Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks

Chris Drummond cdrummon@csi.uottawa.ca
Department of Computer Science,
University of Ottawa,
Ontario, Canada, K1N 6N5

Abstract

This paper discusses a system that accelerates reinforcement learning by using transfer from related tasks. Without such transfer, even if two tasks are very similar at some abstract level an extensive re-learning effort is required. The system achieves much of its power by transferring parts of previously learnt solutions rather than a single complete solution. The system exploits strong features in the multi-dimensional function produced by reinforcement learning in solving a particular task. These features are stable and easy to recognise early in the learning process. They generate a partitioning of the state space and thus the function. The partition is represented as a graph. This is used to index and compose functions stored in a case base to form a close approximation to the solution of the new task. Experiments demonstrate that function composition can produce between one and two orders of magnitude increase in learning rate compared to a basic reinforcement learning algorithm.

1. Introduction

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 paper shows how to exploit the results of prior learning to speed up the process while maintaining the robustness of the general learning method.

The system achieves much of its power by transferring parts of previously learnt solutions rather than a single complete solution. The solution pieces represent knowledge about how to solve certain subtasks. We might call them macro-actions (Precup, Sutton, & Singh, 1997), with the obvious allusion to macro-operators commonly found in Artificial Intelligence systems. The main contribution of this work is in providing a way of automatically identifying these macro-actions and mapping them to new tasks.

This work uses syntactic methods of composition much like in symbolic planning, but the novelty arises in that the parts being composed are multi-dimensional real-valued functions. These functions are learnt using reinforcement 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.

The subtasks are identified on the basis of strong features in the multi-dimensional function that arise during reinforcement learning. The features are not “in the world” but in the system’s interaction with the world. 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. One important aspect of these features is that they largely dictate the shape of the function. If the features differ by a small amount one would expect the function to differ by a small amount.

The features generate a partitioning of the function. A popular technique in object recognition, the snake (Suetens, Fua, & Hanson, 1992), is used to produce this partition. 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. The boundary of the region is a low order polygon demarking an individual subtask. This is repeated until the whole function is covered. The polygons are converted into discrete graphs, a vertex of the polygon becoming a node of the graph. Merging these graphs produces a composite graph representing the whole task.

The composite graph is used to control the transfer by accessing a case base of previously learnt functions. The case base is indexed by graphs. The relevant function is determined by matching a subgraph of the composite graph with one acting as an index to a case. The associated functions are transformed and composed to form a solution to the new task. This is used to reinitialise the lower level learning process. It is not necessary for the transfer to produce an exact solution for the new task. It is sufficient that the solution is close enough to the final solution often enough to produce an average speed up. Reinforcement learning will further refine the function and quickly remove any error.

This paper demonstrates the applicability of transfer in two different situations. In the first, the system learns a task for a particular goal position and then the goal is moved. Although the function itself will change significantly the partition generated on the initial task can be used to compose the function for the new task. In the second situation considered, the system is placed in a different environment within the same domain. Here a new partition has to be extracted to control the composition process.

This paper unifies and significantly extends previous work by the author (Drummond, 1997, 1998). In the main the focus has been in extending the applicability of the partitioning approach introduced in Drummond (1998). One limitation of the original approach was that the snake could only extract polygons that were rectangles. This paper relaxes this restriction, allowing it to be applied to a different environment within the same domain and to a different task domain. Although lifting this restriction removes some desirable bias, the experiments demonstrate that none of the efficacy of the original system is lost. Further, the results are more broadly obtained on the larger set of related tasks and in a different domain. Overall the function composition approach can produce between one and two orders of magnitude increase in learning rate when compared to a basic reinforcement learning algorithm.

The rest of the paper begins with Section 2 giving a very high level discussion of the approach taken. Section 3 gives a more in depth discussion of the techniques used. Section 4 presents experiments demonstrating a substantial increase in learning rate. Subsequent sections deal with limitations, future work and related research.

2. An Overview

The intent of this section is to appeal to the intuitions of the reader, leaving much of the detail to later in the paper. The subsections that follow will demonstrate in turn: that there are features in the function produced by reinforcement learning; that graphs based on these features can be used to control the composition of the function pieces; that these features are easy to detect early in the learning process; that these features exist in multiple domains.

2.1 Features in the Reinforcement Learning Function

This overview begins with a very high level introduction to reinforcement learning and the function it produces. It will show there are features in this function which can be extracted and converted into a graphical representation.

One of the experimental testbeds used is this paper 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 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 are the walls of the rooms, the thick lines the boundary of the state space.

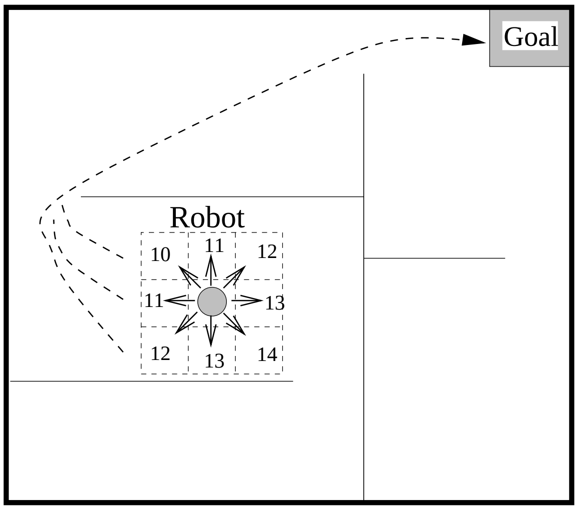

Figure 1: Robot Navigating Through a Series of Rooms

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. 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 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.

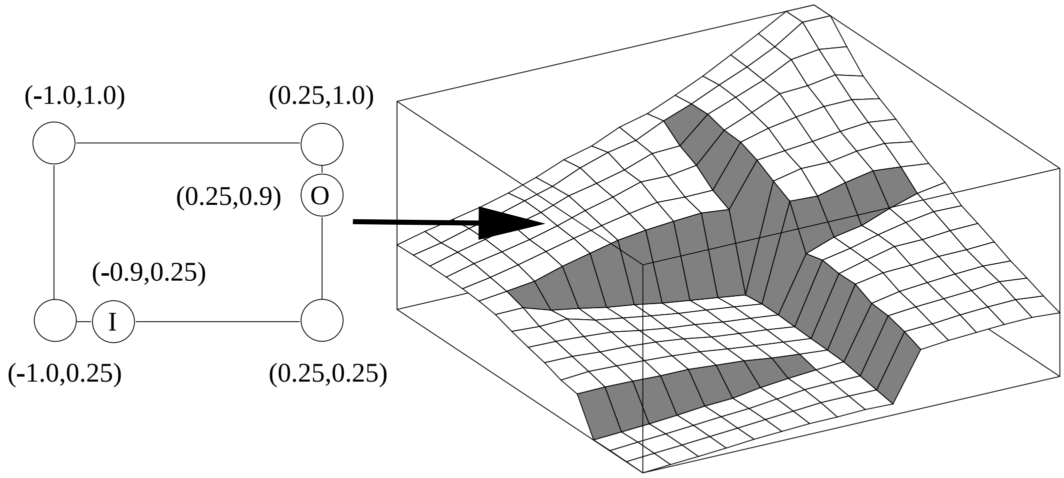

Figure 2: The Reinforcement Learning Function

The function shown in Figure 2 is the result of reinforcement learning on the problem of Figure 1. But instead of it representing the actual distance to goal, it represents essentially an exponential decay with distance to goal. The reasons for this will be made clear in Section 3.1. The shaded areas represent large gradients in the learnt function. Comparing this to the environment shown in Figure 1 it is apparent that these correspond to the walls of the various rooms. These are the *strong features* discussed in this paper. 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.

An edge detection technique called a snake is used to locate these features. The snake produces a polygon, in this instance a rectangle, locating the boundary of each room. The doorways to the room occur where the differential of the function is at a local minimum. The direction of the differential with respect to that of the large gradient determines if it is an entrance or an exit. From this information a plane graph, with an (x, y) coordinate for each node, is constructed. Figure 2 shows one such example, for the room at the top left corner of the state space, subsequent graphs will not show the coordinates. Nodes corresponding to the doorways are labelled “I” or “O” for in and out respectively.

2.2 Composing Function Pieces

This overview continues by showing how the graphs extracted from the features in the function learnt by reinforcement learning can be used to produce a good approximation to the solution for a new goal position. The left hand side of Figure 3 shows plane graphs for all the rooms (ignore the dashed lines and circles for now). The node representing the goal is labelled "G". A directed edge is added from "In" to "Out" or "In" to "G" as appropriate. Associated with this edge is a number representing the distance between the nodes. This is determined from the value of the function at the points of the doorways. Each individual graph is then merged with its neighbour to produce a graph for the whole problem, the right hand side of Figure 3. The doorway nodes have been relabelled to "D". The composite graph represents the whole function. Each individual subgraph represents a particular part of the function. This information is stored in a case base. Each subgraph is an index and the corresponding part of the function is the case.

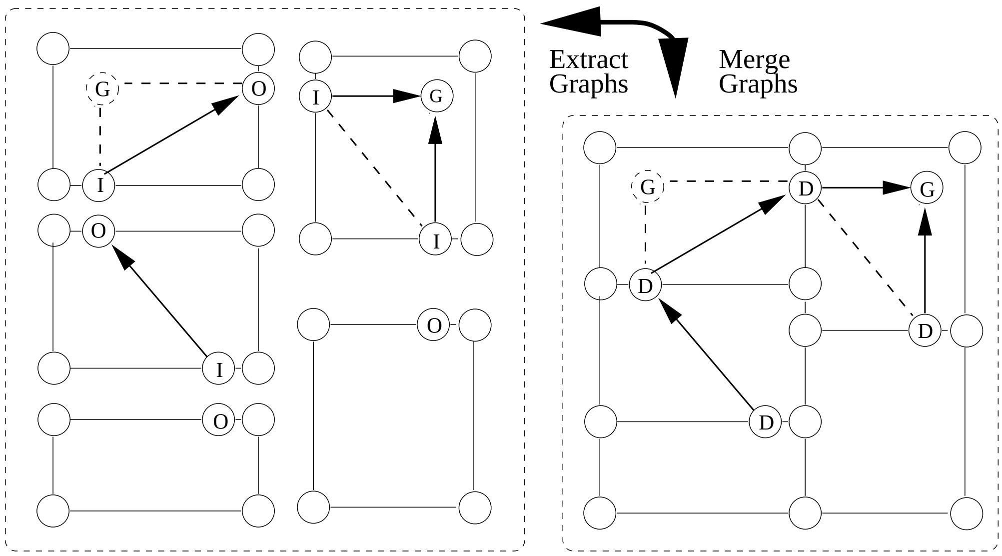

Figure 3: Graphical Representation

Now suppose the goal is moved from the top right corner to the top left corner of the state space. Reinforcement learning in its most basic form would be required to learn the new function from scratch. In this work if the goal is moved, once the new goal position is known the node representing the goal can be relocated. The new goal position is shown as the dashed circle in Figure 3. The edges connecting the doorways and the goal are changed to account for the new goal position. The dashed lines representing the new edges replace the arrows in the same subgraph. To produce a new function, the idea is to regress backwards from the goal along these edges. For each edge, the small subgraph containing the edge is extracted. The extracted subgraph is used to index the case base of functions. The retrieved function is transformed and added to the appropriate region of the state space to form the new function.

In this example some of the existing subgraphs match the new configuration. The two that are 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 graphs in the case base may better match the new task. The best match for the subgraph containing the new goal is, in fact, the subgraph for the goal in the original

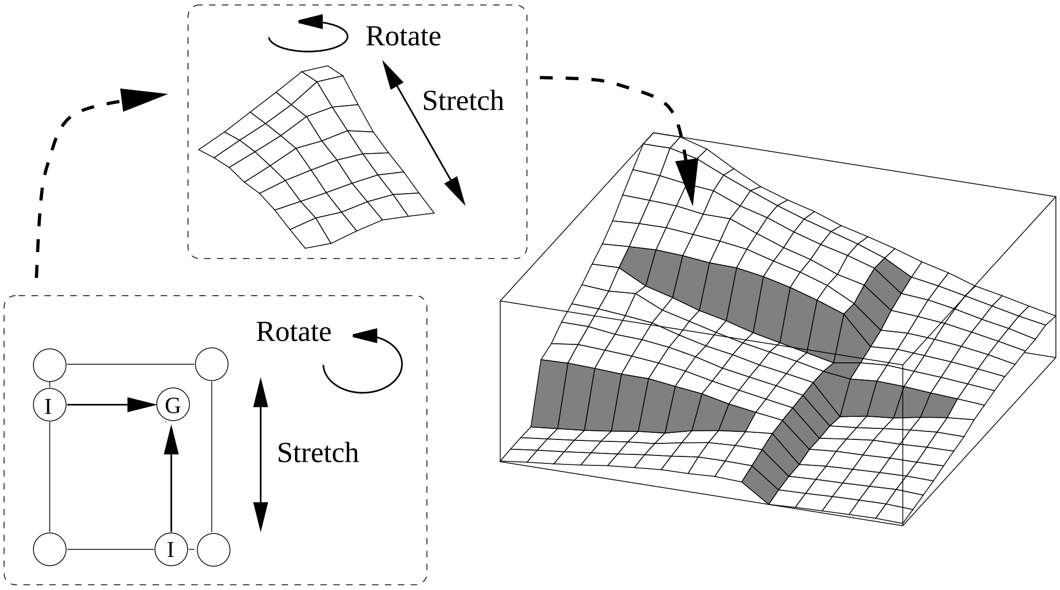

Figure 4: Function Composition

problem. To fit this to the new task the plane graph is rotated and stretched slightly in the new x direction by changing the coordinates of its nodes, see Figure 4. Then this same transformation is applied to the function. But for the room containing the original goal a case obtained when solving another task is a better match. 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 "O" doorway. Because the matching of the subgraphs allows some error and asymmetric scaling may be used 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.

An important stage in this process is locating the new goal. This can be done relatively easily by using the known partition to compose a series of search functions. But first 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. The system then composes a search function, by assuming a particular room contains the goal. The function for the room is set to a constant. This does not bias the search to any particular part of the room and allows some limited learning to encourage exploration of the room. 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 process is repeated until the goal has been located ten times, this ensures 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.

2.3 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.

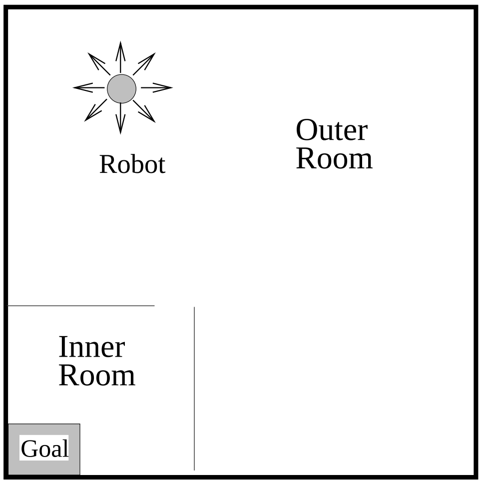

Figure 5: The Old Task

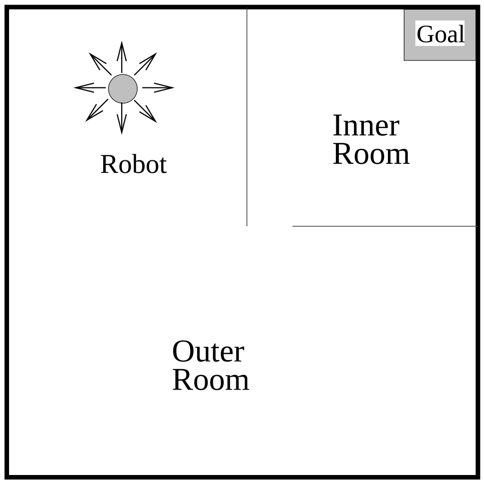

Figure 6: The New Task

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 5 and 6 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 5. We would hope that we could adapt the old solution to fit the closely related "New task" of Figure 6.

The steps in this example are essentially those in the previous one. But now 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. Then we can proceed much as before. First a graph for the inner room is extracted. The best matching graph in the case base from the old task is rotated and stretched to fit the new task. Next a matching graph for the outer L-shaped room is rotated and stretched around the larger inner room. The same transforms are then applied to the associated functions, any height adjustments carried out and the functions composed to form an approximate solution to the new task.

In this example the first step in the process is to locate the goal. As there is no partition to aid search, the search function is set to a mid-range constant value (see Figure 7). This allows some limited learning which encourages the system to move away from regions it has explored previously to prevent 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 8). If such a function does not exist in the case base any rough approximation could be used instead. The "no walls" function is not used exactly as stored in the case base. 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.

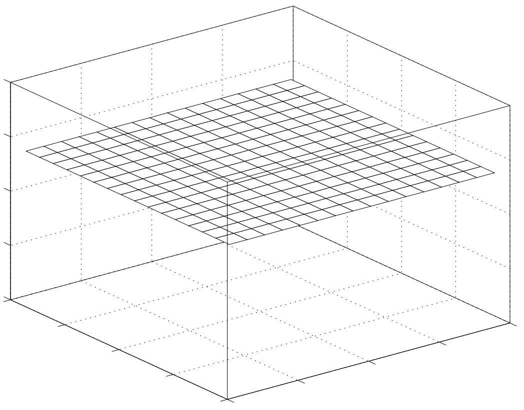

Figure 7: Start Function

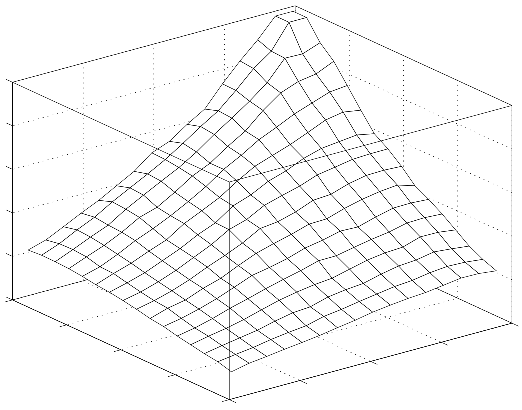

Figure 8: Intermediate Function

Figure 9 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 by taking the magnitude of the gradient vector of this function as shown in Figure 10. This is the absolute value of the largest gradient in any direction, it forms hills corresponding to the large gradients of the function. Figure 11 shows the reinforcement learning function for the new task if it had been allowed to converge to a good solution, Figure 12 shows the gradient. Both gradients have roughly the same form even though the converged function took in excess of 200,000 steps.

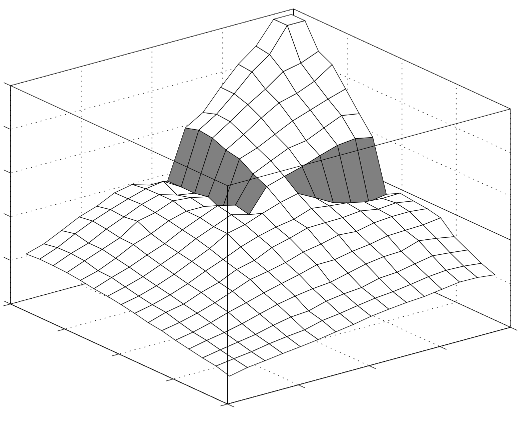

Figure 9: Early Function

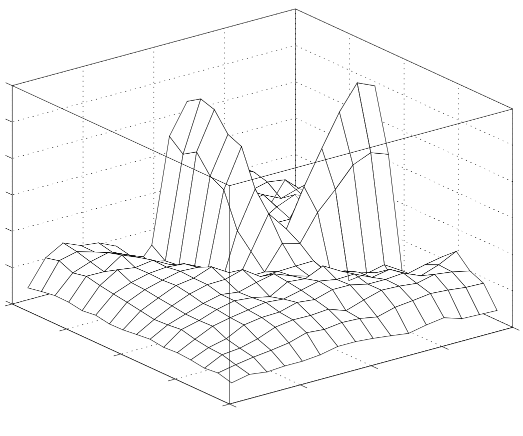

Figure 10: Early Gradient

After the "no walls" function is introduced the system looks for emerging features. The snake will typically filter out features that are too small and not well formed. Additional filtering at the graphical level further constrains acceptable features. The total set of features

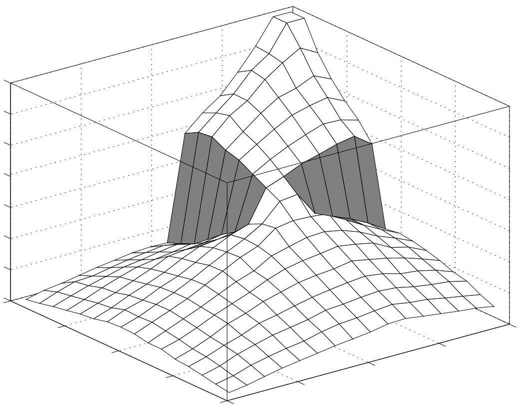

Figure 11: New Task Function

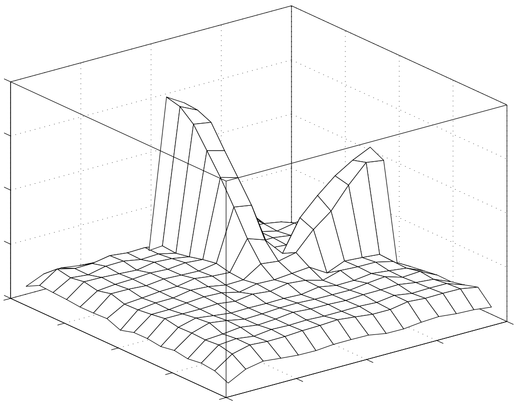

Figure 12: New Task Gradient

must produce a consistent composite graph, the doorways from different subgraphs must align and the graph must overlay the complete state space. There must also be a matching case in the case base.

2.4 A Different Task Domain

The previous sections dealt with a simple robot navigation problem. This section demonstrates that these features also exist in a quite different domain, that of a two degrees of freedom robot arm, as shown in Figure 13. The shoulder joint can achieve any angle between of $\pm\pi$ 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 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 state space for the purposes of reinforcement learning is the configuration space for the arm, sometimes called the joint space (see Figure 14). 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 14. 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 13. 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 such 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.

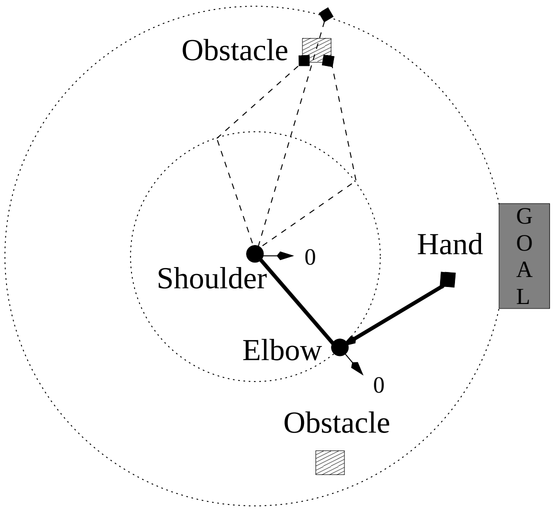

Figure 13: Work Space

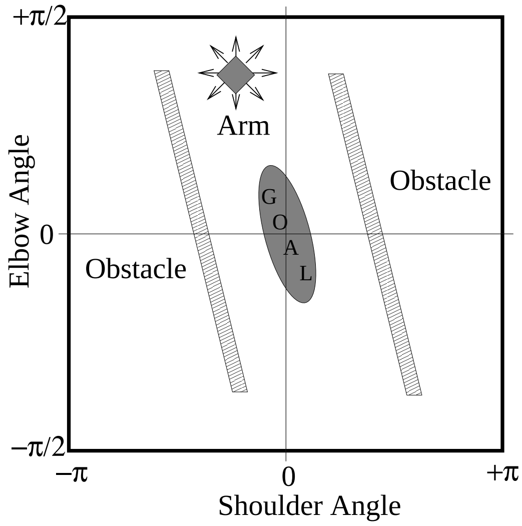

Figure 14: Configuration Space

At the moment the feature extraction method is limited to these simpler shapes, this will be addressed later in the paper in the discussion of limitations and future work of Section 5.

The reinforcement learning function produced by this problem is shown in Figure 15. 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 a 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.

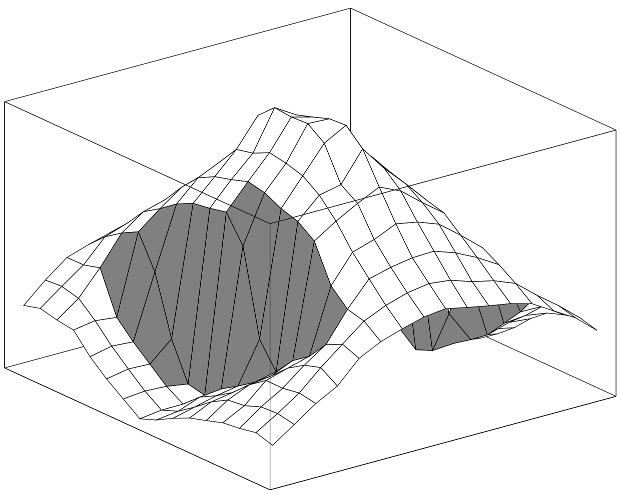

Figure 15: The Reinforcement Learning Function

3. Details of The Techniques Used

This section will discuss in more detail the techniques used. These include: reinforcement learning to produce the initial function, snakes to extract the features producing the graph, and the transformation and composition of the subgraphs and their corresponding functions to fit the new task.

3.1 Reinforcement Learning

Reinforcement learning works by progressively improving estimates of the distance to goal from each state. This estimate is updated by the best local action, i.e. the one moving the robot, or arm, to the new state with the smallest estimated distance. Early in the learning process only states close to the goal are likely to have accurate estimates of true distance. Each time an action is taken, the estimate of the new state is used to update the estimate of the old state. Eventually this process will propagate back accurate estimates from the goal to all other states.

Rather than directly recording the distance to goal, this paper uses the more usual representation of reinforcement learning, the expected discounted reward for each state $E[\sum_{t=1}^{\infty} \gamma^t r_t]$. The influence of rewards, r_t, are reduced progressively the farther into the future they occur by using a γ less than one. In this work, the only reward is for reaching the goal. So the farther the state is from the goal the smaller the value. This forms a function over the state space, as shown in Figure 2. The use of an expectation here allows the actions to be stochastic, so when the robot, or arm, takes a particular action in a particular state, the state reached is not always the same.

To do the reinforcement learning this research uses the Q-learning algorithm (Watkins & Dayan, 1992). This algorithm assumes the world is a discrete Markov process, thus both states and actions are discrete. For each action a in each state s, Q-learning maintains a rolling average of the immediate reward r plus the maximum value of any action a' in the next state s', see Equation 1. The function discussed in previous sections and shown in the figures represents this maximum value. The action selected in each state is usually the one with the highest score. 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.

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

Watkins and Dayan (1992) proved that this will converge to the optimal value with certain constraints on the reward and the learning rate α. The optimal solution is produced by taking the action with the greatest value in any state. Thus in both experimental domains, a greedy algorithm will take the shortest path to the goal once learning is complete. The extension to continuous spaces may be done using function approximation. The simplest method, and the one used here, is to divide the state dimensions into intervals. Each resulting cell then represents the average Q-value of taking a particular action from somewhere within a region of the state space. In many applications this method is successful but there exists no general proof of its convergence.

3.2 Feature Extraction

Feature extraction uses a vision processing technique called a snake. The left hand side of Figure 16 is the gradient from Figure 10 when extracting features early in the learning process. 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 right hand side of Figure 16 the dashed lines are contour lines for the small inner room as indicated.

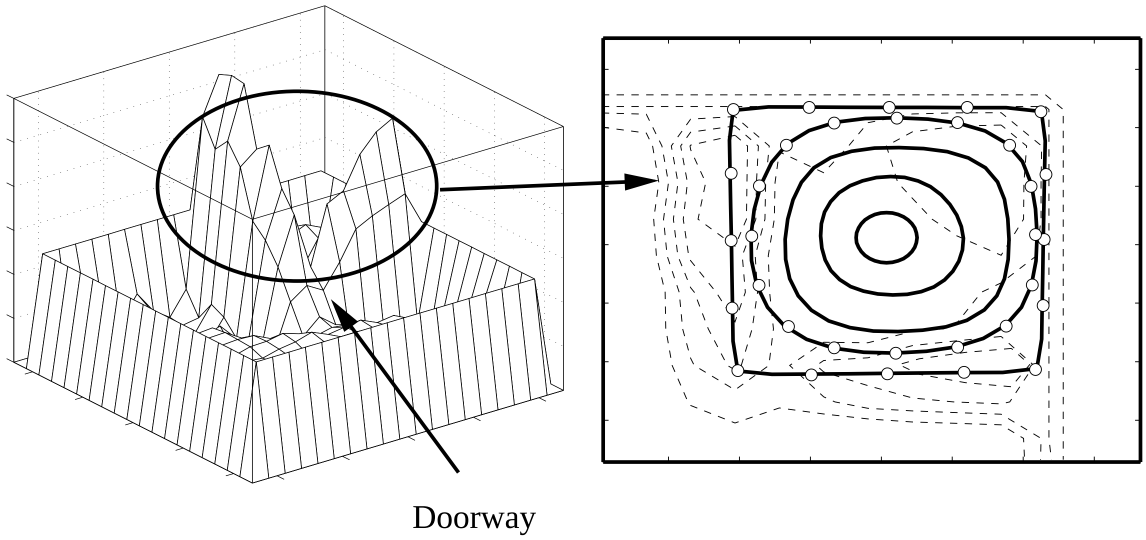

Figure 16: Fitting the Snake

The bold lines in right hand side of Figure 16 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. 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 16. 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. A more detailed mathematical treatment of this approach is given in Cohen and Cohen (1993). Looking at the gradient plot, the doorways are regions with a small differential between the ridges. The position of the doorways can be determined from the magnitude of the gradient along the snake. Taking the corner points of the quadrilateral and the position of the doorways, a plane graph is produced. The quadrilateral delimits a region of the state space, and therefore of the learnt function. This becomes a case in the case base, the corresponding graph its index.

3.2.1 Three Extensions To The Snake Approach

This section introduces three extensions to the basic snake approach to facilitate the extraction of features.

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 constraints. Looking at the gradient function and contour lines of Figure 16, 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 on the left hand side of Figure 17.

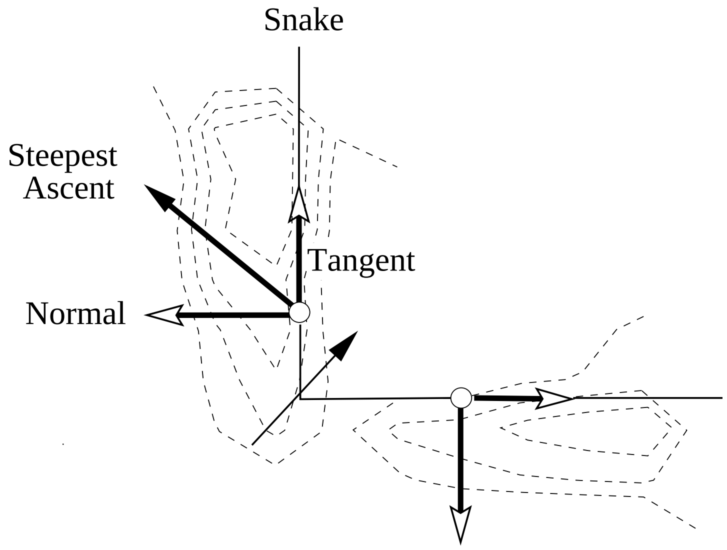

Figure 17: Controlling Forces on The Snake

This force can be broken into two components with respect to the snake, a normal and a tangential force. 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, as indicated by the thin black arrow in Figure 17. 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 Drummond (1998) this degradation of the accuracy was prevented by restricting the snakes to rectangular shapes. But with the weakening of this constraint to more general polygons this effect again becomes a problem. This 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 does not significantly restrict the motion of the snake, all that is being removed is the component along the body of the snake. Thus it mainly prevents the stretching and shrinking of the snake due to the gradient.

The second extension controls the way the snake is expanded to reach the base of the hills. In 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 in Figure 6. 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 18, the bold continuous 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.

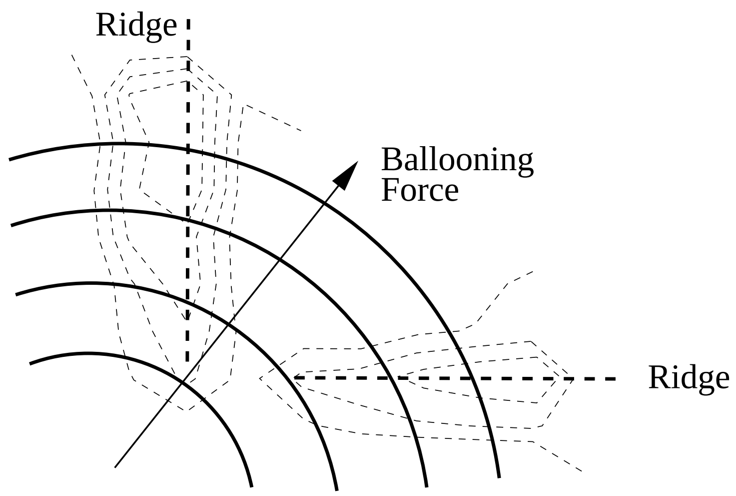

Figure 18: Using The Ballooning Force

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 roughly at 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. To increase the effectiveness of this idea, the absolute value of the differential of the gradient is thresholded, values above the threshold being set to one those below to zero. It is then smoothed with a truncated Gaussian, as shown in Figure 19. This produces bowls associated with each room. In this example the smoothing has almost completely obscured the presence of the doorway, although this is generally not the case.

The snake is initialised as a small circle at the minimum of one of these bowls. This is shown as the circle in the middle of Figure 20, where the dashed lines are the contour lines of this function. It then flows outwards so as to follow the contour lines of the bowl, the largest component of the flow being in the direction of the arrows in Figure 20. This 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 tend to get pushed inwards, those lower pushed outwards. The surface tension of the mercury is produced by various smoothing constraints on the differential and curvature of the snake (see Appendix A).

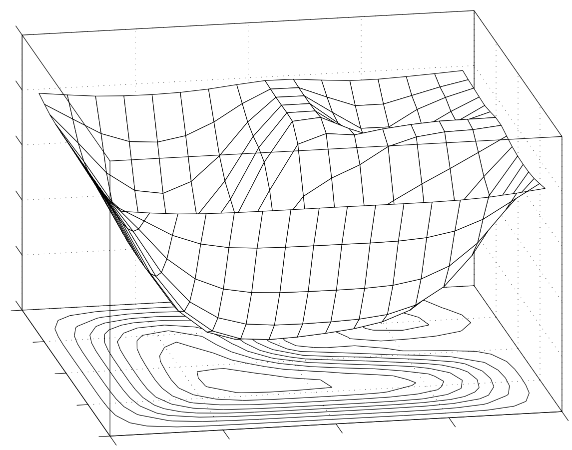

Figure 19: Smoothed Function

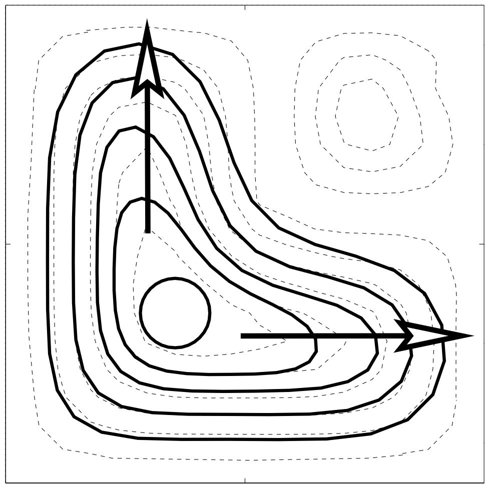

Figure 20: Mercury Flow

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 14 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.

The solution is to limit the amount the snake can change its shape as it grows by holding its curvature proportional to its length. In Figure 20 it is apparent that the snake takes up a good approximation to the shape of the room some time before it reaches the ridges. If the shape can be locked-in before reaching the ridges the problem just described can be avoided. When the snake is initialised the only constraint is smoothness. As the snake is expanded this smoothness constraint is progressively weakened and the curvature constraint progressively strengthened. (If the snake length increases by one quarter the smoothness factor is halved and the curvature factor doubled). This progressively locks in the shape while still allowing the snake to make small local adjustments to better fit the features.

3.3 Transformation

This section discusses the matching process - how a subgraph is used to locate and transform a function from the case base. The matching 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 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. Associated with each node of a subgraph is an (x, y) coordinate. An affine transform, Equation 2, 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.

$$x' = C_0x + C_1y + C_2 \qquad y' = C_3x + C_4y + C_5 \tag{2}$$

Ideally, the transformed nodes would be positioned exactly over the mapped nodes, but this is not always possible. Even with simple rectangular shapes the case base may not contain a graph with exactly the same doorway positions. As the position of certain nodes is more important than others, a weighted sum of squared distances is used, the default weight being one. Figure 21 shows how this is used for fitting the L-shaped rooms of Section 2.3. The left hand side of the figure 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. By setting to ten the weights of the nodes of the corners adjacent to the doorway a good fit is achieved around the inner room by sacrificing fit far from the doorway. Where the new graph overlays the old graph, values are assigned by using bilinear interpolation on the discrete values of the function. Where is does not bilinear extrapolation is used, based on the closest four values.

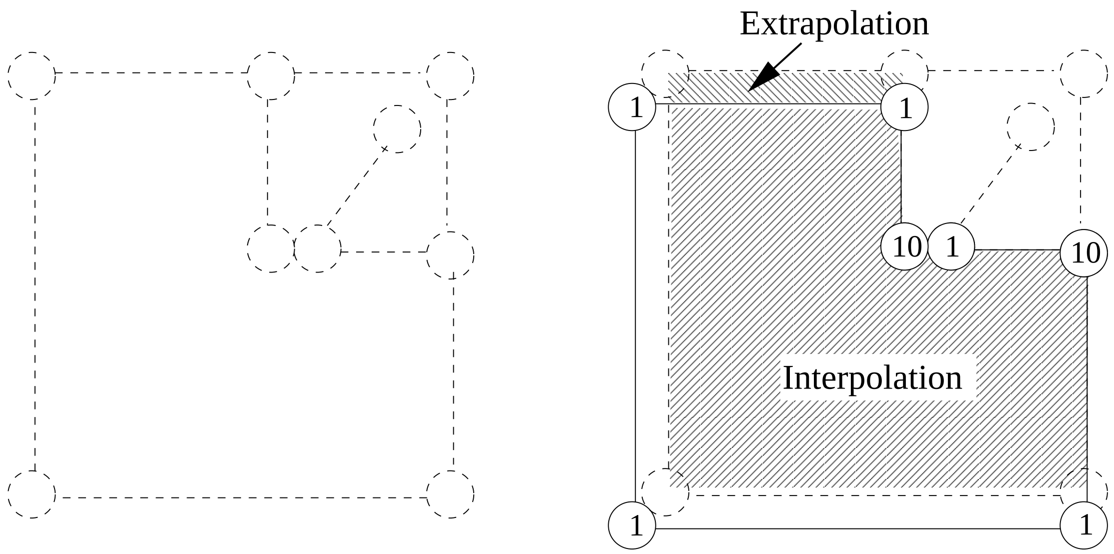

Figure 21: Weighting Graph Nodes

The outer L-shaped room in Figure 21 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 0.25. 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 4.5. But this error should be small and reinforcement learning will correct it.

Not only should the fit be good but we would also prefer that the amount of transformation be small. All transforms 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. If we ignore the effect of shear, which tends to be very small in our examples, we can rewrite the affine transformation in the form of Equation 3.

$$x' = a\cos(\theta)x - a\sin(\theta)y + t_x \qquad y' = b\sin(\theta)x + b\cos(\theta)y + t_y \tag{3}$$

In this form each type of transformation can be more easily identified and penalised individually. Penalty factors for these types of transform can be added to the error for the misalignment of nodes. These are the numerical constants in Equation 4. If no isomorphism is found with an error less than 1.5, a constant function will be used as a default. Otherwise the function associated with the case base graph that produces the smallest error will be used. The same transformation applied to this graph will then be applied to the function.

$$\begin{aligned}
error &= \textstyle\sum_i w_i(x_i^2 + y_i^2) && \text{(node misalignment)}\\
&+ 0.500\left(\log_2\left|\tfrac{a}{b}\right|\right)^2 && \text{(differential scaling)}\\
&+ 0.025\left(\log_2\left(\tfrac{|a+b|}{2}\right)\right)^2 && \text{(scaling factor)}\\
&+ 0.025\,(t_x^2 + t_y^2) && \text{(translation factor)}\\
&+ 0.0125\,((1-H(a)) + (1-H(b))) && \text{(reflection factor)}
\end{aligned} \tag{4}$$

The height of the retrieved function is adjusted in two circumstances. The first is when the retrieved function is added to the function for the whole state space. We need to abut the individual functions so that the result is smooth at the doorways. As the function is an exponential it is just multiplied by the exponential of accumulated distance to the goal, after normalising the function so that it is one at the "out" doorway. The second circumstance is 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 symmetric the result is exact, assuming distance is based on the linear combination of the two dimensions. With asymmetric scaling the result is not exact. But if the difference between the two scale factors is relatively small it is a useful approximation to use their average.

3.4 Composition

This section describes function composition, how the transformation is applied successively to the series of subgraphs extracted from the composite graph. Function composition uses a slightly modified form of Dijkstra's algorithm (Dijkstra, 1959) to traverse the edges between doorway nodes. The left hand side of Figure 22 shows the composite graph after moving the goal in the robot navigation example of Section 2.2. The right hand side shows the graph traversed by Dijkstra's algorithm.

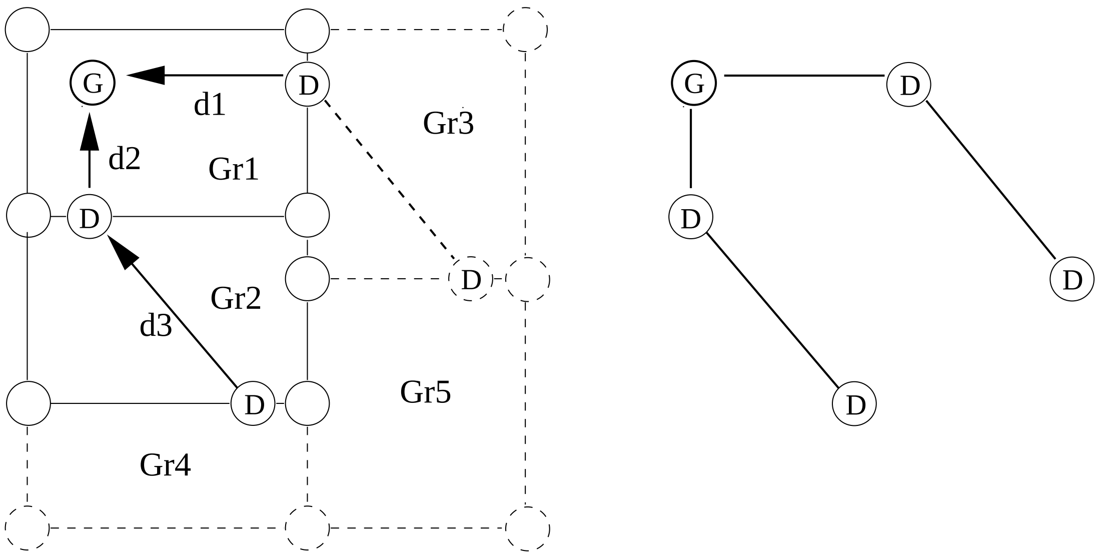

Figure 22: Using Dijkstra's Algorithm

To begin the process, the subgraph which contains the goal is extracted and the 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, d1 and d2 in Figure 22. 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. At each stage when a subgraph is matched, the corresponding transformed function is retrieved and added to the new function in the appropriate region.

In this example 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 on the left hand side of Figure 23, then the graph on the right hand side of Figure 23 would result. There are now two possible paths to the goal 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 as shown in Figure 24: 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, had been 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

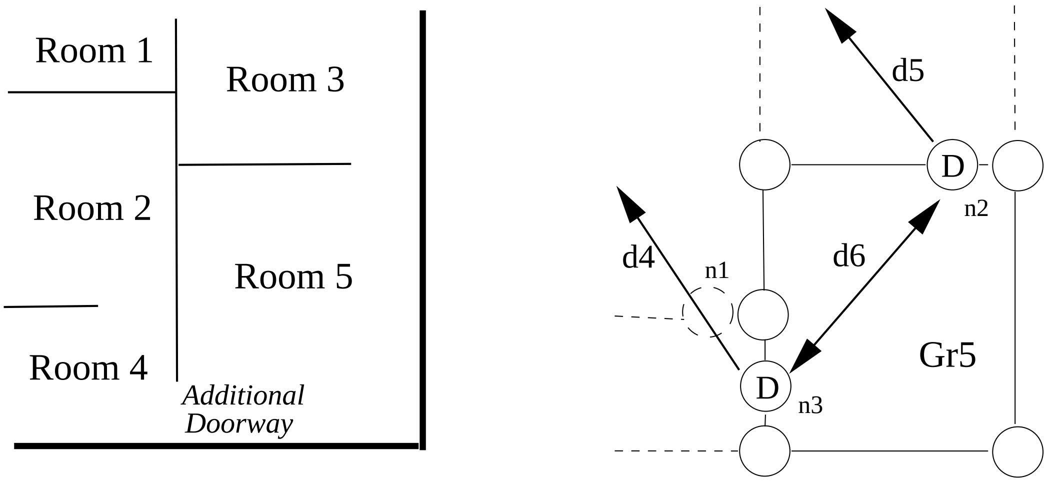

Figure 23: Multiple Paths to Goal

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 23. 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 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.

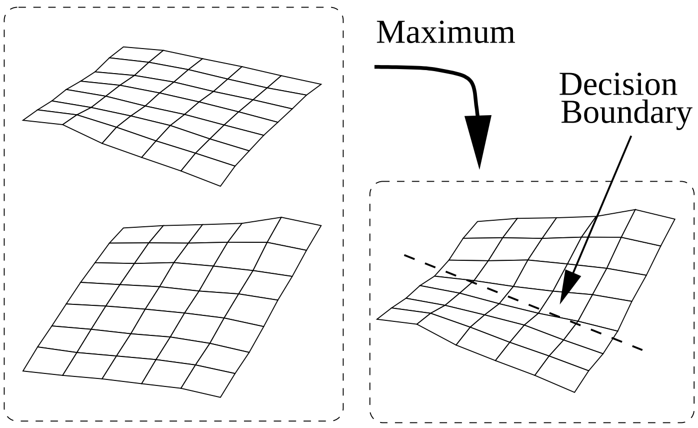

Figure 24: Combining Two Functions

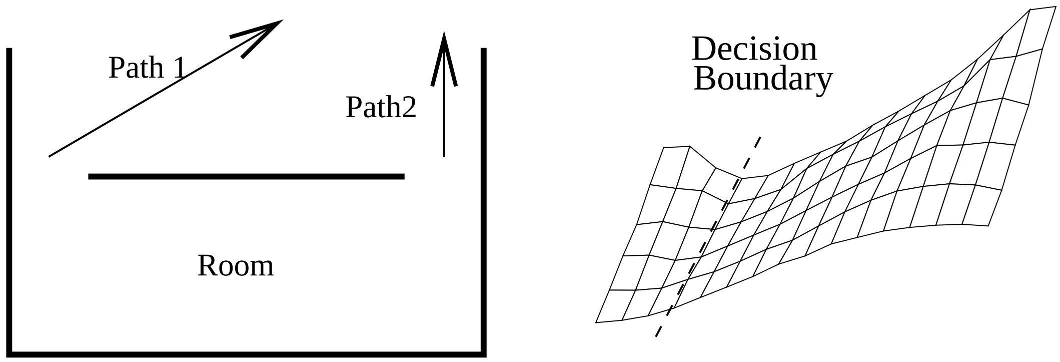

Figure 25: Decision Functions

To construct the decision boundary two functions, for entering by one doorway and leaving by another, were used. By adjusting the heights of the two functions and we can place the decision boundary anywhere within the room. If there are no such functions in the case base, functions that already include decision boundaries may be used. This technique produces a correct decision boundary 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. On the left hand side of Figure 25 there is a room with two doorways. As path 1 is significantly longer than path 2 the decision boundary is far to the left. The shortest path to the goal from most of the room is via the right hand doorway. 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 26. This could be used for the new problem shown on the left hand side of Figure 26 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 25

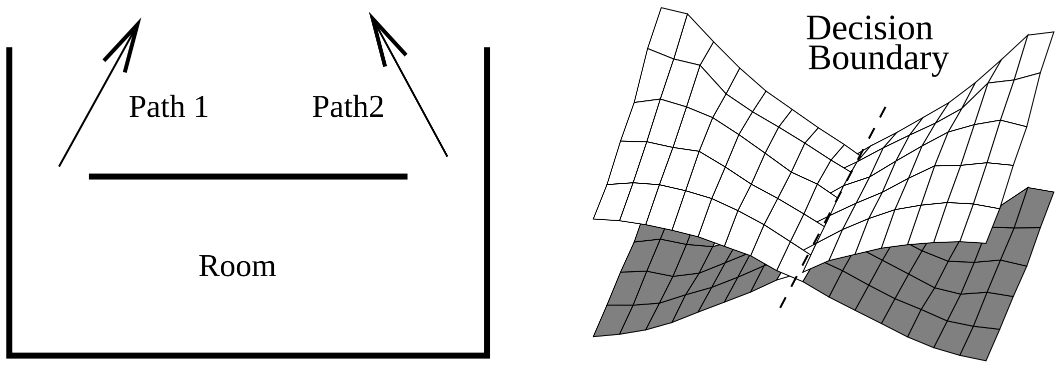

Figure 26: Combining Decision Functions

4. Experiments

This section compares learning curves for function composition and a simple baseline algorithm. Four sets of results are presented; one for each of the two types of related task in each of the two domains. The learning curves represent the average distance to goal as a function of the number of actions 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.

The knee of the function composition system's curve (where the low level learning algorithm is initialised with the composed function) is used as the reference point in estimating speed up. Speed up is calculated by dividing the number of learning steps at specific points on the baseline learning curves by the number of learning steps to achieve the knee. The two points used are when the average distance to goal on the baseline learning curves first reaches 150% and 100% of the distance at the knee. If one or more of these values is not reached the total number of steps of the experiments is used.

The baseline algorithm and the underlying learning algorithm for the function composition system is the basic Q-learning algorithm using a discrete function approximator as discussed in Section 3.1. The learning rate α is set to 0.1, the greedy policy uses an ϵ of 0.1 (the best action is selected 90% of the time), the future discount γ is 0.8 and a reward of 1.0 is received on reaching 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 in each dimension either separately or together, giving the eight possible actions. The actions are stochastic, a uniformly distributed random value between ± 0.125 being added to each dimension of the action.

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.

4.1 Experiment 1: Robot Navigation, Goal Relocation

The first experiment investigates the time taken to correct the learnt function when the goal is relocated in the robot navigation domain. There are nine different room configurations, the number of rooms varying from three to five, and 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 always in the top right corner.

The basic Q-learning algorithm is then rerun on each room configuration with the goal in the top right corner. After 300,000 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 27 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.

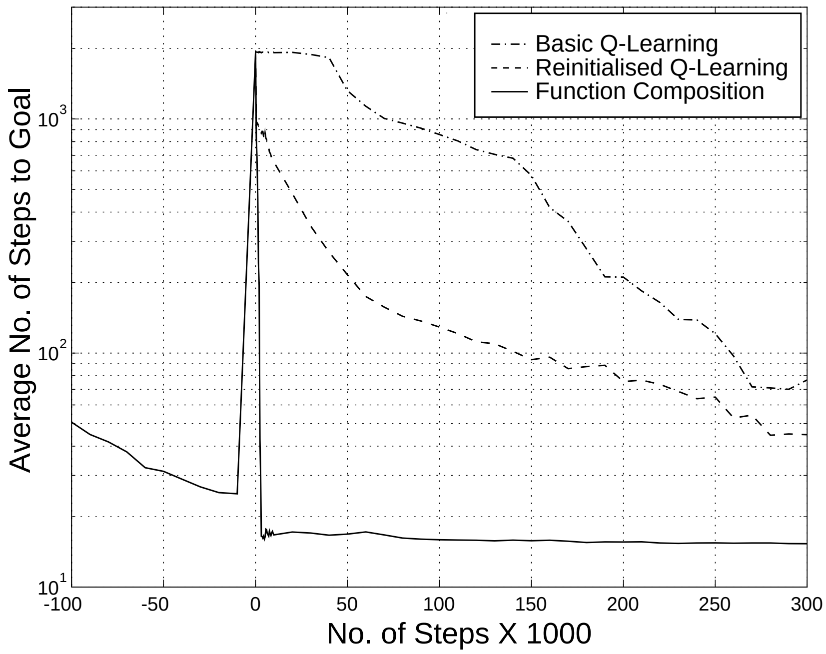

Figure 27: Learning Curves:Robot Navigation, Goal Relocation

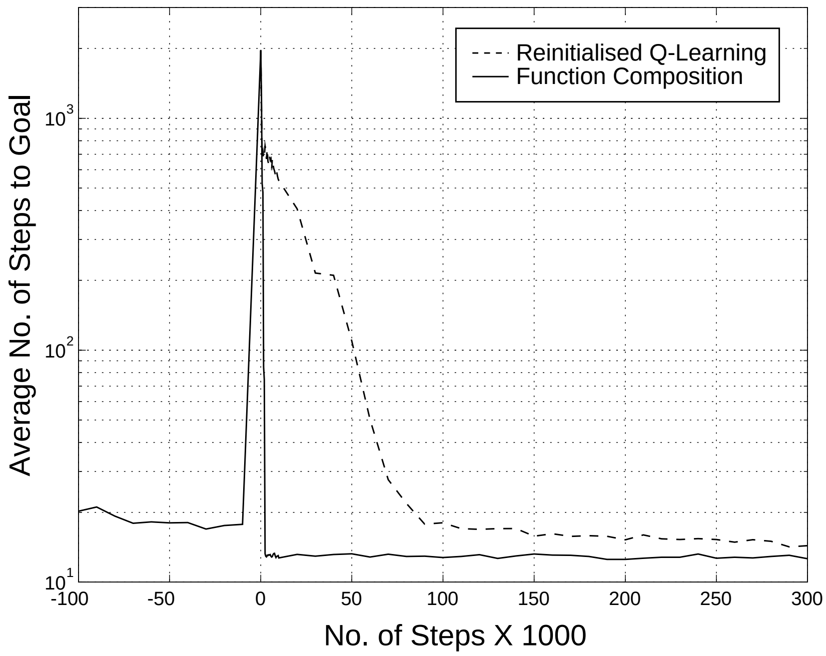

Figure 28: Learning Curves: Robot Arm, Goal Relocation

The basic Q-learning algorithm, the top curve of Figure 27, 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 better.

The function composition system, the lowest curve, performed by far the best. The knee of this curve occurs at 3200 steps. As neither baseline learning curve came within 150% of the value at the knee, the total number of experimental steps is used to estimate speed up. This gives a speed up of about 95 in both instances although the reinitialised algorithm is appreciably faster.

4.2 Experiment 2: Robot Arm, Goal Relocation

The second experiment is essentially a repeat of the first experiment but using the robot arm. As the arm is only two degrees of freedom and with the restrictions discussed in Section 2.4 the number of variations is small. Because of this and the time needed to carry out the experiments only three configurations with two obstacles in each were used. There is also a slight variation on the first experiment in that the initial position of the goal was different for each configuration, this allowed populating the case base with a greater variety of cases. The curves in Figure 28 are the average of 6 experimental runs, two new goal positions for each of the three obstacle configurations. There are only two curves, the basic non-reinitialised Q-Learning being dropped.

As in experiment 1 the function composition system, the lower curve, performed much better than Q-learning, with the knee occurring at 2400 steps. The Q-learning curve reaches 150% of the knee value at roughly 85,000 steps, a speed up of about 35. The 100% value is not reached and using the total number of experimental steps gives a speed up of about 125.

4.3 Experiment 3: Robot Navigation, Detecting Features Early

The third experiment investigates the time taken to learn a new, but related, task in the robot navigation domain. There are seven different inner rooms, all have a single doorway but the size of the room and the position of the doorway are varied. To initialise the case base, a function is learnt for each of these configurations with the goal in the top right corner. The system is however prevented from selecting a case that comes from the same room configuration. So all functions are composed of pieces that come from the other 6 tasks. 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 200,000 steps. The curves in Figure 29 are the average across the seven different rooms.

The top curve is the Q-learning algorithm, the bottom curve the function composition system. For these experiments locating the goal took between 400 and 800 steps. The function composition system then introduces the "no walls" function and a further 800 to 2000 steps are taken before usable features are generated. The knee of the function composition system's curve occurs at 3200 steps. The 150% point on the baseline learning curve occurs at roughly 80,000 steps, the 100% value is not reached. This gives speed up values of about 25 and 60 respectively.

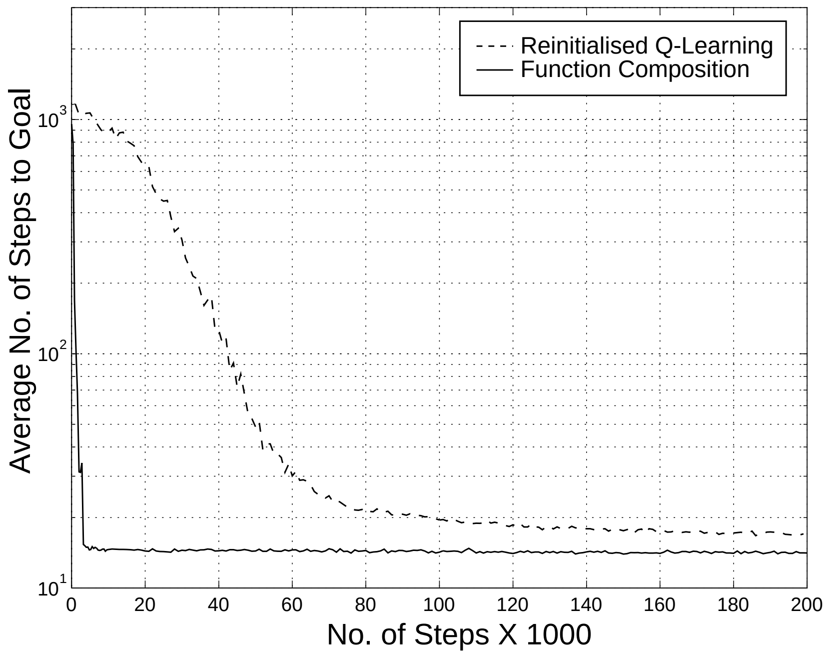

Figure 29: Learning Curves: Robot Navigation, Detecting Features Early

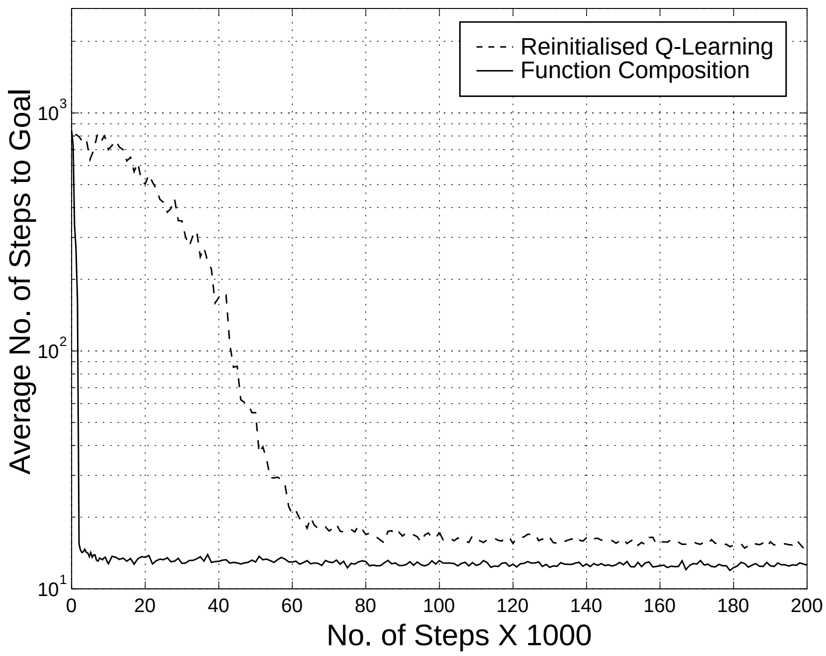

Figure 30: Learning Curves: Robot Arm, Detecting Features Early

4.4 Experiment 4: Robot Arm, Detecting Features Early

The fourth experiment is essentially the same as the third experiment except with the robot arm. Here three configurations of a single obstacle were used. The top curve of Figure 29 is the Q-learning algorithm, the bottom curve the function composition system. The knee of the function composition system's curve occurs at 2200 steps. The 150% point on the baseline learning curve occurs at roughly 60,000 steps, the 100% value at 180,000 steps. This gives speed up values of about 25 and 80 respectively.

4.5 Discussion

The experiments of this section have shown there is a very significant speed up across the different types of related task and across the two domains. The very size of the speed up should mitigate any concerns about the experimental methodology. Nevertheless the following will discuss a number of possible concerns. Some will be, at least partially, addressed now and some will be the subject of future work.

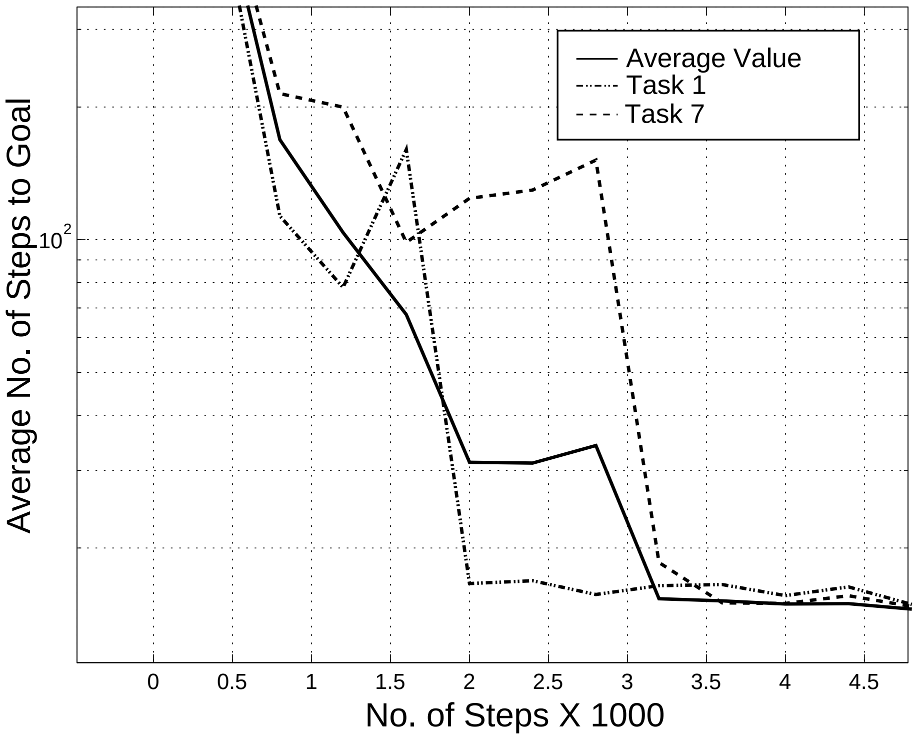

Figure 31: Detailed Learning Curves: Estimating Speed Up

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. The mapping from one curve to another is not clear. Nevertheless a rough analysis of the results for the "Robot Navigation, Detecting Features Early" experiment treating each curve individually was carried out and produced a larger speed up. Figure 31 shows some results from this experiment, expanded around the knee points of the curves. The bold curve is the average value repeated from Figure 29. The knee of this curve occurs at 3200 steps, giving the speed up values of 25 and 60. The other two curves are 2 of the 7 individual tasks. Five of the tasks have curves very similar to the one marked task 1, extracting the features at about 2000 steps. One extracted the features at 1600 steps. Task 7 took significantly longer. The inner room is relatively long and thin and the doorway is in the middle of the wall at the thin end of the room. The significant features are much smaller than in the other tasks and take longer to appear clearly. The results from this task have a strong influence on the shape of the average curve close to the knee point and thus on the estimated speed up. An estimate of the average speed up treating each trial separately gives values of 30 and 85. Thus, at least in these experiments, the effect of averaging on the knee position of the function composition system's curve dominates any effect of averaging on the baseline curves in calculating speed up.

Another 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, basic Q-learning. Generally the function composition system produces a solution which requires little further refinement by reinforcement learning. Even when more refinement is needed none of the trials exceed the limit. Thus the estimates of speed up are, if anything, conservative.

Another concern might be the source of speed up in the "detecting features early" experiments. In Figure 31 it is apparent that after the "no walls" function is introduced, around 600 steps, the average distance to goal drops appreciably. This function is correct inside the inner room and at least somewhat useful outside. It is not clear then overall how much speed up is due to this first step and how much due to the feature extraction step. We might expect however that as the tasks addressed become more complex that this initial stage should become less significant.

There are other concerns that can only really be addressed by further experiments. The effectiveness of feature extraction is dependent on the setting of a number of parameters. The values of these were empirically obtained using some of the examples from the experimental section. The obvious danger is that the parameters might be somewhat tuned to these examples and further experiments are needed with randomly generated examples.

The value of speed up is very much dependent on the baseline algorithm used. The initial function for the main baseline learning algorithm was zero everywhere. In this author's previous work (Drummond, 1998) the function was initialised to a constant value of 0.75. 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 may be that some intermediate value would be a better trade off. Other variations in the parameters of the Q-learning algorithm were not tried. 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 learning rate of the baseline algorithm. The Q-learning algorithm 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.

The rest of this section looks at some individual tasks. It will highlight where basic reinforcement has most difficulty. It will also show where some error is introduced during function composition and must be corrected by further reinforcement learning. Basic reinforcement learning is particularly slow when the room configuration is the one discussed in Section 2.2 and the goal is moved to the lower left hand corner of the state space. If we do not reinitialise the function, the new goal is at the minimum of the existing function, so the learning algorithm has to “flatten” it completely before a new one can be learnt. In this instance, the upper curve of Figure 32 the limit of 2000 steps to the goal was exceeded for the majority of start positions even after 300,000 steps. The reinitialised baseline algorithm performs slightly better. But after the function is reinitialised the likelihood of random actions successfully navigating the rooms to the goal is small and this seriously slows the learning process.

When the goal is moved to the lower right hand corner the effect is similar, learning is slow. Another configuration with four rooms and the goal at the end of a single path produces a similar learning curve. These three examples are by far the slowest for both baseline learning algorithms. If they are removed from the reinitialised Q-learning curve the overall speed up for the 150% point nearly halves dropping from 95 to 50. These results suggest that as the complexity of the tasks increases the learning rate of reinforcement learning can worsen rapidly. Thus function composition should become much more useful as the learning task becomes harder for the baseline algorithm.

For the function composition system, the lowest curve in Figure 32, after introducing the new function a significant amount of refinement is still required. The initial error is due to underestimating the distance to the new goal. The underestimated distance is for the path from the doorway of the room containing the new goal. With only a single path through this doorway from everywhere else in the state space, the functions representing the rest of the rooms are higher than they should be. But even with this error the benefits of the approach can be clearly seen.

In one of the goal relocation experiments the system cannot find a suitable function in the case base for one of the rooms. It uses a default function for that room, a constant value, and combines it with matched functions for the other rooms. In Figure 33 the lowest curve shows that further refinement is needed to correct the inaccuracy caused by using this default function. Intuitively, this might be expected to produce a worse learning rate than in the previous example, where the correct functions were used albeit with some error in their heights. However, comparing the two curves of the function composition system in Figures 32 and 33 one can see that the previous example requires more refinement. In that example there is only a single path to goal and the error is made close to the goal, which has a global effect. In this example the error is further from the goal and there are two paths. Thus the error is local and only affects paths staring within, or very close to, the specific room.

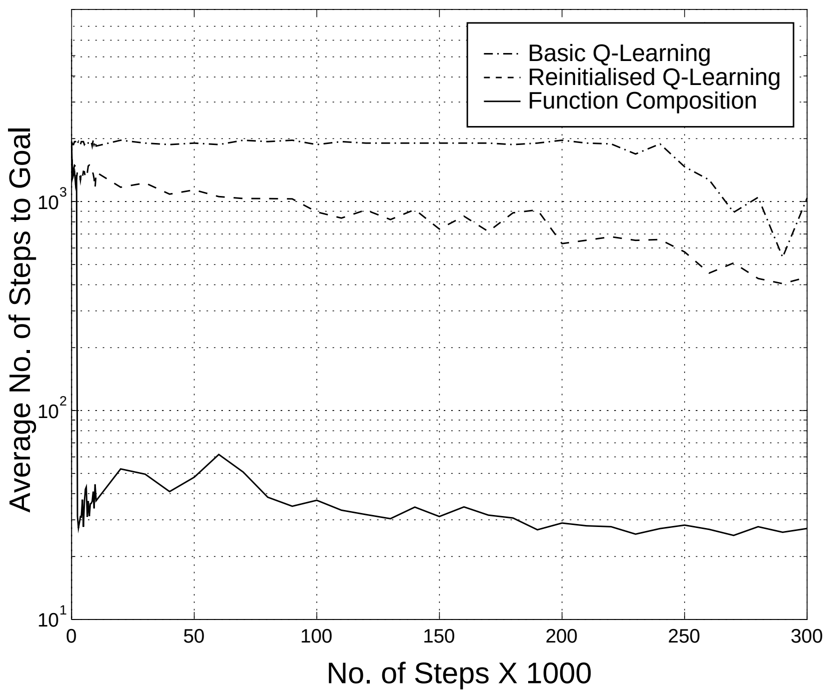

Figure 32: Detailed Learning Curves: Difficult Tasks

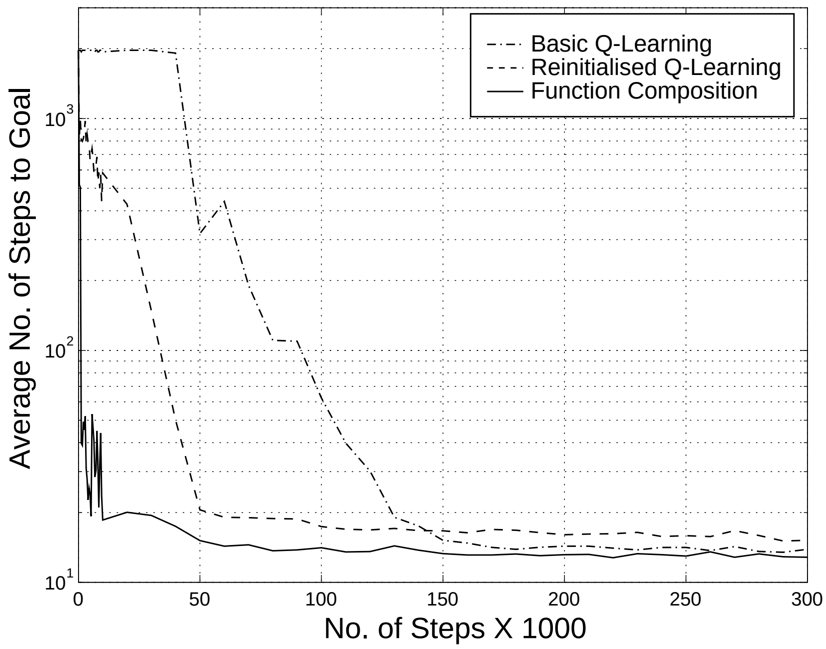

Figure 33: Learning Curves: Using Default Functions

5. Limitations and Future Work

As suggested in the previous section more experiments, preferably with randomly generated examples, are required. The possible goal locations were particularly limited in the experiments. For instance, when learning a brand new task in the robot navigation domain, the goal was always in the top right hand corner. Using other corners is very unlikely to pose any difficulty to the system. The goal relocation experiments made extensive use of the different transforms such as rotation necessary to deal with these variations. Future experiments will use more arbitrary locations in rooms for the robot navigation domain and goal positions other that on the boundary of the robot arm's work space.

This paper has significantly extended the variety of shapes that can be extracted by the snake. But as was apparent in the discussion of the robot arm in Section 2.4 this is still insufficient and further work is needed. It is likely however that the feature extraction algorithm need not be sensitive to all the subtle variations of an obstacle's boundary. It will probably be sufficient to make a reasonable approximation and let low level learning refine the small details. The shape restriction, the use of symmetries in the domain and the settings of various parameters are essentially system biases. Future work will investigate how generally applicable these biases are in a broader range of tasks and domains. Future work will also investigate if the relevant biases can be learnt directly from the domain.

At present the system is told whether the existing partition can be used or the learning process must be started afresh. There are many examples other than goal relocation that could exploit information from the existing task, such as the opening or closing of doors in the robot navigation problem. In future work the aim is that the system should decide what, if any, relationship there is to the existing task and how it might best be exploited. This will allow a more complex interaction of the function composition system with reinforcement learning. For instance, the learning of a brand new task for the robot navigation domain 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 become apparent.

This approach also should handle any errors the system might make with feature extraction. In the experiments with these simple room configurations the filtering discussed in Section 2.3 proved sufficient to prevent problems. But in more complex tasks it is likely that false "doorways" will be detected simply because the system has not explored that region of the state space. A composed function including that extra doorway will drive the system into that region. It should then become quickly apparent that the doorway does not exist and a new function can be composed.

6. Related Work

The most strongly related work is that investigating macro actions in reinforcement learning. Precup, Sutton and Singh (1997, 1998) propose a possible semantics for macro actions within the framework of normal reinforcement learning. Singh (1992) uses policies learnt to solve low level problems as primitives for reinforcement learning at a higher level. Mahadevan and Connell (1992) use reinforcement learning in behaviour based robot control. To learn a solution to a new task, all these systems require a definition for each subtask and their interrelationships in solving the compound task. The work presented here gives one way that macro actions can be extracted directly from the system's interaction with its environment without any such hand-crafted definitions. It also shows how to determine the interrelationships of these macro actions needed to solve the new task. Thrun's research (1994) does identify macro actions, by finding commonalities in multiple tasks. But unlike the research presented here, no mapping of such actions to new tasks is proposed. Hauskrecht et al. (1998) discuss various methods of generating macro actions. Parr (1998) develops algorithms to control the caching of policies that can be used in multiple tasks. But in both cases they need to be given a partitioning over the state space. It is the automatic generation of just such a partition that has been the focus of much of the work presented in this paper. It may well be that this approach to generating partitions and to determining the interrelationships between partitions of related tasks will prove useful to this other work.

Another group of closely connected work is the various forms of instance based or case based learning that have been used in conjunction with reinforcement learning. They have been used to address a number of issues: (1) the economical representation of the state space, (2) prioritising states for updating and (3) dealing with hidden state. The first issue is addressed by Peng (1995) and by Tadepalli and Ok (1996) who use learnt instances combined with linear regression over a set of neighbouring points. Sheppard and Salzberg (1997) also use learnt instances but they are carefully selected by a genetic algorithm. The second issue is addressed by Moore and Atkeson (1993) who keep a queue of "interesting" instances. These are updated most frequently to improve the learning rate. The third issue is addressed by McCallum (1995b) who uses trees which expand the state representation to include prior states removing ambiguity due to hidden states. In further work (McCallum, 1995a) he uses a single representation to address both the hidden state problem and the general problem of representing a large state space by using a case base of state sequences associated with various trajectories. Unlike this other research, in the work presented here the case is not an example of the value function during learning. Instead, it is the result of a complete learning episode, so the method should be complementary to these other approaches.

This work is also related to case based planning (Hammond, 1990; Veloso & Carbonell, 1993), firstly through the general connection of reinforcement learning and planning. But it is analogous in other ways. When there is a small change in the world such as the goal is moved, a composite plan is modified by using sub-plans extracted from other composite plans.

Last but not least is the connection with object recognition in vision research (Suetens et al., 1992; Chin & Dyer, 1986). In the work presented here, many of the methods - if not the final application - has come from that field. The features in the reinforcement learning function are akin to edges in an image. These are located by finding the zero crossing point of the Laplacian as introduced by Marr (1982). In the work presented here, it was proposed that the features largely dictate the form of the function. Mallat and Zhong (1992) have shown that a function can be accurately reconstructed from a record of its steep slopes.

7. Conclusions

This paper described a system that transfers the results of prior learning to significantly speed up reinforcement learning on related tasks. Vision processing techniques are utilised to extract features from the learnt function. The features are then used to index a case base and control function composition to produce a close approximation to the solution of a new task. The experiments demonstrated that function composition can produce between one and two orders of magnitude increase in learning rate compared to a basic reinforcement learning algorithm.

Acknowledgements

The author would like to thank Rob Holte for many useful discussions and help in preparing this paper. This work was in part supported by scholarships from the Natural Sciences and Engineering Research Council of Canada and the Ontario Government.

Appendix A. Spline Representations

This appendix presents some of the underlying mathematics associated with spline representations and the snake. It is not meant to be an introduction to the subject. Rather it is added for completeness to give some discussion of certain important aspects of the system not addressed elsewhere in this paper. Knowledge of these aspects is not necessary to understand the basic principles of the approach discussed in this paper but would be necessary if one wanted to duplicate the system. Some specific papers that address these ideas in much greater detail are: for splines (Terzopoulos, 1986) and for snakes (Cohen & Cohen, 1993; Leymarie & Levine, 1993).

Splines are piecewise polynomials where the degree of the polynomial determines the continuity and smoothness of the function approximation. Additional smoothing constraints can be introduced by penalty terms which reduce the size of various differentials. One way then to view spline fitting is in the form of an energy functional such as Equation 5.

$$E_{spline}(f_{app}) = \int_R (E_{fit}(f_{app}) + E_{smooth}(f_{app}))\, ds \tag{5}$$

Here there is an energy associated with the goodness of fit, some measure of how close the approximating function is to the input function. This is typically the least squared distance between the functions. There is an energy associated with the smoothness of the function. Two very commonly used smoothness controls produce the membrane and thin plate splines by restricting the first and second differentials of the function respectively. To fit the spline to the function the total energy must be minimised. A necessary condition for this is an Euler-Lagrange differential equation such as Equation 6. Here ω_1 controls the tension in the spline (the resistance to stretching) and ω_2 the stiffness (the resistance to bending). Often the error function will be based on individual data points and the left hand side of Equation 6 would include delta functions.

$$-\frac{\partial}{\partial s}(\omega_1(s)\frac{\partial f_{app}(s)}{\partial s}) + \frac{\partial}{\partial s^2}(\omega_2(s)\frac{\partial^2 f_{app}(s)}{\partial s^2}) = f_{in}(s) - f_{app}(s) \tag{6}$$

In this work such splines have been used for a number of purposes. When fitting the snake, measures of the first and second differential are needed. A two dimensional quadratic spline is fitted to the discrete representation of the maximum Q-values. An ω_1 of 0.2 used (ω_2 is zero) to limit overshoot (Drummond, 1996) to prevent false edges. Values from an identical spline except using an ω_1 of 2.0 are squared and then divided into the differential values. This normalises the differentials so that the size of edges is not dependent on where they occur in the function. The same type of spline is used to produce the bowls associated with the rooms as discussed in Section 3.2.1. Here ω_1 is 1.0 and ω_2 is 0.5 giving roughly Gaussian smoothing. The values used to produce this function are weighted. Values close to one are given weights of 200, lower values a weight of 1. This prevents the sides of the bowls collapsing under smoothing.

A one dimensional cubic spline is used in locating the doorways. These are found by steepest descent on the value of the differential along the body of the snake. This differential 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. Two splines with different smoothing coefficients are used to remove these effects. One has an ω_1 of 1.0 and an ω_2 of 0.01. The other has an ω_1 of 0.15 and an ω_2 of 0.0015. Steepest descent is applied to the second spline. Starting points for steepest descent are changes in the sign of the finite difference between coefficients. When a local minimum is found the value is compared to the value for the same position of the smoother function. If it is above this value it is rejected. Using the spline does introduce some error in the position of the doorway due to the smoothing. This is removed by locally searching in the original data.

To represent the snake the model of the spline must be changed somewhat. The snake itself is a one dimensional cubic spline. But the energy minimum that is being sought is in the differential of the Q_{max} function, subject to other constraints. This paper uses the dynamics of the snake as defined by the Euler-Langrange equations given in Leymarie and Levine (1993) and shown in Equation 7.

$$\mu \frac{\partial^2 f_{snake}}{\partial t^2} + \gamma \frac{\partial f_{snake}}{\partial t} - \frac{\partial}{\partial s}(\omega_1(s) \frac{\partial f_{snake}}{\partial s}) + \frac{\partial}{\partial s^2}(\omega_2(s) \frac{\partial^2 f_{snake}}{\partial s^2}) = F(f_{snake}) \tag{7}$$

Here we start the snake with an ω_1 of 0.00375 and an ω_2 of 1.5 thus having a relatively small tension but a large stiffness. A μ of 150 and a γ of 50 controls the momentum and the drag on the snake respectively. As in Cohen and Cohen (1993) a factor is added to the energy associated with the differential that is in the direction normal to the body of the snake, as shown in Equation 8. But instead of it being a constant, a variable is used to produce the mercury model discussed in Section 3.2.1.

$$F(f_{snake}) = M(f_{snake}) \overrightarrow{n}(s) + \nabla(- \left| \nabla Q_{max}(f_{snake}) \right|^2) \overrightarrow{n}(s) \tag{8}$$

The energy minimisation process is carried out iteratively interleaving steps for the x and y directions. The differential of $\nabla \left| Q_{max} \right|^2$ for the x direction is given by Equation 9, a similar equation is used for the y direction.

$$- \frac{\partial \left| \nabla Q_{max} \right|^2}{\partial x} = -2 \left[(\frac{\partial Q_{max}}{\partial x})(\frac{\partial^2 Q_{max}}{\partial x^2}) + (\frac{\partial Q_{max}}{\partial y})(\frac{\partial^2 Q_{max}}{\partial x \partial y}) \right] \tag{9}$$

The snake grows under the forces of the mercury model until it reaches an approximately stable position subject only to small oscillations. It is then converted into a polygon by finding the corners (where the normal passes through $\frac{(2n+1)\pi}{4}$ where $n = 0 \ldots 3$). The coefficient ω_1 is set to zero everywhere. The coefficient ω_2 is set to zero at the corners and 15 between them. This produces a polygon which is flexible at its vertices.

To detect the features as early as possible in the learning process, as discussed in Section 2.4, the height of the gradient is scaled according to the signal to noise ratio. The noise arises from variations in the low level learning process and the stochastic nature of the task. Both the size of the features and the noise grow with time and are somewhat normalised by this scaling process. The idea is to collect uniformly sampled values of the function shown in Equation 9 for both the x and y directions and find the mean of their absolute values. The mean does not include the highest one third of values. This is intended to ignore the size of the features and measure the noise of the regions in between.

References

Chin, C. H., & Dyer, C. R. (1986). Model-based recognition in robot vision. *Computing surveys*, *18*(1), 67–108.

Christiansen, A. D. (1992). Learning to predict in uncertain continuous tasks. In *Proceedings of the Ninth International Workshop on Machine Learning*, pp. 72–81. Morgan Kaufmann.

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.

Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numer. Math.*, *1*, 269–271.

Drummond, C. (1996). Preventing overshoot of splines with application to reinforcement learning. Computer science technical report TR-96-05, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada.

Drummond, C. (1997). Using a case-base of surfaces to speed-up reinforcement learning. In Leake, D. B., & Plaza, E. (Eds.), *Proceedings of the Second International Conference on Case-Based Reasoning in Lecture Notes in Artificial Intelligence*, Vol. 1266, pp. 435–444. Springer-Verlag.

Drummond, C. (1998). Composing functions to speed up reinforcement learning in a changing world. In Nedellec, C., & Rouveirol, C. (Eds.), *Proceedings of the Tenth European Conference on Machine Learning in Lecture Notes in Artificial Intelligence*, Vol. 1398, pp. 370–381. Springer-Verlag.

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. Morgan Kaufmann.

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *The Journal of Cognitive Science*, *14*(3), 385–443.

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 International Conference on Uncertainty In Artificial Intelligence*. Morgan Kaufmann.

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.

MacDonald, A. (1992). Graphs: Notes on symetries, imbeddings, decompositions. Electrical engineering department TR-92-10-AJM, Brunel University, Uxbridge, Middlesex, United Kingdom.

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.

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

McCallum, R. A. (1995a). Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pp. 377–384. The MIT Press.

McCallum, R. A. (1995b). Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelth International Conference on Machine Learning*, pp. 387–395. 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.

Parr, R. (1998). Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings of the Fourteenth International Conference on Uncertainty In Artificial Intelligence*. Morgan Kaufmann.

Peng, J. (1995). Efficient memory-based dynamic programming. In *Proceedings of the Twelfth International Conference of Machine Learning*, pp. 438–439. Morgan Kaufmann.

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 *Tenth European Conference on Machine Learning in Lecture Notes in Artificial Intelligence*, Vol. 1398, pp. 382–393. Springer-Verlag.

Sheppard, J. W., & Salzberg, S. L. (1997). A teaching strategy for memory-based control. *Artificial Intelligence Review, special issue on Lazy Learning*, *11*, 343–370.

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. AAAI Press, The MIT Press.

Suetens, P., Fua, P., & Hanson, A. (1992). Computational strategies for object recognition. *Computing surveys*, *4*(1), 5–61.

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. The MIT Press.

Sutton, R. (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. Morgan Kaufmann.

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. Morgan Kaufmann.

Terzopoulos, D. (1986). Regularization of inverse visual problems involving discontinuities. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, *8*(4), 413–423.

Thrun, S., & Schwartz, A. (1994). Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pp. 385–392. The 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.

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