Enhancing Transfer in Reinforcement Learning by Building Stochastic Models of Robot Actions

Sridhar Mahadevan
IBM T.J. Watson Research Center
Box 704, Yorktown Heights, NY 10598
(sridhar@watson.ibm.com)

Abstract

Recent work has shown that reinforcement learning is a viable method of automatically programming behavior-based robots. However, one weakness with this approach is that the learning typically does not transfer across tasks. Furthermore, there is unnecessary duplication of learning effort because information is not shared among the various behaviors. This paper describes an alternative technique based on action models that attempts to maximize transfer within and across tasks. Action models are inferred using a statistical clustering technique from instances generated by a robot exploring its task environment. Task-specific knowledge is encoded using a reward function for each subtask. A multi-step lookahead strategy using the reward functions as static evaluators is employed to select the most appropriate action. Experiments on simulated and real robots show that useful action models can be learned from a 12 by 12 scrolling certainty grid sensor. Furthermore, on the simulator these models are sufficiently rich to enable significant transfer within and across two tasks, box pushing and wall following.

1 Motivation

There has recently been a surge of interest in using reinforcement learning (RL) to automatically generate behaviors for an autonomous intelligent agent [1, 8, 9, 20, 21]. In particular, we have previously shown that the convergence of RL is significantly improved if the overall task is decomposed into a set of simpler sub-tasks, a transformation that naturally lends itself to robots operating with a modular architecture such as subsumption [10, 11]. For example, a robot task involving pushing boxes is learned quicker and more effectively if it is posed as three separate RL problems – finding boxes, pushing a found box, and unwedging from a stalled situation – rather than as one "monolithic" task.

However, even the above approach suffers from an endemic problem in RL: the learning typically does not transfer across tasks. In particular, the RL technique we used to train our robot, namely Q learning [22], creates learned knowledge that is task-specific. In Q learning, an agent incrementally updates the utility $Q(x,a)$ of performing an action a in a state x by a fraction of the estimated error in the current value. The best action in any given state is obviously the one with the best $Q(x,a)$ value. However, the problem with this approach is that the $Q(x,a)$ values are task-specific. A table of $Q(x,a)$ values for one task may be wholly inappropriate for another. For example, in Sutton's navigation example [21], the $Q(x,a)$ value associated with the best action generally points in the direction of the goal location. Given a new goal location, the old $Q(x,a)$ values are worthless.

Modularization of an agent's architecture is itself a mixed blessing. A new task may require a very different subsumption architecture, with little sharing of constituent modules. Finally, since different modules do not share learned information, transfer of learning *within* a task is also impeded, thus requiring unnecessary duplication of effort.

In an attempt to overcome both these problems, in this paper we describe an alternative method of programming autonomous intelligent agents. Our approach is based on inferring an action model from the states and actions generated by an agent exploring the task environment. An action model is simply a representation of a "next-state" function, mapping states and actions to resulting states. An action model is not task-specific because it applies to all tasks that employ the same sensors and effectors. Furthermore, there is rapid transfer of learned knowledge across different modules of the agent because they share the same action model.

We assume that task-specific knowledge is encoded in a set of reward functions, one for each of the sub-tasks of the overall task. Furthermore, the agent can invoke these reward functions on states it predicts will result from doing a certain action. In other words, unlike the usual situation in RL where an agent *passively* receives a reward upon executing an action, in our case the agent can *actively* use the reward function to discriminate between predicted future states. In particular, the agent can look ahead multiple number of steps to choose the most promising action.

Clearly, our approach embodies a number of important assumptions. It assumes that action models can be learned that apply across several tasks. In this paper we present some empirical evidence on learning action models for a fairly general robot sensor. Our approach also assumes that reward functions return non-zero values on states other than "goal" states. We believe that many complex real world domains (including robotics) admit of such reward functions. In any case, our previous work [10] and Whitehead's theoretical results on Q learning [24] suggest that reward functions that only return a non-zero value when a goal state is reached provide insufficient performance feedback for tractable learning.

The paper is organized as follows. Section 2 shows how action models can be learned using statistical clustering. This extends our previous clustering algorithm [10], which only worked on 18-bit binary state descriptions, to states comprising of 144 real values. Section 3 describes how action models can be used with task-specific knowledge in the form of reward functions to choose actions that will lead to high reward. Section 4 describes a simulated robot testbed, along with two sample tasks – box pushing and wall following. Section 5 evaluates the performance of the technique. The empirical results show that the robot is able to learn both the box pushing and the wall following tasks to satisfactory performance levels. Furthermore, the robot is able to transfer its learned knowledge fairly well from one task to another. Section 6 discusses certain aspects of our technique. Section 7 discusses related work. Finally, Section 8 summarizes the paper.

2 Action Models

This section describes a technique for forming action models using statistical clustering. Action models are formed by merging into clusters the states prior to and succeeding an action. Thus, an action model can be viewed as a description of the preconditions and postconditions of an action. An action may have several action models which specify the consequences of doing an action in differing initial states.

The new version of the clustering algorithm differs from the earlier version (the latter is described in detail in [11]) in several important ways. The new version can deal with real-valued state descriptions, not just binary descriptions. This is vital because a 144 real-valued state description is a sufficiently rich input to learn good action models, whereas the 18 bit description we used in our previous work is not. Furthermore, the new algorithm uses a simple distance-based probability metric for matching states and clusters, whereas the previous one used a Bayesian probability metric. Finally, the clusters contain no task specific information in the form of Q values.

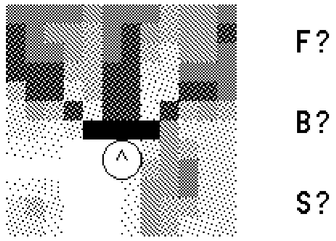

Figure 1: A state description created using a simulated scrolling certainty grid sonar sensor

States In this paper, we assume that a state is an $N \times N$ array of cells generated by a scrolling certainty grid sensor [14]. Certainty grids are a general representation of sensory information that can be computed from several types of robot sensors, such as sonar, infra-red scanners, and vision. Associated with each cell is a "cell occupancy" probability. A scrolling certainty grid is configured such that the position of the robot is slightly below the middle of the grid and facing "upwards". Thus, a scrolling certainty grid is a representation of the local environment surrounding the robot at any given time, which essentially "moves" with the robot (unlike the global certainty grid maps described in [15] in which the robot moves relative to the map).

Figure 1 illustrates a state description created by a 12×12 scrolling certainty grid sensor generated on a simulated robot. The position of the robot is depicted by the circular figure with the arrow showing its current orientation. The robot is assumed to occupy a 2×2 subarray within the overall grid. The cell occupancy probabilities are shown graphically, with darker values being closer to 1 and lighter values being closer to 0. The figure also shows three extra predicates: **F?** is a "front" detector that is true when any one of the cells located in the three rows above the robot and in the same two columns as the robot has an occupancy probability greater than 0.5. **B?** is a "bump" detector that is true when the front of the robot is almost touching an object. Finally, **S?** is a "stuck" detector that indicates that the robot is wedged.

Figure 2 shows a state description created using an infra-red scanner mounted on a real robot (the robot is shown in Figure 12). The real and simulated state

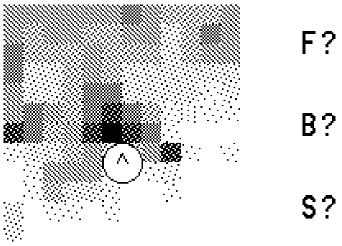

Figure 2: A state description created from an infra-red scanner on a real robot

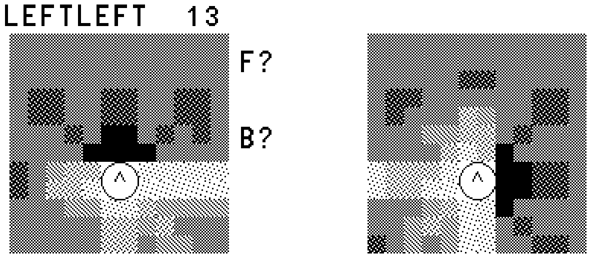

Figure 3: An action model learned by a simulated robot

descriptions are similar, although the "sharp" edges seen on the simulator are not usually obtained in real state descriptions. We will describe how the certainty grid sensor is computed in more detail in Section 4. For convenience, below we will represent the $N \times N$ 2-dimensional certainty grid array as a 1-dimensional array of size N^2.

Action models A *cluster* c is a n tuple $\langle c_1, \ldots, c_n \rangle$, where c_i is an estimate of the probability of the ith state element being occupied. Here n is the number of elements in a state (for an $N \times N$ certainty grid, $n = N^2$). An action model a_j of an action a is a 3 tuple $\langle C_{pre}, C_{succ}, m_j \rangle$ where C_{pre} and C_{succ} are clusters representing generalized descriptions of states immediately preceding and succeeding action a, respectively. m_j represents the number of instances that have matched the model so far.

Figure 3 illustrates an action model learned by a simulated robot. The probability information in a cluster description is depicted graphically, with the darker regions representing higher occupancy probabilities, and the lighter regions representing lower occupancy probabilities. This action model can be interpreted as follows. If the robot is currently in a state that matches the cluster on the left-hand side, then taking the action LEFTLEFT (turning left by 90 degrees) will result in a state matching the cluster on the right-hand side. This action model has been formed from 13 instances. The spatial pattern generated by the scrolling certainty grid sensor makes it easy to interpret the action model. In particular, the solid black stripe above the robot in the left-hand cluster represents an object, such as a wall. The action model can be interpreted as saying that if there is some object to the front of the robot, and the robot is bumped against it, then turning left by 90 degrees will result in the object appearing to the right of the robot in the resulting state. This model is useful when the robot is learning to avoid obstacles, or follow walls.

Matching Action Instances against Action Models An *action instance* $\langle s, a, t \rangle$ of an action a is a pair of states s and t that immediately preceded and succeeded (respectively) an instance of the use of action a by the agent. Deciding if an instance of an action a matches a given action model a_j requires matching the states s and t against the predecessor and successor cluster descriptions in the action model. Thus, we first describe how states are matched against clusters.

A simple criteria for deciding if a state s is an instance of a cluster c is obtained using the following "geometric" view of clusters and states. Basically, states and clusters are n-dimensional real-valued vectors, where each value lies between 0 and 1. Thus they can be interpreted as points within an n-dimensional unit hypercube. So a state can be viewed as an instance of a cluster if it is "close" enough in terms of the Euclidean distance metric on this hypercube. The problem with this criteria is that it treats all elements as equal, which is not true of most sensors. In particular, it is not true of the state description shown in Figure 1 since the three extra predicates **F?**, **B?**, and **S?** clearly have more "semantic" content that a certainty grid cell has.

To account for this disparity, we define the *weighted* Euclidean distance between a state s and a cluster c as $distance(s, c) = \sqrt{\sum_{i=1}^{n} w_i (s_i - c_i)^2}$. Here s_i and c_i are the probabilities of the ith state element and ith cluster element being occupied, respectively. The weights w_i of all the state elements is 1, except for the three extra predicates **F?**, **B?**, and **S?** which have a weight of ϵ^2, where ϵ is a parameter to the algorithm. Given this distance metric, a state s is considered an instance of a cluster c if $distance(s, c) < \epsilon$. Thus, by attaching an equal weight of ϵ to each of the three extra predicates, we are ensuring that a state and a cluster differing in any one of these predicates will not match because the overall distance between them will be at least ϵ. Finally, an action instance $\langle s, a, t \rangle$ matches an action model $a_j = \langle C_{pre}, C_{succ}, m_j \rangle$ if s matches C_{pre}, and t matches C_{succ}.

Merging Action Instances into Action Models Once an action instance has been found to match an action model, the instance is merged into the model as follows. Let $\langle s, a, t\rangle$ be an action instance that matches an action model $a_j = \langle C_{pre}, C_{succ}, m_j\rangle$. Denoting the updated action model as $a'_j = \langle C'_{pre}, C'_{succ}, m'_j\rangle$, we have that $C'_{pre_i} = C_{pre_i}\frac{m_j}{m_j+1} + s_i\frac{1}{m_j+1}$ and that $C'_{succ_i} = C_{succ_i}\frac{m_j}{m_j+1} + t_i\frac{1}{m_j+1}$. In other words, the new cluster probabilities are the weighted sum of the old cluster probabilities and the state probabilities. The number of instances of the model (m_j) and the state (assumed equal to 1) are used as the weighting factor. Finally, the number of instances of the new action model a'_j is simply $m'_j \leftarrow m_j + 1$.

Forming New Action Models If an action instance $\langle s, a, t\rangle$ does not match any existing action models, then a new model $a_j = \langle C_{pre}, C_{succ}, m_j\rangle$ is created as follows. By an empty cluster, we mean a cluster of the form $c = \langle 0.5, \ldots, 0.5\rangle$. Then, the cluster C_{pre} of the new action model is formed by merging state s with an empty cluster. Similarly, the cluster C_{succ} is formed by merging the state t with an empty cluster. Initializing a cluster's probabilities in this manner has the effect of relaxing the state description, while taking account of the particular occupancy probability values of the state. Finally, m_j is set equal to 1.

Merging Existing Action Models Often two existing action models are considered "close" enough that they can be merged to form a new action model. This requires a distance metric on action models. Given models $a_j = \langle C_{pre}, C_{succ}, m_j\rangle$ and $a'_j = \langle C'_{pre}, C'_{succ}, m'_j\rangle$, $distance(a_j, a'_j) = \sqrt{\sum_{i=1}^{n} w_i(C_{pre_i} - C'_{pre_i})^2 + w_i(C_{succ_i} - C'_{succ_i})^2}$. The metric computes the Euclidean distance between two action models in terms of the (weighted) Euclidean distances between the respective predecessor cluster descriptions and the respective successor cluster descriptions. Now given the above distance metric, two action models are merged together if and only if $distance(a_j, a'_j) < \epsilon$, the same parameter used above. Thus, two action models may initially be further apart than ϵ, but will eventually get merged if they come closer than ϵ.

We can now define what it means to merge two action models $a_j = \langle C_{pre}, C_{succ}, m_j\rangle$ and $a'_j = \langle C'_{pre}, C'_{succ}, m'_j\rangle$ into a new action model $a''_j = \langle C''_{pre}, C''_{succ}, m''_j\rangle$. The predecessor cluster description $C''_{pre_i} = C'_{pre_i}\frac{m'_j}{m'_j+m_j} + C_{pre_i}\frac{m_j}{m'j+m_j}$. Similarly, $C''_{succ_i} = C'_{succ_i}\frac{m'_j}{m'_j+m_j} + C_{succ_i}\frac{m_j}{m'j+m_j}$. In these computations, the number of instances that make up the two action models is used to weight the probabilities. This weighting scheme tilts the final values towards those associated with the larger of the two clusters over those associated with the smaller cluster. Finally, the number of instances of the new action model a''_j is $m''_j \leftarrow m_j + m'_j$.

This concludes the description of the technique for forming action models using statistical clustering. We now turn to the issue of temporal projection and action selection using action models.

3 Temporal Projection

Thus far we have described how action models can be learned by an agent from states it perceives, and actions it carries out. In this section we show how these action models can actually be used by the agent to choose the most appropriate action.

The utility of doing an action a in a state s can be defined as shown in Figure 4. The basic idea is to select the action model of a that best matches the current state, and recursively estimate the utility of the next state, where the next state is essentially the successor cluster description of the model. $R(s, a, t)$ is the reward obtained on doing action a in state s which resulted in state t. Given a way of estimating $U(s, a, l)$, then the best action in any given state is simply the one which has the highest $U(s, a, l)$ value.

$U(s, a, l)$:
1. If the lookahead distance $l = 0$, return 0.
2. Retrieve action model $a_j = \langle C_{pre}, C_{succ}, m_j\rangle$ that maximizes the quantity
$P(s, C_{pre}) = \frac{\sqrt{n+3\epsilon^2} - distance(s, C_{pre})}{\sqrt{n+3\epsilon^2}}$
where $\sqrt{n+3\epsilon^2}$ is the maximum weighted Euclidean distance.
3. Let the predicted next state $t = C_{succ}$
4. Return $U(s, a, l) = P(s, C_{pre})R(s, a, t) +$ maximum over all a' of $\gamma U(t, a', l-1)$ where γ is a discount factor less than 1.

Figure 4: Evaluating the utility of an action a in a state s to a given lookahead distance l

Figure 5 illustrates the lookahead procedure just described for a wall following robot. The original state is shown in the first row. The second row shows the possible next states. Only two actions are shown for clarity. The third row shows the possible states obtained by expanding the states in the second row. Again for clarity, only two actions are shown, one of them different from before for diversity. The actions taken are indicated above the state resulting from the action. The first number after the action indicates the probability of match between the state and the best matching model for the action. The second number indicates the reward for the particular action instance.

In the original state, the robot is bumped (**B?**) against some object in front (**F?**) of it. The best matching action model for the FORWARD action predicts with probability 0.93 that moving forward will result in the robot being stuck (**S?**), an undesirable transition indicated by the reward of -1.0. The best sequence of moves for the robot is to turn right by 90 degrees (RIGHTRIGHT), and then to move forward along the "wall" on its left. Note that the topmost state and the rightmost state on the bottom match fairly closely, as they should since RIGHTRIGHT is the inverse of the LEFTLEFT action.

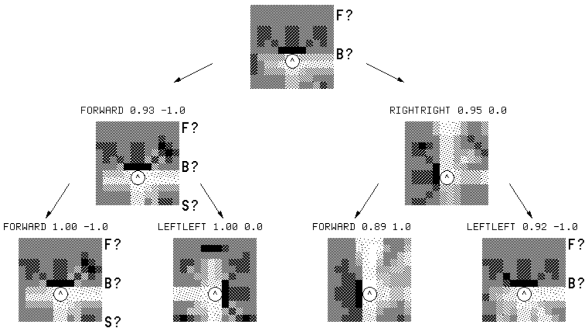

Figure 5: A portion of a lookahead tree created by a simulated wall-following robot

Note that the lookahead procedure in Figure 4 is an approximation to a more accurate (and intractable) lookahead procedure, which would involve exploring every possible action model of every action [22]. Even so, the complexity of the lookahead procedure in Figure 4 is at least $|a|^{l-1}|M|$, where $|a|$ is the number of actions, $|M|$ is the average number of action models, and l is the lookahead distance. Thus, it is exponential in the number of actions, and can get very expensive when the number of actions and the lookahead distance is very large. We have not systematically experimented with heuristics for reducing the lookahead time for two reasons. First, the number of actions in our case is relatively small (5) and we have only needed small amounts of lookahead (less than 5). Furthermore, long lookaheads are undesirable anyway as the quality of temporal projection diminishes as the number of lookahead steps is increased.

Figure 6 gives the main steps of overall algorithm, whose major components were described above. The overall loop consists of repeatedly sensing the state, picking the best action using temporal projection, and updating the action model. One detail omitted from the figure is that in choosing an action, it is necessary to sometimes take a random action as a tradeoff between exploration and policy directed behavior. In our experiments we have chosen a straightforward strategy whereby an agent takes a random action 10% of the time, although more elaborate tradeoff algorithms are possible [8]. The exact values of the parameters to the algorithm will be given in the section on experimental results.

1. Initialize action models under each action a to NIL. Fix clustering parameter ϵ and lookahead distance l.

2.Do Forever:

   a. Observe the current world state s.

   b. Choose an action a that maximizes $U(s, a, l)$ using temporal projection.

   c. Let the new state resulting from action a be t

   d. If there exists an action model a_j which matches the action instance $\langle s, a, t\rangle$ to within ϵ then merge the action instance into a_j. Else create a new action model a'_j from the instance.

   e. Merge any existing models a_j and a'_j under action a if $distance(a_j, a'_j) < \epsilon$.

Figure 6: The Projection/Action Models Algorithm

4 A Simulated Robot Testbed

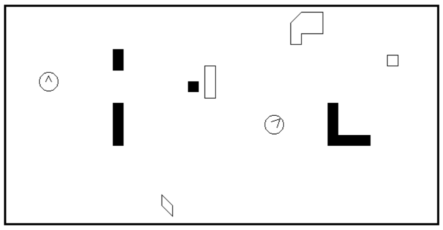

Figure 7: A Simulated Robot Environment

In this section we describe a simulated robot testbed used to experimentally evaluate our technique. Figure 7 illustrates a simulated robot environment that we used earlier for the box-pushing task [11]. Agents are depicted by the circular shapes in the figure with the arrow indicating their orientation (we use only one agent in our experiments). The outline objects represent movable "boxes". The shaded figures are immovable obstacles. This environment can be used to define a number of robot tasks. The box pushing task involves finding and pushing the outline figures to the walls and corners of the environment. The wall following task is to cruise along the walls of the environment while avoiding all the objects (boxes and obstacles).

The scrolling certainty grid sensor is computed as follows. The $N \times N$ array (for $N = 12$) is initialized with every cell occupancy probability at 0.5. After every action the robot computes "range" data representing distances to the nearest objects in several different directions. The range data is then combined with the existing occupancy probabilities in the current certainty grid – after rotating or translating the grid depending on the particular action – to yield a new updated certainty grid. On the simulator, the range information is computed using the data structures storing the entire simulated environment. On the real robot, range data is computed either using sonar sensors or the infra-red scanner. We have tried to ensure that the scrolling certainty grid sensor on the simulator resembles the one obtained on the real robot. A more detailed discussion of certainty grids is given in [14, 15].

4.1 Two Example Robot Tasks

We now describe two robot tasks that we use to illustrate transfer across and within tasks.

The Box-Pushing Task: The specification of the box pushing task remains the same as in our previous work [10]. In particular, the task is decomposed into three separate behaviors: box finding, box pushing, and unwedging. Since finding a box must precede pushing one, the module in charge of finding is given the lowest priority and is always active. If a potential box is found, the agent must try to push it, and thus the pushing module overrides the box finder whenever it is active. Its applicability condition is triggered by the "bump" **B?** predicate. Finally, the agent cannot carry out any of its activities if it stalled against an immovable object. Therefore, the unwedging module overrides the other two whenever it is activated. Its applicability condition is triggered by the STUCK (**S?**) sensor. The pushing or unwedging behaviors remains active for a period of 5 time steps even after their triggering conditions are turned off.

Knowledge of the box-pushing task is specified by a reward function for each behavior. In particular, if the agent is finding a box, it receives a reward of +3 for going forward and turning on the "front" predicate **F?**. If **F?** is false, the reward received is -1. For the pushing behavior, the robot gets a reward of +1 if it went forward and **B?** remained true. If **B?** turned off, it gets a reward of -3. Finally, if the agent is trying to unwedge from a stalled state, it gets a reward of +1 for going forward and being in a state where **S?** is off. It gets a reward of -3 if **S?** is on. The default reward for all three behaviors is 0.

The Wall Following Task The wall-following task is decomposed into two separate behaviors: wall finding, and wall following. Since finding a wall must precede following one, the module in charge of finding is given the lowest priority and is always active. If a potential wall is found, the agent must try to follow it, and thus the wall-following module overrides the wall finder whenever it is active. Its applicability condition is triggered by either the bump predicate **B?** being true, or when a "wall" predicate **W?** is true. **W?** is true when two contiguous cells in the same set of rows as the robot, and in a column on its left side or its right side have occupancy probabilities greater than 0.6. The wall follower remains active for 5 steps even after its triggering condition turns off.

Knowledge of the wall-following task is specified by a reward function for each behavior. In particular, if the agent is finding a wall, it receives a reward of +3 for going forward and turning the **F?** sensor described above. If **F?** is false, the reward received is -1. Finally, the default reward for the wall finder is 0. For the wall following task, the robot gets a reward of -1 if either **B?** is true or **W?** is false. Also, the robot gets a reward of +1 for going forward in a state in which **B?** is false and **W?** is true. Finally, the default reward for the wall follower is 0.

5 Experimental Results

Task Performance: We begin the experimental analysis of the action models technique by studying how well it learns the box pushing task and the wall following task. To measure task performance, we use the same metrics as in our earlier study [10]. Specifically, we examine how well each individual behavior was learned by the agent, as well as study its overall performance at the task. We measure both of these on the simulator over a learning run of 2000 steps, which is broken up into 20 equally long trials of 100 steps. At the beginning of each trial, the simulator environment is reset back to its original state. All the experiments were conducted with $\epsilon = 2.57$, $\gamma = 0.9$, and a 3-step lookahead.

We define the *average reward* at a given step for a behavior to be the total reward received by the behavior upto that step divided by the number of steps the behavior was active. Figure 8 shows the average reward at each step of a learning run of 2000 steps for the box pushing task on the simulator. The horizontal axis represents percentage of the learning run. Figure 9 show the same information for the wall following task on the simulator. There is clear improvement in performance over the learning run for each of the behaviors.

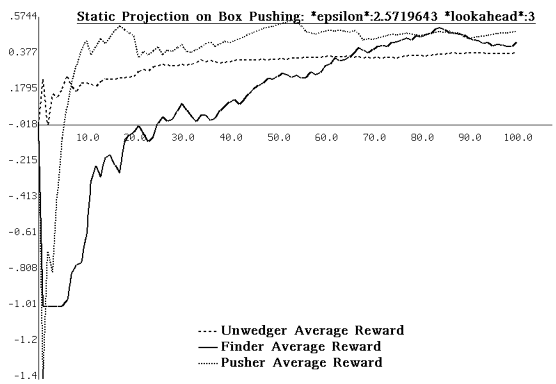

Figure 8: Performance improvement at box pushing on simulator

The *average performance* at the end of a learning run

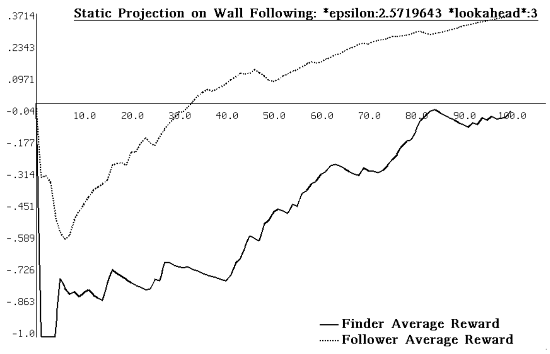

Figure 9: Performance improvement at wall following on simulator

Behavior	Act. Models	Q/Cluster.	Hand
Wall Finder	24%	?%	32%
Wall Follower	69%	?%	61%
Box Finder	36.5%	55%	48%
Box Pusher	87.5%	88%	88%
Unwedger	87.5%	75%	75%

Table 1: Average reward at the end of a learning run. A "?" indicates the data is currently not available.

is $\frac{AverageReward - LowestReward}{BestReward - LowestReward}$. The average performance measures the improvement in a behavior as a fraction of the maximum possible improvement. Table 1 lists the average performance (in percentage form) at the end of a learning run for three different techniques on each of the five behaviors on the two tasks, wall following and box pushing.

With the exception of wall finding and box finding, the improvement achieved by the action models technique is as good or better than the Q learning/clustering technique and the handcoded program. This is remarkable given the low level of lookahead (3), and the more complex sensor used (144 real values instead of 18 bits). Furthermore, a lower average reward at box finding does not necessarily translate to poorer overall task performance. To emphasize this point, Table 2 shows that the overall performance of the action models technique on the box pushing task is much better than the best result we obtained in our previous study with the Weighted Hamming learning technique [10].

Technique	Box-Pushing Steps
Action Models	16%
Q Learn./Wt. Hamming	10%

Table 2: Overall performance at box pushing task

Across Task Transfer: Another important dimension on which to evaluate the action models technique is transfer across tasks. Figure 10 shows the performance of the robot at box pushing over a run of 2000 steps after it was trained on the wall following task. No additional learning was allowed to take place. The curves in Figure 10 show some variation because the robot is still taking a random action 10% of the time. Notice that the curves directly start to go above zero, indicating that the robot is clearly transferring a lot of information from the wall following task to the box pushing task. It is instructive to compare this graph with that in Figure 8. Table 3 summarizes the average performance at box pushing after training on wall following. Comparing with the figures in Table 1 shows that the performance is close to that when the robot was directly trained on the box pushing task.

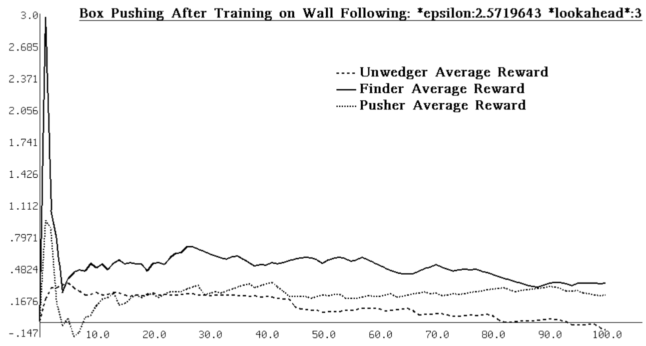

Figure 10: Performance on box pushing on simulator after training on wall following

Task	Finder	Pusher	Unwedger
Box Pushing	35%	82%	73%

Table 3: Average performance at box pushing on simulator after training on wall following

Conversely, Figure 11 and Table 4 show the data for transfer from the box pushing task to the wall following task. Once again, the curves in Figure 11 show some variation because the robot is still taking a random action 10% of the time. Interestingly, here the performance at wall finding is superior to the case when the robot was directly trained on the wall finding task! However, performance at wall following is not as good as when the robot was directly trained on the wall following task. In any case, the data clearly shows significant transfer is taking place.

Task	Finder	Follower
Wall Following	34%	48%

Table 4: Average performance at wall following on simulator after training on box pushing

5.1 Data from a Real Robot

We have conducted preliminary tests of our technique on a real robot (shown in Figure 12) equipped with

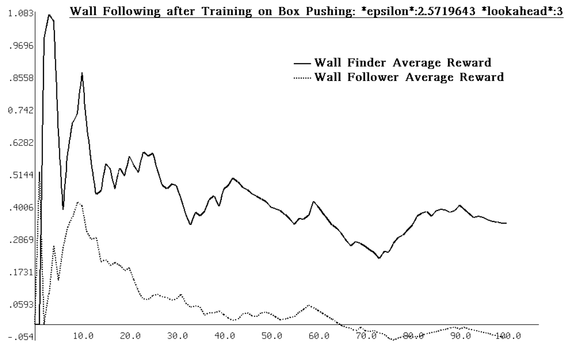

Figure 11: Performance at wall following on simulator after training on box pushing

Figure 12: A real robot testbed

a variety of sensors, including sonar and an infra-red scanner. Figure 13 shows an action model learned by the real robot using an infra-red scanner. The model illustrates the result of the robot pushing a box – the predicates "bump" **B?** and "front" **F?** are true before and after the action, and "stuck" **S?** is not true either before or after the action. The action model shows a clear outline of the box in front of the robot.

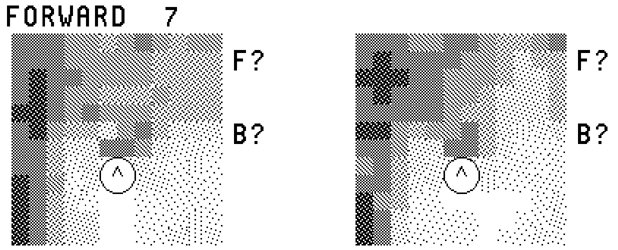

Figure 13: Action model learned on real robot for pushing boxes

One problem in testing the action models technique on the real robot is that in our current implementation, it takes upto half a minute to integrate range data into the certainty grid. This delay implies that a complete learning run would take over 16 hours, which makes it difficult if not impossible to carry out detailed experiments of the form described above. We hope to carry out more systematic experiments on the real robot, including studying transfer of learning across tasks, using faster sensor integration software in the near future.

Summarizing, the experimental data on the simulator shows that the action models technique is capable of learning a variety of different behaviors, and also capable of transferring the learned information across tasks.

6 Discussion

Scrolling Certainty Grids: There is an apparent circularity in learning action models from scrolling certainty grid state descriptions. The routine that translates or rotates the grid at every step (depending on the particular action taken) in some sense already has an action model built into it! There are several reasons why learning action models still makes sense. The built-in action model is an idealized one. In contrast, the learned action model can predict certain features of real environments, such as the fact that objects are extended in space. Learned models can also account for limitations in the sensors used, such as specular reflections from sonar. Since learned models are built using information obtained after carrying out actions, they can incorporate information (such as the new values of cells at the boundaries of a grid) that is unavailable to the built-in action model. Finally, the learned action model also projects three extra predicates, at least one of which (the stuck bit **S?**) cannot be predicted using the built-in action model.

Selective Projection: Our temporal projection algorithm attempts to project every state element, which is surely computationally inefficient. A better strategy would be to project only those state elements that are deemed "relevant" to the agent's current goals. For example, if a "box-like" object appears on the scrolling certainty grid image, then the agent could project the elements comprising the object to determine a sequence of actions that would bring it closer to the object.

Composing Action Models: One interesting possibility we would like to explore is composing sequences of "one-step" action models to yield "multi-step" composite models. Composite action models enable the robot to carry out deeper searches of the tree of future states. Such composite action models are called "macros" in planning [5] and machine learning [13]. A central question here is which sequences of actions is it worth forming macros of? One approach is to form macros of those action sequences that lead to high reward.

7 Related Work

There are several recent papers on using action models in the RL literature. Whitehead [23] describes a genetic classifier system that embodies a projection technique to enhance prediction of utilities in situations where the uncertainty is high. Sutton [21] describes a system called DYNA that built a world model by storing states prior to and resulting from doing an action. By alternating between transitions in the world model and the real world, Sutton showed that learning converged much quicker than using Q learning alone. However, DYNA was limited in that states were not generalized. Also, DYNA had no mechanism for projection over action sequences. Lin [9] describes a neural net approach to learning an action model, but his system only performs one step lookahead. None of these studies focus on across-task transfer of learned information.

Singh [20] describes a RL system that is able to transfer learning across tasks. One of the strengths of his system, which is based on an architecture originally developed by Jacobs [7] and Nowlan [17], is that it can also automatically learn the decomposition of a composite task. However, his system seems specific to sequential tasks, such as "go to place A and then go to place B". Watkins' thesis [22] contains an excellent discussion of lookahead-based action selection, which he calls "model-based control" to distinguish it from one-step lookahead used in Q learning which he calls "primitive control".

The clustering technique used to form action models is similar in spirit to other state generalization techniques used in RL [1, 9, 16]. However, its use in forming a general action model for multi-step lookahead across different tasks seems novel in the RL literature.

There is also a fairly large set of papers in the robotics literature on inferring the inverse kinematics of robot arms, which can be viewed as a type of action model. For example, Mel [12] describes a program called MURPHY that learns an action model using a neural net for computing the inverse kinematics of a jointed arm. The possible joint angles generated by a robot arm moving through a cluttered workspace seems a more restrictive and tractable hypothesis space than the possible certainty grid configurations generated by a robot exploring its task environment.

Several robot learning systems have been implemented that learn action models. Zhou [26] describes a genetic classifier system that learns in a simulated robot navigation domain. However, he assumes perfect sensors on the robot that can detect its current x-y location, heading, and presence of obstacles. Christiansen et. al. [2] describe a program for learning action models for a tray-tilting task. While their system deals with non-deterministic actions and noisy state descriptions, the size of the state space is fairly small (9 regions). Furthermore, they use a rote learning technique that does not generalize across states. Pierce [18] describes a system that starts with an uninterpreted sensors and effectors, and discovers the physical placement of the sensors and the different classes of actions. One difference between his system and ours is that he does not address the issue of learning a predictive model of the primitive actions for the purposes of lookahead planning. Finally, Drescher [4] describes a technique called the *schema mechanism* which learns action models for a simulated agent. One problem with the schema mechanism is that it requires exhaustively storing correlational data across state elements and actions. It seems that it would require a lot of sensory pre-processing to deal with a real robotics task.

Many examples of lookahead-based action selection can be found in work on planning and problem-solving [3]. Samuel's [19] early work on a program that learned to play checkers program is a classic example of a self-improving multi-step lookahead system. However, in games like checkers, actions have deterministic side effects, which are known in advance.

Yang and Fisher [25] describe a machine learning system that uses conceptual clustering [6] to group together actions with similar preconditions and effects. The standard assumption underlying this work and others like it is that the effects of primitive actions are known and can be specified using a logical STRIPS-like formalism. In this paper we are addressing a different problem, namely learning a stochastic action model of primitive actions of a mobile robot from noisy sensors.

8 Conclusions

This paper describes an implemented technique that attempts to enhance the transfer within and across tasks in reinforcement learning by using action models. A statistical clustering technique is used to learn action models from instances generated by an agent exploring its task environment. The action model is used to generate possible future states, and task-specific reward functions are used as static evaluators to select the most promising action. Experiments on a simulated robot environment showed that action models could be learned from a 12 by 12 scrolling certainty grid robot sensor that enabled significant transfer within and across two tasks, box pushing and wall following.

9 Acknowledgements

I thank Jonathan Connell for implementing the scrolling certainty grid sonar, and for his useful comments on an earlier draft. The idea of using a distance-based clustering technique originated in a discussion

with Nils Nilsson. Finally, I thank the reviewers for their comments.

References

[1] D. Chapman and L. Kaelbling. Learning from delayed reinforcement in a complex domain. In *Proceedings of the IJCAI*, 1991.

[2] A. Christiansen, T. Mason, and T. Mitchell. Learning reliable manipulation strategies without initial physical models. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1224–1230. Morgan Kaufmann, 1990.

[3] P. Cohen and E. Feigenbaum. *The Handbook of Artificial Intelligence.* William Kaufmann, 1982.

[4] G. Drescher. *Made-up Minds: A Constructivist Approach to Artificial Intelligence.* PhD thesis, MIT, 1990.

[5] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artifical Intelligence*, 3(4), 1972.

[6] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[7] R. Jacobs. *Task Decomposition Through Competition in a Modular Connectionist Architecture.* PhD thesis, University of Massachusetts, Amherst, May 1990.

[8] L. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University., 1990.

[9] L. Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. Technical Report CMU-CS-90-109, Carnegie-Mellon University., 1990.

[10] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of the Ninth AAAI*, pages 768–773. Morgan Kaufmann, 1991.

[11] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 1992. To Appear.

[12] B. Mel. Further explorations in visually-guided reaching: Making MURPHY smarter. In *Neural Information Processing Systems (NIPS)*, pages 349–355, 1988.

[13] T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-Based Generalization: A unifying view. *Machine Learning*, 1(1), 1986.

[14] H. Moravec. Certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[15] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.

[16] N. Nilsson. An approach to learning sequences of actions: Combining delayed reinforcement and input generalization. Unpublished draft notes.

[17] S. Nowlan. Competing experts: An experimental investigation of associative mixture models. Technical Report CRG-TR-90-5, Univ. of Toronto, September 1990.

[18] D. Pierce. Learning a set of primitive actions with an uninterpreted sensorimotor apparatus. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 338–342. Morgan Kaufmann, 1991.

[19] A. Samuel. Some studies in machine learning using the game of checkers. II – recent progress. *IBM Journal of Research and Development*, 11:601–617, 1967.

[20] S. Singh. Transfer of learning across compositions of sequential tasks. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 348–352. Morgan Kaufmann, 1991.

[21] R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[22] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, 1989.

[23] S. Whitehead. A role for anticipation in reactive systems that learn. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 354–357. Morgan Kaufmann, 1989.

[24] S. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings of the Ninth AAAI*, 1991.

[25] H. Yang and D. Fisher. Conceptual clustering of means-ends plans. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 232–234. Morgan Kaufmann, 1989.

[26] H. Zhou. CSM: A computational model of cumulative learning. *Machine Learning*, 5:383–406, 1990.