Composing Functions to Speed up Reinforcement Learning in a Changing World

Chris Drummond

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

Abstract. This paper presents a system that transfers the results of prior learning to speed up reinforcement learning in a changing world. Often, even when the change to the world is relatively small an extensive relearning effort is required. The new system exploits strong features in the multi-dimensional function produced by reinforcement learning. The features generate a partitioning of the state space. 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. The experimental results investigate one important example of a changing world, a new goal position. In this situation, there is close to a two orders of magnitude increase in learning rate over using a basic reinforcement learning algorithm.

1 Introduction

The aim of this work is to maximise the transfer of learning from previous tasks to the current one. At the base of the system is a standard reinforcement learning algorithm. One advantage of reinforcement learning is that it learns even when the information available is very limited. It requires only knowledge of its present state and infrequent numerical rewards to learn the actions necessary to bring a system to some desired goal state. As often occurs in achieving this level of generality, the learning rate is slow. It is important therefore to exploit the results of prior learning to speed up the process while maintaining the robustness of the general learning method.

This work uses syntactic methods of composition much like in symbolic planning, but the novelty arises in that the components of this composition are continuous functions. The functions needed for composition are either learnt individually or extracted from 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 [1], so although individual actions may be highly uncertain the overall result is largely predictable.

The central intuition of this work is that there are strong features in the multi-dimensional function learnt using reinforcement learning. The important aspect of these features, for the purposes of this paper, is that they largely dictate the shape of this function. A popular technique in object recognition, the snake [12], is used to locate and characterise these features. The snake is then converted into a discrete graph. The graph and its constituent subgraphs act as an index into a case base of previously learnt functions. When a new task is being learnt planning occurs at the graphical level. The relevant cases are determined by graph matching and then a transform is applied to adapt the retrieved function to the new task. The new function is used to initialise the lower level learning process, which may further refine the function to better fit the new task. Thus the planning stage need not be exact. It need only be accurate enough to produce a significant speed-up in the learning process, averaged over many learning episodes.

The rest of the paper begins with section 2 giving a very high level discussion of the approach taken. The intent is to appeal to the intuitions of the reader, with section 3 giving 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 experimental testbed 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 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.

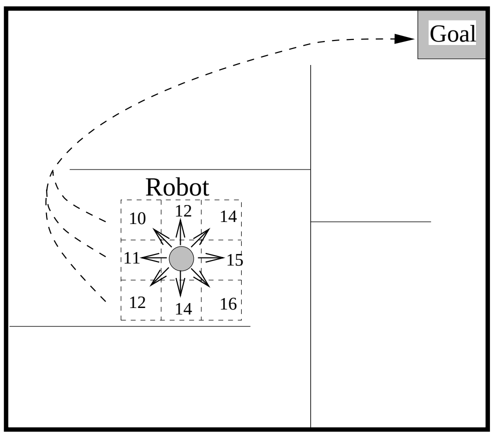

Fig. 1. Robot Navigating Through a Series of Rooms

If each action is independent, 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 try 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 [18, 14] which progressively improves estimates of the distance to goal from each state until they converge to the correct values. This paper is not primarily concerned with learning in a radically new environment, the system incorporates a reinforcement learning algorithm for that purpose. This paper rather addresses the transfer of learning between closely related tasks. The principal example for the purposes of this discussion and the experiments of section 4 is the transfer that occurs when the goal position is changed.

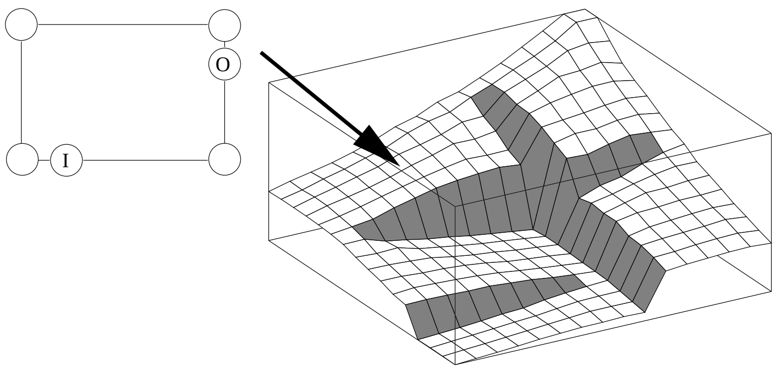

Fig. 2. The Reinforcement Learning Function

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 discontinuities 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 the inside of the room on the path to the goal. These features are visually readily apparent to a human, so it seems intuitive 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 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 discontinuity 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. Nodes corresponding to the doorways are labelled “I” or “O” for in and out respectively.

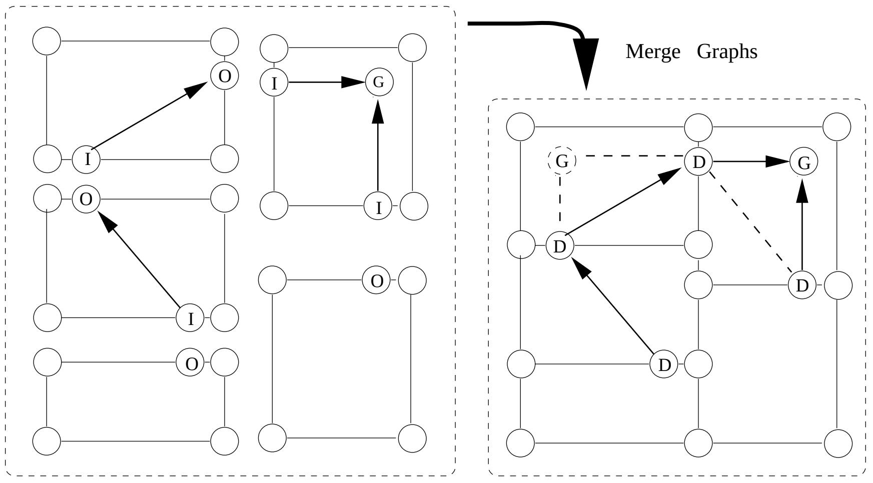

Fig. 3. Graphical Representation

The left hand side of figure 3 shows plane graphs for all the rooms. 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, the corresponding part of the function is the case.

Now suppose, the goal is moved from the top right corner to the top left. 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 solid lines 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.

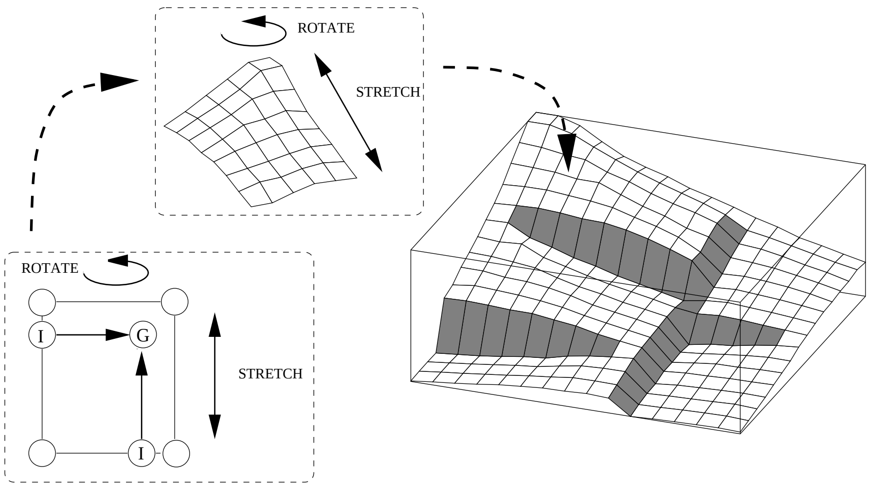

Fig. 4. Function Composition

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 may be possible to exchange these two, using an appropriate transform. But if there are many more cases in the case base other graphs may better match the new task. In this example, the best match for the subgraph containing the new goal is the subgraph for the goal in the original 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. A case obtained when solving another task is used for the room containing the original goal. The other three rooms use the functions from the original problem, though their heights 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.

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, one moving the robot 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 takes a step in a particular direction from a particular state, the state reached is not always the same.

To do the reinforcement learning this research uses the Q-learning algorithm [18]. This algorithm assumes the world is a discrete Markov process thus both states and actions 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 paragraphs 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 [13] 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 [18] proved that this will converge to the optimal value with certain constraints on the reward and the learning rate α. The optimal solution is to take the action with the greatest value in any state. Thus in this robot problem, a greedy algorithm will take the shortest path to the goal once learning is complete. The extension to continuous spaces can 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 right hand side plot of figure 5 is the magnitude of the gradient vector of the function in figure 2. This is the absolute value of the largest gradient in any direction, it forms hills corresponding to the steep slopes of the function. To locate the features a curve is found that lies along the ridge of the hills. On the left hand side of figure 5 the dashed lines are contours for one of the rooms as indicated, the system adds a gradient around the rooms to represent the state space boundary.

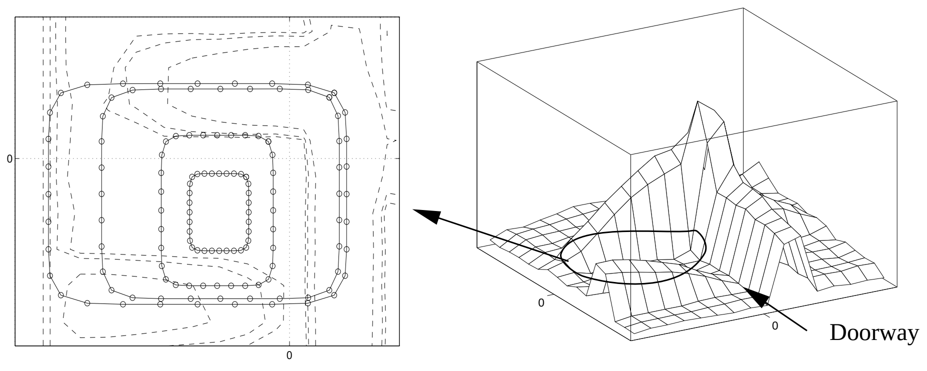

Fig. 5. Fitting the Snake

The dark lines in figure 5 are the snake. Starting from the initial position, the smallest rectangle, the snake is expanded by a ballooning force [2] until it reaches 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 circles. But instead of being allowed to climb independently their movement relative to each other is constrained, in this instance to maintain a rectangular shape. This prevents individual points getting trapped in local maxima and collecting at the hills' apices. When the snake reaches the top of the ridge, the largest rectangle in figure 5, 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 [2]. 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 rectangle and the position of the doorways, a plane graph is produced. The rectangle 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.

3.3 Transformation and Composition

This section discusses the transformation and composition of individual subgraphs and their corresponding functions to form a solution to the new task. The system finds all subgraphs in the case base isomorphic to a subgraph extracted using the snake and all possible isomorphic mappings between their nodes, using a labelling algorithm [6]. Associated with each node of a subgraph is an (x, y) coordinate. A similarity transform is applied to each of the isomorphic subgraphs to minimise the squared distance between the coordinates of the mapped nodes. The transform permits translation, rotation and independent scaling in each dimension. The subgraph selected is a compromise between the best fit and the least scaling, particularly asymmetric scaling. The corresponding function from the case base is then modified using the same transform.

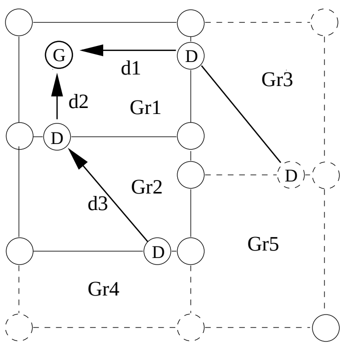

Fig. 6. Using Dijkstra's Algorithm

Function composition uses Dijkstra's algorithm [3] to traverse the edges between doorway nodes. Figure 6 shows the composite graph for the new task. To begin the process, the subgraph which contains the goal is selected 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 subgraph, d1 and d2 in figure 6. As d2 is less than d1, the next subgraph selected, 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 updated, d3. The shortest path is again determined, as d1 is less than d2 + d3 subgraph Gr3 is selected. The process is repeated until all subgraphs have been updated. At each stage when a subgraph is matched, the corresponding transformed function is added to the new function in the appropriate region.

Here there is one path to the goal from each room. If a doorway was added to the lower left corner of room 5, the graph on the left of figure 7 would result. There are now two possible paths, lengths d4 and d5. If the distance across room

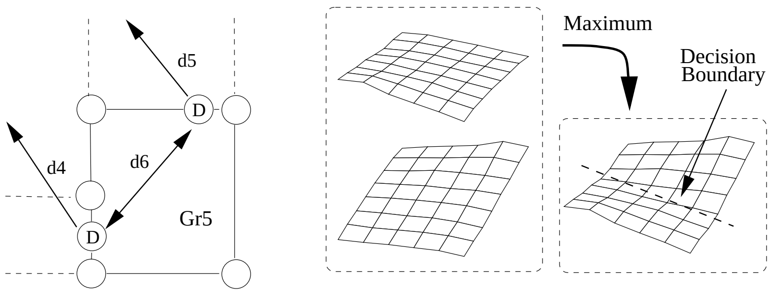

Fig. 7. Combining Two Functions

5, d6, is greater than the absolute difference between d4 and d5, the choice of path will be determined by a decision boundary in this room. This is produced by taking the maximum of two functions one for each path, see figure 7. This principle could be repeated for additional paths to the goal from this room.

When a scaling transform is used the height of the retrieved function must be adjusted. Imagine that to fit a new room the length of the function must be doubled in each dimension. The distance between the doorways will therefore double. As the function is an exponential in this distance the height must be squared, after normalising the function so that it is one at the out doorway. In general, the height is raised to the power of the scale factor. When scaling is symmetric the result is exact, assuming distance is a linear metric. With asymmetric scaling it is not. But if the difference is relatively small it is a useful approximation to use the average of the two scale factors. The height is also adjusted when the retrieved function is added to the function for the whole state space. We need to abut individual functions so that the result is smooth at the doorways. Starting with a normalised function, as it is an exponential the function is multiplied by the exponential of accumulated distance to the goal.

4 Experiments

The experiments investigate the time taken to correct the learnt function when the goal is relocated. The basic Q-learning algorithm is used ($\alpha = 0.1, \epsilon = 0.1, \gamma = 0.8, r = 1.0$ at the goal) with a discrete function approximator as discussed in section 3.1. There are 9 different room configurations, the number of rooms varying from 3 to 5, and four different goal positions. Each room has 1 or 2 doorways and 1 or 2 paths to the goal. The state space is a unit square. A step is ± 0.0625 plus a random value between ± 0.03125 along one or both dimensions, giving the eight possible actions . 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

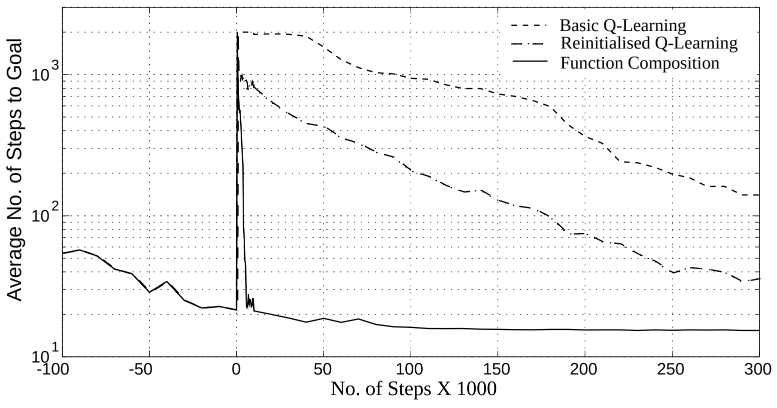

Fig. 8. Learning Curves: Steps to Goal

is moved to one of the three remaining corners of the state space. 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 average is across 64 different start positions distributed uniformly throughout the state space. The maximum number of steps for each start location is limited to 2000. The curves in figure 8 are the average for three new goal positions for each of the nine room configurations. Zero on the x axis is where the goal is moved.

The basic Q-learning algorithm, the top curve of figure 8, performs poorly. When the goal is moved the existing function pushes the robot towards the old goal position. This is particularly apparent when the room configuration is the one of section 2. If the new goal is at lower left hand corner of the state space, the minimum of the existing function, the learning algorithm has to "flatten" it completely before a new one can be learnt. Here the limit of 2000 steps to the goal was exceeded for the majority of start positions even after 300,000 steps.

A variant of the basic algorithm reinitialises the function to 0.75 everywhere on detecting the relocation of the goal, it being no longer at the maximum of the existing function. This reinitialised Q-learning, the middle curve, performed better and all the start locations took under 2000 steps to the goal after 200,000 steps. Room configurations with a single path to goal, such as the example of section 2, still took the most time. 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.

The function composition system, the lowest curve, performed by far the best. It first detects when the goal is moved, locates its new position and composes the new function. The number of paths has no impact on learning. It reached the same average steps to goal as the reinitialised Q-learning at about 5000 steps and as the basic Q-learning at about 3000 steps. This is an increase in learning rate of between times 60 and 100.

5 Limitations and Future Work

Future work will address limitations in the present experimental validation. A more thorough comparison to other approaches is needed. An alternative straw man such Dyna-Q+ [14], specifically designed to deal with changing worlds, would unquestionably reduce the speed-up experimentally obtained. Also, experiments using a random positioning of the goal would be more realistic.

The system exploits symmetries in the domain. What happens when these symmetries only partially hold or do not hold at all remains to be investigated. The system at present only works for rectangular rooms, but the snake is not so restricted and this work should readily extend to much more general shapes. In addition, the system only detects a change in goal position. Other changes such as the opening and closing of doors are also being investigated.

Previous work by this author [4] investigated using the features discussed in this paper to recognise if a similar task had been solved previously. Functions stored in a case base were then used to initialise and thus speed up the initial learning process. These two ideas will combined in future work. When a new task is being learnt, the system will progressively build up a solution by function composition, as different features become apparent.

6 Related Work

The most strongly related work is that investigating macro actions in reinforcement learning. Precup and Sutton [9] propose a possible semantics for macro actions within the framework of normal reinforcement learning. Singh [11] uses policies learnt to solve low level problems as primitives for reinforcement learning at a higher level. The work presented here gives one way macro actions can be extracted from the systems interaction with its environment without external help. Thrun's research [16] identifies macro actions used in multiple tasks. But unlike the research presented here, no mapping of such actions to new tasks is proposed. Mahadevan and Connell [7] use reinforcement learning in behaviour based robot control. Although in a much simpler domain, the work presented here does not require rewards for individual macro actions. Rather the macro actions are identified and extracted from the solution of a compound task.

Previous work that combines instance based or case based learning with reinforcement learning has principally addressed the economical representation of the state space. Peng [8] and Tadepalli [15] use learnt instances combined with linear regression over a set of neighbouring points. Sheppard and Salzberg [10] also use learnt instances but they are carefully selected by a genetic algorithm. Unlike this other research, in the work presented here the case is not an example of the value function during learning. Rather it is a function representing a macro action and the principle should be complementary to these other approaches.

This work is also related to case based planning [5, 17], 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, a composite plan is modified by using sub-plans, extracted from other composite plans.

7 Conclusions

This paper described a system that transfers prior learning to significantly speed up reinforcement learning in a changing world. Features extracted from the learnt function are use to index and compose functions from a case base to produce a close approximation to the solution of a new task. The experiments demonstrated that the system can rapidly respond to the situation when the goal is relocated.

Acknowledgements The author would like to thank Rob Holte for all his help. This work was supported by NSERC and Ontario Government scholarships.

References

1. A. D. Christiansen. Learning to predict in uncertain continuous tasks. *ICML* pp 72–81, 1992.
2. L. D. Cohen and Isaac Cohen. Finite element methods for active contour models and balloons for 2-d and 3-d images. *PAMI* 15(11):1131–1147, Nov 1993.
3. E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.* 1:269–271, 1959.
4. C. Drummond. Using a case-base of surfaces to speed-up reinforcement learning. *LNAI* volume 1266, pp 435–444, 1997.
5. K. J. Hammond. Case-based planning: A framework for planning from experience. *Journal of Cognitive Science* 14(3):85–443, July 1990.
6. A. MacDonald. Graphs: Notes on symetries, imbeddings, decompositions. Elec. Eng. Dept. TR-92-10-AJM, Brunel University, Uxbridge, Middx, U. K., Oct 1992.
7. S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence* 55:311–365, 1992.
8. J. Peng. Efficient memory-based dynamic programming. *ICML* pp 438–439 1995.
9. D. Precup and R. S. Sutton. Multi-time models for temporally abstract planning. *NIPS* 10 1997.
10. J. W. Sheppard and S. L. Salzberg. A teaching strategy for memory-based control. *Artificial Intelligence Review* 11:343–370, 1997.
11. S. P. Singh. Reinforcement learning with a hierarchy of abstract models. *AAAI* pp 202–207, 1992.
12. P. Suetens, P. Fua, and A. Hanson. Computational strategies for object recognition. *Computing surveys* 4(1):5–61, 1992.
13. R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *NIPS* 8 pp 1038–1044, 1996.
14. R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *ICML* pp 216–224, 1990.
15. P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating learning by approximating. *ICML* pp 471–479, 1996.
16. S. Thrun and A. Schwartz. Finding structure in reinforcement learning. *NIPS* 7 pp 385–392 1994.
17. M. M. Veloso and J. G. Carbonell. Derivational analogy in prodigy: Automating case acquisition, storage and utilization. *Machine Learning*, 10(3):249–278, 1993.
18. C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.

This article was processed using the LaTeX macro package with LLNCS style