to fit the new goal position. Another transformed function representing the next subtask solution is abutted to the first function so that the result is smooth at the doorway, the black square in figure 6.9. To achieve this the function is first normalised so that it is one at the "out" doorway, i.e. its maximum height is one. Then as the function is an exponential it is just multiplied by the exponential of accumulated distance to the goal at the doorway.

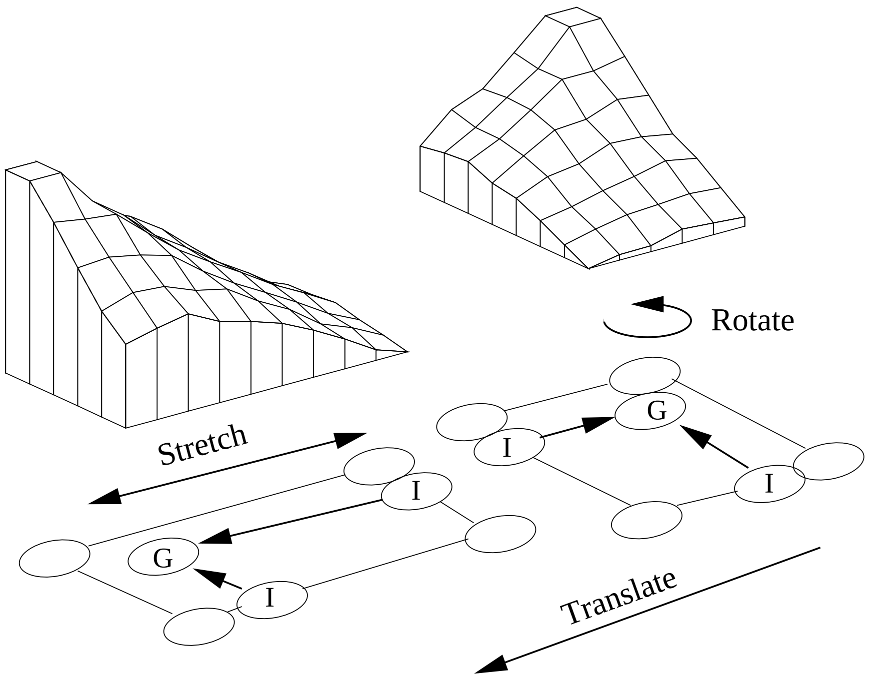

Figure 6.8: Transformation

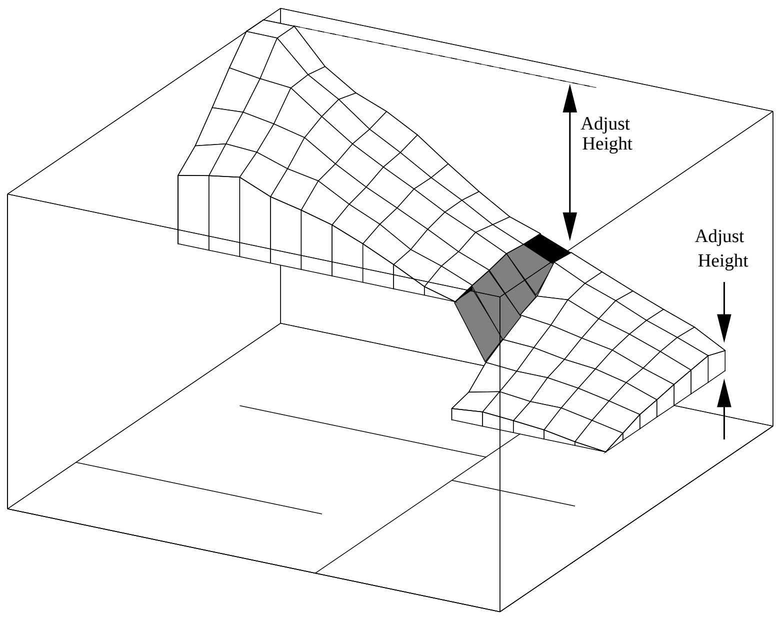

Figure 6.9: Concatenating Functions

The height must also be adjusted when a scaling transform is used. Imagine that to fit a new room the size of the region of the state space covered by the function must be doubled in each dimension. The distance between the doorways will therefore double. Again starting with a normalised function and remembering it is an exponential raised to a power of this distance, the height must be squared. In general, the height is raised to the power of the scale factor. When scaling is the same in each dimension, the result is exact With asymmetric scaling the result is not. But if the asymmetry is relatively small using the maximum works well in practice. This will be discussed in further detail in section 6.2.2

6.2.1 Decision Boundaries

In the example of section 6.2 there is only a single path to the goal from each room. Often there will be multiple paths. Suppose room 5 had an additional doorway in the lower left corner of the room, as shown in the bottom right on the left hand side of figure 6.10. The graph on the right hand side of figure 6.10 would result. There are now two possible paths to the goal from room 5 of lengths d4 and d5. If the length across room 5, d6, is greater than the absolute difference between d4 and d5, the choice of path from this room will be determined by a decision boundary inside the room. This is produced by taking the maximum of two functions (shown in figure 6.11): one for entering by the top doorway and leaving by the bottom left doorway; one for entering by the bottom left doorway and leaving by the top doorway. This principle can be repeated if there are more than two paths to the goal from a given room.

If the cross-room distance, d6, is smaller than the difference (|d4-d5|) the decision boundary would have to be in another room. In general we want to find the room in which the cross-room distance is larger than the difference between the incident paths. This is repeated for every cycle in the path graph. A cycle is detected when a node is visited twice, indicating that it is reachable by two separate paths. Let us suppose this is node n3 in the graph of figure 6.10. As Dijkstra's algorithm is being used, we know that all previous nodes, on either path, such as n1 and n2 are already closed. This must be true for both paths to have reached n3. All the rooms on paths

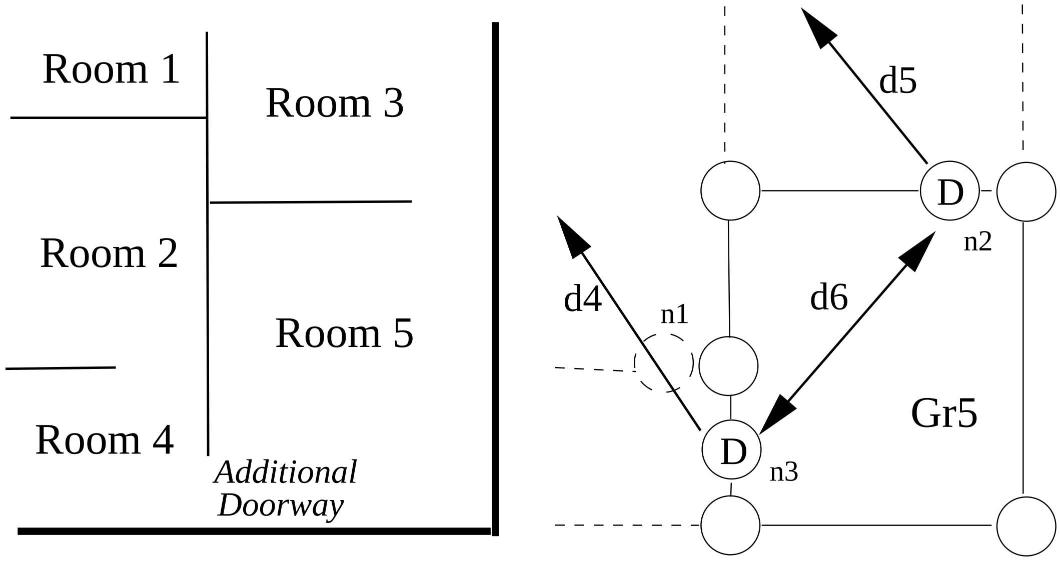

Figure 6.10: Multiple Paths to Goal

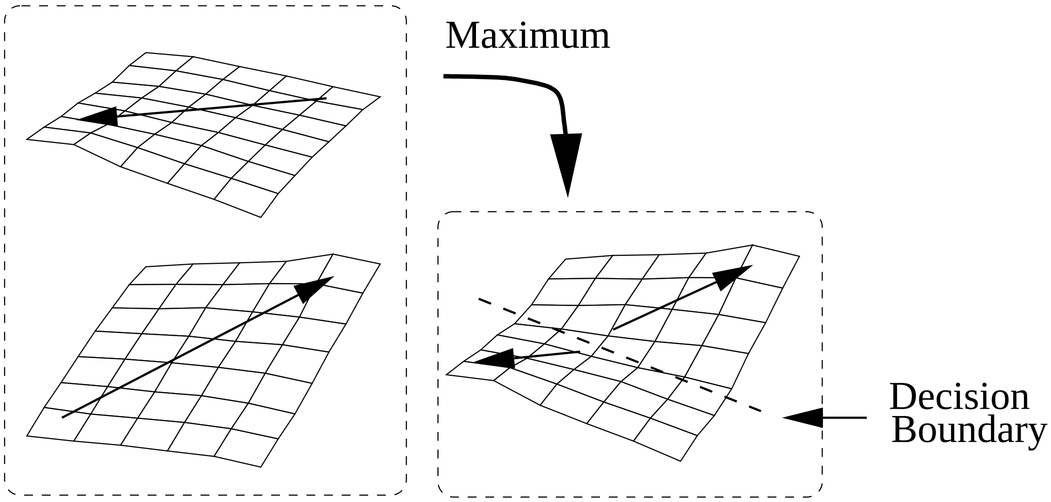

Figure 6.11: Combining Two Functions

up to these nodes cannot contain the decision boundary so it must be in either room 4 or 5. To decide which remaining room it is in, we compare the two path lengths. If d4 is longer than d5 plus d6 then the decision boundary will be in room 4. Otherwise it will be in room 5. If the paths are of equal lengths, taking the maximum of the two functions will correctly put the decision boundary at the doorway.

The case base may not always contain the necessary functions for entering the room by one door and leaving by another. But it may include a function already containing a decision boundary. This function cannot be used directly, avoiding the need to combine two functions, unless the decision boundary is in the right position for the two incident path lengths. But it can be used with another function if the difference in the path lengths entering the room is less than the difference between the heights of the function at the "out" doorways.

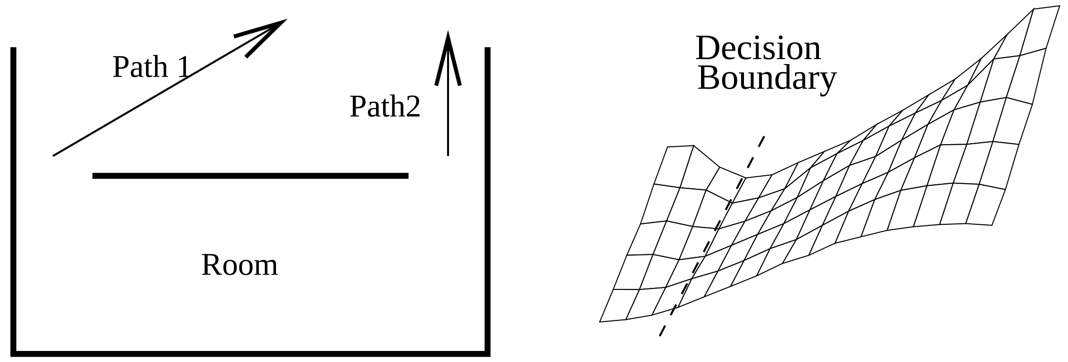

Figure 6.12: Decision Functions

On the left hand side of figure 6.12 there is a room with two doorways. If path 1 is significantly longer than path 2 the shortest path to the goal from most of the room is via the right hand doorway. The function learnt for solving this subtask will contain a decision boundary on the far left. If this function is combined with a mirror image of itself it will produce a decision boundary in the middle of the room, as shown on the right hand side of figure 6.13. This could be used for the new problem shown on the left hand side of figure 6.13 where the two paths are the same length. Again the heights of the two functions can be changed to move the decision boundary. But it cannot be moved to anywhere in the room. The decision boundary can be moved no closer to a particular doorway than in the original function shown in figure 6.12

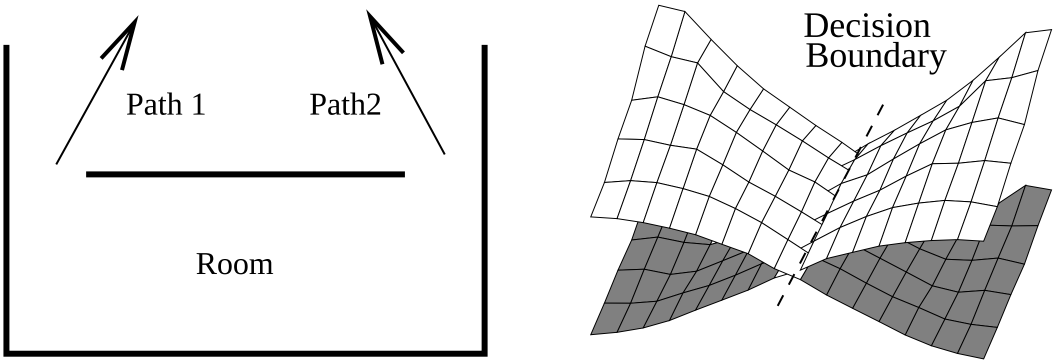

Figure 6.13: Combining Decision Functions

6.2.2 The Matching Process

This section discusses the matching process - how a subgraph is used to select and transform a function from the case base. Selecting the most suitable function must take into account how accurately the transformed function fits the new task, and how much error the transform introduces. The next two sections will address these issues in turn; the third section will address applying the transform to the selected function.

6.2.2.1 Selecting Similar Shaped Subgraphs

The selection process first finds all subgraphs in the case base isomorphic to the extracted subgraph and all possible isomorphic mappings between their nodes, using a labelling algorithm (MacDonald, 1992). The algorithm produces a labelling of nodes according to their adjacency relationships. When these labels are sorted lexicographically and concatenated they form a string which is used to identify isomorphic graphs. Figure 6.14 shows one example of an isomorphism for the L-shaped subgraph in the robot navigation problem of two rooms.

At the top of figure 6.14 is the extracted subgraph, an isomorphic subgraph and the mapping between nodes. It is important to note that this is a general graph isomorphism which does not take into account the (x, y) coordinates of the plane

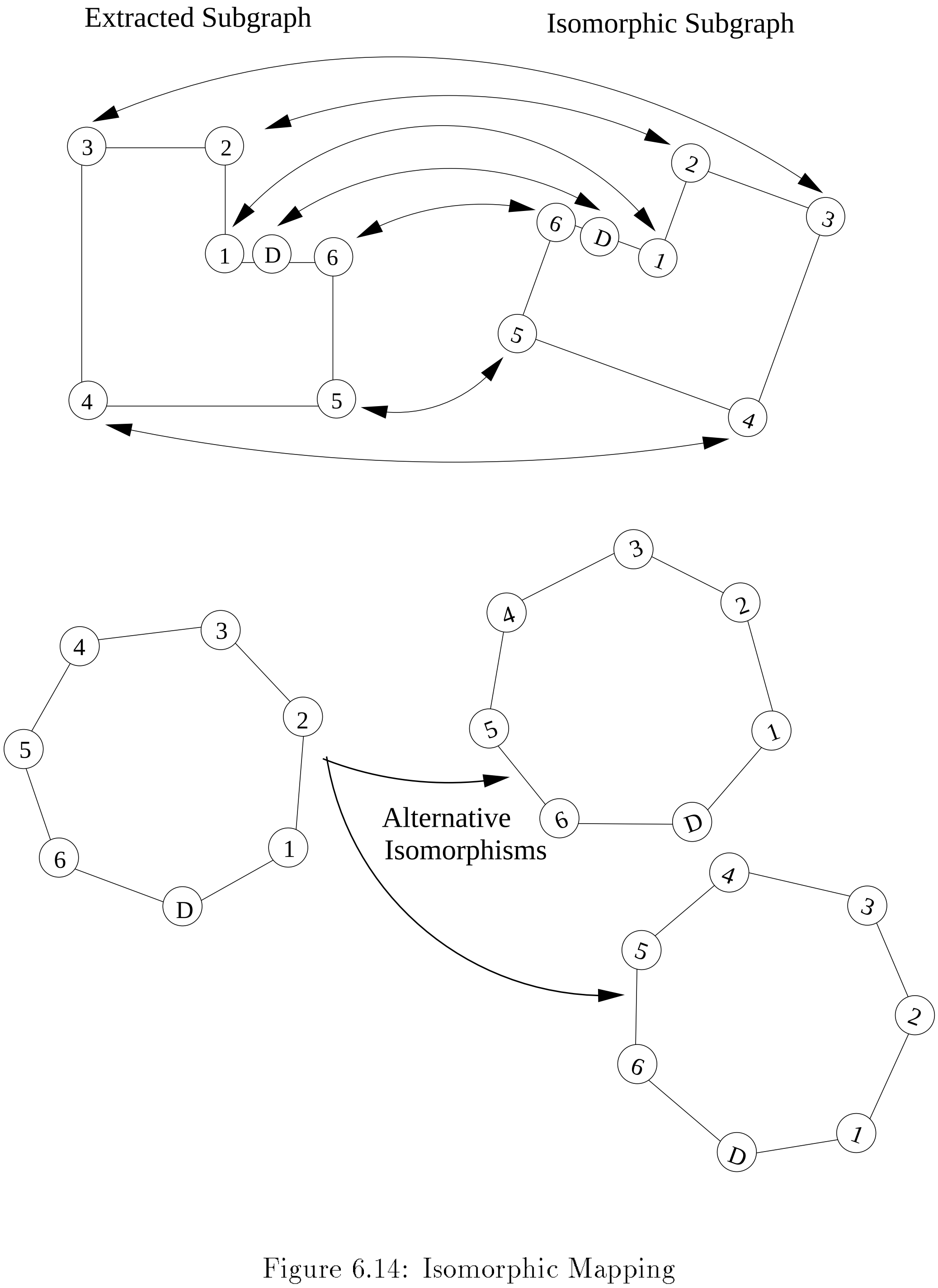

Figure 6.14: Isomorphic Mapping

graph and is therefore insensitive to shape. If the extracted graph is displayed in a normalised form, shown at the bottom right of the figure, it is clear there are two possible isomorphic mappings, one the reflection of the other. In fact, the number of isomorphic mappings is potentially exponential in the number of nodes. But here the graphs typically have only a few nodes and a few symmetries, so only a few isomorphic mappings. Although the general problem is NP-complete there is much active research in speeding up matching on the average or in special cases (Gold & Rangarajan, 1996; Gallip, 1986).

To take account of the shape of the plane graph the (x, y) coordinates of its nodes must be considered. An affine transform, equation 6.1, is found that minimises the distances between the coordinates of the mapped nodes for each of the isomorphic subgraphs. The advantage of this transform is its relative flexibility while having a simple form. The new value of each coordinate x', y' is just the linear combination of the old coordinates given by the coefficients $(a, \cdots, f)$. If the two graphs are of similar shape, the error of fit after the transform and the amount of distortion the transformation introduces will be small.

$$x' = ax + by + c \qquad y' = dx + ey + f \tag{6.1}$$

To measure the error of fit, the squared Euclidean distance between mapped nodes is used. Equation 6.2 shows this error measure where (x^e, y^e) are the coordinates of the extracted subgraph and (x^t, y^t) the coordinates of the isomorphic subgraph after the affine transformation. A weighting term is added that depends on the node. This is used, for instance, to emphasise the importance of accuracy of fit of "out" doorway nodes over corner nodes.

$$E = (\sum_n w_i((x_i^e - x_i^t)^2 + (y_i^e - y_i^t)^2))^{1/2} \tag{6.2}$$

Representing the affine transformation in homogeneous coordinates allows all transformations to be carried out by matrix multiplication (Zwillinger, 1998). The coordinates on the right hand side of equation 6.3 are for the nodes of the isomorphic subgraph, those on the left hand side for the nodes of the transformed subgraph.

$$\underbrace{\begin{bmatrix} x_1^t, x_2^t \dots x_n^t \\ y_1^t, y_2^t \dots y_n^t \\ 1, 1 \dots 1 \end{bmatrix}}_{\mathbf{X_t}} = \underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} x_1, x_2 \dots x_n \\ y_1, y_2 \dots y_n \\ 1, 1 \dots 1 \end{bmatrix}}_{\mathbf{X_i}} \tag{6.3}$$

The aim is to find the set of coefficients in matrix $\mathbf{C}$ of equation 6.3 that will transform the isomorphic subgraph so that its nodes best align with the extracted subgraph. This can be done by solving a matrix equation, equation 6.4 where the $\mathbf{W}$ is the weight matrix, containing the weights from equation 6.2, $\mathbf{X_e}$ the coordinates of the extracted subgraph and $\mathbf{X_i}$ the coordinates of the isomorphic subgraph.

$$\mathbf{X_i^T W X_i C} = \mathbf{X_i^T W X_e} \tag{6.4}$$

Ideally the transformed nodes would be positioned exactly over the mapped nodes, but this is not usually possible. Even with simple rectangular shapes the case base may not contain a graph with exactly the same doorway positions. Using a graph that is not an exact match will introduce some error in the composed function for the new task. By weighting some nodes more than others, where the error occurs can to some extent be controlled. An obvious aim is to minimise the introduction of errors that affect the overall path length. But of equal importance is to only introduce errors that normal reinforcement learning can easily correct. Figure 6.15 shows how differential weighting is used for fitting the L-shaped rooms of Section 3.2.2.

The left hand side of figure 6.15 shows the composite graph for the new task. The right hand side shows the result of overlaying it with a graph from the case base. If the fit at the "out" doorway of the outer L-shaped room is in error the robot will tend to miss the doorway and collide with the wall on one side. The farther the doorway is out of position, the longer normal reinforcement learning will take to correct the error. To encourage a good fit at the "out" doorway, a weight of 4 is used. Nodes adjacent to the "out" doorway are given a weight of 2, all other nodes have a weight of one. This is based on the intuition that more trajectories from different parts of the state space will be pass through the region close to the doorway. Any error here is likely to have a broader effect, and take longer for normal reinforcement learning

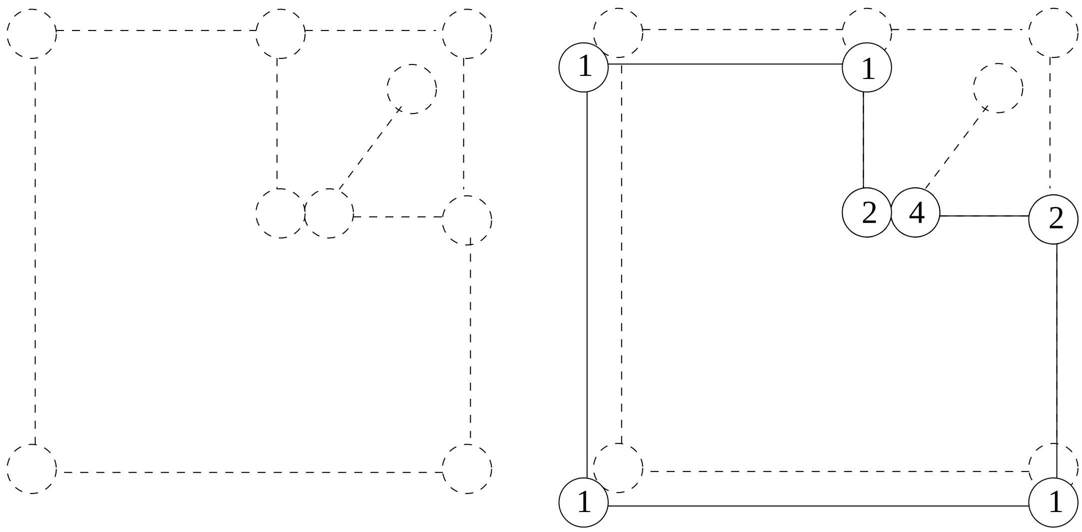

Figure 6.15: Weighting Graph Nodes

to correct, than in regions far from the "out" doorway. So the fit around the inner room is improved by sacrificing fit far from the doorway.

The outer L-shaped room in figure 6.15 has no "in" doorway but the inner room does. The exact position of this "in" doorway is not critical and its weight is set to 0.5. Whatever the position of the doorway the shape of the function will be correct inside the room. However, the further the doorway is from its correct position, the greater the error in the edge length. This will produce some error in the composed function as demonstrated in Section 7.3. But again the expectation is that this error will be small and reinforcement learning will quickly correct it.

The above procedure for fitting an isomorphic subgraph to the extracted subgraph is applied to each graph with a matching string produced by the labelling algorithm. Equation 6.2 is then used as a measure of the accuracy of fit. But before the best function can be selected the error introduced by the transformation must be determined.

6.2.2.2 Determining the Transformation Error

Not only should the fit be good but we would also prefer that the amount of transformation be small. All transformations produce some error and this is particularly true of asymmetric scaling as discussed later in this section. Generally the transform produces translation, reflection, rotation, shearing and independent scaling in each dimension. We can rewrite the affine transformation in the form of Equation 6.5.

$$x' = a\cos(\theta_x)x - a\sin(\theta_y)y + t_x \qquad y' = b\sin(\theta_x)x + b\cos(\theta_y)y + t_y \tag{6.5}$$

This transformation is a rotation (θ_x and θ_y allow a differential rotation which is a component in shearing), followed by a scaling (a and b allow asymetric scaling), followed by a translation (t_x and t_y allow translations in each dimension). In this form, it is easier to see the effect of different transformations and therefore penalise them. Unfortunately this decomposition is not unique, and different operations in different orders will produce the same overall transformation (Rothe, Susse, & Voss, 1996).

The rest of this section investigates a way of uniquely defining, and measuring, the overall transformation which is still intuitive appealing. In the robot navigation domain, the distance between points in the state space is just the normal Euclidean distance. The reinforcement learning function is an exponential decay in the distance to goal. If the transformation does not change the Euclidean distance, the transformed function should be directly applicable.

The affine transformation is just one family in a hierarchy of transformations. At the bottom of this hierarchy, shown in equation 6.6, are the symmetric transformations. The symmetric transformations, also known as rigid body transformations, applied to an L-shape are shown in figure 6.16. These transformations do not change the Euclidean distance.

$$Symmetric \subset Similar \subset Affine \subset Projective \tag{6.6}$$

The next step up the hierarchy introduces scaling, equal in each dimension. This will affect the Euclidean distance but only by a multiplicative factor. Thus the only

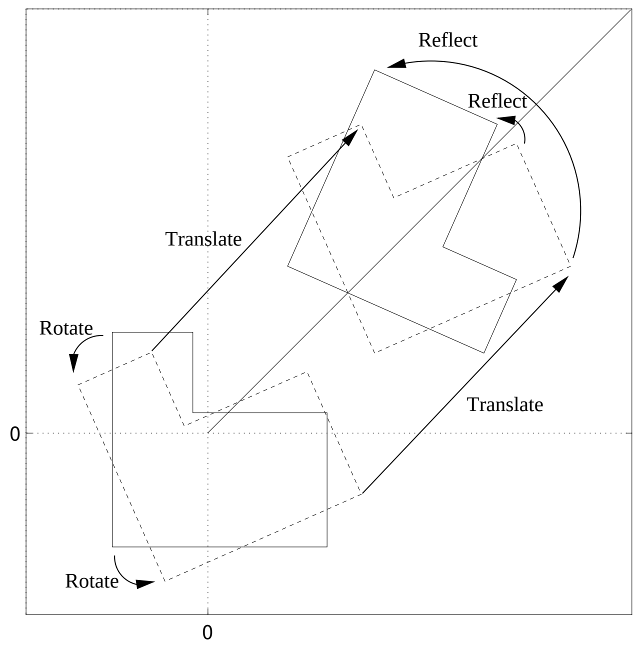

Figure 6.16: Symmetric Transforms

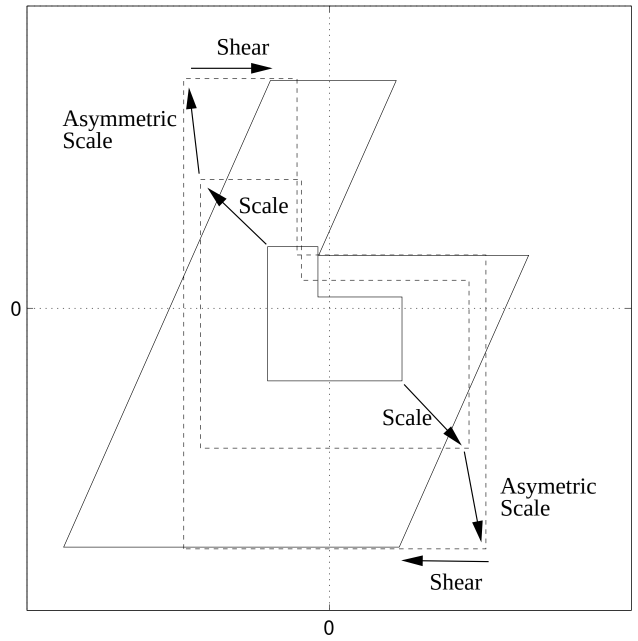

Figure 6.17: Affine Transforms

change needed to the transformed function is to scale the height. Rather than using the most general transformation in this hierarchy, the next level down is used the affine transformations. This allows the addition of asymmetric scaling and shear. Figure 6.17 shows the additional transformations over and above the symmetric ones. Asymmetric scaling and shear will distort the Euclidean distance, what needs to be determined is how much.

Ignoring the effect of translation, let us look at the result of applying the affine transformation to the unit circle, equation 6.7. The symmetric, rigid body, transformations will not alter the circle. But as shown in figure 6.18 the other transformations will. The symmetric scaling transform just changes the diameter of the circle. The asymmetric scaling and shear transformations change the circle into the ellipse shown in equation 6.8

$$x^2 + y^2 = 1 \tag{6.7}$$

$$\underbrace{(a^2 + b^2)}_{c_x} x^2 \underbrace{-2(ad + be)}_{c_{xy}} xy + \underbrace{(d^2 + e^2)}_{c_y} y^2 - \underbrace{(ae - bd)^2}_{c_r} = 0 \tag{6.8}$$

The amount of distortion of the Euclidean distance introduced by the transform can be determined by the ratio of lengths of the major and minor axes. The length of these axes can be calculated by finding the roots of equation 6.9 (Geary, Lowry, & Hayden, 1967), i.e. solutions for r^2.

$$\frac{c_r^2}{r^4} + (c_x + c_y)\frac{c_r}{r^2} + c_x c_y - c_{xy}^2 = 0 \tag{6.9}$$

The error of fit from the previous section is now combined with the transformation error using the lengths of the major and minor axes, r_{maj} and r_{min} respectively, of the ellipse. There is a penalty for Euclidean Distortion from asymmetric scaling and shear. The log factor is added directly to the error of fit as shown in equation 6.10. There is a much smaller penalty for scaling. Log factors are used so that the penalty functions are symmetric. If no isomorphic graph is found with a total error less than 1.5, a constant function will be used as a default.

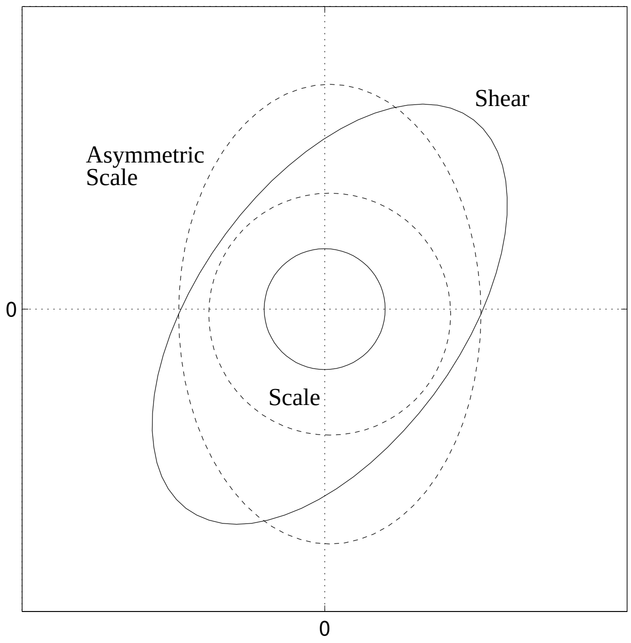

Figure 6.18: Affine Transforms on a Unit Circle

$$
\begin{aligned}
error &= sqrt(\textstyle\sum_i w_i(\Delta x_i^2 + \Delta y_i^2)) && \text{(node misalignment)} \\
&+ \left(\log_2 \left|\frac{r_{maj}}{r_{min}}\right|\right)^2 && \text{(Euclidean Distortion)} \\
&+ 0.05\left(\log_2\left(\frac{|r_{maj}+r_{min}|}{2}\right)\right)^2 && \text{(scaling factor)}
\end{aligned} \tag{6.10}
$$

At present there is no penalty for any of the symmetric transformations. In the robot navigation problem this is desirable. In the robot arm problem the different dimensions are not interchangeable, representing the angle of the shoulder and elbow joints. Transformations that have this effect should not be used. At present this is not explicitly prevented. It does not create a problem, however, as the only cases that match have the same orientation as required in the new task. For instance there are no subgraphs in the case base of the same shape but rotated ninety degrees. But this is unlikely to be always true and future work will investigate determining which symmetries hold in a domain and which transformations should be penalised or prevented altogether.

6.2.2.3 Transforming the Function

Finding the best matching subgraph produces a set of affine coefficients. These coefficients can be applied to the associated function transforming it to fit the new task. The function is presently represented by a box spline, essentially an array of values. As there is not an exact match between the extracted and the isomorphic subgraphs, there will not be an exact mapping from cells of the stored function to those of function for the new task. Where the new graph overlays the old graph, as shown in figure 6.15, values are assigned by using interpolation on the discrete values of the function. Where it does not extrapolation is used.

To interpolate or extrapolate, a set of cells is selected and their values used to determine the coefficients of a function which produces values for new points. Here, bilinear approximation is used. Bilinear approximation takes four values, one from each cell forming the corners of a square. This defines the set of coefficients in equation 6.11. Values for any other point in the state space can be found by solving the equation.

$$v = c_1 x + c_2 y + c_2 xy + c_3 \tag{6.11}$$

The only remaining problem is choosing the appropriate four cells. The cells have to be inside the polygon which delimits the function to be transformed. So the overall aim is choose the closest four cells inside the polygon. Figure 6.19 shows how this is done, the points to be approximated are marked by crosses. When interpolating the four nearest cells to the cross are selected. When extrapolating, the closest point on the polygon is found and the nearest four points on a grid centred on the cells is determined. Typically this defines some cells inside the polygon and some cells outside. The process then moves the four points in increments of grid size until all are within the polygon. The result is four cells inside the polygon, whose values determine the coefficients for extrapolation.

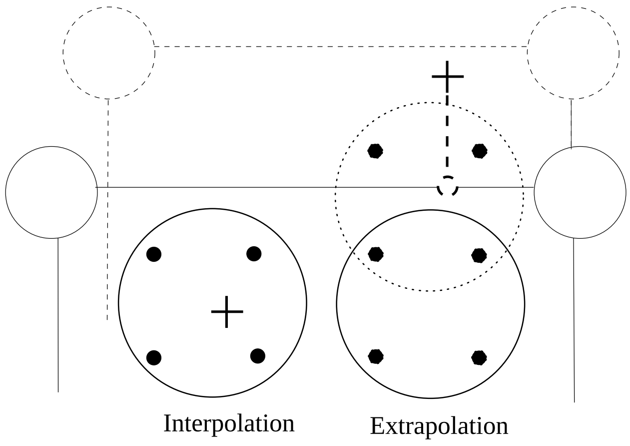

Figure 6.19: Interpolation and Extrapolation

6.2.3 Composing Search Functions

The previous sections discussed how to compose a new control function for a new task. This approach can also be used to compose a search function to help in locating the goal. When learning a new task the system has no knowledge about how to break it down into subtasks. So in section 3.2.2 the search function used was simply a constant value. This encourages exploration of new regions of the state space but gives no information about how to move between rooms. In the situation where the goal is moved the partition of the existing task can be used to construct a more efficient search function, in fact a series of search functions. Sutton and Barto (Sutton & Barto, 1998) argue that one of the critical aspects of reinforcement learning is the exploration exploitation trade off. Being able to construct search functions through planning gives a way to do this at much higher level.

The system composes a search function by assuming a particular room contains the goal. The procedure is very similar to composing a solution for a new task, except the function for the room is set to a constant value which is less than the normal goal value but higher than that of the surrounding function. This does not bias the search to any particular part of the room. It allows some limited learning to encourage exploration of the room without the possibility of being driven out of the room. Figure 6.20 shows the function assuming the goal is in the top left room in the

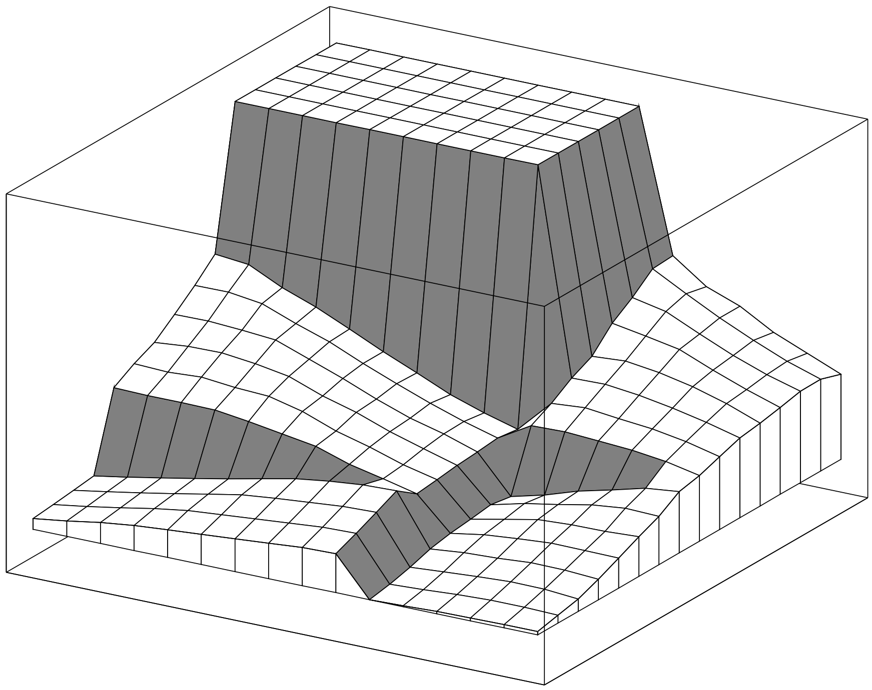

Figure 6.20: A Search Function

navigation problem of five rooms. The composed function drives the robot into the room from anywhere else in the state space. If it fails to find the goal after a fixed number of steps a new search function is composed with another room assumed to contain the goal. This guarantees that the search is equally distributed in each room rather than being mostly focussed in the room where the search started. This process is repeated until the goal has been located ten times to ensure a good estimate of the "centre of mass" of the goal. The "centre of mass" is used as the new position of the goal node in the composite graph.

6.3 In Summary

This chapter has discussed how the partition generated by the snake can be converted into a plane graph that represents a complex task. Individual subgraphs are tied to regions of the function delimited by the partition and represent solutions to subtasks. These solutions together with their associated subgraphs are stored in a case base. This chapter has shown how a process much like symbolic planning can compose functions from the case base to produce the solution to a new task. It has demonstrated how the most appropriate functions are selected and transformed to fit the new task. The planning process can also compose functions that serve other purposes, such a searching for the goal when its location has changed.

The algorithm shown in figure 6.21 summarises the steps taken in the planning process. To emphasise the relationship between the planning process and a basic shortest path algorithm, Pearl's "best first algorithm" (Pearl, 1984, p48) has been followed as closely as possible. Some steps have been deleted as Dijkstra's algorithm uses a simple distance measure from the start node rather than a more general heuristic. It is important to note that the start node in this algorithm is actually the "goal" node in the graph representing the solution to the task. Steps from the basic algorithm are shown as normal text, additional steps taken in the planning process are shown in a bold font. Section numbers indicate where a more detailed explanation of the related steps can be found.

1. Put the start node s on a list called OPEN of unexpanded nodes.
2. If OPEN is empty, **exit (planning completed)**.
3. Remove from OPEN a node n at which **the distance from the start node** s is minimum, and place it on a list called CLOSED to be used for expanded nodes.
4. Expand node n, **its successors are the doorway nodes, other than** n**, sharing the subgraph not containing** n**'s predecessor**.
   - **(a) Update or add edges to indicate that** n **is the predecessor of all its successors.**
   - **(b) Extract the common subgraph with its new or updated edges.**
   - **(c) For all the isomorphic subgraphs in the case base and for all possible isomorphisms find the most accurate affine transformation. Apply the best transformation to the associated function and compose with existing functions (Sec 6.2.2).**
5. For every successor n' of n:
   - (a) Calculate **the distance from the start node** s **to** n' **through** n.
   - (b) If n' was neither on OPEN nor on CLOSED, add it to OPEN. Assign the newly computed **distance** to node n'.
   - (c) If n' already resided on OPEN, compare the newly computed **distance** with the value previously assigned to n'. If the new value is lower, substitute it for the old (n' now points back to n instead of its previous predecessor). **Repeat step 4 once using node** n' **instead of** n **and then go step 2 (Sec 6.2.1)**.
6. Go to step 2.

Figure 6.21: The Planning Algorithm

Chapter 7

Experiments

Previous chapters in this thesis have addressed the individual processes that constitute the system and demonstrated their effectiveness at their alloted activities. This chapter brings together those processes into a coherent whole. It will present a series of experiments, demonstrating the effectiveness of the system at speeding up the learning of control tasks. It begins by describing the experimental set up. It then details experiments in two different learning situations and two different domains. It concludes with an analysis of the experimental results and discusses the implications for the overall approach.

7.1 Experimental Set up

This section, while discussing the experimental set up, also shows how the individual processes interact in the learning of control tasks. Figure 7.1 shows the experimental steps overlayed on a simplified system architecture. Before the experiments begin the case base is created using the following steps. In step A, Q-learning is applied to a selected task. A reward of one is received on reaching the goal. This value, discounted by the expected distance to the goal, propagates backwards until the learning function has converged. In step B, snakes group features in the reinforcement learning function, forming a partitioning of the state space into individual subtasks. Each region of the function associated with a subtask and its related subgraph are loaded into the subtask library, the case base. This is repeated across all tasks for the particular type of learning problem for the particular domain.

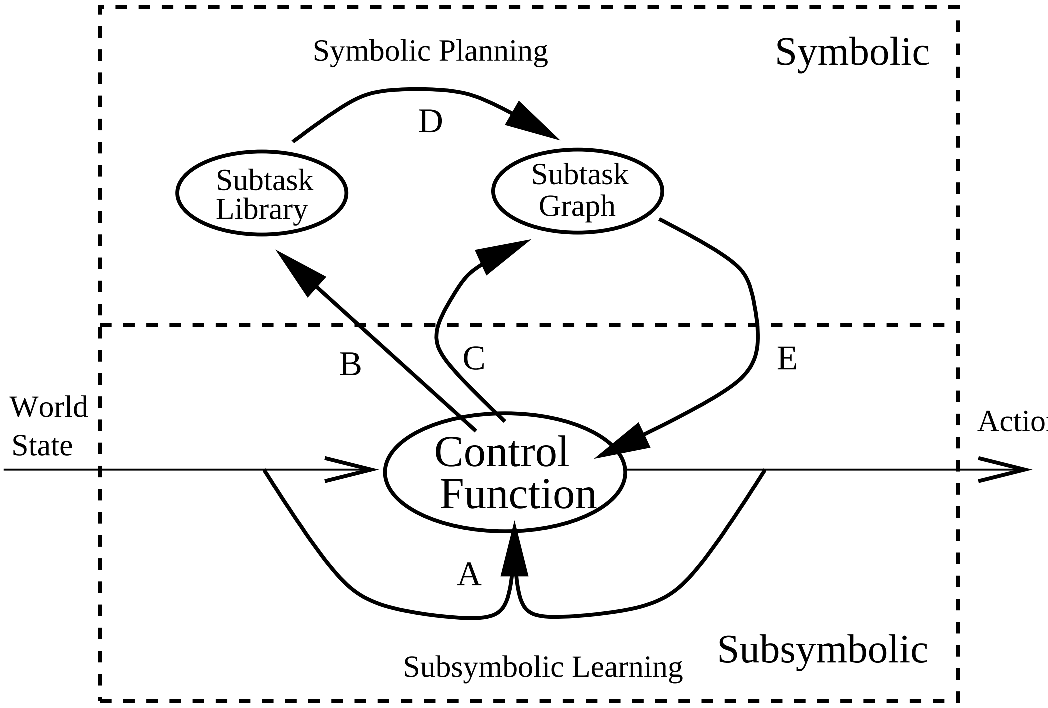

Figure 7.1: Steps in the Experiment

Generally the experiments measure the speed of function composition and the quality of the solution produced. There are differences in the experimental procedure for the different types of learning problem. But in all experiments the following cycle is used. In step A, some initial learning is done on a new task. In step C, the function is partitioned, the goal and doorways located and a graph built representing the whole task. In step D, symbolic planning regresses backwards down the edges connecting doorways to the goal. At each stage the appropriate subgraph is extracted and matched with those indexing the case base. A plan is constructed with the best matches. In step E, the transformations applied to the subgraphs are applied to functions representing subtasks. They are composed to form a solution to the new task which is used to initialise the control function. Step A is then repeated, with learning further refining the function.

This cycle is used a number of times to produce a series of different control functions. All bar the final solution can be viewed as search functions. In the experiments on moving the goal, search functions are used to determine the new goal position. The series of functions, discussed in section 6.2.3, guide the system into each of the rooms in turn where a constant function encourages exploration of the room itself. In the experiments on learning in a new environment, search functions are used to determine the position of the goal and the features. Firstly the goal is located using a constant value function, discussed in section 3.2.2. Once the goal is located, a partition is produced with a single region covering the whole state space. The resultant graph matches a solution in the case base when there are no walls, in other words the state space is a single room.

This cycle is not always run to completion. If executing step C does not produce a consistent composite graph, then the cycle terminates and is run again after a fixed number of steps. To be consistent, the doorways from different subgraphs must align (a small error of 0.1 is allowed) and the graph must overlay the complete state space. This is of particular importance when learning in a new environment. Generally the snake will filter out features that are too small and not well formed. But as the features become stronger not all snakes may accurately capture the features. When there is this degree of uncertainty in the how the task should be partitioned, the present approach is to allow Q-learning to further refine the function and then repeat step C.

The baseline algorithm and the underlying learning algorithm for the function composition system is the basic Q-learning algorithm. A discrete function approximator was used, realised by B-spline box functions as discussed in section 3.1.1. Why higher order splines were not used will be discussed in section 7.3.3. The learning rate α is set to 0.1, the greedy policy uses an ϵ of 0.1 (the best action is selected about 91% of the time) and a reward of 1.0 is received for any action inside the goal. Although the state spaces for the different domains represent two quite different things - the robot's $< x, y >$ location and the angle of the arm's two joints - the actual representation is the same. The state space ranges between ± 1 for each dimension. A step is ± 0.25 or zero in each dimension giving nine possible actions. The actions are stochastic, a uniformly distributed random value between ± 0.125 being added to each dimension of the action. The discount factor γ is 0.8 raised to the power of the step size, the Euclidean sum of the size in each dimension or 0.25 which ever is the larger.

In the robot navigation examples if the robot hits the wall it is positioned a small distance from the wall along the direction of its last action. This has not been implemented for the robot arm as it is a somewhat more complex calculation. Instead, if a collision with an obstacle occurs the arm is restored to its position before taking the action.

7.2 Experimental Results

This section compares learning curves for function composition and a simple baseline algorithm. Four sets of results are presented. The first two demonstrate learning in a fixed environment when the goal is moved, the second two learning in a new environment. Each type of learning activity is first applied to the robot navigation domain and then to the robot arm domain. The learning curves represent the average number of steps to goal as a function of the number of steps taking during learning. The average is across 64 different start positions distributed uniformly throughout the state space. The maximum number of steps for each start location is 2000. If a trial takes 2000 steps and has not yet reached the goal it is stopped and the distance to goal recorded as 2000.

Speed up is calculated by dividing the number of learning steps at one specific point on the baseline learning curve by the number of learning steps at an equivalent point on the function composition system's learning curve. The knee of the function composition system's curve occurs where the low level learning algorithm is initialised with the composed function. The main point of comparison is the approximate position of knee of the baseline curve.

7.2.1 Robot Navigation, Goal Relocation

The first experiment investigates the time taken to correct a learnt function when the goal is relocated in the robot navigation domain. The system establishes that the goal has moved by determining that it is no longer at the maximum of the existing function. Due to some uncertainty in the exact boundary of the goal, this is required to occur ten times with no intervening occurrence of the goal being detected at the maximum.

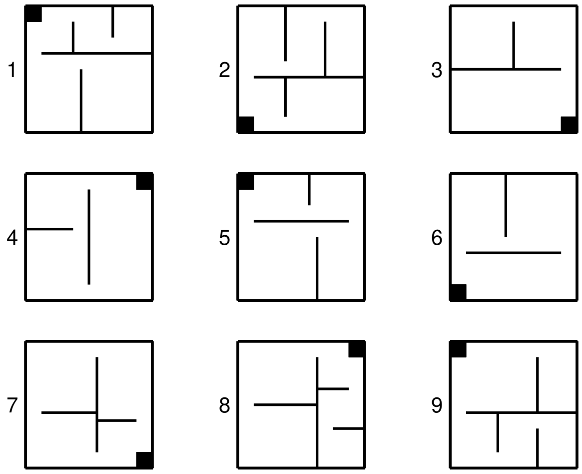

Figure 7.2: The Different Suites of Rooms

There are nine different room configurations as shown in 7.2, the number of rooms varying from three to five, and there are four different goal positions. Each room has one or two doorways and one or two paths to the goal. To initialise the case base, a function is learnt for each of these configurations with the goal in the position shown by the black square. The rooms were generated randomly with some constraints on the configuration of the rooms and doorways, discussed in greater detail in section 7.3. The case base also included functions generated for the experiments discussed in section 7.2.3. This was necessary to give a sufficient variety of cases to cover most of the new tasks. Even with this addition not all subgraphs were matched. Constant valued default functions were used when there was not a match. This reduced speed up significantly but did not eliminate it altogether.

Once the case base is loaded the basic Q-learning algorithm is then rerun on each room configuration with the goal in the position shown. After 400,000 steps the goal position is moved to one of the three remaining corners of the state space, a task not included in the case base. Learning continues for a further 300,000 steps. At fixed intervals, learning is stopped and the average number of steps to reach the goal is recorded. The curves in figure 7.3 are the average of 27 experimental runs, three new goal positions for each of the nine room configurations. Zero on the x-axis is where the goal is moved.

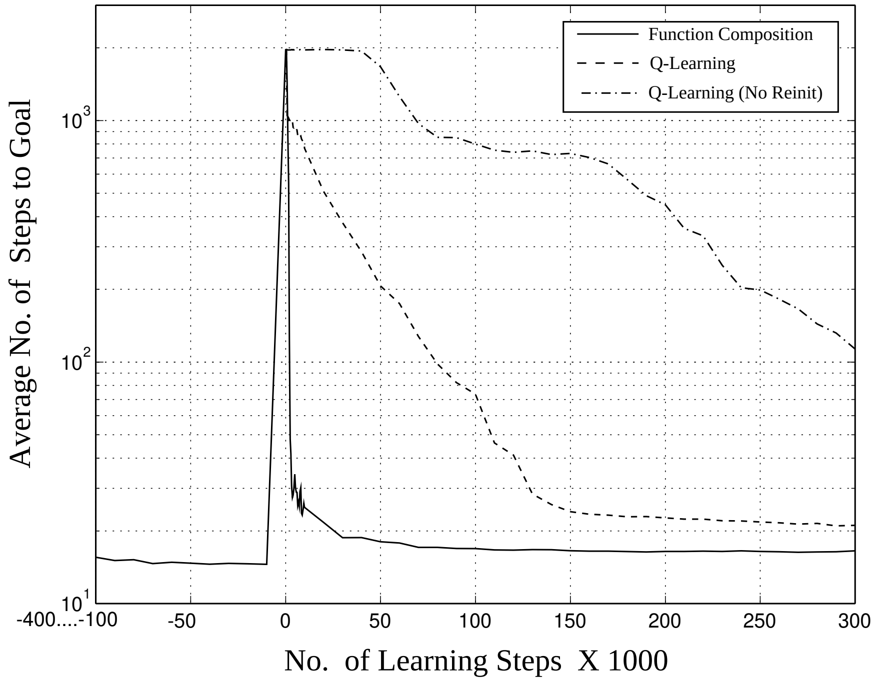

Figure 7.3: Learning Curves: Robot Navigation, Goal Relocation

The basic Q-learning algorithm, the top curve of figure 7.3, performs poorly because when the goal is moved the existing function pushes the robot towards the old goal position. A variant of the basic algorithm reinitialises the function to zero everywhere on detecting that the goal has moved, it being no longer at the maximum of the existing function. This reinitialised Q-learning, the middle curve, performed much better.

The function composition system, the lowest curve, performed by far the best. The precise position of the knee of this curve is difficult to determine due to the effect of using default functions. If only those examples using case base functions are considered, the knee point is very sharp at about 3000 steps. The average number of steps to goal at 3000 steps for all examples is 40. The non-reinitialised Q-learning fails to reach this value within 300,000 steps giving a speed of over 100. The reinitialised Q-learning reaches this value at about 120,000 steps giving a speed up of about 40. Function composition generally produces accurate solutions. Even if some error is introduced further Q-learning quickly refines the function towards the asymptotic value of about 17. After about 150,000 steps normal Q-learning reaches an average value of 24 steps and then slowly refines the solution to reach an average value of 21 after 300,000 steps.

7.2.2 Robot Arm, Goal Relocation

The second experiment is essentially a repeat of the first experiment but in the robot arm domain. The initial number of steps before the goal was moved was reduced to 300,000 to speed up the experiments. As the arm is only two degrees of freedom and with the restrictions discussed in section 3.2.3.1 the number of variations is small. So only three obstacle configurations were used, constructed by hand, with two obstacles in each. To increase the number of experiments, to allow for greater statistical variation, each configuration was repeated with the goal in each of three possible positions, as shown in figure 7.4. The black diamonds represent the obstacles, the black rectangles the goal. Solutions to all these tasks were loaded into the case. When composing a function, however, the system is prevented from selecting a case that comes from the same goal and obstacle configuration.

The curves in figure 7.5 are the average of 18 experimental runs, two new goal positions for each of the three original goal positions in the three obstacle configurations shown in figure 7.4. There are only two learning curves, non-reinitialised Q-Learning being dropped. As in the first experiment the function composition system, the lower curve, performed much better than Q-learning. The knee of the function composition system occurs at 2000 steps, the knee of Q-learning at 50,000 steps giving a speed up of 25. In this experiment the case base contained subgraphs that matched for all new tasks, so default functions were not needed. The composed functions tend to be very accurate and little further refinement is necessary.

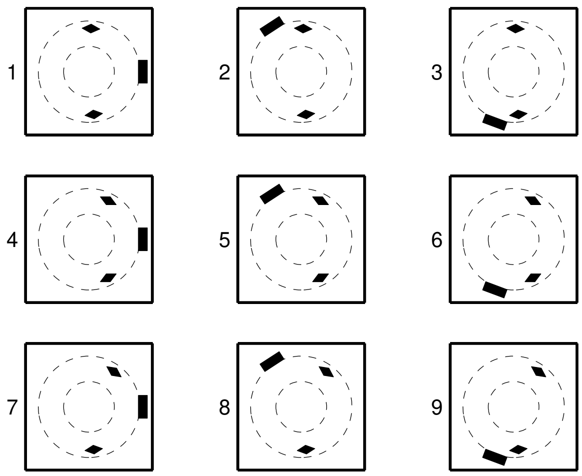

Figure 7.4: The Robot Arm Obstacle and Goal Positions

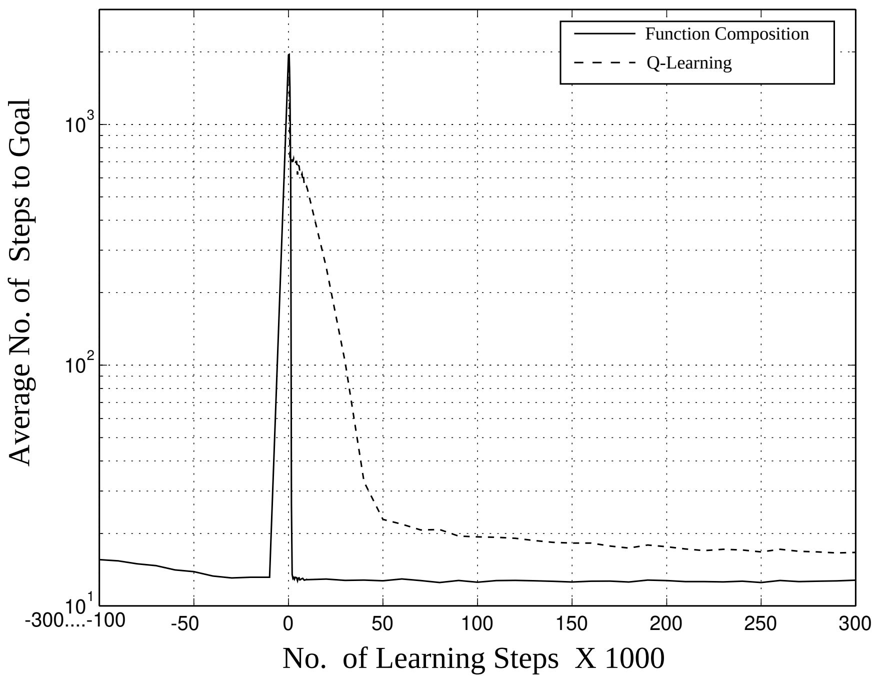

Figure 7.5: Learning Curves: Robot Arm, Goal Relocation

7.2.3 Robot Navigation, New Environment

The third experiment investigates the time taken to learn in a new, but related, environment in the robot navigation domain. Nine different inner rooms were generated randomly, again under some constraints. All have a single doorway but the size and position of the room and the location of the doorway are varied as shown in figure 7.6. To initialise the case base, a function is learnt for each of these configurations with the goal inside the small room as indicate by the dark square. Learning is then repeated on each of the room configurations in turn. However, when composing the new function the system is prevented from selecting a case learnt from the same goal and room configuration. Experimental runs for the Q-learning algorithm and the function composition system are initialised with a flat function of zero and 0.75 everywhere respectively, denoted as zero on the x-axis. Learning continues for 100,000 steps. To improve the statistical variation, experiments for each configuration were repeated three times, each time with a new random seed. The curves in figure 7.7 are therefore the average across 27 experimental runs.

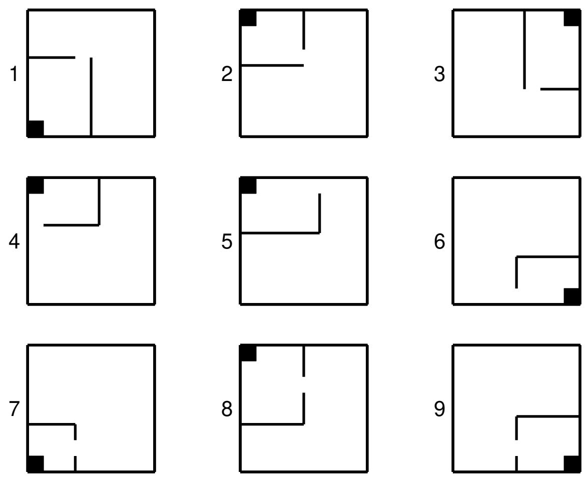

Figure 7.6: The Single Rooms

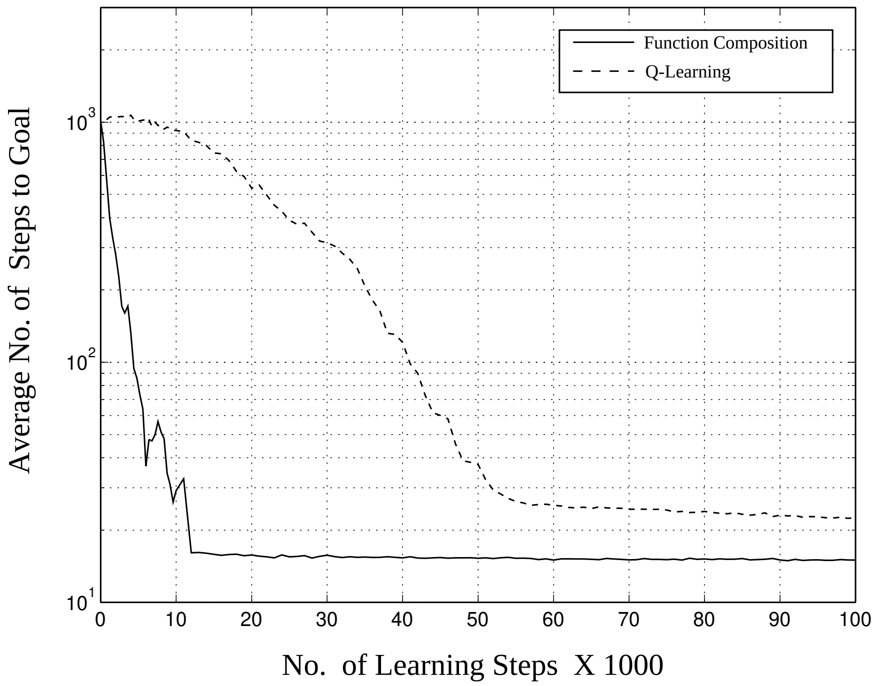

Figure 7.7: Learning Curves: Robot Navigation, New Environment

The top curve is the Q-learning algorithm, the bottom curve the function composition system. For these experiments locating the goal took typically between 400 and 1200 steps although some took 2000 steps. The function composition system then introduces the "no walls" function and typically a further 800 to 4000 steps are taken before usable features are generated. Again certain experimental runs took longer, this will be discussed in section 7.3. Due to these runs the knee of the function composition system's curve occurs at 12000 steps. The knee of the basic Q-learning curve occurs at approximately 54000 steps giving a speed up of 4.5. As in previous experiments once initialised the function is very accurate and little further refinement is necessary. Basic Q-learning on reaching the knee takes a long time to remove the residual error.

7.2.4 Robot Arm, New Environment

The fourth experiment is essentially the same as the third experiment except in the robot arm domain. Here three, hand crafted, configurations of a single obstacle with the goal in a fixed position were used, as shown in figure 7.8. To increase the statistical variation each configuration was run five times with a different random seed. The curves in figure 7.9 are therefore the average across 15 experimental runs.

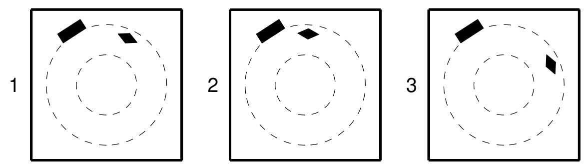

Figure 7.8: The Different Obstacle Positions

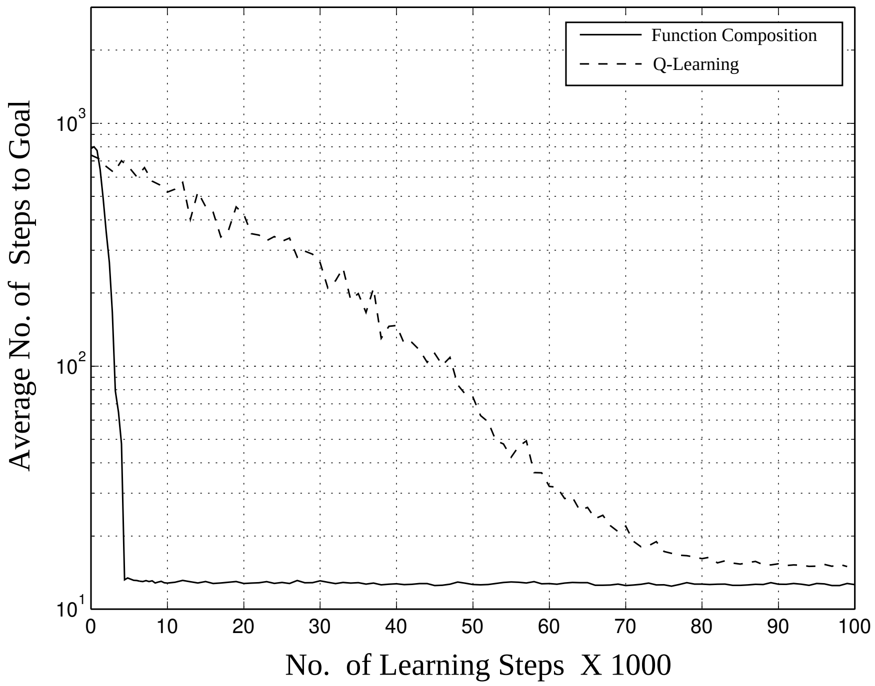

Figure 7.9: Learning Curves: Robot Arm, New Environment

The top curve of figure 7.7 is the Q-learning algorithm, the bottom curve the function composition system. The knee of the function composition system's curve occurs at about 4400 steps. The knee of the basic Q-learning algorithm at about 68000 steps giving a speed up of about 15.

7.3 Analysis of Results

The experiments of this chapter have shown that function composition produces a significant speed up across two different types of related task and across two domains. In addition, the composed solutions tend to be very accurate and little further refinement is required. This section begins by looking at possible limitations of the experimental set up. It looks at the difficulties with the doorway detection and how these were manifest in these experiments. It then looks at situations where function composition was the most and the least successful.

7.3.1 Limitations of the Experimental Set Up

The speed up obtained using function composition is sufficiently large that small variations in the experimental set up are unlikely to affect the overall result. Nevertheless there a number of concerns that might be raised about the experimental methodology. Some will be, at least partially, addressed in this section and others will be the subject of future work.

The first concern might be how the estimated value of speed up is measured. The value is a measure of the speed up of the average of a set of learning tasks, rather than the average of the speed up in each of the tasks. One of the difficulties of estimation with curves for single tasks is that the average distance to goal may oscillate up and down as learning progresses even though the general trend is downwards. This makes judging the position of the knee of the curves difficult and any estimate of speed up questionable. Even experimental runs using the same configuration but with different random seeds exhibit a considerable variation. It is likely in some instances that the speed up measured on individual curves would benefit the function composition system, in others the baseline algorithm. Nevertheless probably overall most of these effects will cancel out.

The second concern might be the effect on speed up of the limit of 2000 steps when measuring the distance to the goal. Comparing two averages of values limited in this way is sometimes misleading (Gordon & Segre, 1996). But this limit primarily affects only the baseline algorithm, and was only significant when the goal was moved and the function not reinitialised. As estimation of speed up is principally concerned with comparing the position of the knees of the different curves, where the average distance to goal is relatively small, this is likely to have little effect.

The third concern might be that the value of speed up is very much dependent on the baseline algorithm used. Certainly, experience arising from work in this thesis has shown that variation in how the function is initialised and how actions are selected can have an impact on the speed of learning. The initial function for the baseline learning algorithm used in the experiments was zero everywhere. In this author's previous work (Drummond, 1998) the function was initialised to a constant value of 0.75 and tie breaking between actions of the same value was achieved by adding a small amount of noise (circa $\pm 5 \times 10^{-5}$). Using zero instead of 0.75 increased the speed up for the goal relocation experiments in the robot navigation domain. But in the other experiments it had the opposite effect. Generally using zero improved the learning rate of the baseline algorithm on the easier tasks, while degrading performance on the more difficult tasks. It was then discovered, however, that replacing the noise factor with a strict tie-breaker, randomly selecting amongst actions with the same value, produced a significant speed up in the baseline learning algorithm. So this set up was used for the preceding experiments, but this did not always benefit the baseline algorithm.

Figure 7.10 shows two baseline learning curves for one configuration in the robot arm domain. The upper curve occurred during the experiments for moving the goal. As it had such a large impact on the average learning curve, it was replaced by the lower curve produced by repeating the experiment with a different random seed. This very slow learning rate arises from the interaction of the partial observability of the robot arm domain with the use of an initial value of zero. The top of figure 7.11 shows one of the problems with the goal indicated by the shaded oval. The dotted lines demark the cells in the box function representation. The bold dashed lines are the "walls" produced by mapping the obstacles into configuration space. A few of the box functions straddle the "walls" and can be updated by actions on either side. The region delineated by the circle has been expanded on the lower left of the figure and shows one of these functions indicated by the solid square. Contour lines of the value function early in the learning process are shown on the lower right of figure 7.11. The function close to the goal is quite high as it grows upwards towards the goal value. The value leaks through the "wall" to the other side, via the straddling box function, causing greedy actions to move the arm towards the obstacle. Only a reasonably long series of non-greedy actions will allow learning to find the correct path to the goal.

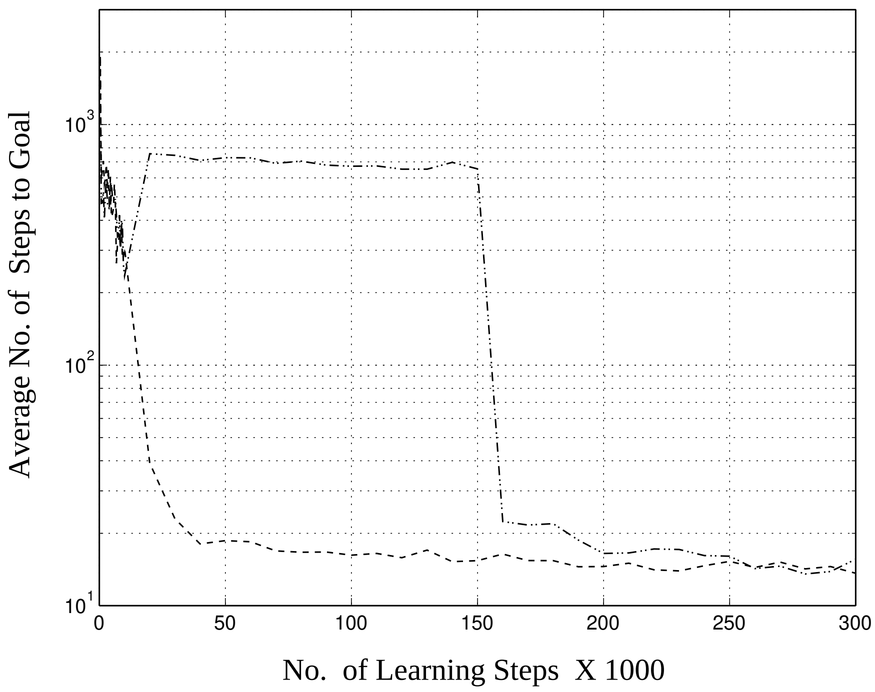

Figure 7.10: Learning Curves in a Partial Observable Domain

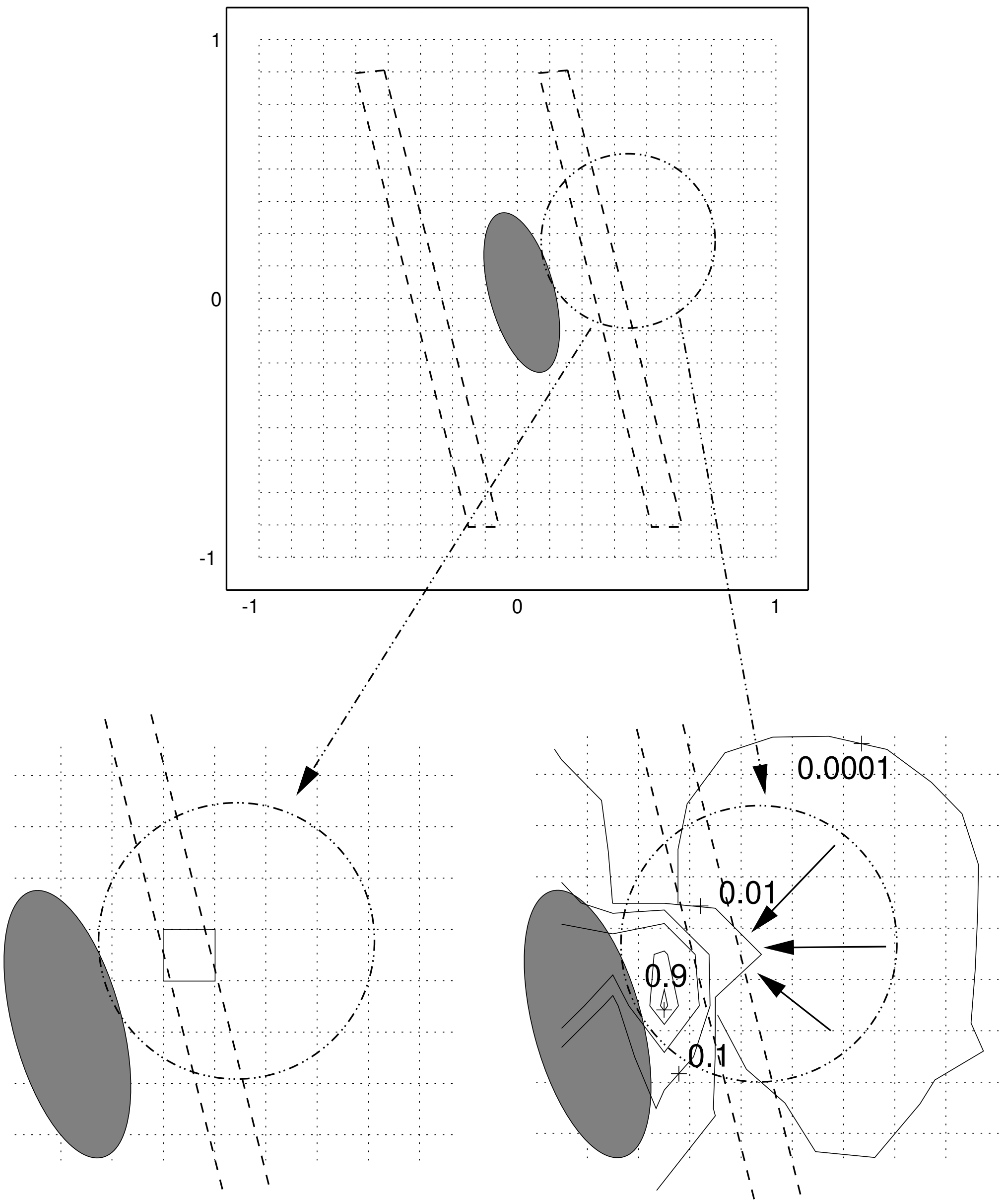

Figure 7.11: Value “Leak Through”

In early experiments the obstacles were small, with more cells straddling the "walls". Yet with Q-learning initialised to a value of 0.75, no problems were observed. It is probable that local learning quickly corrected any "leak-through" and drove the system away from these states. When this value was changed to zero, the problem appeared. To attempt to alleviate this, the size of the obstacles was increased. Again initial experiments indicated no apparent problem. Only in these recent experiments using strict tie-breaking has the problem reemerged. The difficulty that arises when starting with a zero value is that once an action receives some value it will remain the best action for some time. Continual update of this action will push this value down, but it can only asymptotically approach zero. So until other actions for the same state are updated it will always be selected as the greedy action. It may well be that in these sorts of domain, where there is some degree of partial observability, small initial values are better than zero or some means of improving exploration for very small values might be necessary.

This problem did not occur in the robot navigation domain as the position of the walls was carefully chosen to sit between the box functions representing the reinforcement learning function. So there was no mismatch between the representation and the environment to create partial observability. This was the main reason higher order splines were not used the experiments, as they would have produced such a mismatch. Having a mismatch seems to have little effect on function composition but does, at least occasionally, have a serious impact on the baseline learning algorithm. With more randomly positioned walls there would be a mismatch with box functions and higher order splines might well produce a better representation.

Other variations in the parameters of the baseline algorithm have not been explored in this thesis. For instance, a constant learning rate of 0.1 was used. Alternatives, such as starting with a higher rate and reducing it as learning progresses might also improve the overall speed of the baseline algorithm. The Q-learning algorithm used is the most basic and a more sophisticated one would unquestionably reduce the speed up experimentally obtained. For instance, some form of reinforcement learning using eligibility traces (Singh & Sutton, 1996) might be used. For the experiments when the goal was moved, a baseline such Dyna-Q+ (Sutton, 1990) which was specifically designed to deal with changing worlds would probably be a better reference point.

7.3.2 Limitations with Doorway Detection

This section looks at the limitations of the doorway detection algorithm and how the present system deals with them. Future work addressing these limitations will be discussed at the end of this section and in section 9.2.1. The algorithm locates doorways by searching for local minima along the ridges of the gradient of the reinforcement learning function. The main difficulty arises early in the learning process. The "no walls" function might be also imagined as a "doorways everywhere" function. As the features emerge, points along the ridge not visited frequently have a gradient that has not changed significantly from the "no walls" value. This may give the appearance of wider doorways or there being doorways where there are, in fact, none.

If a doorway appears a bit wider than it should be, the detection algorithm may offset it, producing some error in the composed function. Typically this does not produce a large error and normal reinforcement learning quickly eliminates it. In one of the experiments, however, a relatively small mispositioning of the doorway had a more significant effect. The left hand side of figure 7.12 shows one of the room configurations from section 7.2.3. In one of the experimental runs, the feature representing the lower wall had not completely emerged when the partition was generated. This produced what looked to be a shorter wall, as shown in the middle of the figure. The position of the doorway appears to be almost exactly at the corner. The algorithm, in fact, positioned the doorway just on the wrong side of the corner, and composed a function for the configuration shown on the right of this figure.

Figure 7.13 shows various learning curves for this configuration. The solid line that drops gradually and appears noisy is produced by this error. For comparison, the two solid lines that drop sharply are for the correct composition, produced using different random seeds. The dashed curves are for three different seeds for the baseline algorithm. The composed function still produced speed up. But it is unclear why reinforcement learning took so long to correct what seems, on the surface at least, to be a local error. Reiterating the point first made in section 6.2.2.1, it is important to

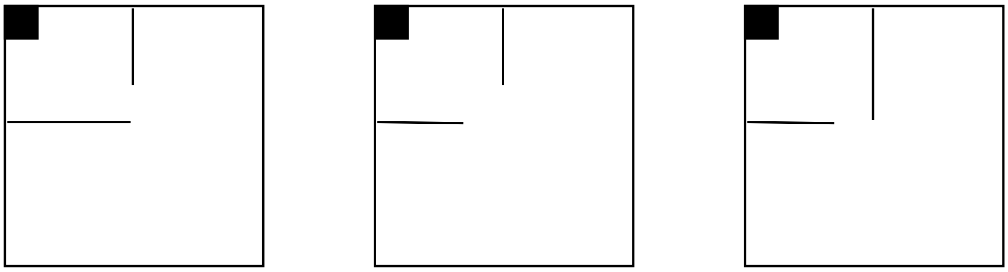

Figure 7.12: A Wrongly Positioned Doorway

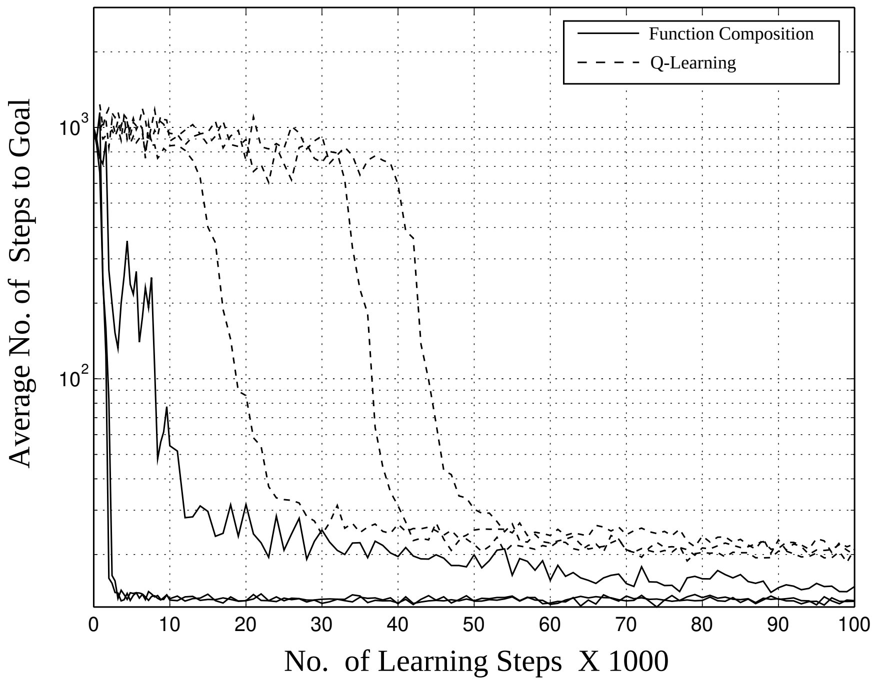

Figure 7.13: Effect of Doorway Positioning on Learning Curves

study what errors are easy, and what errors are hard, for reinforcement learning to eliminate. Systems designed to speed up reinforcement learning should concentrate on avoiding the hard errors.

At present two things prevent the problem of false doorways ultimately leading to an incorrect composition. Firstly, what often occurs is that the doorway detection algorithm associated with one snake disagrees with that associated with another snake. This is due to different normalisation and smoothing in the two processes. As the graph is not consistent, the system waits for further refinement of the function and often the problem is eliminated. Occasionally, however, the graph does include a false doorway. This might produce a composed function for the room configuration shown on the left hand side of figure 7.14 when learning the room configuration on the left hand side of figure 7.12. Such a function would drive the robot towards a false doorway for paths starting in region A of the state space until normal reinforcement learning corrected the error. This occurs infrequently, but at present does not impact the learning curves as there is no appropriate solution in the case base and the function does not get updated. A better way of addressing this problem will be proposed at the end of this section.

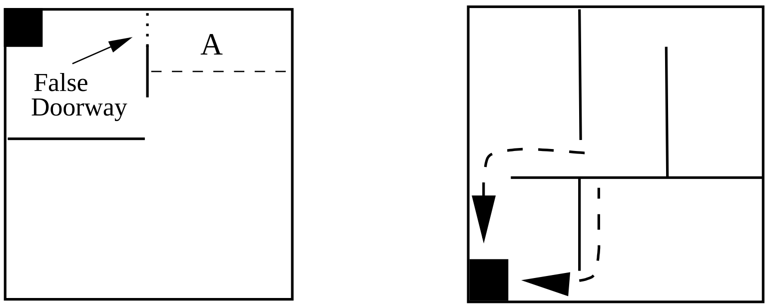

Figure 7.14: False Doorways

When dealing with a largely converged function, such as when the goal is moved, false doorways are unlikely to occur. But they may still happen when the distance to goal from either side of a wall is approximately equal. An example, taken from the suites of rooms in section 7.2.1, is shown on the right hand side of figure 7.14. The dashed lines are two paths to goal of approximately the same length. The gradient separating the start of each path is a local minimum and relatively small. The simple scaling and thresholding does not eliminate it as a possible doorway. An extra filter, discussed in section 6.1.1, which samples the gradient a short distance from the doorway is intended to eliminate this problem. This distance was set at two and a half times the width of the underlying box functions which was effective on the hand crafted configurations. This proved insufficiently close for this particular configuration and the length was reduced to the width of one box function to eliminate the problem. This was the only change made to the system as a result of the randomly generated configurations.

Generally to be able to plan successfully, the system must locate true doorways, and only true ones, reasonably accurately. False doorways are, in fact, the least problematical. Let us suppose that a false doorway is included in the plan and the appropriate function composed. At present the system only produces a new function once. But a simple extension is to continue to observe the learning function on the new task. The composed function would try to push the system through the non-existent doorway. Observing the gradient along the partition, it would become quickly apparent that this doorway was blocked. A new function could then be composed without the doorway.

What is more problematical is if the system fails to detect a true doorway. In the present tasks this is unlikely occur. Moving through doorways incurs no additional cost. They are therefore always used from at least some part of the state space, unless the doorway contains the goal or a decision boundary. If there is a cost associated with going through a doorway, solving one task might avoid the doorway altogether. It would then not be apparent in the reinforcement learning function, but it would be important in a related task. The composed solution would work but it might be far from optimal. This shows one weakness of using the value function of a single task to determine the structure of the environment. One way of dealing with this is to combine structural information from multiple tasks in the same environment. But this would lessen the opportunities for transfer. An alternative approach will be discussed in section 9.2.1

7.3.3 Performance Variation with Room Configuration

From the first set of experiments, discussed in section 7.2.1, it is clear that some sort of special response is needed when there is a significant change in the world such as the goal moving. Relying on normal reinforcement learning to correct the existing function to fit the new task is likely to take a long time. In the experiments, the probability that an exploratory step, rather than a greedy step, is taken was kept fixed at 0.1. Increasing this value would go some way towards mitigating this effect. But to successfully reach the new goal would likely require a long series of non greedy steps. To achieve this, the exploration rate might have to be increased substantially, again raising the spectre of the possibility of divergence as discussed in chapter 4. Detecting that the goal has moved is a relatively simple task. Reinitialising the function to zero produces a significant speed up but the speed up produced by function composition is much larger.

The only change in the world this thesis has investigated experimentally, is moving the goal. Other significant changes like the opening and closing of doors would also be problematical to basic reinforcement learning. Having knowledge of the goal position is a very simple of model of the world. Detecting other significant changes in the world requires a more complex model. The partitioning information extracted by the snake is a simple form of model and lends itself to easily detecting these other changes. Future work exploring this idea will be discussed in section 9.2.1.

The top of figure 7.15 shows learning curves that are the average of the three configurations of section 7.2.1 that were most difficult for the baseline algorithm and where the most speed up was obtained. The bottom of figure 7.15 shows these configurations with the new goal position, the numbers refer to the original configurations shown in figure 7.2. The configurations are all very similar, consisting of five rooms, with a single path to the new goal located in a small room. The knee of the function composition curve is at 2400 steps and the knee of the basic Q-learning curve at 150,000 steps giving a speed up of just over 60. Generally the more rooms there are and the fewer alternative routes to goal the longer basic reinforcement learning will take and the more speed up will be obtained by function composition.

The top of figure 7.16 shows the average of four learning curves when function

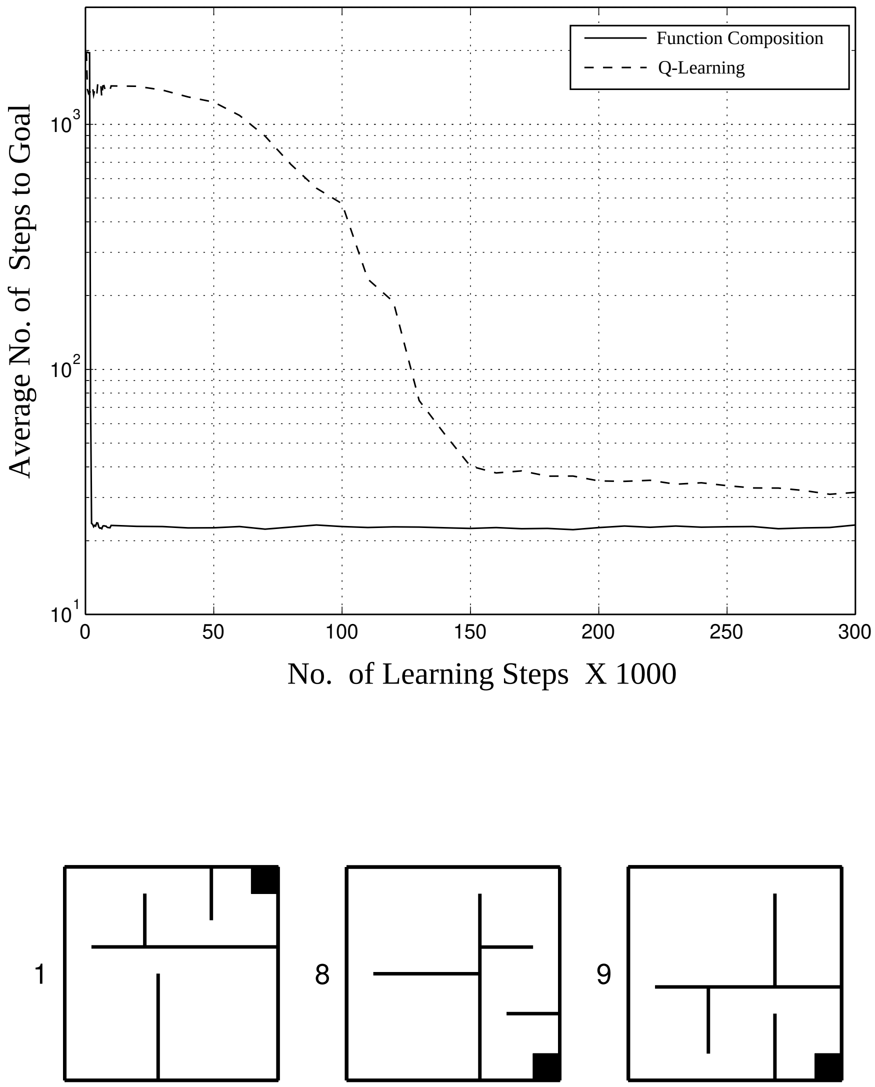

Figure 7.15: Success in Robot Navigation Moving Goal

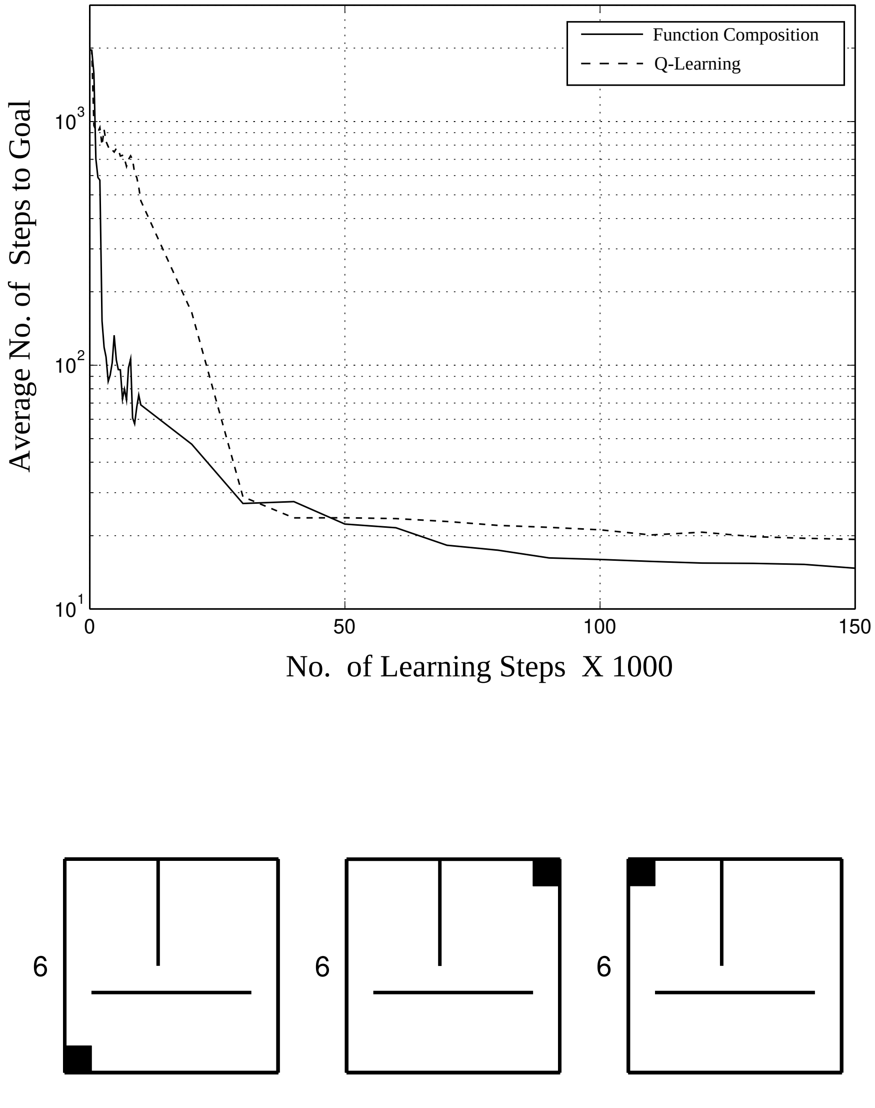

Figure 7.16: Failure in Robot Navigation Moving Goal

composition does the worst. The bottom of figure 7.16 shows one of the room configurations that produced these curves. On the far left is the original problem, on the right are the two goal positions that are problematical. There are two reasons for this. Firstly, the configuration consists of three rooms and there are two paths to goal. This makes it one of easiest tasks from the experimental set for the baseline algorithm. Secondly, a solution for the lowest room, with two doors at the top, does not exist in the case base. There are no isomorphic subgraphs of this form. Rather than not composing a solution, the system introduces a constant value function for this room. This room represents almost half the state space and so much additional learning is required. As the top of figure 7.16 shows initially there is significant speed up. Further refinement reduces the advantage and in fact for a short while the baseline algorithm is better. But later function composition gains the upper hand and converges more quickly than the baseline algorithm towards the asymptotic value.

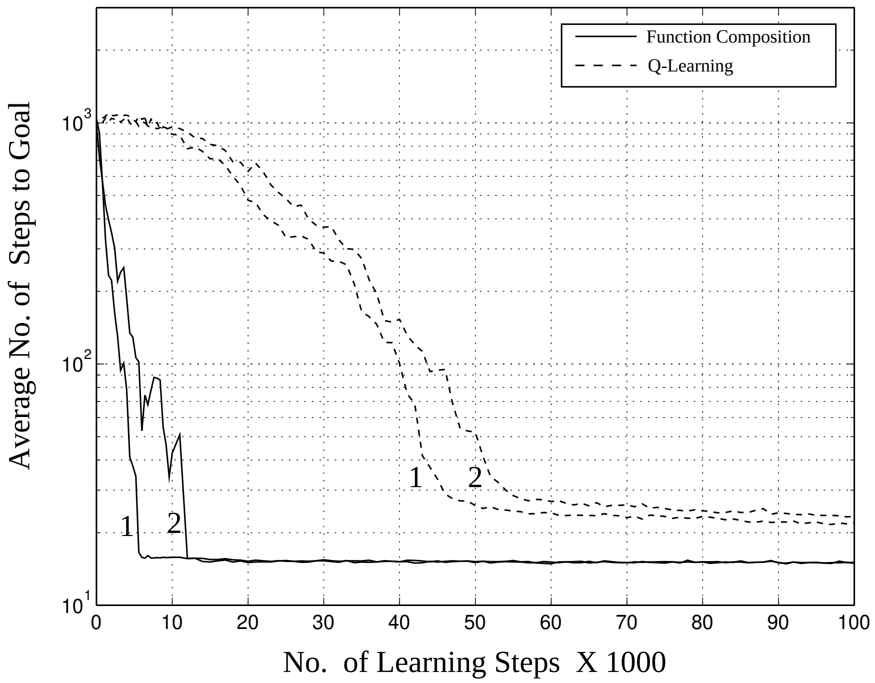

Figure 7.17: Success in Robot Navigation New Environment

Figure 7.17 shows the variation in speed up when learning in a new environment in the robot navigation domain. The room configurations have been divided into two groups. This first contains the smaller rooms from figure 7.6 (numbers 2,4,6,7,9), the average learning curves are labelled "1" in figure 7.17. The run when the doorway was mispositioned has been left out. The second contains the larger rooms (numbers 1,3,5,8), the average learning curves are labelled "2". Function composition is most successful when the inner room is small. Here a speed up of close to 10 is achieved. There are a number of reasons that larger rooms take longer for function composition, with a smaller speed up of 4.5. If a wall is long the feature takes more time to develop, more refinement by Q-learning is needed to make it apparent. This effect is exacerbated for a configuration like 6 in figure 7.6. In this example the short wall also takes a long time to develop. Paths from only a relatively small part of the state space are likely to reach this wall. Most paths move along the longer wall and then can enter the inner room through the doorway. This configuration accounted for the two longest runs. Another effect that was also observed was that a false door appeared an the end of the short wall which the doorway detecting algorithm failed to eliminate, as discussed in the previous section.

7.4 In Summary

This chapter has demonstrated that function composition produces a significant speed up in different types of related task and in different domains. The parameters established on hand crafted examples have been shown to be robust when applied to randomly generated examples. The composed functions are typically very accurate solutions to the new task. If there is some error normal reinforcement learning quickly eliminates it.

Chapter 8

Philosophical Implications:

Symbols, Systematicity and Synergisms

This thesis has proposed a hybrid architecture of closely coupled symbolic and subsymbolic layers for the learning of control tasks. This chapter positions this approach within a broader philosophical context. A crucial property of the symbolic model of cognition is systematicity, that having some thoughts is intrinsically related to having others (Fodor & Pylyshyn, 1988). This thesis takes the position that being able to solve some tasks is intrinsically related to being able to solve others. In the proposed hybrid architecture systematicity arises from the symbolic control of the composition of solutions to subtasks to form a solution to a new task. But this is not the end of the process, subsymbolic learning produces a more synergistic solution by increasing the interdependence of the solutions to the constituent subtasks.

In many ways this chapter is a defence of the classical symbolic view of cognition, which is taken to be defined by the "Language of Thought" hypothesis (Fodor, 1976). Where it most strongly differs from the classical view is in addressing low-level interactions with the world. Such interactions are characterised by the continuous, rapid interplay of an agent's actions and feedback from the world. Not surprisingly it is here that the symbolic approach is most vulnerable and has been subject to the most criticism. The response of this thesis is to define a clear role for symbolic processing in such interactions. Its usefulness comes from manipulating representations, learnt by a subsymbolic process, directly connecting feedback to action. These representations are the solutions to subtasks, composed to form the solutions to more complex tasks. The complexity of the composed solution is potentially unlimited, subtasks can be composed ad infinitum. The productive nature of the process results in a language where more complex interactions with the world are composed from simpler ones.

8.1 Symbolism and its Critics

The symbolic view has been central to artificial intelligence for some considerable time. Its inception occurred with the combination of Alan Turing's idea of the universal Turing machine and the refinement of philosophical logical forms into the mathematical logic of Boole and Frege. Newell and Simon (1976), two of the main exponents of the symbolic view, proposed "The Physical Symbol System Hypothesis" which stated that a physical symbol system has the necessary and sufficient means for intelligent action. Although some initial work was done with neural modelling, particularly the perceptron of Rosenblatt (1958), it was the symbolic side of the research that grew the faster. Many feel that this was largely due Minsky and Papert's (1969) strong critique of limits of the neural models of that time.

Although symbolism has always had its opponents, the criticisms have become more vocal recently. This is due to the growth of interest in approaches based on quite different principles: connectionism exemplified by work from the PDP group (Rumelhart & McClelland, 1986), behavioural modelling approaches exemplified by Brooks (1991a) and dynamic system approaches exemplified by van Gelder (1995). The feeling among many in these areas is that the classical symbolic approach has spent a long time to achieve relatively little and that this brings the research methodology and its underlying theories into doubt.

But this attack on symbolism focuses too narrowly on the Artificial Intelligence research field. There is a much broader philosophical context. The more modern view of intelligence being situated seems to borrow much from the empiricist approaches. The dynamic system model shares much in common with cybernetics (Port & van Gelder, 1995). Theories of connectionism have much in common with the older theories of associationism, the crux of Fodor and Pylyshyn's (1988) criticism of connectionism. In fact, these approaches generally fall under the umbrella of associative theories, which certainly have a long history themselves.

Arguments against associative theories of intelligence have typically focussed on language. From Chomsky's (1964) attack on Skinner to the more modern attack of Fodor and Pylyshyn (1988) on connectionism the human use of language has been the central theme. Fodor and Pylyshyn (1988) do suggest that animals exploit symbolic processes, but such arguments lack the strong intuitive appeal of those based on language. This thesis has explored this more extended role for symbolic processes away from language. To defend against criticisms of the inadequacies of the symbolic approach, it has demonstrated that there is a role for symbolic processing even in low level tasks like motor control.

8.2 Relationship to the Language of Thought

This section discusses the arguments for Fodor's "Language of Thought" hypothesis, the basis of the classical symbolic processing model. The main tenet of this hypothesis is that "mental representations ...... have *a combinatorial syntax and semantics*" (Fodor & Pylyshyn, 1988, p 12). The hypothesis is a computational theory of the mind, "Computers show us how to connect semantical properties with causal properties for *symbols* ...... you connect the causal properties of a symbol via its syntax" (Fodor, 1987, p 18). Although the hypothesis is principally aimed at propositional attitudes, this thesis takes the position that it is much more generally relevant. But when connecting attitudes to actions in the world other subsymbolic processes come into play.

8.2.1 Arguments for a Language of Thought

Fodor and Pylyshyn (1988) point to three essential characteristics that are a natural part of the classical symbolic processing model: productivity, systematicity and compositionality. They argue that these characteristics are so fundamental to cognition that any model that does not exhibit them as a natural consequence of its form is of limited merit. This criticism is aimed principally at connectionism, arguing that the best such an approach can achieve is to exactly implement the classical model.

Productivity denotes an unbounded representational capacity achieved by a finite process. The construction of composite symbols by the recursive application of rules starting with atomic symbols certainly gives this facility. Even if productivity is not accepted, much could be said for a compact representation. It would seem unlikely that any practical system for real world problems could use, for instance, a look-up table. Some form of iteration would seem to be the best answer for representational economy.

Systematicity means that entertaining certain thoughts is intrinsically linked to entertaining other thoughts. This is most apparent in natural language where understanding some sentences is intrinsically connected to understanding others. For instance it is hard to conceive of a person being able to understand the sentence "John loves Mary" but being unable to understand the sentence "Mary loves John". In much the same way it seems unlikely that a person can think "John loves Mary" but is unable to think "Mary loves John". The notion of systematicity is also relevant to logical inference. It would be unlikely that a person would infer that "John went to the store" from "John and Mary went to the store" but not from "John and Mary and Susan went to the store". For systematicity to occur, compositionality is required. Compositionality refers to the idea that the constituents of a sentence (or thought) make approximately the same semantic contribution wherever they appear. Thus John in the above examples has essentially the same meaning in all sentences.

These arguments seem convincing and many connectionists have responded by proposals of how to include structure sensitivity in connectionist systems. This has led to continued debate between the symbolists (Fodor & McLaughlin, 1990; Fodor, 1997) and connectionists (Smolensky, 1987; Sharkey & Jackson, 1994; Elman, 1990). To prevent the connectionist schemes being strict implementations of the symbolic model much has been made about the distributive properties of the connectionist approach and how this results in a quite different compositional scheme (Chalmers, 1990b, 1990a; van Gelder, 1990). The main argument has been that Fodor's approach assumes a concatenative structure whereas the connectionist systems do not. It is unclear to what extent the Language of Thought hypothesis depends on concatenation. Aydede (1997) argues that it is an implementation issue and thus not a critical aspect of the hypothesis. So to the extent that connectionist systems are successful in structure sensitive processing they are implementing the "Language of Thought" hypothesis.

8.2.2 Symbols and Propositional Attitudes

The whole notion of a language of thought is centred around propositional attitudes, such things as beliefs, desires and expectations, representing attitudes towards particular propositions. Thus John may believe that the proposition "John loves Mary" is true but Mary may just hope it is. But are such propositional attitudes limited to language users? Certainly Fodor believes that other animals have propositional attitudes. This is evidenced by his discussion of "Greycat" in the preface to his book "Psychosemantics" (Fodor, 1987). But what is the classical stance on other issues? Do language-like processes mediated such tasks as walking or are these processes of no importance to cognition? As Chalmer's (1990a) points out "The fact that connectionism might implement Classical theories of *composition* does not imply that connectionism would be implementing Classical theories of the *mind*. Compositionality is just one aspect of the mind after all." But any cognitive theory will focus on the particular aspects of the mind held to be of primary importance in explaining human behaviour. Certainly Fodor takes it that propositional attitudes are such core cases and within this area compositionality is a critical property.

The thesis proposes that compositionality is a much more generally important issue. It aims to address the issue that Pylyshsyn (1984, p xvii) voices "If we can set up situations demonstrating that certain stimulus-response regularities can be altered in ways that follow these rational principles, we can say that the input-output is *cognitively penetrable*, concluding that at least some part of this function cannot be explained directly in terms of properties of the functional architecture." But Pylyshsyn argues for fixed cognitively impenetrable transducers that connect the world to the symbolic processes. This chapter argues the merits of a more flexible and cognitively penetrable connection to the world. The aim is to show how a hierarchy of representations allows symbolic processes to manipulate the functional architecture to aid in interacting successfully with the world.

8.2.3 From Attitudes to Actions

The position of this thesis is that it is not only the higher level cognitive processes that are language-like. It is the aim of this work to show that lower level processes have these properties too. Fodor (1987, p 20) takes the "Language of Thought" to be a theory of "How is rationality mechanically possible?" Fodor (1987, p 23) discusses Representational Theories of the Mind (RTM) saying "Some - but not all - versions of RTM borrow more than this; not just a theory of rationality but a theory of intelligence too. According to this story, intelligent behaviour typically exploits a 'cognitive architecture' constituted of hierarchies of symbol processes". Fodor (1976, p 173) regards this as an empirical question. He writes "There may be - perhaps there must be - some end to this hierarchy of rational decisions. But the end is not in sight. For all we now know, cognition is saturated with rationality through and through." This thesis is, in part, an investigation of where this "rationality" bottoms out and how language-like processes might direct but not control lower level processes without such properties.

This thesis takes the view, that even in low-level processes such as motor control, syntactic operations are useful. But how might propositional attitudes be connected to such processes? Fodor (1987, p 137) talks about the intention to carry out a number of actions. "Whereas according to the LOT theory if I intend to raise my left hand and hop on my right foot, I must put in the intention box a formula which contains, inter alia, a subexpression that means *I raise my left hand* and a subexpression *I hop on my right foot* " . The intentions are represented by distinct subexpressions, but the relationship between the behaviours they produce is more complex. Raising one's hand affects how one balances when hopping on one foot. Certainly we would expect to be able to execute this combined skill given that we could execute each skill independently. But with practice the hopping skill may be tailored to better suit the raised hand.

Fodor (1987, p 143) certainly acknowledges the existence of such more specific skills, "To put the point quite generally, psychologists have a use for the distinction between segmented behaviours and what they call 'synergisms'. (Synergisms are cases where what appear to be behavioral elements are fused together to one another, so that the whole business functions as a unit; as when a well practised pianist plays a fluent arpeggio.) Since it's empirically clear that not all behaviour is synergistic ......" . The aim of this chapter is to trace a path from symbols to synergisms. The amount of practice will determine the degree of fusion between the behavioral elements. In learning a complex skill, initially it may be just the combination of simpler skills. With more practice, however, the simpler skills become more interdependent making the complex skill more synergistic. In this thesis, the subtasks solutions are such skills, the composite solution after further refinement by reinforcement learning is the synergism.

8.3 A Layered Approach

To produce this path from symbols to synergisms, this thesis has proposed a hybrid architecture with closely coupled symbolic and subsymbolic layers. This is also a coupling between different levels of representation. At the lowest level are functions learnt by reinforcement learning which are partitioned into solutions to subtasks. These are mapped to a symbolic representation through an analogical representation, as shown in figure 8.1, and stored in a case base. This allows a syntactic method of composition much like symbolic planning to combine these subtask solutions to form solutions to a new task. Both symbolic planning (Cohen & Feigenbaum, 1982) and case based learning (Aamodt & Plaza, 1994) have been proposed as important parts of human cognition. Reinforcement learning is an associative learning method and has much common with operant conditioning except the reinforcement is typically considerably delayed. With the addition of eligibility traces it is also strongly related to classical conditioning (Sutton & Barto, 1990).

8.3.1 Levels of Representation

This section discusses the different levels of representation employed by the hybrid architecture. The role, in fact the very existence, of representations in intelligence has the subject of much recent controversy. Fodor and Pylyshyn (1988) present the representationalist's view, "there are states of the mind which function to encode states of the world." Markman and Dietrich (1998) suggest that even anti-representationalists support the idea of internal state but it is additional properties that cause the controversy. They argue that different properties may be important in different cognitive tasks: "(1) being enduring, (2) being discrete (and therefore composable), (3) having compositional structure, (4) being abstract, and (5) being rule-governed."

In the hybrid architecture proposed in this thesis, all these properties are apparent. At the lowest level reinforcement learning progressively refines a control function improving an agent's ability to reach a goal quickly. The control function is an internal representation, but it represents not the world states themselves but the best actions to take in these states. It is a mapping from perception to action, similar in spirit to Clark's (1997) action-oriented representations. It is certainly enduring, once learnt it is stored in a case base for future use. It also has structure which this thesis exploits to increase the opportunities for transfer. This structure is identified by locating features within the control function which are extracted to form a plane graph. This is another internal representation but of a more abstract kind, which captures analogically certain properties of the agent's interaction with its environment. This plane graph includes a graph of the paths connecting its subgraphs and representing the topological structure of the task. This is used as the basis for symbolic planning. The nodes represent states of the world at an abstract level, the edges abstract actions. To produce the solution to a new task, a rule-governed process composes abstract actions learnt when solving previous tasks.

8.3.2 Coupling the Layers

In this section the work presented in this thesis is positioned in a broader context, allowing for a larger range of possible divisions of labour between the layers. In

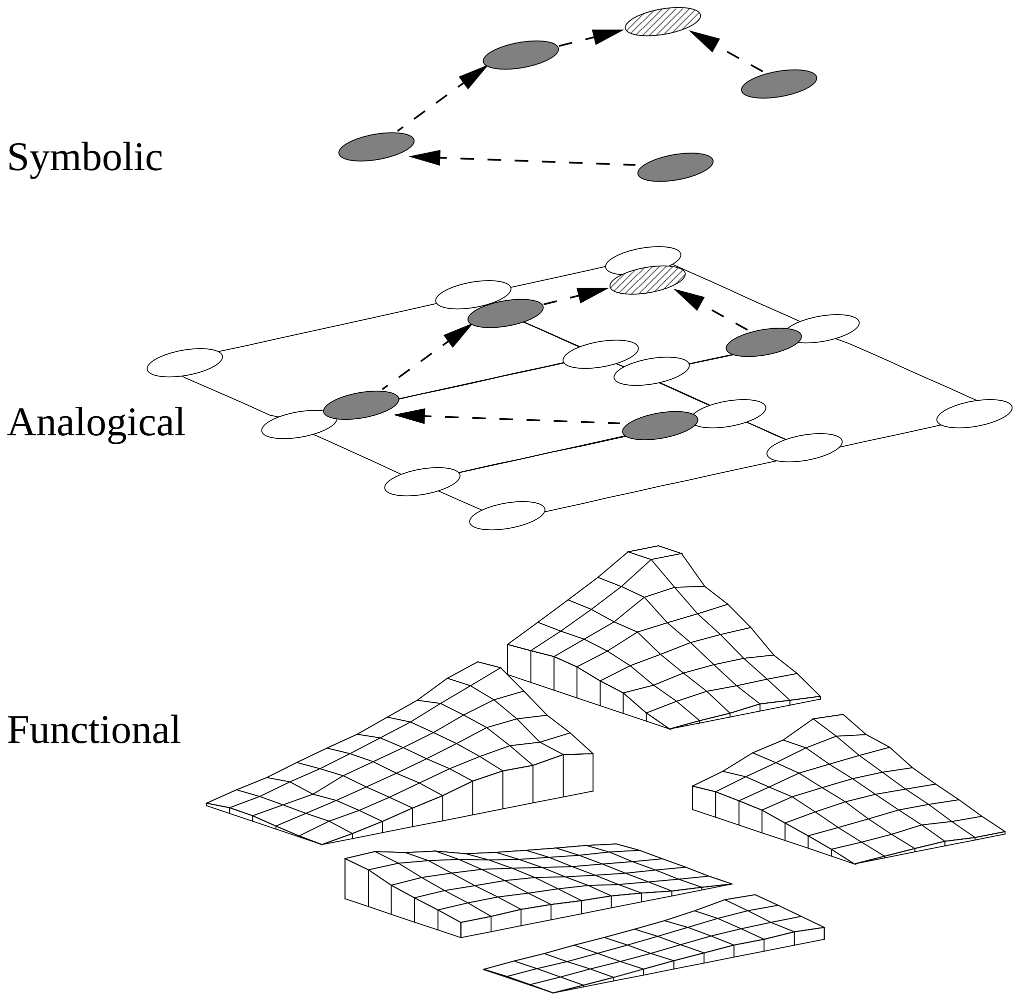

Figure 8.1: Coupling Representations

figure 8.2, the symbolic layer is wrapped in a subsymbolic layer of varying thickness. At one extreme the processing is largely done symbolically. The thickness of the subsymbolic layer between the outside world and the symbolic process increases, as more of the processing is passed over to the subsymbolic layer. At the other extreme the subsymbolic layer connects perception directly to action. But even in this case the symbolic processes have a role to play in determining the form of this connection. This has been the focus of the work in this thesis.

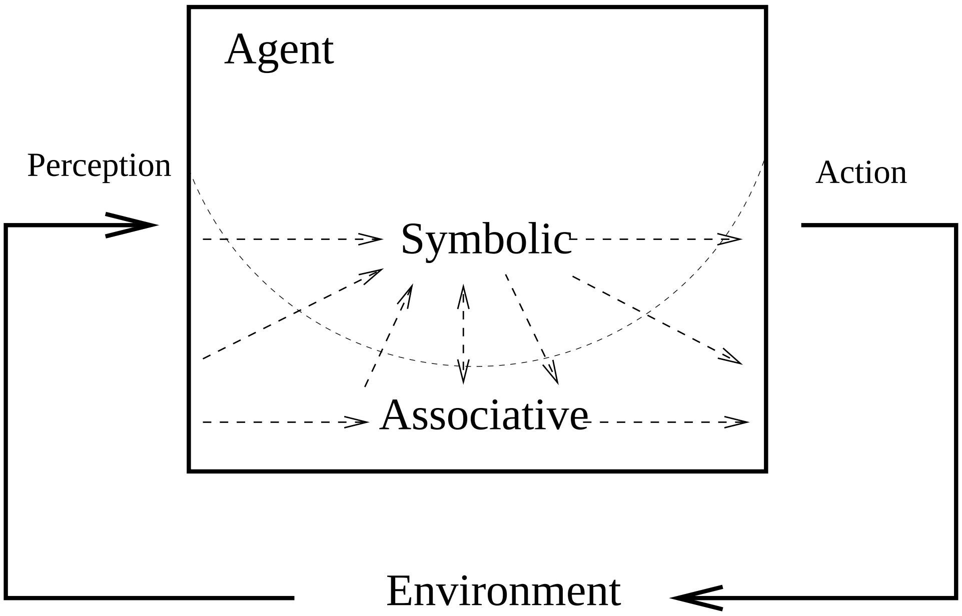

Figure 8.2: Coupling Layers

This division into layers shares much in common with psychological models of skill acquisition. Fitts proposed three stages in skill acquisition: cognitive, associative and autonomous (Fitts & Posner, 1969). In Anderson's ACT system (Anderson, 1982) the cognitive stage uses a general purpose set of productions to solve a problem. The associative stage is the progressive compilation of productions used in the solution by combining multiple productions and instantiating the variables. The autonomous stage is reached when compilation is complete but allows further tuning of rules to speed up the process.

The model presented here includes the same three stages, but unlike the ACT system the layers are not all symbolic. This thesis has not investigated using the symbolic level to directly solve the problem, but the architecture in figure 8.2 certainly allows for this. Rather, compilation replaces the steps in the symbolic plan with functional representations mapping world states to actions. The cognitive stage is this initial planning that produces the function. There is then a stage where the function is refined by the low-level learning process but if and when new features emerge replanning occurs with a significant update of the function. Ultimately at the autonomous stage the function is solely refined by the low-level learning process.

The idea of a layered architecture has recently become more prevalent in Artificial Intelligence research largely influenced by Brooks's (1986) subsumption architecture. For instance Sloman's (1997) approach has three layers: reactive, deliberative and reflective. The question remains as to how these processes are mediated. Brooks (1997) certainly sees one of the key ideas of his approach as to "Minimize interaction between layers", so this mediation occurs largely in the outside world. In Touretzky and Pomerlau's (1994) view the lower level processes may act autonomously but may also be overridden by higher cognitive processes. The view of this thesis is that there is a significant degree of "cognitive penetrability" of low level processes in human cognition and a much larger degree of internal interaction between the layers.

8.4 Defining A Position

This section aims to clarify how the view presented in this thesis differs from those held by various philosophers, neuroscientists, cognitive scientists and Artificial Intelligence researchers. To this end, the positions of various individuals prototypical of the principal schools of thought within these research fields are considered with respect to certain questions.

- How strongly is intelligence dependent on the environment?
- Is intelligence constrained by its physical implementation?
- What is the role and nature of representations in intelligence?

- What behaviours define intelligence?

These questions define dimensions spanning a space of possible positions. The positions are not necessarily clearly delineated and there is often considerable overlap. In addition, many of the subspaces are empty, the position on one dimension being highly correlated to that on another.

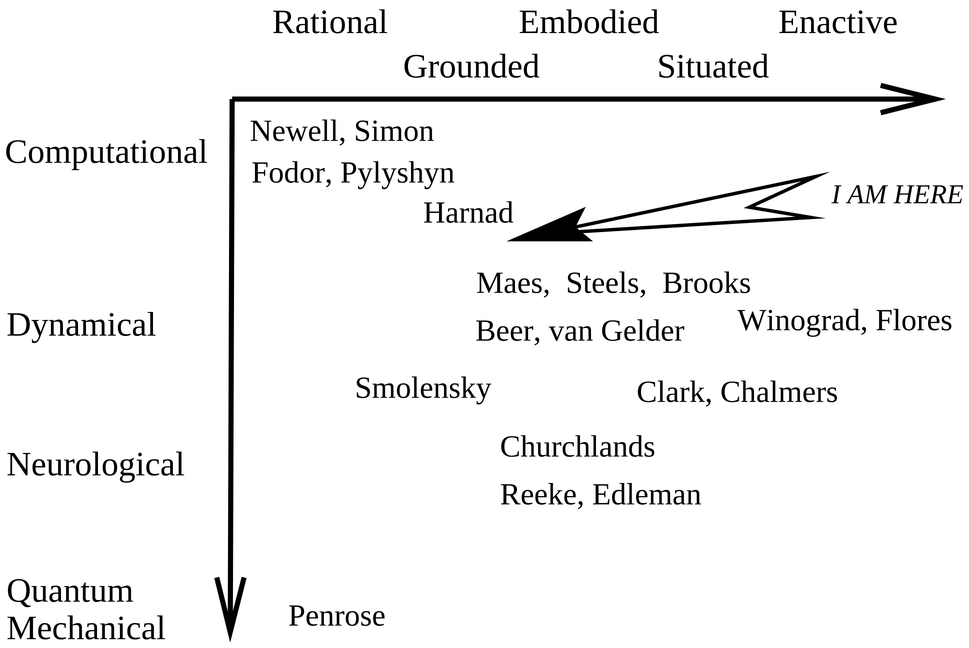

Figure 8.3: Grounding

The first two of these questions form a pair of dimensions that are roughly orthogonal, as shown in figure 8.3. They might both be called grounding or as Fodor (1987) calls them "two types of reductionism". The top dimension is the degree of grounding in the world. It concerns the extent to which intelligence of an individual can be isolated from its environment. The side dimension is the degree of grounding of intelligence in its physical substrate. It concerns the extent to which any theory of intelligence is constrained by its physical implementation.

8.4.1 Grounding in the World

Grounding in the world addresses the question "How strongly is intelligence dependent on the environment?" A closely related question is "Where does the mind stop and the rest of the world begin?" (Clark & Chalmers, 1998). There is a very large range of opinions on how to circumscribe intelligence: is it within the head, the body, the local environment or the social group? At one extreme is the rationalist view where intelligence is positioned firmly inside the head. It is connected to its environment through transducers, but this connection is not necessarily the main influence on human behaviour. For instance, Pylyshyn (1984, p 12) claims "the reason behaviour is stimulus free is that what people do depends to a great extent on what they believe at the moment, how they see the situation at the moment, on what they think will be the consequences of their behaviour and so on"

Yet some feel it is premature to address higher-level problems until the nature of this connection is well understood. In Harnad's (1990) view, the symbol does not have any intrinsic meaning until it is connected to the world, which he christened the "symbol grounding problem". He sees grounding symbols as a way of addressing the problems raised by Searle (1980) in his famous Chinese room argument. Here, Searle likened a symbolic system to a person in a room with a book which contains pairs of Chinese phrases. The person would receive written Chinese phrases through a slot in the wall, look them up in the book and pass back the result. Although the result might to a Chinese person be totally coherent, the person inside the room need have no understanding of Chinese. Thus any semantics in symbolic systems is purely due to the interpretation of an external human observer. But Harnad proposes that symbols correctly grounded in the world produce the necessary semantics so that the whole system could be said to understand.

This idea is not necessarily in conflict with many views of classical symbolism where it has long been recognised that symbols must be connected in the "right way" to the external world. Determining the logical properties of these transducers is important in establishing a semantic theory. The whole issue of how meaning gets inside the head, if it does at all (Dennett, 1987; Putnam, 1977), is a major philosophical concern as yet unresolved. As Fodor says (1990) "But of the semanticity of mental representations we have, as things now stand, no adequate account". But the importance of transducers to semantics does not necessarily bring them into the realm of cognition, they might still be seen as part of the "functional architecture" (Pylyshyn, 1984) and their exact form not the subject of cognitive science. While side-stepping the complex issue of semantics, this thesis does propose one way of connecting internal representations to the world. It also espouses a model where the "functional architecture" can be manipulated at the symbolic level and thus falls clearly within the boundary delimiting intelligence.

The next step after including transducers within this boundary, is embodiment. Many view intelligence as defined by how an agent responds in a timely fashion to environmental demands (Brooks, 1991a). Bodily form certainly affects how this is done, so the boundary can be redrawn at the skin. Exponents of "situated" cognition downplay the need of an internal model of the world, for instance Brooks (1991a) suggests that "The world is its own best model". This, in effect, moves the boundary even further, offloading much of what is seen as intelligence into the environment. Clark and Chalmers (1998) discuss the use of tools "Thus consider the use of pen and paper to perform long multiplication, ... and the general paraphernalia of language, books, diagrams, and culture. In all these cases the individual brain performs some operations, while others are delegated to manipulations of external media." The use of tools extends the envelope of the body, and therefore intelligence, further and further into the environment. The environment will also include other people, Winograd and Flores (1986) propose that the interaction within a society is a crucial part of intelligence. The whole complex interaction of an agent with its environment and its altering of the environment over a long period of time might all form a part of the intelligence envelope (Varela, Thompson, & Rosch, 1991).

It seems inarguable that intelligence needs some connection to the world. It is less clear however that more than a symbolic interface is needed to support cognitive processes. An interesting example of a system with relatively poor grounding is given in Sacks's book (1985) which looks at various cases of neurological damage. One particular case involves a woman who is blind from birth, chair bound and has not used her hands since early childhood. In fact when presented objects in her hands is unable to identify them. Her symbolic grounding appears minimal, yet Sacks reports she is capable of carrying out a perfectly normal, in fact particularly intelligent, conversations. Yet all her knowledge of the world had been through conversations and books read by others. This thesis shares Hillis's (1988) view "Sensory-motor functions are clearly important for the application of intelligence and for its evolution ..... But much more apparatus is probably necessary to exercise and evolve intelligence than to sustain it."

8.4.2 Grounding in the Physical Substrate

Grounding in the physical substrate addresses the question "Is intelligence constrained by its physical implementation?" Symbolists, dynamicists, behavioural modellers are all functionalists of one type or another. They may disagree on the types of functions necessary to produce intelligence but they would agree that the physical substrate that implements the functionality is of limited importance. The computer metaphor is a strong motivator for this view, the same software can run on many different types of machine. But even those who are not strict computationalists would view their functional theories as multiply realisable, many different substrates being capable of producing the same result. Certainly the notion of functionalism is apparent in many common objects. As Block (1996) points out "What it is for something to be a carburetor is for it to mix fuel and air in an internal combustion engine". This says nothing about how it is constructed, so function under-determines form.

With connectionists the situation is not so clear. Smolensky (1988) views connectionism not as a neural model but as a sub-conceptual model. This allows him to distance himself from criticisms from neuro-scientists, who view connectionism as much too dissimilar from the true workings of the brain to be considered a neural model. For instance, proponents of a view of neural development called Neural Darwinism (Reeke & Edelman, 1988; Reeke & Sporns, 1990) accuse connectionists of "looking sideways at biology". Such researchers argue that more careful attention to how the brain actually processes information is the path most likely to lead to insights into cognition. Neuro-scientists are not alone in this view, it is also held by philosophers such as Patricia and Paul Churchland (Churchland, 1986; McCauley, 1996).

Neurological grounding is not necessarily the lowest level to consider. Penrose (1989) proposes that cognition is critically dependent on Quantum Mechanical effects.

While perhaps most researchers would not accept the Quantum Mechanical view, few would claim that neurological data will shed no light on cognitive theories. It might be that a completely anti-reductionist stance is untenable, to explain some intelligent behaviours a neurological level description might be necessary. The essence of human intelligence might be captured by a substrate neutral competence theory, the actual substrate only becoming important in the performance theory, to borrow Chomsky's (1965, p 4) distinction. So even if there was a multiple realisability of intelligence a "provincial, reduction of psychologies" (Schwartz, 1992) would offer some specific explanation with a specific group. This thesis has not in any way tried to pursue a neurologically plausible architecture, as such is firmly committed to the functionalist view.

8.4.3 Representations

As suggested at the beginning of this section the positions taken by different people on different issues are far from orthogonal to those taken on other issues. For instance opinion on the question "What is the role and nature of representations in intelligence?" is roughly correlated with that taken on grounding in the world as shown in figure 8.4. Certainly, symbolists and connectionists alike are committed to internal representations that represent things about the external world. The form of these representations differ however, being atomistic or holistic respectively. For the symbolist complex representations consist of atoms (primitive symbols) composed syntactically. For connectionists, however, representations are distributed across micro-features. A representation is a point in feature space and may be manipulated using some sort of vector operator.

To borrow the terms from Pinker and Prince (1988) these proponents of holistic representations are revisionists with respect to the representations proposed by the symbolic approach. A more extreme posture is to be an eliminativist, denying the existence of internal representations. Brooks (1991b) certainly feels that the internal states used in his subsumption architecture are not representations, saying "However

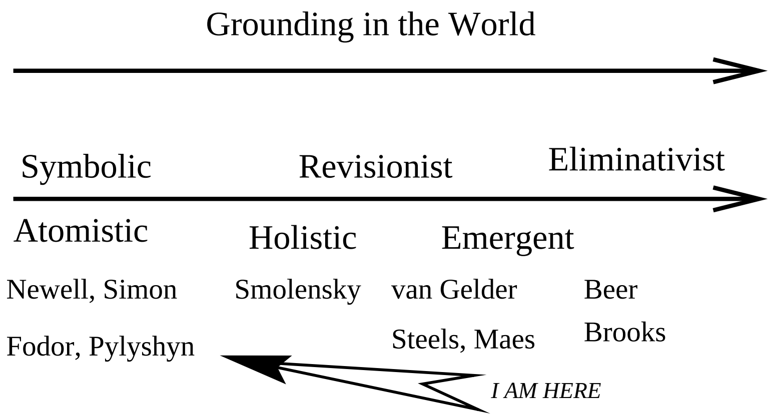

Figure 8.4: Internal Representations

we are not happy with calling such things representations. They differ from standard representations is so many ways." But other people from the same school take a less extreme position (Steels, 1990, 1995; Maes, 1994). They advocate a lesser role for representations than the symbolists propose and of a different kind, typically favouring analogical or task dependent feature based representations. Dynamicists (Beer, 1995b; van Gelder, 1995) seem at first glance to have a view similar to Brooks that there is no need for internal representations. But van Gelder (1995) at least seems to believe that dynamic systems can form representations but again of a radically new kind. Borrowing from dynamic system theory these might be attractors, trajectories and so on, although their exact form is still speculative.

This thesis views higher level cognition as symbolic, exploiting internal representations with well defined syntactic structure. Certainly the extreme anti-representationalist stance is open to criticism. As Clark (1994) points out many of the problems that have been addressed by people in this camp are cases where the connection to the world is simple and immediate. As the tasks become less dependent on an immediate response from the world the need for internal representation of at least some sort becomes strong.

As to the form of these representations, some debate has centred on the granularity of symbols versus the microfeatures of connectionism. In some people's minds "symbol" has become synonymous with "word". But in much research primitive symbols are much lower level features. Even in linguistics the smallest unit of meaning is a morpheme, often just part of a word. This thesis argues that if at a low level features are distinct then syntactic operations would seem highly desirable. Without such distinctiveness, such nascent symbols would become entangled leading to the more holistic form the connectionists claim.

Still below a certain level distinctiveness disappears. This level may be task sensitive. It may be appropriate to regard things as distinct at certain times and not at others. In the discussion on productivity in section 8.2.1 the parsimony of nature was held as an argument against non-compositional representations. When a task however is carried out many times parsimony might well be traded off against accuracy. Thus representations do become intertwined. The parts of the functional form becoming less clearly delineated producing the synergisms discussed in this chapter.

8.4.4 Intelligence Tests

Another issue that parallels grounding in the world is the disagreement on the question "What behaviours define intelligence?" as shown in figure 8.5. When Turing (1950) first considered the question "Can machines think?" he introduced the notion of what later became known as the Turing Test. If a human judge could not tell a human from a machine then the machine might reasonably be thought intelligent. The judge was to determine this by asking questions but "the answers should be written, or better still, typewritten". Thus intelligent behaviour was seen as the ability to express reasonable amounts of knowledge in a linguistically coherent manner. A recently suggested extension to this test by Harnad (1993) is the Total Turing Test. Here the computer must be replaced by a robot that can "interact robotically with (i.e., to discriminate, identify, manipulate and describe) the objects, events and states of affairs that its symbols are systematically interpretable as denoting". Thus the test would focus less on linguistic performance and more on the ability to interact successfully with the world.

Although not explicitly proposing a test, Brooks aims to go further in adding human-like properties to the robot by "building a full human level intelligence that

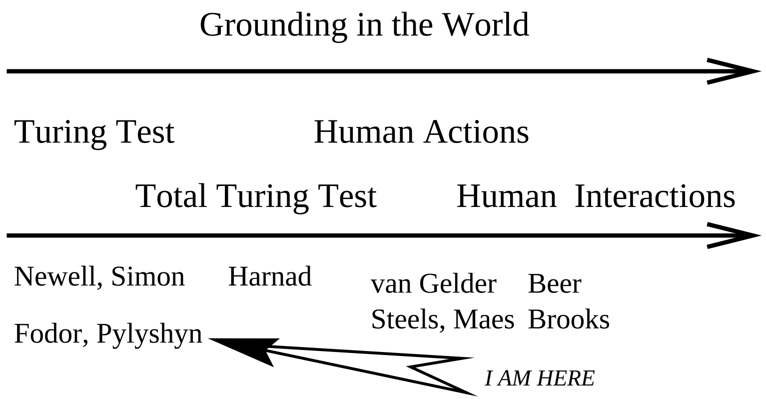

Figure 8.5: Important Behaviours

is able to operate and interact in much the same way a human would operate and interact" (Brooks, 1997). Brooks justifies this on the pragmatic grounds of the robot needing to interact with people, but also with guarded reference to the view of Lakoff (1987) who claims that the form of the body has an influence on the representations used in thought. Thus to embody human intelligence a system must be able to act physically in many ways like a human. One might go further. If human intelligence is dependent on the complex interaction with the environment and other humans, a test for intelligence would have to take into account such interactions.

While this chapter has focused on human intelligence, this is not to deny intelligence in other animals. But even in research involved in modelling animal behaviour the final aim is often to understand human intelligence. One important goal of the Turing's original test was the removal of extraneous human behaviours not important in capturing human intelligence. As is apparent in the above discussion, which behaviours are extraneous is an area of much debate. The gamut runs from placing language centre stage to seemingly including all behaviours bar language. It is noteworthy that although blind and deaf people interact with the world in a very different way than sighted and hearing people, they are usually all capable of some sort of language. Any test that excluded Sacks's patient discussed earlier would seem to be of limited merit and would seem to miss the abstract notion of intelligence that Turing aimed to capture.

Certainly language in many ways is what defines us as uniquely human, separating us from other animals. But is the intelligence of language users as Dennett suggests (1996) radically different from that of other animals? An alternative view is that language is just one more cognitive skill built upon countless others produced during our evolutionary past. The supporting argument emphasises that evolution has taken a long time to produce language and other forms of human expertise (Brooks, 1990). Reeke and Sporns state (1988) "Clearly, we, as humans, do carry out certain activities based on symbol manipulation ...... however [this] is an invention made late in evolution, in societies of individuals communicating through language." So to understand humans, the proposal is that we must first understand lower life forms. This is evident in the title of Brooks's (1997) paper "From Earwigs to Humans".

This argument seems based on the idea of an evolutionary ladder. Evolution has progressed from simple to more complex creatures, building on its successes by adding higher and higher level cognitive structures. But as Gould (1989) points out evolution is not a matter of progression from lower to higher forms. Evolution is not a straight line and may move from the complex to the simple if that improves survivability. The connection between relatively simple animals (even assuming they are strictly our forebears) and humans is surely too long to expect this clean delineation of skills to survive. It seems more reasonable to look at our closer relatives for clues to ourselves. For instance Calvin (1993) proposes that throwing behaviour exhibited by chimpanzees might indicate an evolutionary precursor of language. Throwing requires a sequence of muscle actions too fast to rely on feedback from the world and some form of planning is required. This Calvin relates to the production of language, an idea very much the focus of this chapter.

8.5 In Summary

This chapter has argued for the importance of symbolic representations and processes not only in the building of practical systems but also in terms of human cognition. Certainly there is more to intelligence than propositional attitudes or linguistic performance, an intelligent agent must act in the world. But crucial to this intelligence is the ability to recognise and exploit structure and it is here symbolic systems show their power. The symbols presented in this thesis are not in the world but in the agent's interaction with the world. Thus this thesis is accepting that the agent is situated in the world. But this is not to accept the more radical thesis that a human's interaction with the world is what defines human intelligence. As Brewer (1974) suggests in humans even conditioning processes are steeped in higher level cognition. Extreme situatedness may be true for very simple creatures but overall this is of much less importance to human intelligence than the radicals suggest. It is certainly not clear that because certain animals solve a task in one way that human beings must adopt the same approach. As Dennett (1994) argues "[this view] ignores the independently well-evidenced possibility that there are two profoundly different ways of building dams: the way beavers do and the way we do." This thesis proposes that it is "cognitive penetrability" of low level sub-symbolic processes, the ability to "reach down" into more reactive processes, that might well be the defining characteristic of human intelligence, and perhaps that of other higher animals.

Chapter 9

Conclusions

The thesis has defined a role for both symbolic and subsymbolic processing in the learning of control tasks. It has proposed a novel hybrid architecture with a tight coupling between a variant of symbolic planning and reinforcement learning. This architecture combines the strengths of the function approximation of subsymbolic learning with the more abstract compositional nature of symbolic learning. The former is able to represent mappings of world states to actions in an accurate way. The latter allows a more rapid solution to problems by exploiting structure within the domain. Viewed from a symbolic standpoint, this thesis has shown how a planning system might learn its operators and planning space. Viewed from a subsymbolic standpoint, this thesis has shown how the solution to subtasks might be identified and later composed to speed up the learning of a new task. The efficacy of the hybrid architecture in the learning of control tasks has been experimentally demonstrated. It was shown to produce a significant speed up over basic reinforcement learning, in learning in a new environment and when dealing with a changing world.

Earlier chapters in this thesis detailed the components that form the hybrid architecture. To summarise how these various components interact, figure 9.1 shows the high level algorithm for learning a new task. The chapters detailing the most important steps are indicated at the end of the relevant lines. A similar algorithm is used when the goal is moved. The differences are that steps 1 to 5 occur when the goal is in its original position and the search method discussed in section 6.2.3 is used to locate the new goal before the final steps 6, 7 and then 9 are carried out.

1. **Initialise the function approximator. Set GRAPH to null.**
2. **Apply reinforcement learning to the task for a fixed number of steps, using Q-learning adapted for function approximation (Chap 4)**
3. **If the goal position is known:**
   (a) **Apply the partitioning algorithm (Chap 5).**
   (b) **Construct a new graph by adding nodes for the goal, doorways and corners to the partition (Chap 6).**
4. **If the new graph is the same as GRAPH or contains errors go to step 2.**
5. **Set GRAPH to the new graph.**
6. **Apply the symbolic planning algorithm to compose a new control function (Chap 6).**
7. **Reinitialise the control function with the new function.**
8. **When this step is first reached go to step 2.**
9. **Apply reinforcement learning to further refine the function.**

Figure 9.1: The High-level Algorithm

Although the main contribution of this thesis is the complete architecture, each component either alone or in tandem with others has broader applicability. This thesis has shown how an important family of function approximators, namely B-splines, can be used in reinforcement learning. It has shown how regularisation can prevent oscillatory behaviour and even divergence in the learning function. This thesis has identified features, naturally produced by reinforcement learning, that can be used as a way of partitioning the function into subtasks. It has shown how a particular vision processing algorithm, the snake, can form this partition. It has proposed various extension of the snake representation that improve its robustness. This thesis has proposed a method much like symbolic planning to control the composition of functions to build a new solution from subtasks. It has defined a set of transformations to fit old subtask solutions to a new task.

The following sections discusses the limitations of the approach and ways these limitations might be addressed. It also discusses various potential avenues of future research.

9.1 Limitations

Limitations come, roughly, in two kinds: those arising from the approach and those arising from the implementation. In the former case, ways to address these limitations may be highly speculative or impossible without abandoning some of the fundamental ideas behind the approach. In the latter case, there is a reasonable expectation that future work will address these limitations. The following sections will deal with these cases in turn.

9.1.1 Limitations in the Approach

To explore the possibilities of limitations in the approach, this section reviews the assumptions that the approach makes.

The first assumption is that features arise in the reinforcement learning function that qualitatively define its shape. The features used in this thesis are the violation of a smoothness assumption, that neighbouring states have very similar utility values.

A wall, by preventing transitions between neighbouring states, typically causes such a violation. Other things such as actions with a significant cost would have a similar effect. Smaller and much more varied costs will not generate the features required by this approach, so it offers little in the way of speed up in these cases. If there is a mixture of large and small costs, it is expected that the system will capture features generated by the former, initialise the function and normal reinforcement learning will address the latter.

The smoothness assumption is less clear if the dimensions are not numeric. The neighbourhood relation that has been used here is a predefined distance metric over a continuous space. In nominal, binary or mixed domains it is not obvious how such a metric would be defined, although there is some work on such metrics for other applications (Osborne & Bridge, 1997). If the dimensions are mixed, feature location might be limited to the continuous ones. If the dimensions are purely nominal or binary, assuming the appropriate metric exists, a generalisation of the snake may be appropriate. The snake is, at an abstract level, a constrained hill climber. But whether or not this idea would usefully generalise in this way is at present somewhat speculative.

The second assumption is that the features clearly delimit subtasks. In the domains investigated in this thesis, the obstacles and walls subdivide the state space into regions with small "doorways" connecting them. The subtask of getting to one doorway is not greatly affected by the subsequent subtask. But in other domains this may not be the case. As the doorways become larger, the context sensitivity increases. As long as the composed solution is reasonably accurate and reinforcement learning can easily correct the error then the result is just the more synergistic solutions defended in this thesis. At some point, however, the advantage of dividing into subtasks will become questionable, due a very large amount of context sensitivity. It would be possible to add some sort of context dependency in the graph matching stage. One might look at larger units than subgraphs. If two adjacent subgraphs match the new problem, it would be better to use them as pair, thereby including any contextual relationship between them. Even if single subgraphs were used, the context in which they appear, i.e. the shape of neighbouring subgraphs, could be taken into account.

In the limit, graph matching the whole task might be used. But as was argued at the beginning of this thesis, this would considerably limit when transfer is applicable and thus its overall usefulness. In addition, unless the task matched almost exactly the affine transformations used in this thesis would be inappropriate and more complex locally sensitive transformations would have to be used.

The third assumption is that position of the features is not critical, rather it is the shape of the delimited region that matters. To increase the effectiveness of transfer, solutions to subtasks have been subjected to a variety of transformations. In some domains many, if not all, of these transformations will be invalid. For instance if, as discussed in the first paragraph of this section, many small costs affect different regions of the state space the effectiveness of transfer will be reduced. This would be to some extent addressed by additional penalties for different transformations but this would again limit the opportunities for transfer. Which transformations are appropriate in which domains is the subject of future research.

9.1.2 Limitations in the Implementation

Previous chapters in this thesis have discussed the various limitations of the components of the hybrid architecture and suggested some ways these might be overcome. These limitations raise the question of the general applicability of the implementation, and perhaps the approach, in other domains. Other domains are likely to differ from those presented in this thesis in a number of ways.

The first difference is that the dimensionality of the space may be higher than the two dimensions of the applications investigated in this thesis. In section 5.4.3, the snake was shown to work in three dimensions and the problem of higher dimensional spaces was briefly discussed. The mathematics behind the snake is not limited to three dimensions. There also seems nothing in principle that would prevent other processes such as graph matching, planning or transformation from working in higher dimensions. The main problem area is the speed and principally then only of the snake. This is an issue of great importance in the vision processing community. Current research is investigating this problem, at least in two or three dimensions. The results of such research will undoubtedly be of advantage here.

Learning in very high dimensional domains is likely to be slow whatever technique is used. Normal reinforcement learning will take time to navigate the much larger space, slowing down the emergence of the features. So although the time taken to partition the function will increase, the frequency with which partitioning is applicable will decrease. Thus the amortised cost will rise more slowly. Further, as high dimensional spaces are generally problematical, methods such as principal components analysis and projection pursuit (Nason, 1995) can be used to reduce dimensionality. It may prove in practice that the dimensionality which is important and is the focus of feature extraction is much smaller than the actual dimensionality of the space.

The second difference is that many small differences in the reinforcement learning function might render the snake ineffective at locating the features with the present parameter settings. The values of the parameters were empirically determined using hand crafted examples from the robot navigation and the robot arm domains. The obvious danger is that the parameters might be somewhat tuned to these examples. To demonstrate that this is not the case, configurations for the experiments in the robot navigation were generated randomly. As configurations for the robot arm domain are more tightly constrained the hand crafted examples were used in the experiments. Nevertheless, the experiments have shown that the parameters selected work successfully for random examples in the robot navigation domain. They also work successfully in a second domain the robot arm. Section 5.4.2 also demonstrated that they were reasonably effective in a quite different domain, the "car on the hill". Further, it is anticipated that using the results of current research into snakes will automate the selection of many parameters.

The third difference is that the shape of various regions in the partition may be more complex than can be dealt with by the present snake. The randomly generated examples of section 7.2.1 were subject to certain constraints: a room could not be too small or narrow, a door could not be too large. Configurations with narrower rooms were tried informally, but the snake did not reliably locate the features. The configurations in chapter 7 plus those discussed in section 5.4 represent the limit of the complexity of partition the snake can produce at present. It is expected that using ideas from the large body of already published research into snakes will go a long way towards addressing this limitation. For complex regions, locating all the subtleties of the underlying shape may be, in fact, unnecessary or even undesirable. The aim is to speed up low level learning. As long as the solution is reasonably accurate speed up should be obtained. Being too sensitive to minor variations in shape may severely limit the opportunities for transfer and thus reduce speed up overall.

9.2 Future Work

Future work will address many of the limitations discussed in the previous sections. The following sections discuss other future work.

9.2.1 Iterative Updating

This thesis has demonstrated how the hybrid architecture can speed up learning when the goal has moved. But there are other relatively small changes in the world that can have a large impact on the present solution and take a long time to correct. Sutton (1990) talks about the "problems of changing worlds" such as an already learnt path becoming blocked or a shortcut opening up. Section 7.3.2 discussed the problem of false doorways and failing to detect true ones. Dealing with doorways that become blocked is much the same as dealing with false doorways. Locating new doorways is much the same as dealing with undetected doorways. By observing the existing partition, a normally open doorway becoming closed would be quickly detected, as it will block some path to goal. A new doorway, or one that was missed, would also become apparent by just watching the partition. But this likely to take some time, most existing paths to the goal may go nowhere near this doorway. Still, occasionally random actions would go through the doorway changing the gradient at that point. In both cases, once the change is detected, a new function can be composed that solves the new task.

Being able to plan at the abstract level, gives the system the ability to compose many different functions for many different purposes. For instance to find short cuts, the system could compose a more preemptive exploratory, or "what-if", function. A "what-if" function might embody what would happen if a wall was not there. The left hand side of figure 9.2 shows the configuration discussed in section 7.3.2. The system could compose a function with the wall, indicated by the dashed line, missing. If the function is compared to the one for the wall present, because of the similar path lengths to goal, there will be little difference. So even if there was a doorway in this wall, there is little point in changing the function. But suppose a function was composed for the goal in the top left corner as shown of the right hand side of figure 9.2. Now there is a difference but only for paths starting in room A. If this new function is used, when the system is in room A, and only room A, it will move towards the apparently open wall. Whether or not the wall includes a doorway should become quickly apparent by observing the gradient along the partition.

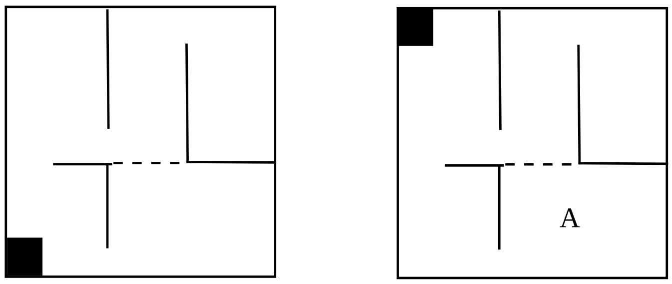

Figure 9.2: Looking For Shortcuts

Repeatedly updating the function as new information becomes available might be viewed as form of theory revision or refinement (Towell, Shavlik, & Noordewier, 1990). The composed function is a theory of how to efficiently solve a task, further learning exposes errors in the theory and a new function is composed. This raises the question of when to do small refinements to the theory and when to make more radical revisions. For instance, if a goal is close to a doorway it is not only the doorway that might be missed. The gradient across the wall adjacent to the doorway is likely to be small and the system may fail to detect the wall itself. If the goal is moved the composed function will not take this wall into account. However, as in the case of false doorways, the system can observe the function (although not along the existing partition) looking for newly emerging features and then compose a new function.

Looking for additional emerging features is also important when learning in new environments. The different environments, discussed in chapter 7, used the relatively simple situation of two rooms. The function composition system initialised the low level algorithm once on detecting suitable features. In future, to address more complex tasks, an incremental approach will be used. When a new task is being learnt, the system will progressively build up a solution by function composition, as different features emerge. The most revision is necessary at the beginning when the new environment is first encountered and the old theory must be discarded. At present the system is told when this occurs. Of course, this could be detected by watching the partition itself but it would probably take some time. It seems likely that there would be additional indicators of such a situation which might be used in conjunction with the existing partition to determine what has happened.

9.2.2 Representing the Control Function

This thesis has advocated the use of B-splines to reduce the number of bases required to represent the control function. Selecting the number of basis functions in advance, as was done in this thesis, is a form of bias the choice of which requires domain knowledge. If too few basis functions are used the accuracy with which the true function can be represented is limited. If too many functions are used the processing time needed to update functions will be unnecessarily long. In fact, the learning rate may also suffer as there is less generalisation of learning to nearby points. One possible direction of future research is to start with a "simple" spline, with just a few basis functions, and a large degree of smoothness. This accords with the intuitive notion that similar actions are associated with similar states and with little information the "radius of similarity" should be large. As more data is accumulated, if the fit of the existing function to the new data is poor, new bases can added in just those areas to correct this discrepancy.

Using basis functions only where they are needed addresses another problem in reinforcement learning often termed the "curse of dimensionality". The "curse of dimensionality" arises from the rapid increase in the complexity of a representation with an increasing number of state dimensions. If a naive sub-division scheme is used and each dimension is divided into a fixed number of intervals the number of states is exponential in the number of dimensions. Of course it is quite possible that the actual function is relatively simple even with high state dimensionality. In fact, the critical dimensionality of the function is not the dimensionality of the state space but the complexity of the representation that can accurately model it. The idea of using a relatively small set of features to represent a large space of values is not a new topic in machine learning. Aha (1991) has looked at keeping a small set of exemplars in his instance based system. Salganicoff (1993) investigated using KD trees to divide up the input space economically. Moore (1992) has used the same technique in reinforcement learning. So economy of representation is a very general issue.

In function approximation, one way to address this problem is by using wavelets. Wavelets are hierarchical, representing the function at progressively finer scales. The highest levels represent gross characteristics (essentially statistical moments over large regions of the state space). As one moves down the hierarchy finer and finer detail is added to the representation. Using the wavelet representation of splines, additional basis functions can be added to the existing set when and where more accuracy is required (Gortler, Schroder, Cohen, & Hanrahan, 1993; Gortler, 1995). Thus the actual number and position of the basis functions can be decided during learning. One advantage of wavelets is that they are localised in both frequency and space (or time). Thus they are well suited to approximating local discontinuities, such as the features that have been the focus of this thesis.

Using a hierarchical representation ties neatly with approaches to speed up the snake. One direction snake research has taken is fitting the snake to finer and finer resolution scales which is exactly what the wavelet representation provides. It is also particularly suited to the ideas of transfer proposed in this thesis. The features that the hybrid architecture uses are where a greater density of basis functions are required. The transfer would be therefore not only the solution to various subtasks but also the underlying representation needed to accurately fit the complex task.

9.3 A Few Final Words

This thesis has proposed a particular hybrid architecture combining symbolic and subsymbolic processes. It has demonstrated its effectiveness on different tasks in different learning domains. It has shown that symbolic planning can produce not only good functions that aid the exploitation of the space, it can compose search functions that aid the exploration. It is also anticipated that existing and future work in planning will improve the effectiveness of the system. Coupling this sort of planning with a robust learning method such as reinforcement learning addresses many of the limitations of planning, like requiring an accurate model of the world. Present and future reinforcement learning research will do much to improve the system's effectiveness.

Bibliography

Aamodt, A., & Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, *7*(1), 39–59.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-Based Learning Algorithms. *Machine Learning*, *6*(1), 37–66.

Almor, A. (1992). A Grounded Modular Architecture for Cognitive Models- What Symbols Need and What are They Needed for. In *Working Notes: The AAAI Workshop on Integrating Neural and Symbolic Processes: The Cognitive Dimension.*

Amini, A. A., Weymouth, T. E., & Jain, R. C. (1990). Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *12*(9), 855–867.

Anderson, J. R. (1982). Acquisition of Cognitive Skill. *Psychological Review*, *89*, 396–406.

Aumann, G. (1997). Subdivision of Linear Corner Cutting Curves. *Journal for Geometry and Graphics*, *1*(2), 91–103.

Aydede, M. (1997). Language of Thought: The Connectionist Contribution. *Minds and Machines*, *7*(1), 57–101.

Backstrom, C., & Jonsson, P. (1995). Planning with Abstraction Hierarchies can be Exponentially Less Efficient. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1599–1604.

Baird, L. C. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 30–37.

Barto, A., Bradtke, S., & Singh, S. (1995). Learning To Act Using Real-time Dynamic Programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Basye, K., Dean, T., & Kaelbling, L. P. (1995). Learning Dynamics: System Identification for Perceptually Challenged Agents. *Artificial Intelligence*, *72*, 139–171.

Baxter, J. (1995). Learning Internal Representations. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pp. 311–320.

Baxter, J. (1997). A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling. *Machine Learning: Special issue on inductive transfer*, *28*(1), 7–40.

Beer, R. D. (1995a). A Dynamical Systems Perspective On Agent-environment. *Artificial Intelligence*, *72*(1-2), 173–261.

Beer, R. D. (1995b). Computational and dynamical languages for autonomous agents. In Port, R., & van Gelder, T. (Eds.), *Mind as Motion: Explorations in the Dynamics of Cognition*, chap. 5, pp. 121–147. MIT Press.

Bellman, R. E. (1957). *Dynamic programming.* Princeton University press.

Birnbaum, I., & Lapidus, L. (1978). Splines and Control via Discrete Dynamic Programming. *Chemical Engineering Science*, *33*, 415–426.

Bishop, C. M. (1992). Curvature-Driven Smoothing: A Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, *4*(5), 882–884.

Blackmore, J., & Miikkulainen, R. (1995). Visualizing High-Dimensional Structure with the Incremental Grid Growing Neural Network. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 55–63.

Block, N. (1996). What Is Functionalism?. In *The Encyclopedia of Philosophy Supplement.* Macmillan.

Boddy, M., & Dean, T. L. (1989). Solving Time-Dependent Planning Problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 979–984.

Booker, L., Goldberg, D., & Holland, J. (1989). Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, *40*, 235–282.

Boutilier, C., & Dearden, R. (1994). Using Abstractions for Decision-Theoretic Planning with Time Constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1016–1022.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting Structure in Policy Construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1104–1111.

Boyan, J. A., & Moore, A. W. (1995). Generalization in Reinforcement Learning: Safely Approximating the Value Function. In *Advances in Neural Information Processing Systems 7*, pp. 369–376.

Branting, L. K., & Aha, D. W. (1995). Stratified Case-Based Reasoning: Reusing Hierarchical Problem Solving Episodes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 384–390.

Brewer, W. F. (1974). There is no convincing evidence for operant or clasical conditioning in adult humans. In Weimer, W. B., & Palermo, D. S. (Eds.), *Cognition and the Symbolic Process*, pp. 1–42. Lawrence Erlbaum Associates.

Britanik, J., & Marefat, M. (1995). Hierarchical Plan Merging with Applications to Process Planning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1677–1683.

Brooks, R. A. (1986). A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics And Automation*, *2*(1), 14–23.

Brooks, R. A. (1990). Elephants Don't Play Chess. *Robotics and Autonomous Systems*, *6*, 3–15.

Brooks, R. A. (1991a). Intelligence Without Reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 569–595.

Brooks, R. A. (1991b). Intelligence Without Representation. *Artificial Intelligence Journal*, *47*, 139–159.

Brooks, R. A. (1997). From Earwigs to Humans. *Robotics and Autonomous Systems*, *2-4*, 291–304.

Buchanan, B., Barstow, D., Bechtal, R., Bennett, J., et al. (1983). Constructing an Expert System. In *Building Expert Systems*, chap. 5, pp. 127–167. Addison-Wesley.

Bylander, T. (1991). Complexity Results for Planning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 274–279.

Calvin, W. H. (1993). The unitary hypothesis: A common neural circuitry for novel manipulations, language,plan-ahead, and throwing?. In Gibson, K. R., & Ingold, T. (Eds.), *Tools, Language, and Cognition in Human Evolution,*, pp. 230–250. Cambridge University Press.

Caruana, R. (1996). Algorithms and Applications for Multitask Learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 87–95.

Caruana, R. (1997). Multitask Learning. *Machine Learning: Special issue on inductive transfer*, *28*(1), 41–76.

Chalmers, D. J. (1990a). Syntactic Transformations on Distributed Represenations. *Connection Science*, *18*(1-2), 53–62.

Chalmers, D. J. (1990b). Why Fodor and Pylyshyn Were Wrong: The Simplest Refutation. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pp. 340–347.

Chang, S. G., & Vetterli, M. (1997). Spatial Adaptive Wavelet Thresholding for Image Denoising. In *Proceedings of the International Conference on Image Processing*.

Chang, S. G., Yu, B., & Vetterli, M. (1998). Spatially Adaptive Wavelet Thresholding with Context Modeling for Image Denoising. In *Proceedings of the International Conference on Image Processing*.

Chin, C. H., & Dyer, C. R. (1986). Model-based recognition in Robot Vision. *Computing Surveys*, *18*(1), 67–108.

Chomsky, N. (1964). Review of Skinner. In Fodor, J. A., & Katz, J. J. (Eds.), *The structure of language; readings in the philosophy of language*, pp. 1–42. Prentice-Hall.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. The MIT Press.

Christiansen, A. D. (1992). Learning to Predict in Uncertain Continuous Tasks. In *Proceedings of the Ninth International Workshop on Machine Learning*, pp. 72–81.

Chui, C. K., & Wang, J. Z. (1992). On compactly supported spline wavelets and a duality principle. *Transactions of the American Mathematical Society*, *330*, 903–915.

Chui, C. K. (1992). *An Introduction to Wavelets*. Academic Press.

Churchland, P. S. (1986). *Neurophilosophy: toward a unified science of the mind-brain*. MIT Press.

Clark, A. (1997). *Being There: Putting Brain, Body, and World Together Again*. MIT Press.

Clark, A., & Chalmers, D. J. (1998). The Extended Mind. *Analysis*, *58*(1), 7–19.

Clark, A., & Toribio, J. (1994). Doing Without Representing. *Synthese*, *101*, 401–431.

Cohen, L. D. (1991). On Active Contour Models and Balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, *53*(2), 211–218.

Cohen, L. D., & Cohen, I. (1993). Finite element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *15*(11), 1131–1147.

Cohen, P. R., & Feigenbaum, E. A. (Eds.). (1982). *The Handbook of Artificial Intelligence*, chap. XI C, pp. 22–27. HeurisTech Press.

Collins, G., & Pryor, L. (1995). Planning under uncertainty: some key issues. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1567–1573.

Cormier, S. M., & Hagman, J. D. (Eds.). (1987). *Transfer of Learning: Contemporary Research and Applications*. Academic Press.

Cramer, A. E., & Gallistel, C. R. (1997). Vervet monkeys as travelling salesman. *Nature*, *387*, 464.

Crites, R. H., & Barto, A. G. (1995). Improving Elevator Performance using Reinforcement Learning. In *Advances in Neural Information Processing Systems 7*, pp. 1017–1023.

Dahlquist, G., & Bjorck, A. (1969). *Numerical Methods*. Prentice Hall.

Daniel, J. W. (1976). Splines and Efficiency in Dynamic Programming. *Mathematical Analysis and Applications*, *54*.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, pp. 271–278.

de Jong, K. (1988). Learning with Genetic Algorithms: An Overview. *Machine Learning: Special issue on genetic algorithms*, *3*(2-3), 121–138.

Dean, T., Kaelbling, L., Kirman, J., & Nicholson, A. (1995). Planning Under Time Constraints in Stochastic Domains. *Artificial Intelligence*, *76*(1-2), 35–74.

Dean, T., & Lin, S.-H. (1995). Decomposition Techniques for Planning in Stochastic Domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1121–1127.

Dejong, G., & Bennett, S. (1995). Extending Classical Planning to Real-World Execution with Machine Learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1153–1159.

Dennett, D. C. (1987). *The Intentional Stance*. MIT Press.

Dennett, D. C. (1994). What is Intelligence?. In Khalfa, J. (Ed.), *The Darwin College Lectures*. Cambridge Univ. Press.

Dennett, D. C. (1995). *Darwin's Dangerous Idea: Evolution and the Meanings of Life*, chap. 13, pp. 370–400. Peguin Press.

Dennett, D. C. (1996). *Kinds of minds: toward an understanding of consciousness*. Basic Books.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference of Machine Learning*.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*, 269–271.

Donoho, D. L., & Johnstone, I. M. (1993). Adapting to Unknown Smoothness via Wavelet Shrinkage. Tech. rep. Statistics Technical Report 425, Department of Statistics, Stanford University.

Dononho, D. L., & Johnstone, I. M. (1994). Ideal Spatial Adaption via Wavelet Shrinkage. *Biometrika*, *81*, 425–455.

Drummond, C. (1997). Using a Case-Base of Surfaces to Speed-Up Reinforcement Learning. In *Proceedings of the Second International Conference on Case-Based Reasoning*, Vol. 1266 of *LNAI*, pp. 435–444.

Drummond, C. (1998). Composing Functions to Speed up Reinforcement Learning in a Changing World. In *Proceedings of the Tenth European Conference on Machine Learning*, Vol. 1398 of *LNAI*, pp. 370–381.

Drummond, M. (1989). Situated Control Rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pp. 103–113.

Drummond, M., Swanson, K., Bresina, J., & Levinson, R. (1993). Reaction-First Search. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1408–1414.

Dyer, M. G. (1994). Grounding Language in Perception. In *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, chap. XIX, pp. 455–482. Academic Press.

Ellman, T. (1993). Abstraction via Approximate Symmetry. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 916–921.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*, 179–211.

Esch, R. (1983). Functional Approximation. In Pearson, C. E. (Ed.), *Handbook of Applied Mathematics: Selected Results and Methods*, chap. 17, pp. 928–987. Van Nostrand Reinhold Company.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., & Williamson, M. (1993). An Approach To Planning With Incomplete Information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pp. 115–125.

Eubank, R. L. (1988). *Spline Smoothing and Nonparametric Regression*, Vol. 90 of *Statistics, textbooks and monographs*. Marcel Dekker.

Fikes, R., & Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fitts, P. M., & Posner, M. I. (1969). *Human Performance*. Wadsworth.

Fodor, J. A. (1976). *The Language of Thought*. Harvester Press.

Fodor, J. A. (1987). *Psychosemantics: the problem of meaning in the philosophy of mind.* MIT Press.

Fodor, J. A. (1990). *A Theory of Content and Other Essays.* MIT Press.

Fodor, J. A. (1997). Connectionism and the problems of systemacity (continued): why Smolensky's solution still doesn't work.. *Cognition*, *62*, 109–119.

Fodor, J. A., & McLaughlin, B. (1990). Connectionism and the problems of systemacity: why Smolensky's solution doesn't work. *Cognition*, *35*, 183–204.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, *28*, 3–71.

Forbus, K. D., Nielsen, P., & Faltings, B. (1987). Qualitative Kinematics: A Framework. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 430–435.

Fransconi, P., Gori, M., Maggini, M., & Soda, G. (1996). Representation of Finite State Automata in Recurrent Radial Basis Function Networks. *Machine Learning*, *23*(1), 5–32.

Gallant, S. I. (1993). *Neural Network Learning and Expert Systems.* MIT Press.

Gallip, Z. (1986). Efficient Algorithms for Finding Maximum Matching in Graphs. *ACM Computing Surveys*, *18*(1), 23–38.

Geary, A., Lowry, H. V., & Hayden, H. A. (1967). *Advanced Mathematics for Technical Students.* Longmans.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural Networks and the Bias Variance Dilemma. *Neural Computation*, *4*, 1–58.

Gentner, D. (1990). The Mechanism of Analogical Reasoning. In *Readings In Machine Learning*, pp. 601–622. Morgan Kaufmann.

Gervasio, M. T., & Dejong, G. F. (1994). An Incremental Learning Approach for Completable Planning. In *Proceedings of the Eleventh International Conference of Machine Learning*, pp. 78–95.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization Theory and Neural Networks Architectures. *Neural Computation*, *7*(2), 219–269.

Gold, S., & Rangarajan, A. (1996). A Graduated Assignment Algorithm for Graph Matching. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *18*(4), 377–388.

Gordon, G. J. (1995a). Stable Fitted Reinforcement Learning. In *Advances in Neural Information Processing Systems 8*, pp. 1052–1058.

Gordon, G. J. (1995b). Stable Function Approximation in Dynamic Programming. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 261–268.

Gordon, G. J., & Segre, A. M. (1996). Nonparametric Statistical Methods for Experimental Evaluations of Speedup Learning. In *Proceedings of the Thirteenth International Conference of Machine Learning*, pp. 200–206.

Gortler, S., Schroder, P., Cohen, M., & Hanrahan, P. (1993). Wavelet Radiosity. *Computer Graphics, Annual Conference Series Siggraph*, *1*, 205–212.

Gortler, S. J. (1995). *Wavlet Methods For Computer Graphics.* Ph.D. thesis, Princeton University.

Gould, S. J. (1989). *Wonderful life: the Burgess Shale and the nature of history.* W.W. Norton.

Graf, D. H., & LaLonde, W. R. (1989). Neuroplannners and Their Applications To Eyes/Head/Neck Coordination. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1022–1028.

Green, B. F., Wolf, A. K., Chomsky, C., & Laughery, K. (1963). Baseball: An Automatic Question Answerer. In *Computers and Thought*, pp. 207–216. McGraw-Hill.

Grigni, M., Papadias, D., & Papadimitriou, C. (1995). Topological Inference. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 901–906.

Hall, L. O., & Romaniuk, S. G. (1990). A Hybrid Connectionist, Symbolic Learning System. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 783–788.

Hammond, K. J. (1990). Case-Based Planning: A Framework for Planning from Experience. *The Journal of Cognitive Science*, *14*(3), 385–443.

Handelman, D. A., Lane, S. H., & Gelfand, J. J. (1989). Integrating Knowledge-Based System and Neural Network Techniques for Robotic Skill Acquisition. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 193–198.

Harnad, S. (1990). The Symbol Grounding Problem. *Physica D*, *42*, 335–346.

Harnad, S. (1993). Artificial Life: Synthetic Versus Virtual. In *Artificial Life III. Proceedings, Santa Fe Institute Studies in the Sciences of Complexity*, Vol. XVI.

Harnard, S., & Hanson, S. J. (1994). Learned Categorical Perception in Neural Nets: Implication for Symbol Grounding. In Uhr, V. H. L. (Ed.), *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, chap. IX, pp. 191–206. Academic Press.

Hart, P., Nilsson, N., B., & Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum-Cost Path. *IEEE Trans. on Systems Science and Cybernetics*, *4*(2), 100–107.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L. P., & Dean, T. (1998). Hierarchical Solution for Markov Decision Processes Using Macro-Actions. In

Proceedings of the Fourteenth Conference on Uncertainty In Artificial Intelligence, pp. 220–229.

Hendler, J., Tate, A., & Drummond, M. (1990). AI Planning: Systems and Techniques. *AI Magazine*, *10*(2), 61–77.

Herrmann, C. S. (1995). A Hybrid Fuzzy-Neural Expert System for Diagnosis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 494–500.

Herve, J.-Y., Sharma, R., & Cucka, P. (1991). The Geometry of Visual Coordination. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 732–737.

Hillis, W. D. (1988). Intelligence As An Emergent Property; or, The Songs of Eden. In Graubard, S. R. (Ed.), *The Artificial Intelligence Debate: False Starts, Real Foundations*, pp. 143–173. MIT Press.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Holland, J. H. (1976). Escaping Brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J., & Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach 2*, pp. 593–623. Morgan Kaufmann.

Holland, J. H. (1986). Adaptation. In Rosen, R., & Snell, F. N. (Eds.), *Progress in Theoretical Biology IV*, pp. 263–293. Academic Press.

Holland, J. H. (1992). Complex Adaptive Systems. *Daedalus*, 25.

Holte, R., Drummond, C., Perez, M., Zimmer, R., & MacDonald, A. (1994). Searching With Abstractions: A Unifying Framework and New High-Performance Algorithm. In *Proceedings of the Canadian Artificial Intelligence Conference*, pp. 263–270.

Holte, R., & Hernadvolgyi, I. T. (1999). A Space-Time Tradeoff for Memory-Based Heuristics. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*.

Honavar, V., & Uhr, L. (1995). Integrating Symbol Processing and Connectionist Networks.. In Goonatilake, S., & Khebbal, S. (Eds.), *Intelligent Hybrid Systems*, chap. 1, pp. 177–208. Wiley.

Ivanova, I., & Kubat, M. (1995). Initialization of neural networks by means of decision trees. *Knowledge-Based Systems*, *8*(6), 333–344.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, *6*(6), 1185–1202.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kaelbling, L. P. (1993). Hierarchical Learning in Stochastic Domains: Preliminary Results. In *Proceedings of the Tenth International Workshop on Machine Learning*, pp. 167–173.

Kass, M., Witkin, A., & Terzopoulus, D. (1987). Snakes: Active contour models. *International Journal of Computer Vision*, *1*, 321–331.

Knoblock, C. A. (1991). Search Reduction in Hierarchical Problem Solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 686–691.

Knoblock, G. A. (1995). Planning, Executing, Sensing and Replanning for Information Gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1686–1693.

Koenig, S., & Simmons, R. G. (1995). Real-Time Search in Non-Deterministic Domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1660–1667.

Korf, R. E. (1990). Real-Time Heuristic Search. *Artificial Intelligence*, 189–211.

Kuipers, B. J. (1993). Reasoning with Qualitative Models. *Artificial Intelligence*, *59*, 25–132.

Kuipers, B. J. (1996). A hierarchy of qualitative representations for space. In *Working Papers of the Tenth International Workshop on Qualitative Reasoning*.

Kuipers, B. J., & Levitt, T. (1988). Navigation and mapping in large scale space. *AI Magazine*, *9*(2), 25–43.

Lacher, R. C. (1992). The Symbolic/Sub-Symbolic Interface: Hierarchical Network Organisations For Reasoning.. In *Working Notes: The AAAI Workshop on Integrating Neural and Symbolic Processes: The Cognitive Dimension*.

Lakoff, C. (1987). *Women, Fire and Dangerous Things: What Categories Reveal about the mind*. University of Chicago Press.

Lancaster, P., & Salkauskas, K. (1986). *Curve and Surface Fitting An Introduction*. Academic Press.

Leroy, B., Herlin, I. L., & Cohen, L. D. (1996). Multi-resolution algorithms for active contour models. In *Proceedings of the Twelfth International Conference on Analysis and Optimization of Systems Images, Wavelets and PDE'S*.

Leymarie, F., & Levine, M. D. (1993). Tracking Deformable Objects in the Plane Using an Active Contour Model. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *15*(6), 617–634.

Lin, L.-J. (1993). Scaling Up Reinforcement Learning for Robot Control. In *Proceedings of the Tenth International Conference of Machine Learning*, pp. 182–189.

Ma, Z., & Harrison, R. F. (1995). GR2- A Hybrid Knowledge-based System Using General Rules. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 488–493.

MacDonald, A. (1992). Graphs: Notes on Symetries, Imbeddings, Decompositions. Tech. rep. Electrical Engineering Department TR-92-10-AJM, Brunel University, Uxbridge, Middlesex, United Kingdom.

Maes, P. (1990). Situated Agents Can Have Goals. In *Designing Autonomous Agents*, pp. 49–70.

Maes, P. (1994). Modeling Adaptive Autonomous Agents. *Artificial Life Journal*, *1*(1-2), 135–162.

Maes, P., & Brooks, R. A. (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 796–802.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, *55*, 311–365.

Mallat, S., & Zhong, S. (1992). Characterization of Signals from Multiscale Edges. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *14*(7), 710–732.

Markman, A. B., & Dietrich, E. (1998). In Defense of Representation as Mediation. *Psycoloquy*, *9*(48).

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information.* W.H. Freeman.

Mataric, M. J. (1990). Behavioral Synergy Without Explicit Integration. In *Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures*, pp. 130–133.

Mataric, M. J. (1991). Comparative Analysis of Reinforcement Learning Methods. Tech. rep. A.I. Memo 1322, M.I.T. Artifiical Intelligence Lab.

Mataric, M. J. (1994). Reward Functions for Accelerated Learning. In *Proceedings of the Eleventh International Workshop on Machine Learning*, pp. 181–189.

McCallum, R. A. (1995a). Instance-Based State Identification for Reinforcement Learning. In *Advances in Neural Information Processing Systems 7*, pp. 377–384.

McCallum, R. A. (1995b). Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 387–395.

McCauley, R. N. (Ed.). (1996). *The Churchlands and their critics*. Blackwell.

Minsky, M., & Papert, S. (1969). *Perceptrons: an introduction to computational geometry*. MIT Press.

Mitchell, T. (1990). The Need for Biases in Learning Generalizations. In *Readings In Machine Learning*, pp. 184–191. Morgan Kaufmann.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time.. *Machine Learning*, *13*, 103–130.

Moore, A. W. (1992). Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State Spaces. In *Proceedings of the Ninth International Workshop on Machine Learning*.

Moriarty, D. E., & Miikkulainen, R. (1996). Efficient Reinforcement Learning through Symbiotic Evolution. *Machine Learning: Special issue on reinforcement learning*, *22*(1-3), 11–32.

Narenda, K. S., & Thathachar, M. A. L. (1974). Learning Automata - A Survey. *IEEE Transactions On Systems Man And Cybernetics*, *4*(4), 323–334.

Nason, G. (1995). Three-dimensional projection pursuit. Tech. rep., Department of Mathematics, University of Bristol.

Nau, D. S., Gupta, S. K., & Regli, W. C. (1995). AI Planning Versus Manufacturing-Operation Planning: A Case Study.. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1670–1676.

Newell, A., & Simon, H. A. (1963). GPS, A Program That Simulates Human Thought. In *Computers and Thought*, pp. 279–293. McGraw-Hill.

Newell, A., & Simon, H. A. (1976). Computer Science as Empirical Inquiry: Symbols and Search. *Communications Of The Association Of Computing Machinery*, *3*(19), 113–126.

Omlin, C. W., & Giles, C. L. (1994). Extraction and Insertion of Symbolic Information in Recurrent Neural Networks. In Uhr, V. H. L. (Ed.), *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, chap. XII, pp. 271–300. Academic Press.

Osborne, H., & Bridge, D. (1997). Similarity Metrics: A Formal Unification of Cardinal and Non-Cardinal Similarity Measures. In *Proceedings of the Second International Conference on Case-Based Reasoning*, Vol. 1266 of *LNAI*, pp. 235–244.

Parr, R. (1998). Flexible Decomposition Algorithms for Weakly Coupled Markov Decision Problems. In *Proceedings of the Fourteenth Conference on Uncertainty In Artificial Intelligence*, pp. 422–430.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley.

Peng, J. (1995). Efficient Memory-Based Dynamic Programming. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 438–439.

Peng, J., & Williams, R. J. (1994). Incremental Multi-Step Q-Learning. In *Proceedings of the Eleventh International Conference of Machine Learning*, pp. 226–232.

Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics.* Oxford Univ. Press.

Piaget, J., & Inhelder, B. (1967). *The child's conception of space.* W. W. Norton.

Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. In Pinker, S., & Mehler, J. (Eds.), *Connections and Symbols*, chap. 3, pp. 73–194. MIT Press.

Poggio, T., & Girosi, F. (1989). A Theory of Networks for Approximation and Learning. Tech. rep. AI Memo 1140, Massachusetts Institute of Technology, Massachusetts, United States.

Poggio, T., & Girosi, F. (1990). Networks for Approximation and Learning. *Proceedings of the IEEE*, *78*, 481–149.

Poggio, T., Voorhees, H., & Yuille, A. (1985). A Regularized Solution to Edge Detection. Tech. rep. AI Memo 833, Massachusetts Institute of Technology, Massachusetts, United States.

Port, R. F., & van Gelder, T. (Eds.). (1995). *Mind as Motion: explorations in the dynamics of cognition*. MIT Press.

Pratt, L. Y. (1994). Non-literal transfer Among Neural Network Learners. In Mammone, R. (Ed.), *Artificial Neural Networks for Speech and Vision*, pp. 143–169. Chapman and Hall.

Precup, D., Sutton, R. S., & Singh, S. P. (1997). Planning with Closed-Loop Macro Actions. In *Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems*, pp. 70–76.

Precup, D., Sutton, R. S., & Singh, S. P. (1998). Theoretical Results on Reinforcement Learning with Temporally Abstract Options. In *Proceedings of the Tenth European Conference on Machine Learning*, Vol. 1398 of *LNAI*, pp. 382–393.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1993). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

Prieditis, A. E. (1993). Machine Discovery of Effective Admissable Heuristics. *Machine Learning Journal*, *12*(1), 117–141.

Putnam, H. (1977). Meaning and Reference. In Schwartz, S. P. (Ed.), *Naming, Necessity and Natural Kinds*. Cambridge Univ. Press.

Pylyshyn, Z. W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science*. MIT Press.

Quinlan, J. R. (1994). Comparing connectionist and symbolic learning methods. In Rivest, R. (Ed.), *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*, pp. pp. 445–56. MIT Press.

Ramsay, J. O., & Heckman, N. (1996). Spline smoothing with model-based penalties. In *Behavior Research Methods, Instruments, & Computers*.

Reeke, G. N., & Edelman, G. M. (1988). Real Brains and Artificial Intelligence. In Graubard, S. R. (Ed.), *The Artificial Intelligence Debate: False Starts, Real Foundations*, pp. 143–173. MIT Press.

Reeke, G. N., & Sporns, O. (1990). Selectionist Models of Perceptual and Motor Systems and Implications for Functionalist Theories of Brain Function. *Physica D*, *42*, 347–364.

Ring, M. B. (1997). CHILD: A First Step Towards Continual Learning. *Machine Learning: Special issue on inductive transfer*, *28*(1), 77–104.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–408.

Rothe, I., Susse, H., & Voss, K. (1996). The Method of Normalization to Determine Invariants. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *18*(4), 366–376.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition.* Computational models of cognition and perception. MIT Press.

Russell, S. J. (1989). Execution architectures and compilation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 15–20.

Sacerdoti, E. (1974). Planning In A Hierarchy Of Abstraction Spaces. *Artificial Intelligence*, *5*, 115–135.

Sacks, O. W. (1985). *The man who mistook his wife for a hat and other clinical tales.* Summit Books.

Sakakibara, S. (1994). A Practice of Data Smoothing by B-spline Wavelets. In *Wavelets Theory, Algorithms and Applications*, pp. 179–196.

Salganicoff, M. (1993). Density Adaptive Learning and Forgetting. In *Proceedings of the Tenth International Conference of Machine Learning*, pp. 276–283.

Samuels, A. (1959). Some Studies In Machine Learning Using The Game Of Checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.

Schmidhuber, J., Zhao, J., & Wiering, M. (1997). Shifting Inductive Bias with Success Story Algorithm, Adaptive Levin Search and Incremental Self-Improvement. *Machine Learning: Special issue on inductive transfer*, *28*(1), 105–130.

Schnabel, J. A. (1997). *Multi-Scale Active Shape Description in Medical Imaging.* Ph.D. thesis, University of London, London, United Kingdom.

Schwalb, E. (1993). Compiling Bayesian Networks into Neural Networks. In *Proceedings of the Tenth International Workshop on Machine Learning*, pp. 291–297.

Schwartz, J. (1992). Who's Afraid of Multiple Realizability?: Functionalism, Reductionism, and Connectionism. In Dinsmore, J. (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap*, chap. 5, pp. 89–112. Lawrence Erlbaum Associates.

Searle, J. R. (1980). Minds, brains and programs. *Behavioral and Brain Sciences*, *3*, 417–424.

Sejnowski, T., & Rosenberg, C. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, *1*, 145–168.

Setioni, R., & Liu, H. (1995). Understanding Neural Networks via Rule Extraction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 480–485.

Sharkey, N. A., & Jackson, S. A. (1994). Three horns of the representational trilemma. In Honavar, V., & Uhr, L. (Eds.), *Artificial Intelligence and Neural Networks: Steps towards Principled Integration*, chap. VIII, pp. 155–190. Academic Press.

Shavlik, J. W. (1994). Combining Symbolic and Neural Learning. *Machine Learning*, *14*(3), 321–331.

Shavlik, J. W., Towell, C. G., & Noordewier, M. O. (1991). An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. *Machine Learning*, *6*(2), 111–143.

Simon, H. A. (1957). *The Sciences of the Artificial*. MIT Press.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*, 123–158.

Singh, S. P. (1992). Reinforcement Learning with a Hierarchy of Abstract Models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 202–207.

Singley, M. K., & Anderson, J. R. (1989). *The Transfer of Cognitive Skill*. Harvard University Press.

Sloman, A. (1997). What sort of architecture is required for a human-like agent?. In Woodbridge, M., & Rao, A. (Eds.), *Foundations of Rational Agency*. Kluwer.

Smolensky, P. (1987). The constituent structure of connectionist mental states: a reply to Fodor and Pylyshyn. *Southern Journal Of Philosophy*, *XXXVI(supp)*, 137–163.

Smolensky, P. (1988). On the Proper Treatment of Connectionism. *Behavioural and Brain Sciences*, *11*, 1–74.

Stakgold, I. (1979). *Green's functions and boundary value problems*. John Wiley and Sons.

Steels, L. (1990). Exploiting Analogical Representations. *Robots and Autonomous Systems*, *6*(1-2), 71–88.

Steels, L. (1995). Intelligence - Dynamics and Representations. In Steels, L. (Ed.), *The Biology and Technology of Intelligent Autonomous Agents*. Springer-Verlag.

Stenz, A. (1995). The Focussed D* algorithm for Real-Time Replanning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1652–1659.

Stewart, D. E. (1992). *Meschach: Matrix Computations in C.* Centre for Mathematics and its Applications, School of Mathematical Sciences, Australian National University, Canberra, Australia.

Suetens, P., Fua, P., & Hanson, A. (1992). Computational strategies for object recognition. *Computing Surveys*, *24*(1), 5–61.

Sun, R. (1992). A Connectionist Model for Commonsense Reasoning Incorporating Rules and Similarities. *Knowledge Acquisition*, *4*, 293–321.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224.

Sutton, R. S. (1995). TD Models: Modeling the world at a Mixture of Time Scales. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 531–539.

Sutton, R. S. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044.

Sutton, R. S., & Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In Gabriel, M., & Moore, J. (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pp. 497–537. MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Tadepalli, P., & Ok, D. (1996). Scaling up Average Reward Reinforcement Learning by Approximating the Domain Models and the Value Function. In *Proceedings of the Thirteenth International Conference of Machine Learning*, pp. 471–479.

Tchoumatchenko, I., & Ganscia, J.-G. (1994). A Bayesian Framework to Integrate Symbolic and Neural Learning. In *Proceedings of the Eleventh International Workshop on MachineLearning*, pp. 302–308.

Terzopoulos, D. (1986). Regularization of Inverse Visual Problems Involving Discontinuities. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *8*(4), 413–423.

Tesauro, G. (1988). Neurogammon: a neural network backgammon program.. In *IJCCN Proceedings III*, pp. 33–39.

Thrun, S. (1994). Extracting Rules from Artificial Neural Networks with Distributed Representations. In *Advances in Neural Information Processing Systems 7*, pp. 505–512.

Thrun, S., & Schwartz, A. (1994). Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems 7*, pp. 385–392.

Tolman, E. C. (1973). Cognitive Maps in Rats and Men. In *Image and environment: cognitive mapping and spatial behavior*, chap. 2, pp. 27–50. Aldine Pub. Co.

Touretzky, D., & Pomerleau, D. (1994). Reconstructing physical symbol systems. *Cognitive Science*, *18*(2), 345–353.

Towell, G. G., Shavlik, J. W., & Noordewier, M. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the Eigth National Conference on Artificial Intelligence*, pp. 861–866.

Tsitsiklis, J. N., & Roy, B. V. (1997). An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Tsoi, A. C., So, D. S. C., & Sergejew, A. (1994). Classification of Electroencephalogram Using Artificial Neural Networks. In *Advances in Neural Information Processing Systems 6*, pp. 1151–1158.

Turing, A. M. (1950). Computing Machinery and Intelligence. In *Mind*, pp. 433–460.

Unser, M. (1997). Ten good reasons for using spline wavelets. In *Proceedings of SPIE: Wavelet Applications in Signal and Image Processing*, Vol. 3169, pp. 422–431.

Unser, M. (1998). Splines: A perfect fit for signal processing. Tech. rep. EPFL report, Biomedical Imaging Group, Swiss Federal Institute of Technology, Lausanne, Switzerland.

Utgoff, E., & Mitchell, T. M. (1982). Acquisition of appropriate bias for concept learning. In *Proceedings of the Second National Conference On Artificial Intelligence*, pp. 414–418.

van der Smagt, P. P., & Krose, B. J. A. (1991). A Real-Time Learning Neural Robot Controller. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, pp. 351–356.

van Gelder, T. (1990). Compositionality: A Connectionist Variation on a Classical Theme. *Cognitive Science*, *14*, 355–384.

van Gelder, T. (1995). What Might Cognition be, if not Computation. *The Journal of Philosophy*, *91*(7), 345–381.

Varela, F., Thompson, E., & Rosch, E. (1991). *The Embodied Mind: Cognitive Science and Human Experience*. MIT Press.

Veloso, M. M., & Carbonell, J. G. (1993). Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage and Utilization. *Machine Learning*, *10*(3), 249–278.

Veloso, M. M., Munoz-Avila, H., & Bergmann., R. (1996). General-purpose case-based planning: Methods and systems. *AI Communications*, *9*(3), 128–137.

Watkins, C. J., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, *8*(3-4), 279–292.

Weld, D. S. (1994). An Introduction to Least Commitment Planning. *AI Magazine*, *15*(4), 27–61.

Williams, D. J., & Shah, M. (1992). A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics and Image Processing: Image Understanding*, *55*(1), 14–26.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: a new foundation for design.* Ablex.

Zhang, W., & Dietterich, T. (1995). A Reinforcement Learning Approach to Job-shop Scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1114–1120.

Zwillinger, D. (Ed.). (1998). *Standard Mathematical Tables and Formulas* (30th edition). CRC.