Cumulative Learning Using Functionally Similar States

M. M. Hassan Mahmud
Department of Computer Science
University of Illinois At Urbana Champaign
Urbana, IL 61801, USA
mmmahmud@uiuc.edu

Sylvian Ray
Department of Computer Science
University of Illinois At Urbana Champaign
Urbana, IL 61801, USA

January 2005

Technical Report
UIUCDCS-R-2005-2511

Abstract

In this paper we propose a Cumulative Learning System for artificial agents that uses the idea of Functional Similarity between states. The general idea of Cumulative Learning is to build a *cognitive architecture* for an artificial agent that 'lives' for a long time and solves many related tasks during its lifetime. Two states (or situations) are said to be functionally similar (FS) with respect to an action if the action induces the same change on both the states. We define the notion of FS for Markov Environments, and then use that to develop a Predictive Model (PM) that given states and actions observed so far, predicts the next state when an action is taken in some novel state (state never observed before or often) - i.e. the PM is a novel type of forward model. We also describe a planning mechanism for goal directed MDPs with multiple goals that uses the PM to solve tasks quicker using information from solutions to similar tasks solved previously by the agent. After establishing some necessary theoretical properties of both we perform experiments that shows the efficacy of our method. We also outline how the current system, which can actually be categorized as a Lifelong Learning system, may be extended to a complete cumulative learning system.

1 Introduction

A chief reason humans are good at behaving intelligently in the real world is that they can use knowledge learned in one situation in another similar situation. For example we can use knowledge acquired from playing squash to learn how to play racquetball. One reason we can do this is because the two situations are *functionally similar* - that is they 'behave' similarly with respect to actions. That is both squash balls and racquetball balls behave similarly with respect to particular racquet strokes (but not quite the same way as a racquetball ball bounces quite a lot more than a squash ball). In this paper we investigate how we may impart this human ability to exploit functional similarity between situations to artificial agents. We do so in the context of *Cumulative Learning*.

The basic idea of Cumulative Learning in general is to design a cognitive architecture for an agent that 'lives' for a long time in some environment and solves many similar tasks during its lifetime. The agent then speeds up learning one particular task by using knowledge gained when learning related tasks previously. One way to look at Cumulative Learning is as a way to set bias for a new task using knowledge accumulated from solving previous tasks. Since the performance (in terms of no. of examples required to learn) of an learning agent depends to a large extent on the bias given to it in the beginning ([Thrun and Pratt, 1998]), Cumulative Learning helps speed up learning. However, along with this, the difference between current lifelong learning systems ([Thrun and Pratt, 1998],[Thrun, 1995], [Caruana, 1997],[Drummond, 2002], [Mitchell and Thrun, 1993]) and Cumulative Learning is the cognitive architecture aspect of CL. The latter implies a system that incorporates sophisticated methods to acquire, store and reuse knowledge in a way that is appropriate for complex domains. We say more on this in this section a little later.

The CL system we present in this paper is in effect a type of forward model (e.g. [Jordan and Rumelhart, 1992], [Mitchell and Thrun, 1993], [Kawato, 1999], [Karniel, 2002] and [Flanagan and Wing, 1997]) for Markov Environments , i.e. a model that predicts the effects of actions in novel states (states not seen before ever or often); the predictions can then be used to behave more intelligently in novel states. The difference between our method and previously developed models is that, it uses functional similarity between states in addition to structural similarity (similarity in terms of features describing the states) to predict effects of actions in novel states. The use of functional similarity results in a model in which distinctions between groups of similar states are made at a finer grain than in previous methods. The model also predicts a distribution over next states (instead of just a specific next state), is more principled than existing methods and gives a measure of prediction error. In addition, the classifier employed has to learn the much simpler mapping of states $\rightarrow \{-1, 1\}$, rather than states $\rightarrow$ states, which results in more accurate classification. We discuss the notion of functional similarity and the predictive model in section 3 and we compare the PM with other forward models in section 3.3.4.

As an application of the predictive model, we present a planning mechanism for goal directed Markov Decision Problems ([Barto *et al.*, 1995], [Moore *et al.*, 1999]) with task independent cost signals ([Moore *et al.*, 1999]). The planning mechanism uses the predictive model to solve a particular task much faster using information learned from solving similar tasks. We present the planning mechanism in section 4. A block diagram showing the relationship between the predictive model and the planning mechanism is given in figure 1. We present our characterization of the MDPs we consider in this paper in section 2. We present preliminary empirical results on the effectiveness of our method in section 5, and discuss the implications of the results, possible future avenues of research and related work in section 6. We summarize our results in section 7.

Finally, we should note that, although the method we describe in this paper is a perfectly acceptable learning mechanism for interesting multi-task/lifelong learning problem domains, it is nonetheless incomplete as a CL system since it does not include the methods needed for sophisticated long term behavior. We give an outline for what needs to be done to extend this into a full-fledged Cumulative Learning System in sections 3.1 and 6.1.

A Note on Notation: In the following we sometimes use the same lowercase Greek letter to represent different positive constants. We rely on the context to distinguish between the different usages.

2 Problem Definition

In this paper we consider an agent in a particular type of discrete Markov Decision Problem, which has also been considered in [Barto *et al.*, 1995] and [Moore *et al.*, 1999]. Such a MDP is specified by a set of states S and a set of probabilistic actions A. When an $a \in A$ is applied to a state s, a takes the agent to a state s' with some probability, denoted by $P(s'|s, a)$. The distribution over next states in turn is denoted by $P_a(s)$. In addition each state $s \in S$ is described by a finite set of features $F = \{F_i\}$, i.e. there is a bijective

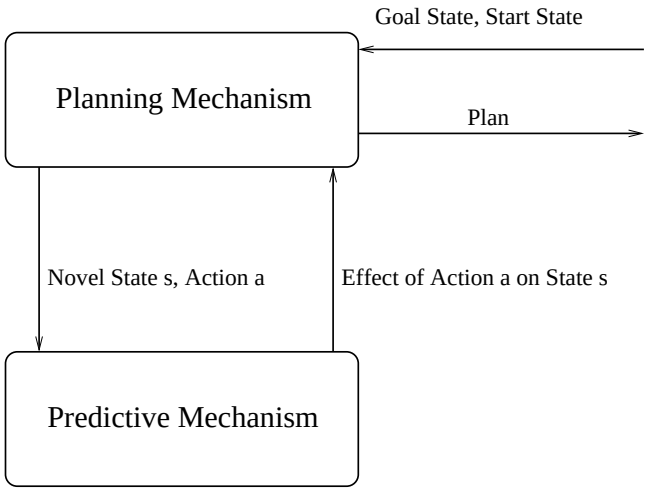

Figure 1: **CL System based on Functionally Similar States.**

mapping $S \rightarrow F_\times$ where $F_\times = F_1 \times F_2 \times \cdots F_n$ is the product space of the features. Eventually, we will consider continuous features, but for the moment we are considering domains with discrete features. In addition there is a cost $0 \leq d_{s,a,s'} < \infty$ associated with each transition from any state s to another state s' caused by an action a. A single task T of the agent consists of getting to some state $s \in T_G \subset S$ from some state $s' \in T_S \subset S$ - where T_G and T_S are, respectively, the goal states and start states corresponding to T - while keeping the costs $d_{s,a,s'}$ incurred during the transfer as low as possible (more on this in section 4). In the Cumulative Learning setting, the agent will need to solve two or more such tasks in the same domain. One further requirement we make, that was not made in prior works mentioned above, is that we require that for each task, the agent be given a description of the goal states (i.e. the acceptable values of each feature of the goal states). The reason we are considering this type of MDPs is that we believe the planning mechanism , combined with the predictive model, we will be able to solve many interesting problems. We will strive to demonstrate this in future.

At this point we note that the concept of Functional Similarity, and thus the Predictive Model, to be presented in section 3 is applicable to any domain constructed on top of a Markov Environment, where a ME is the system described above excluding the cost signals and the tasks. The planning mechanism described in section 4, on the other hand, is specific to the type of MDP described above.

Now, in the following we define the effect of actions in our model more explicitly. We introduce this rather explicit characterization of an action because it is necessary for our description of Functional Similarity later on. Let F be the set of features $\{F_1, F_2, \cdots, F_n\}$ that describe states and $F_\times$ their product space. In our model we assume that number of distinct values of each feature is countable. Note that the set of states S represented in terms of the features, $\{F(s)\} = \{(F_1(s), F_2(s), \cdots, F_n(s))\}$ is the same as $F_\times$. Let $\mathcal{F}$ denote the set of *feature functions* - i.e. the set of functions of the form $f : F_\times \rightarrow F_\times$. So when an action a is applied in a state $s = F(s)$, a applies a function g_a of the form $g : F_\times \rightarrow P_\mathcal{F}$, where $P_\mathcal{F}$ is the space of distributions over $\mathcal{F}$. Let us denote these set of *action distribution* functions by $A_P = \{g_a : a \in A\}$. Thus given a state s, the action selects a distribution over the space of feature functions $\mathcal{F}$ using $g_a(s)$, and then uses that distribution to select the $f \in \mathcal{F}$ to apply to the state. The output of the selected $f \in \mathcal{F}$ is the resulting next state. So to summarize, when an action a is applied at state $s = F(s)$, then

- let $P = g_a(s)$, where P is a distribution over $\mathcal{F}$, and g_a is the action distribution function corresponding to a;

- let $f \in \mathcal{F}$ where f is selected according to the distribution P obtained in the previous step;
- then the effect of the action s' is given by $f(F(s)) = F(s')$.

It is important to note that this description results in exactly the same type of actions as in a ME.

3 Functional Similarity and the Predictive Model

In this section, we define our notion of similar states and describe the Predictive Model that is based on this notion of similarity; we proceed as follows. First we describe Functional Similarity, which is the core idea of this paper. We start with a qualitative description of the idea, and then we give a more formal description. We then describe the predictive model that learns to identify states that are functionally similar and use that ability to determine the effect of an action on a novel state.

3.1 Motivation for Functional Similarity

Let us first describe FS qualitatively with two examples. In the first example, consider two states for an artificial agent, one with a wooden block on a table and the other with a wooden block on the floor. Both these states are functionally similar with respect to action PUSH-BLOCK because the change induced in both these states by PUSH-BLOCK is the same i.e. in both cases the block moves forward a little. Now consider another pair of states consisting of the agent on a flat grassy field and the agent on the pavement - again both these states are functionally similar with respect to the action MOVE-FORWARD. So qualitatively speaking, we say two states are functionally similar with respect to an action if the action induces similar changes in both the states.

Philosophical Aside: From a philosophical perspective, it seems to us that, functional similarity plays a big part in the mechanism we use to understand the world (derive semantics). As Minsky pointed out in [Minsky, 1988], when we say we 'understand' something what we really mean is we know how it relates to everything else we know. For instance when we say we understand what pencil is, we mean that we know we can make marks on paper with it, poke someone we do not like with it and so on. The 'basis' object we can relate anything we understand is ourselves. And this is what functional similarity captures - it describes how everything in the world relates to the agent. Thus a model of the world in terms of functional similarity leads to, in a non-phenomenal sense, basic semantics. In fact, in my opinion, functional similarity is more important than physical similarity in some sense because it is functional similarity that we use to exploit the world.

We may further speculate on the possibilities of the notion of functional similarity. In human interactions with the real world, situations are usually functionally similar if they contain the same or similar objects (e.g. the block in the first example above). Thus we can use functional similarity between situations to identify objects - which will usually be the commonality between the parts of the environment that changed on application of the action on the functionally similar situations. Once we have identified objects, we can then determine the relationship/semantic connection between different objects in terms of how, on application of an action, change in one object relates to change in another object. As an example, application of the WRITE action on a pencil results in a paper object having marks on it. We can then form a semantic connection between the two objects by describing the effect of the action on the objects. Thus functional similarity can help us develop a non-phenomenal semantic network that is grounded in the domain. The agent can then behave intelligently in novel situations that contain objects modeled in the semantic network by using the relationships among the objects to predict the effect of its actions, and then formulating plans for the novel situations based on the predictions.

3.2 Formal Description

As will be shown below, the notion of 'change induced' we mentioned above is the algorithmic representation of the function applied by the action to change one state to the next. The more technical material in this section is dedicated towards showing that such an algorithmic description exists for any function an action may apply if the domain conforms to the specification given in section 2. This idea is fairly intuitive; since each feature is countable, and the number of features is finite, any function mapping from finite number of feature to finite number of features will have at most countable number of pairs. And therefore it will be possible to give an algorithmic description of the function. This material then culminates in a description of functional similarity between states, according to which two states are functionally similar with respect to an action if the most compact algorithmic description of the function applied by the action to both the states are the same.

Recall that $\mathcal{F}$ is the set of feature functions, i.e. functions of the form $f : F_\times \rightarrow F_\times$. Before proceeding further, we need to characterize this set as our description depends on it. Let the set of functions in $\mathcal{F}$ that alter only a single feature of its argument, and has a known algorithmic description, be $\mathcal{F}1$; i.e. if $f \in \mathcal{F}1$ and $f(a) = b$ then a and b differ in the value of only one feature. We call $\mathcal{F}1$ the set of *simple feature functions*. Now let $\mathcal{F}1_\mathcal{C}$ be the set of functions such that each $f \in \mathcal{F}1_\mathcal{C}$ is a composition of a finite sequence of members of $\mathcal{F}1$. Consider the set-of-pair representation of each $f \in \mathcal{F}$: $\{(x_i, y_i)\}$. We assert that each pair $(x_i, y_i) \in f$ can be described by a $f_c \in \mathcal{F}1_\mathcal{C}$:

Lemma 1. *For each $(x_i, y_i) \in f$ where $f \in \mathcal{F}$, there is a $f_c \in \mathcal{F}1_\mathcal{C}$ such that $f_c(x_i) = y_i$.*

Proof. Consider a particular pair $(F(s), F(s')) \in f$. Now, for each feature F_i, there is a $f_i \in \mathcal{F}$ such that, $\forall s_1 \in S$, $f_i(F(s_1)) = F(s_2)$ such that for all $j \neq i$, $F_j(s_2) = F_j(s_1)$ and $F_i(s_2) = F_i(s')$. In other words, f_i is the function in $\mathcal{F}$ that changes the i^{th} feature of any input to the value of the i^{th} feature in s' and keeps all the other features the same. Since f_i changes only one feature, and since it has an algorithmic representation (just described), $f_i \in \mathcal{F}1$. The composition of all such f_i for each $F_i \in F$ gives us the required function f_c and since $|F|$ is finite, the sequence is also finite. □

Given a particular problem domain let $\mathcal{F}_\mathcal{U} \subset \mathcal{F}$ be the set of functions that is actually used in the domain. This means that for each $f \in \mathcal{F}_\mathcal{U}$, $\exists a \in A, s \in S$, such that g_a, the action distribution function corresponding to a, generates a distribution with non-zero probability for f. Now for each $f_u \in \mathcal{F}_\mathcal{U}$ consider a set $J_{f_u} \subset \mathcal{F}1_\mathcal{C}$ such that 1) the members of J_{f_u} cover all the pairs of f_u and 2) the members of J_{f_u} are mutually disjoint. We can always find such a disjoint set of functions in $\mathcal{F}1_\mathcal{C}$:

Lemma 2. *For each $f \in \mathcal{F}$ (and thus for each $f_u \in \mathcal{F}_\mathcal{U}$ since $\mathcal{F}_\mathcal{U} \subset \mathcal{F}$), there is at least one subset J_f of $\mathcal{F}1_\mathcal{C}$ such that members of J_f covers all the pairs of f and the members of J_f are mutually disjoint.*

Proof. Consider the set of functions $f_c \in \mathcal{F}1_\mathcal{C}$ described in lemma 1. So each f_c covers all pairs of the form $(\cdot, y_i) \in f$ for some y_i. Let us call the function $f_c \in \mathcal{F}1_\mathcal{C}$ corresponding to y_i $f_{c_{y_i}}$. Now redefine $f_{c_{y_i}}(x)$ to be undefined whenever $(x, y_i) \notin f$. Consider the set $J_f = \{f_{c_{y_i}}\}$, where each member of J_f corresponds to one y_i. Now consider any pair $(a_j, b_j) \in f$ - since the function of the form $f_{c_{b_j}}(a_j) = b_j$ is in J_f by the above construction, members of J_f cover all pairs in f. In addition when $f_{c_{y_i}}, f_{c_{b_j}} \in J_f$, $f_{c_{y_i}}$ covers pairs of the form $(\cdot, y_i)$ and $f_{c_{b_j}}$ covers pairs of the form $(\cdot, b_j)$ and are thus disjoint. This proves the lemma. □

Since we have an algorithmic description for each $f_c \in \mathcal{F}1_\mathcal{C}$ (described above) and since each $f_u \in \mathcal{F}_\mathcal{U}$ is covered by some of these f_c, we have thus proven the intuitive idea mentioned above that each $f_u \in \mathcal{F}_\mathcal{U}$ has an algorithmic description.

Let $K(\lambda)$ denote the Kolmogorov Complexity of an algorithm λ. Now let $\{\Lambda_j\}$ be a family of set of algorithms with each $\Lambda_j = \{\lambda_i\}$ such that there is a bijective mapping between $\mathcal{F}1_{\mathcal{CU}} \rightarrow \Lambda_j$ and the λ_i corresponding to a f_u implements f_u. Let

$$\mathcal{A}_{\mathcal{U}} = \arg\min_{\Lambda_j} \left(\sum_{\lambda_i \in \Lambda_j} K(\lambda_i) \right) \tag{3.1}$$

i.e. $\mathcal{A}_{\mathcal{U}}$ is the set of algorithms from the family $\{\Lambda_j\}$ that has the lowest combined Kolmogorov Complexity and also implements the functions $f_u \in \mathcal{F}_{\mathcal{U}}$.

Example 3.1. In the simple domain with $S = \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, let $\mathcal{F}_{\mathcal{U}} = \{\{(1,1),(2,4),\cdots\}\}$. So the algorithmic description can simply be $f(x) = x^2$ or a search through the sequence. However, the former is the most compact description and so corresponds to $\mathcal{A}_{\mathcal{U}}$ for this domain, i.e. $\mathcal{A}_{\mathcal{U}} = \{f(x) = x^2\}$. Furthermore, consider another domain with the same state space but with $\mathcal{F}_{\mathcal{U}} = \{\{(1,1),(2,4),(3,9)\cdots\},\{(2,8),(3,27),\cdots\}\}$. Then $\mathcal{A}_{\mathcal{U}} = \{f(x) = x^2, f(x) = x^3 \text{ if } x \neq 1\}$. □

Now we assume we are given a function of the form $\Delta : F_{\times} \times F_{\times} \times A \rightarrow \mathcal{A}_{\mathcal{U}}$. So given two states in terms of their features and the action applied, the $\Delta(F(s), F(s'))$ outputs the algorithm from $\mathcal{A}_{\mathcal{U}}$ corresponding to the feature function used to generate s' from s. Given this we define similarity between two states as follows.

Let $a(s)$ be the random variable denoting the possible next states when action a is taken in state s. Therefore $\Delta_a(s, a(s))$ denotes the random variable corresponding to the possible algorithms that generates the states represented by $a(s)$. Let $L1(p(x), q(x))$ denote the $\mathcal{L}1$ norm between distributions $p(x)$ and $q(x)$ of the random variable x:

$$L1(p(x), q(x)) = \sum_x |p(x) - q(x)|$$

Let $L1[P(\Delta_a(s, a(s))), P(\Delta_a(s', a(s'))]$ denote the $\mathcal{L}1$ norm between distribution $P(\Delta_a(s, a(s)))$ of $\Delta_a(s, a(s))$ and distribution $P(\Delta_a(s', a(s')))$ of $\Delta_a(s', a(s'))$.

Definition 1. *(Functional Similarity) Then we say two states s and s' are ϵ similar with respect to action a if*

$$L1[P(\Delta_a(s, a(s))), P(\Delta_a(s', a(s')))] \leq \epsilon$$

where ϵ is some user defined constant.

Thus, in essence, the "change induced when action a is taken in state s" is the algorithm applied to s by a to get the next state. So two states are similar w.r.t. an action if the distribution of change induced is the same in both states. For the purposes of this paper, we may say the requirement that the algorithms have the lowest Kolmogorov Complexity (i.e. be a member of $\mathcal{A}_{\mathcal{U}}$) was used to get a canonical description of the algorithms, and to help reinforce the intuition that two are similar if the same 'procedure' is applied by the action in both states the states - e.g. PUSH-BLOCK increments/decrements the x or the y coordinate of the box. However, we will show in future work, when we extend the basic notion of functional similarity given here to address *similar actions* and *parameterized feature functions*, that this requirement is actually important one. The above two additions gives us a principled way to handle situations where feature functions we consider equivalent may not be the same. This is the case in the squash/racquetball example in the introduction - although the effect of racquet strokes (the feature functions) are similar they are not quite the same. Now next we need to consider how exactly we obtain the Δ function. We have the following options:

1. We can assume that we are given the right Δ function.
2. We can assume that we are given a set of possible algorithms (perhaps parameterized versions of the same algorithm) and based on experience we determine which algorithm most succinctly describes the changes being induced by the actions.

The second option is more useful, since, as the experimenter selects the state representations, he/she is likely to be able to define the space of feature functions. While it is an interesting learning problem, we are not however interested in this at this point. So, for the moment, we are going develop the rest of our method based on the first assumption. However, the remainder of the development in this paper will be independent of which option is used and we will address the second option in future extensions of this work. The above also raises the question of how do we identify which feature function was applied during a state transition when the feature functions overlap (i.e. $f_i \cap f_j \neq \emptyset$). The following lemma describes a simple way to deal with this problem and its implications. More effective solutions will be addressed in future work.

Lemma 3. *For a given action a and any $s \in S$, divide up $\mathcal{F}_\mathcal{U}$ into subsets $\{\mathcal{F}_\mathcal{U}(s)_i\}$ such that for each such set, $\bigcap_{f_j \in \mathcal{F}_\mathcal{U}(s)_i} f_j \neq \emptyset$ and $P(f_j|s,a) > 0$ for some $f_j \in \mathcal{F}_\mathcal{U}(s)_i$ and $|\mathcal{F}_\mathcal{U}(s)_i|$ is maximal. Then assign the probability of all elements of a $\mathcal{F}_\mathcal{U}(s)_i$ to one arbitrary element of that set; i.e. for some $f_{i'} \in \mathcal{F}_\mathcal{U}(s)_i$ set*

$$\begin{aligned} P'(f_{i'}|s,a) &= \sum_{f_k \in \mathcal{F}_\mathcal{U}(s)} P(f_k|s,a) \\ f_k \in \mathcal{F}_\mathcal{U}(s)_i, k \neq i', P'(f_k|s,a) &= 0 \end{aligned}$$

The following then holds true for all states s_p and s_t:

$$\sum_{f_j \in \mathcal{F}_\mathcal{U}} |P(f_j|s_p,a) - P(f_j|s_t,a)| \geq \sum_{f_j \in \mathcal{F}_\mathcal{U}} |P'(f_j|_k,a) - P'(f_j|s_t,a)| \tag{3.2}$$

with equality only when all the functions are non-overlapping. In addition, the reverse of (3.2) is not necessarily true. The above expressed in terms of $\mathcal{L}1$ norms translates to the following:

$$L1[P(f_j|s_p,a), P(f_j|s_t,a)] \geq L1[P'(f_j|_k,a), P'(f_j|s_t,a)] \tag{3.3}$$

Proof. First, the $\mathcal{F}_\mathcal{U}(s)_i$ are pairwise disjoint, for if any two $\mathcal{F}_\mathcal{U}(s)_j$ and $\mathcal{F}_\mathcal{U}(s)_k$ are not disjoint, we can combine them together to get a larger set, which implies the sets would have violated the maximal requirement on the size of the sets. So we can write

$$\begin{aligned} \sum_{f_j \in \mathcal{F}_\mathcal{U}} |P(f_j|s_p,a) - P(f_j|s_t,a)| &= \sum_{\mathcal{F}_\mathcal{U}(s)_i} \sum_{f_j \in \mathcal{F}_\mathcal{U}(s)_i} |P(f_j|s_p,a) - P(f_j|s_t,a)| \\ &\geq \sum_{\mathcal{F}_\mathcal{U}(s)_i} \left| \sum_{f_j \in \mathcal{F}_\mathcal{U}(s)_i} (P(f_j|s_p,a) - P(f_j|s_t,a)) \right| \\ &= \sum_{\mathcal{F}_\mathcal{U}(s)_i} |P'(f_{i'}|s_p,a) - P'(f_{i'}|s_t,a)| \\ &= \sum_{f_j \in \mathcal{F}_\mathcal{U}} |P'(f_j|s_p,a) - P'(f_j|s_t,a)| \end{aligned} \tag{3.4}$$

where the third and last line follows from the fact that in P' all the probabilities are assigned to a $f_{i'}$. Also note that we get equality in (3.4) only when each $\mathcal{F}_\mathcal{U}(s)_i$ consists of a single

function, because then terms on the left and right hand side are identical. The reverse is evidently false from the above derivation. □

Basically, the above lemma is saying if two states are ϵ similar, then they are also ϵ similar if we ignore overlap between feature functions. In addition it is also saying that the reverse is not true - that is if two states are ϵ similar after they overlap between feature functions is ignored, then the actual $\mathcal{L}1$ norm might be much higher. We discuss further implications of this in section 3.3.1 when we describe the predictive model. We will now illustrate the ideas presented so far with a concrete example:

Example 3.2. Consider a simple gridworld navigation domain with no obstacles (figure 2). In this domain, the task of the agent is to go from the start state to the goal state. There are four actions available to the agent - MOVE-NORTH, MOVE-SOUTH, MOVE-EAST and MOVE-WEST. Each action takes the agent in the appropriate direction with probability $2/3$, and any of the other direction with probability $1/9$. Of course after the direction is selected, the agent stays in the same position if the direction selected takes it outside of the world.

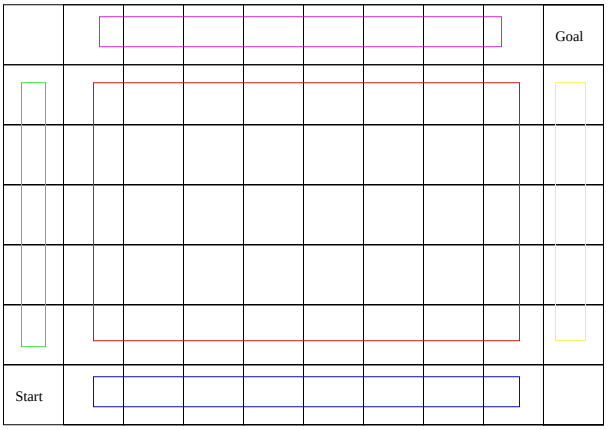

Figure 2: **Example Domain.**

Let the states be represented in terms of x, y coordinate of the agent. Thus in this domain, the $\mathcal{A}_{\mathcal{U}}$ is the set of algorithms of the form $+(1,0), +(-1,0), +(0,1), +(0,-1)$ and $+(0,0)$ - i.e. algorithms that either increments or decrements either the x or the y coordinate. So consider two states $(2,2)$ and $(7,20)$ - these two states are $\epsilon = 0$ similar with respect to any of the actions since the distribution of change induced is the same in both cases (figure 3 shows the case for action MOVE-NORTH). The important thing to note here is that although the states are completely dissimilar in terms of the distribution of next states, they are exactly the same in terms of the algorithm applied. In figure 2 the states that are $\epsilon = 0$ similar with respect to all the actions are covered by colored rectangles.

□

3.3 Using Functionally Similar States

As mentioned in the introduction, the idea is to use functionally similar states to develop a *predictive model* (PM) that will predict the effect of actions on states not visited before or often (a *novel state* - see section 3.3.2). Qualitatively the idea is to identify the observed states that are functionally similar with respect to some particular action. Then the agent trains a classifier to distinguish between these clusters of similar states. When a novel state is encountered, the classifier is used to determine which cluster the novel state belongs to.

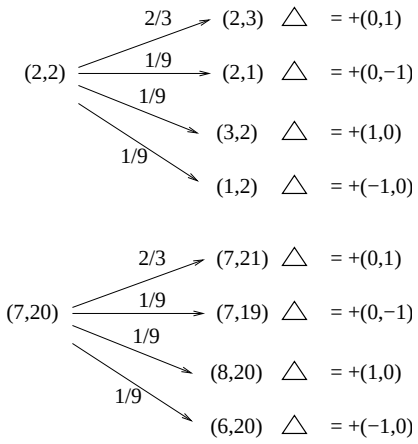

Figure 3: **Functional Similarity Example.**

Then the distribution of the states in that particular cluster is predicted as the distribution of the novel state. We describe this more precisely in the next sub-sections. As we stated earlier, the PM is in fact a type of forward model. We discuss the relationship between PM and other forward models at the end of this section. Section 4.4 describes how the predictive model is used in conjunction with the planning mechanism.

3.3.1 The Predictive Model

We informally described the predictive model in the introduction to this section. More formally, in the ideal case when we have all the probability distribution for each state, the predictive model will consist of for each action $a \in A$ a set $Sa = \{Sa_1, Sa_2, \cdots, Sa_m\}$ such that $\bigcup_i \{Sa_i\} = S$ and for all $Sa_i, Sa_j \in Sa$, $Sa_i \cap Sa_j = \emptyset$. Each subset Sa_i will have the property that the each state $s_k \in Sa_i$ is ϵ similar to all $s_j \in Sa_i$ for some ϵ. We describe how to determine the value ϵ which is "optimal" in a certain sense, and thus the optimal members of each cluster, below. In addition for each action there will be a classifier (Support Vector Machine, neural network etc.) Ca trained with Sa with each Sa_i corresponding to a particular class, so that given any state it will determine which cluster in Sa the state belongs to. Therefore during learning and behaving, the idea will be to estimate Sa and then learn the classifier Ca. Ca will be used to determine which Sa_i a novel state s_n belongs to. Then the prediction for distribution $P(a(s_n)|s_n, a)$ will be the distribution $P(a(s)|s, a)$ for some $s \in Sa_i$. The error for this prediction is, given the Ca classifies correctly, at most the ϵ parameter used to construct Sa_i.

Thus, the predictive model is based on the central assumption that the action distribution functions A_P assigns ϵ close distributions to states that are in some subset of S that is learnable by a classifier. That is, the function does not assign distributions arbitrarily - for $g_a \in A_P$ if $L : S \times S \rightarrow [0, 2]$ is defined by $L_a(s, s) = L1(g_a(s), g_a(s'))$, then L_a is in a sense a continuous mapping with respect to the product topology on $S \times S$ generated by the subsets of S learnable by a classifier and the usual topology on $[0, 2]$. We will explore the relationship between this continuous function and the PM in future work. We also note here that when feature functions overlap, and we ignore the overlap using the method described in lemma 3, then by lemma 3, we will simply consolidate together different classes of states.

As mentioned above, states in a Sa_i are ϵ similar to each other. To determine what the value of ϵ should be we need to consider exactly what use the predictive model will be put to. So let us assume that during a certain point in the agent's operation we have distributions of some set of states $\hat{S}$ with respect to action a. That is we know the distributions $P(a(s)|s, a)$, and thus $P(\Delta_a(s, a(s)))$ for each $s \in \hat{S}$. Then let $\hat{Sa} = \{\hat{Sa}_1, \hat{Sa}_2, \cdots, \hat{Sa}_m\}$ denote the

current set of clusters constructed from $\hat{S}$. Also assume that we have constructed our clusters using some method to select ϵ. Now let us assume that we have observed a novel state s_n and would like to predict the distribution $P(a(s_n)|s_n, a)$ using the predictive model. Now let $\hat{Sa}_i$ be the cluster that the PM, i.e. the classifier C_a, *correctly* determines as s_n belongs to. Therefore we select some state $s_k \in \hat{Sa}_i$ and we wish to use $P(\Delta_a(s_k, a(s_k)))$ to approximate $P(\Delta_a(s_n, a(s_n)))$ and thus $P(a(s_n)|s_n, a)$. Also let us assume the reason we wish to perform the approximation is that eventually we wish to compute a function h of $P(a(s_n)|s_n, a)$. So we choose ϵ_{Sa_i} for a particular cluster $\hat{Sa}_i$ so that the

$$|h(;, P(\Delta_a(s_n, a(s_n))) - h(;, P(\Delta_a(s_k, a(s_k)))| < \eta \tag{3.5}$$

where η some user defined (application dependent) constant and $h(;,)$, in this particular instance, denotes that the other arguments that are independent of $(\Delta_a(s_k, a(s_k)))$. In the rest of these sections we do not show the $h(;,$ part of the notation. Ideally, of course, we would like to choose ϵ_{Sa_i} that minimizes the left hand side of the above equation. We address this issue in section 3.3.3. A trivial example of an h function is $h(x) = L1(x, y)$ where $L1$ is the $\mathcal{L}1$ norm function, y is some fixed distribution and x is the given distribution. We discuss a non-trivial h function (and hence the κ function described below, and the selection of the associated η constant) in section 4.4.

Now since we are choosing s_k arbitrarily, $P(\Delta_a(s_k, a(s_k)))$ might be any distribution with $L1(P(\Delta_a(s_n, a(s_n))), P(\Delta_a(s_k, a(s_k)))) < \epsilon_{Sa_i}$. Therefore we can rewrite the above as

$$|h(P(\Delta_a(s_n, a(s_n))) - h(P(\Delta_a(s_k, a(s_k)))| = \kappa(\epsilon_{Sa_i}) < \eta \tag{3.6}$$

Note that here we are assuming that the Ca classifies correctly. Therefore, denoting the error in classifier Ca for members of $\hat{Sa}_i$ by $P(e|Ca, \hat{Sa}_i)$, we need to select ϵ_{Sa_i} satisfying:

$$(1 - P(e|Ca, \hat{Sa}_i))\kappa(\epsilon_{Sa_i}) + P(e|Ca, \hat{Sa}_i))\kappa(2) < \eta \tag{3.7}$$

where the $\kappa(2)$ term follows from the fact that $\mathcal{L}1$ norm is bounded by 2. Now when adding a state s to our predictive model (i.e. to one of the clusters) we add it to any cluster $\hat{Sa}_j$ satisfying (3.7) after the addition of s_k to $\hat{Sa}_j$. We summarize the results on constructing the predictive model that were obtained in this section in the following lemma:

Lemma 4. *Let the current estimations of clusters of the PM for an action a be $\hat{Sa} = \{\hat{Sa}_1, \hat{Sa}_2, \cdots, \hat{Sa}_m\}$. Let h be the function of $P(a(s)|s, a)$ we wish to approximate using the PM. To ensure that the error in the approximation is $< \eta$, the ϵ_{Sa_i} for each $\hat{Sa}_i \in \hat{S}$ should be set so that it satisfies (3.7) where κ is defined in (3.5) and (3.6). Indeed, when adding a state s_k to the PM, the state should be added only to a cluster $\hat{Sa}_j$ such that $\hat{Sa}_j \cup \{s_k\}$ satisfies (3.7). If no such cluster exists, a new cluster should be created with s_k as the only member.*

3.3.2 Confidence in Predictive Model

Often it may be necessary to determine what the confidence in a predictive model is at some point in time. This section describes how one might compute the confidence. To that end, for each cluster $\hat{Sa}_i$, we also maintain a confidence parameter $C(\hat{Sa}_i)$ which is a measure of the confidence we have in the distributions of members of s - i.e. it is product of the confidences of $P(a(s_j)|s_j, a)$ for each $s_j \in \hat{Sa}_i$. The confidence in each s is estimated using Theorem 12.1.4 in [Cover and Thomas, 1991]. The theorem uses the method of types to bound the probability of observing a particular sequence of values of a random variable given its actual distribution in terms of the KL divergence between the *type* of the observation and the type of the actual distribution. We describe the theorem and its application in our problem below.

Let X be a discrete random variable with N possible values, $\{X_1, X_2, \cdots, X_N\}$ and distribution P_X. Let $T = \{x_1, x_2 \cdots x_n\}$ be a sequence of values of X drawn according to P_X. Then the *type* or *empirical probability distribution* of the sequence T is defined as the proportion of times each symbol X_i appears in T. So for example, if X has possible values $\{a, b, c\}$, and $T = \{a, a, b, a, b, c\}$, then the type of T is $\{\frac{3}{6}, \frac{2}{6}, \frac{1}{6}\}$. The set of all sequences with the same type as T is called its type class. For example, the sequences $\{a, a, b, a, b, c\}$ and $T = \{a, c, a, a, b, b\}$ are in the same type class.

Now let P_T denote the distribution corresponding to the proportion of times each X_i appears in T. Then the aforementioned theorem states that if the actual distribution of a r.v. is P_X, then the probability of a sequence of type T under P_X, denoted by $P_X(T)$, is

$$P_X(T) \leq 2^{-nD(P_X||P_T)} \tag{3.8}$$

where $D(P_X||P_T)$ is the KL divergence between P_T and P_X and n is the length of the sequence T. In other words this says that the probability of a sequence T with KL divergence between P_T and P_X greater than, say, α is less than $2^{-n\alpha}$. This in turn implies probability of a sequence T' with $D(P_X||P_{T'})) < \alpha$ is

$$P_X(T') > 1 - 2^{-n\alpha} \tag{3.9}$$

The most interesting aspect of this equation is that the probability that the observed distribution differs significantly from the actual distribution exponentially with n. In the following we use (3.9) to estimate the confidence in the observed distribution of $P(a(s)|s, a)$.

Now note that a particular sequence of outcomes of taking action a in state s corresponds to a type. Using this type, we will attempt to estimate the probability that the KL divergence between the true distribution $P(a(s)|s, a)$ and the type $\hat{P}(a(s)|s, a)$ is less than α where α is some arbitrarily selected KL divergence that ensures that the distribution is not too far off. So given a particular sequence of observation, we use the following as confidence in a distribution:

$$C(a, s)_n = 1 - 2^{-n\alpha} \tag{3.10}$$

where n is the number of observations with a, s made so far and $C(a, s)$ is the confidence in the distribution of a, s. Note this simple approximation is dependent only on the number of observations made. To obtain the confidence in a particular cluster $\hat{Sa}_i$ is the product

$$C(\hat{Sa}_i) = \prod_{s \in \hat{Sa}_i} C(a, s)_{n_s} \tag{3.11}$$

n_s is the number of observations with a, s made so far. This also gives us a way to define a *novel* state.

Definition 2. *(Novel State) We call a state s novel if $C(a, s)_n < C(\hat{Sa}_j)$ where $\hat{Sa}_j$ is the cluster that the classifier Ca determines s as belonging to.*

Now given $C(\hat{Sa}_i)$, we can incorporate this in condition (3.7) thus:

$$C(\hat{Sa}_i)(1-P(e|Ca, \hat{Sa}_i))\kappa(\epsilon_{Sa_i}) + \Big(1 - C(\hat{Sa}_i)(1 - P(e|Ca, \hat{Sa}_i))\Big)\,\kappa(2) < \eta \tag{3.12}$$

where the first term on the left hand side deals with the case that PM model is correct, and the second term when the PM is incorrect.

3.3.3 Minimizing h

In this section we discuss the problems encountered when we try to minimize the error in approximating h function instead of ensuring the error is below some user-defined constant as in lemma 4. Recall that the problem is to choose ϵ_{Sa_h} for each cluster $\hat{Sa}_i \in \hat{Sa}$ such that the error

$$|h(P(\Delta_a(s_n, a(s_n))) - h(P(\Delta_a(s_k, a(s_k)))| \tag{3.13}$$

in approximating a function h of $P(a(s)|s, a)$ is minimized. Here $s_k \in \hat{Sa}_i$ where $\hat{Sa}_i$ is the cluster that the classifier Ca correctly identifies as the novel state s_n belonging to. In order to minimize the error we should obviously minimize ϵ_{Sa_i} since this minimizes $L1(P(\Delta_a(s_n, a(s_n))), P(\Delta_a(s_k, a(s_k))))$ the distance between the two real distribution and the approximating distribution and hence (3.13). However at the same time, reducing ϵ_{Sa_i} has the potential of decreasing *coverage* of each cluster - i.e. the proportion of novel states that the cluster will be able to predict correctly for in the future - because the greater the ϵ_{Sa_i}, the greater the number of states that will likely belong to $\hat{Sa}_i$. This is defined more precisely below.

From this point we proceed by constructing a formal description of the error in terms of coverage and classification accuracy. It is obvious how classification error may be described formally and quantified, so we focus on coverage. For each $\hat{Sa}_i \in \hat{Sa}$, we define the *coverage* of the $\hat{Sa}_i$ to be the probability that a state seen by the agent in future will belong to $\hat{Sa}_i$. This is denoted by $P(\hat{Sa}_i)$ and is given by:

$$P(\hat{Sa}_i) = \sum_{s \notin \hat{S}} P(s, s \in \hat{Sa}_i)$$

where $P(s, s \in \hat{Sa}_i)$ denotes the probability of seeing the heretofore unobserved state s, and s belonging to $\hat{Sa}_i$. We may define the expected error $Err(PM)$ of the Predictive Model with respect to h as follows:

$$\begin{aligned} Err(PM) = \sum_i P(\hat{Sa}_i)\beta_i \quad + \\ (1 - \sum_i P(\hat{Sa}_i))[P(e|Ca)\min(maxCost, \kappa(2)) \quad + \quad (1 - P(e|Ca))\kappa(2)] \end{aligned} \tag{3.14}$$

where β_i refers to the left hand side of 3.12, and $P(e|Ca)$ the overall error of the classifier. Equation (3.14) can be understood as follows. To compute the expected error, we need to consider the errors that may arise when trying to determine which cluster a particular novel state s_n belongs to. The first term on the right corresponds to the case when s_n belongs to some $\hat{Sa}_i$ and the β_i term gives the error in that instance. The second term on the right corresponds to the case when a novel state does not belong to any of the clusters. The first part of this term corresponds to the case when the classifier is able to identify the novel state as not belonging to any of the existing $\hat{Sa}_i$s, and the error is minimum of $maxCost$ (the cost incurred when no prediction regarding h is made) and the $\kappa(2)$. This is because if the cost of making no prediction is different than making a wrong prediction, then we choose whichever is the minimum. The second part in the second term of the right hand side corresponds to the case when the classifier is incorrect the error is $\kappa(2)$ as in the β_i term.

Although we can derive this expression for expected error, it is not possible for us to estimate $P(\hat{Sa}_i)$, the coverage, because the states are not drawn according to this distribution; but rather some distribution which is a function of the current policy of the agent. As before we summarize the results of this section in the following lemma.

Lemma 5. *Let the current estimations of clusters of the PM for an action a be $\hat{Sa} = \{\hat{Sa}_1, \hat{Sa}_2, \cdots, \hat{Sa}_m\}$. Let h be the function of $P(a(s)|s,a)$ we wish to approximate using the PM. Then if some prior knowledge is available which helps the agent accurately estimate the probability of coverage $P(s \in \hat{Sa}_i)$ as defined in this section, then (3.14) may be used to construct clusters that minimize error in approximating h. Indeed, when adding a state s_k to the PM, the state should be added only to a cluster $\hat{Sa}_j$ such that $\hat{Sa}_j \cup \{s_k\}$ minimizes (3.14). If no such cluster exists, a new cluster should be created with s_k as the only member.*

3.3.4 PM and Other Forward Models

As we discussed in the section 1, the predictive model is in fact a kind of forward model (e.g. [Jordan and Rumelhart, 1992], [Mitchell and Thrun, 1993], [Kawato, 1999], [Karniel, 2002] and [Flanagan and Wing, 1997]) - i.e. a model of the world that predicts the effect of actions in states that have not been seen before. In this section we discuss the differences between the older models referred to above and our model. In the following we refer to the older models as FMs and our model as the PM.

The fundamental conceptual difference between our model and the older models is important, since the PM is constructed in a way that agrees more with the aim of forward models. The way the FMs construct a forward model is by training a classifier on the observed state transitions. i.e. if a state transition $s \rightarrow^a \rightarrow s'$ occurs then the classifier is trained with s as the input and s' as the output (sometimes an encoding of the action is included in the input). The classifier is then used to predict the next states for novel states. Thus, there is an underlying assumption that, states close to each other in feature space (physically similar) should have next states that are also close to each other in feature space. The PM does not make any such assumption, instead it attempts to identify commonality in terms of features space among states that behaves in a similar way, and then uses that identified commonality to predict behavior of new states.

So, the main conceptual difference between FMs and the PM is that FMs assumes that physically similar states are functionally similar, while the PM actually tries to identify physical similarity between functionally similar states.

Obviously the PM is able to handle cases where the assumption made by FMs holds true and also cases where it does not i.e. physically disparate states are functionally similar and physically close states are functionally disparate. Therefore the PM is more accurate. The behavior of PM is more in line with the purpose of forward models - which is to find correspondence between *state transitions* and states. We discuss this in more concrete terms below.

As mentioned above the FMs learn a mapping from states to next states - so basically, in the FMs the mapping being learned for predictions for a particular action is some type of combination of all the feature functions $\mathcal{F}$ that the observed state transitions correspond to (we described and characterized $\mathcal{F}$ in sections 2 and 3.2 respectively). Therefore, one difference between previous models is that instead of learning one monolithic function (which is a strange combination of $f \in \mathcal{F}$) to cover all observed state-state mappings, we assume that we have some mechanism to derive the individual $f \in \mathcal{F}$ being used. As we mentioned in section 3.2 this is a reasonable assumption to make. Now, if we consider deterministic domains, then each of the clusters Sa in the PM contain states which are applied the same $f \in \mathcal{F}$ by a. Therefore, the PM uses the feature functions at a much finer grain than FMs and hence performs better. Also, as mentioned in the previous paragraph, in the FMs, the feature function applied to a novel state s_n to get the next state s'_n is determined by whichever previously observed state s s_n is closest to. The PM on the other hand determines the feature function to apply to s_n by determining which previously observed states s_n is most likely to be functionally similar to. The PM thus, does not make the assumption made by the FMs but is also able to handle cases when the assumption does hold true and is more accurate.

In addition, it is difficult for the FMs to predict a distribution over next states since these compute a direct state-state mapping. On the other hand, the PM can predict a distribution over next states quite easily. The PM can also ensure that the prediction error is below some threshold - which the FMs cannot do. Finally, the mapping states $\rightarrow \{1, -1\}$ learned by the classifiers Cas is a much simpler mapping than state-state mappings learned by FMs. The Ca classifiers are thus more accurate. In other words the PM solves a classification problem which is much simpler than the regression problem that the FMs solve.

4 Planning Mechanism for the CL System

As mentioned in the introduction, the planning mechanism for our CL system operates on goal directed MDPs, which were described in section 2. However the mechanism itself is not the typical DP/Reinforcement Learning Mechanism used for this type of MDP (as in [Barto *et al.*, 1995]). Instead, it uses direct graph search using Dijsktra's algorithm and we show that in the limiting case the method produces plans/policies that are same as those in [Barto *et al.*, 1995]. We also outline how in the non-limiting case, it is actually better in all but non-pathological cases and not demonstrably worse in pathological cases. We use this method because this allows straightforward integration with the predictive model described in the previous section, and we believe it will be useful in many interesting domains. The method as presented in this section is independent of the PM. We demonstrate in section 4.4 how it may be used more effectively in conjunction with the predictive model described above for problems with multiple goal states. Since our planning mechanism is somewhat different than typical RL methods, we refer to our value function by ES (standing for expected number of steps). We discuss the differences between our method and the typical RL method for goal directed MDPs in section 4.5.

4.1 Expected Number of Steps and Plans

First, we call a state s' *reachable* from state s by action a if $P(s'|s,a) > 0)$. We define the discounted expected number of steps required to get from state s to s_g when action a is taken as follows:

$$ES_a(s, s_g) = \sum_{s' \in S} P(s'|s,a) \left(d_{s,a,s'} + \gamma \min_{a' \in A} ES_{a'}(s', s_g) \right) \tag{4.1}$$

where $d_{s,a,s'}$ is the cost associated with the transition $s \rightarrow^a \rightarrow s'$ and $0 \leq \gamma < 1$ is the discount factor. Note that (4.1) is very similar in spirit to the familiar Bellman equations in dynamic programming. It is simply saying that the expected number of steps required to reach state s_g after taking action a from state s is the expected distance from each state s' reachable from state s on after performing action a weighted by probability $P(s'|s,a)$ of reaching the state s'. Also note that since the $d_{s,a,s'}$ are bounded and $0 \leq \gamma < 1$, the ES values are also bounded - specifically by $M/(1-\gamma)$ where $M = \sup_{s,a,s'} d_{s,a,s'} < \infty$, since ES becomes a geometric sum with first term M and rate γ in the worst case. The following theorem (proof in appendix A) shows how we may compute ES in the ideal case.

Theorem 1. *Given the correct values of $d_{s,a,s'}$ and the state transition distributions, the update rule,*

$$\hat{ES}_a(s, s_g)_n = \sum_{s' \in S} P(s'|s,a) \left(d_{s,a,s'} + \gamma \min_{a' \in A} \hat{ES}_{a'}(s', s)_{n-1} \right) \tag{4.2}$$

,where $\hat{ES}$ denotes the estimate of ES, and $\hat{ES}_a(s,s')_n$ denotes the value of $\hat{ES}_a(s,s')$ after the n^{th} update converges to the actual $ES_a(s, s_g), \forall a, s, s_g$ as $n \rightarrow \infty$ and given also that $ES_a(s,s')$ for each triple (s,a,s') is also updated infinitely often.

A plan is defined as a sequence of state action pairs, written for the sake of convenience without the parenthesis demarcating each pair: $\phi = (s_1, a_1, s_2, a_2 \ldots s_n, a_n, s_g, a_\emptyset)$ where $a_\emptyset$ denotes an action that does not have any effect on the domain. So when the agent follows the plan, it performs action a_i in state s_i in the plan and expects to reach s_{i+1} in the plan. A plan becomes invalid if at any point during executing the plan the observed next state s_{i+1} does not agree with the next state given in the plan. Let Φ be the set of all plans; then we define a plan $\phi^* = (s_1, a_1, s_2, a_2 \ldots s_n, a_n, s_g, a_\emptyset)$ to be optimal if

$$\phi^* = \arg\min_{\phi \in \Phi} ES_\phi(s_1, s_g) \tag{4.3}$$

where

$$ES_\phi(s_1, s_g) = \sum_{i=1}^{n} ES_{a_i}(s_i, s_{i+1}) \tag{4.4}$$

So this is a simple extension of ideas in classical planning to our probabilistic domain where we use ES instead of deterministic costs of actions to create our plan. In the next section (specifically through corollary 1) we show that if $a_{min} = \arg\min_{a \in A} ES_a(s_1, s_g)$ then $\phi^* = \{s_1, a_{min}, s_g\}$ when we have all the information - i.e. we know $ES_a(s_i, s_j)$ for all state action triples (s_i, a, s_j). This is a very satisfactory result because this shows that when we have all the information, our notion of an optimal plan reduces to what we would normally consider to be the optimal action.

4.2 Planning with ES

In this section we describe how we may plan using the concept of ES introduced in the previous section. In the following first we describe how the agent will plan in case all the necessary information (i.e. the ES values) are available - i.e. the ideal case. And then we discuss how the agent may plan when it has to learn all the necessary information, which is what is required in a practical setting.

4.2.1 Planning in the Ideal Case

For the moment, let us assume we have the $ES_a(s_x, s_y)$ values for all state-action-state triples. Then the algorithm to compute the optimal plan from state s_1 to state s_g is simply this: apply Dijkstra's shortest path algorithm to find the shortest path between the nodes corresponding to s_1 and s_g on a graph constructed as follows:

Construct Graph

1. Form nodes with the states
2. Form directed edges between each pair of states.
3. For each directed edge (s_x, s_y) going from node/state s_x to s_y, assign cost of $\min_{a \in A} ES_a(s_x, s_y)$ (or ∞ if no such action exists).

Algorithm 4.1: Constructing the Graph in the Ideal Case Planning

---

This algorithm finds the shortest path with the ES_a used as the cost on the edges. This is exactly the optimal plan as defined is (4.3) and thus this algorithm finds the optimal plan as described in (4.3). We now give a theorem (proof in appendix A) the corollary to which gives us the relationship between the optimal plan and the optimal action:

Theorem 2. *The following inequality holds for all states s, s_{mid} and s_{goal},*

$$\min_{a_1 \in A} ES_{a_1}(s, s_{goal}) \leq \min_{a_2 \in A} ES_{a_2}(s, s_{mid}) + \min_{a_3 \in A} ES_{a_3}(s_{mid}, s_{goal}) \tag{4.5}$$

So basically the theorem is saying that the minimum cost of going from state s to another state s_{goal} is less than or equal to the minimum cost of going from state s to some other state s_{mid} plus the minimum cost of going from state s_{mid} to state s_{goal}. It is fairly easy to show this is true for deterministic actions - but a little bit more involved to show this for probabilistic actions. The following corollary shows that in the case when all the ES values are known, then for a particular goal state, the planning mechanism outputs the plan $\phi = \{s, a, s_g\}$ where s is the current state, a is the optimal action, and s_g the goal state - i.e. the plan is one that would be output by a typical Reinforcement Learning algorithm.

Corollary 1. *The algorithm for computing the optimal plan described in this section (section 4.2.1), given the value of $ES_a(s_x, s_y)$ for all state-action-state triple (s_x, a, s_y), will output the plan $\phi = \{s, a, s_g\}$ where s is the current state, s_g is the goal state and a is the optimal action:*

$$a = \arg\min_{a \in A} ES_a(s, s_g)$$

Proof. This is because now the algorithm will know the $\min_a ES_a(s, s_g)$ for all states $s \in S$ and for any particular goal state s_g. And thus from theorem 2, in the graph constructed, the edge corresponding to the action with minimum cost will be the edge output by Dijsktra's algorithm. □

4.2.2 Planning while Learning using ES

In this section we show how an agent may learn ES values and plan while acting or behaving in the domain (that is online learning and planning). First we present Robins-Monroe version of equation (4.2) which can be used to learn ES values while experiencing the world. Assume that the agent observes that the it reaches state s_y after taking action a in state s_x. Then the update rules are:

$$\hat{ES}_a(s_x, s_y)_k = \hat{ES}_a(s_x, s_y)_{k-1} + \alpha(d_{s_x,a,s_y} - \gamma \hat{ES}_a(s_x, s_y)_{k-1}) \tag{4.6}$$

and $\forall s_p \in S, s_p \neq s_y$

$$\begin{aligned} \hat{ES}_a(s_x, s_p)_k \quad = \quad & \hat{ES}_a(s_x, s_p)_{k-1} + \\ & \alpha(\hat{ES}_a(s_x, s_p)_{k-1} - d_{s_x,a,s_p} - \gamma \min_{a' \in A} \hat{ES}_{a'}(s_y, s_p)_{k-1}) \end{aligned} \tag{4.7}$$

with $0 \leq \alpha < 1$ (4.6) handles the case when $s' = s_g$ in the left hand side of (4.2), while (4.7) handles the case when $s' \neq s_g$ in the left hand side of (4.2). The validity of the equations is based on the validity of (4.2). Now we can give our algorithm for learning and planning while behaving:

ES Agent

1. For each goal state s_g, perform **Get To State**.

Get To State

1. While current state is not s_g
   (a) If current plan is not valid, perform **Compute Plan** to get plan ϕ_{cur} to go from the current state to the goal state s_g.
   (b) Perform action according to plan ϕ_{cur}.
   (c) Perform **Update Model**.
   (d) Check validity of current plan by checking whether next state matches next state given in ϕ_{cur}.

Compute Plan

1. Create graph using **Construct Graph** (section 4.2.1).
2. Apply Dijsktra's algorithm to get plan.

Update Model

1. Assume state-action-state triple observed was (s, a, s').
2. Update ES values according to (4.6) and (4.7).

Algorithm 4.2: Algorithms for Planning in the Practical Setting

---

We end this section with some final comments on what happens while this algorithm plans using imperfect information as in this section. First some definitions to make the presentation clearer. As mentioned above, we define a state s' to be *reachable* from state s, if there is an action $a \in A$ such that $P(s'|s, a) > 0$. The set of reachable states from s via a is denoted by $R(s, a)$. We use the term *pairwise cost* of a state s to refer the set of the costs $ES_a(s, s')$ such that $s' \in R(s, a)$.

Therefore, during the initial stages of the planning algorithm when the ES values have not been propagated back, the agent plans using only pairwise costs. Now obviously, if the actions are not probabilistic, then using only pairwise costs results in a plan that is optimal, as in the ideal case - i.e. $\arg\min_{a \in A} ES_a(s_1, s_g) = a_1$, where a_1 is the very first action in the plan, s_1 is the start state and s_g is the goal state (this is just classical planning). And intuitively, the error increases the "more probabilistic" the domain becomes. Or in other words given two states $s_i, s_j \in R(s, a)$ for some s, a, the error increases the more s_i and s_j differ in terms of distance to other states and each other - the domain becomes more "messy". We will quantify and state this intuition precisely in our future work.

4.3 Random Valued Action Cost

As mentioned in section 2, we have so far dealt only with constant valued action cost signals $d_{s,a,s'}$. In this section we describe how we may handle a randomly varying cost signal. As a first step we define ES as follows:

$$ES_a(s, s_g) = \sum_{s' \in S} P(s'|s, a) \left(E(d_{s,a,s'}) + \gamma \min_{a' \in A} ES_{a'}(s', s_g) \right) \tag{4.8}$$

where $E(d_{s,a,s'})$ is the expected value of $d_{s,a,s'}$. Thus all our discussion and theorems so far remain unchanged. The only thing we need to derive is the Robins Monro version of the above, which is given below:

$$\hat{ES}_a(s_x, s_y)_k = \hat{ES}_a(s_x, s_y)_{k-1} + \alpha(d^k_{s_x,a,s_y} - \gamma \hat{ES}_a(s_x, s_y)_{k-1}) \tag{4.9}$$

where $d^k_{s_x,a,s_y}$ is the observed cost at the k^{th} step, and $\forall s_p \in S, s_p \neq s_y$

$$\begin{aligned} \hat{ES}_a(s_x, s_p)_k &= \hat{ES}_a(s_x, s_p)_{k-1} + \\ &\quad \alpha[\hat{ES}_a(s_x, s_p)_{k-1} - d^k_{s_x,a,s_p} - \gamma \min_{a' \in A} \hat{ES}_{a'}(s_y, s_p)_{k-1}] \end{aligned} \tag{4.10}$$

The reasons for the derivation are same as those given in section 4.2.2.

4.4 Planning with ES using the Predictive Model

In this section we show how the planning mechanism may be used in conjunction with the Predictive Model described in section 3.3.1. The basic idea is that the agent learns the predictive model that predicts what the distribution over next state is going to be when some particular action is applied at a state. This information can then be used to predict ES values for these novel states. The ES method can then use these predicted ES values to form plans that contain the novel states. Thus the agent is able to behave more intelligently in the novel states , and in some cases able to plan to get to states it has never seen before. However, in order to do so, the system also needs to be able to compute what the cost signal $d_{s,a,s'}$ for novel states are . In the following, first we assume that all the $d_{s,a,s'} = 1$. Later we demonstrate possible ways we may compute the actual values and use them in constructing the h function for the predictive model.

Therefore, this combination of PM and ES is useful in multiple goal problems when solution paths to the different goal states contain similar states. So let us consider a problem where the solution paths to some goal state s_{g1} contain states similar to states in solution paths to goal state s_{g2}. If the agent has already solved the task s_{g1}, and then when it tries to solve task s_{g2}, it will be able to plan or behave intelligently in states in solution paths to s_{g2} that are similar to states in solution paths to s_{g1}. This is illustrated more clearly in section 5. Hence, the similarity between tasks, and the benefit of using the PM, increases monotonically with the number of similar states shared by the solutions to the different tasks. This measure of similarity between tasks will be explored more fully in the future.

4.4.1 Planning with $d_{s,a,s'} = 1$

Let us assume that after having observed states $\hat{S}$, we have constructed a predictive model consisting of the clusters $\hat{Sa} = \{\hat{Sa}_1, \hat{Sa}_2, \cdots, \hat{Sa}_m\}$ and the corresponding classifier Ca for each action a. The basic idea is that we would like to use (4.1)

$$ES_a(s_n, s_g) = \sum_{s' \in S} P(s'|s_n, a) \left(d_{s_n,a,s'} + \gamma \min_{a' \in A} ES_{a'}(s', s_g) \right) \tag{4.11}$$

to compute the estimate $\hat{ES}_a(s_n, s_g)$ of $ES_a(s_n, s_g)$ for some state s_g, but use $P(\Delta_a(s_j, a(s_j)))$ instead of $P(\Delta_a(s_n, a(s_n)))$ for some $s_j \in \hat{Sa}_i$ where $\hat{Sa}_i$ is the cluster that Ca determines s_j as belonging to. Using $P(\Delta_a(s_j, a(s_j)))$ in place of $P(\Delta_a(s_n, a(s_n)))$ results in using an estimated $\hat{P}(a(s_n)|s_n, a)$:

$$\hat{ES}_a(s_n, s_g) = \sum_{s' \in S} \hat{P}(s'|s_j, a) \left(d_{s_n,a,s'} + \gamma \min_{a' \in A} ES_{a'}(s', s_g) \right)$$

We can rewrite the above in terms of Δ as follows:

$$\hat{ES}_a(s_n, s_g) = \sum_{s' \in S} \hat{P}(\Delta_a(s, s')) \left(d_{s_n,a,s'} + \gamma \min_{a' \in A} ES_{a'}(s', s_g) \right) \tag{4.12}$$

Therefore, the error in doing this approximation is :

$$\begin{aligned} |\hat{ES}_a(s_n, s_g) - ES_a(s_n, s_g)| &\leq \gamma \sum_{s' \in S} |\hat{P}(s'|s_n, a) - P(s'|s_n, a)| \min_{a' \in A} ES_{a'}(s', s_g) \\ &= \gamma \sum_{s' \in S} |P(\Delta_a(s_j, s') - P(\Delta_a(s_n, s'))| \min_{a' \in A} ES_{a'}(s', s_g) \\ &\leq \gamma \max_{s' \in R(s,a)} \min_{a' \in A} ES_{a'}(s', s_g) \sum_{s' \in S} |P(\Delta_a(s_j, s') - P(\Delta_a(s_n, s'))| \\ &\leq \gamma \epsilon_{Sa_j} \max_{s' \in R(s,a)} \min_{a' \in A} ES_{a'}(s', s_g) \end{aligned} \tag{4.13}$$

where $R(s, a)$ refers to states reachable from state s after performing action a and ϵ_{Sa_j} is the maximum $\mathcal{L}1$ norm amongst the distributions $P(a(s_j)|s_j, a)$ for $s_j \in \hat{Sa}_j$. The final step follows from the assumption that $s_n \in \hat{Sa}_j$. Therefore we can write,

$$\begin{aligned} |ES_a(s_n, s_g) - \hat{ES}_a(s_n, s_g)| &\leq \gamma \epsilon_{Sa_j} \max_{s,s' \in S} \min_{a \in A} ES_a(s, s') \\ &= \epsilon_{Sa_j} k \end{aligned} \tag{4.14}$$

where the second line is simply a rewrite of the first. At this point we note that $\hat{ES}_a(s, s')$ is a function of $P(a(s)|s, a)$ i.e. a h function and $\epsilon_{Sa_j} k$ is in fact a upper bound on the corresponding κ function from section 3.3.1. We can thus set the η parameter from (3.12) to $\min_{s,a,s'} d_{s,a,s'}$ (when all the $d_{s,a,s'} \neq 1$) ensuring the expected error is less than the minimum cost of a particular transition. Any other value may also be chosen depending on the application. Furthermore, the reader may notice that, when computing the error, we assumed that the correct values for the $\min_{a \in A} ES_a(s, s')$ terms in the right hand side of (4.13) were known, or that the current estimates are acceptable (whether obtained from the OM or the PM). If we wish to take into account error from estimating the ES values using the PM, we need to recompute the error in (4.14). This computation is given in appendix A.

We can now give our algorithm for planning using the predictive model that 1) learns the predictive model and 2) uses the predictive model to plan while behaving or solving tasks. The basic outline of the algorithm follows the algorithm given in section 4.2.2. The difference is that, at each step when updating the observed model in **Update Model**, the predictive model is also updated (this addresses point 1 above). And, in addition, in **Compute Plan**, when constructing the graph, the predictive model is used to add edges to the graph for nodes corresponding to novel states (and this addresses point 2 above). In the following we describe each of the two necessary modifications.

Update Model

1. Assume state-action-state triple observed was (s, a, s').
2. Update ES values according to (4.6) and (4.7).

3. **Update Predictive Model**

Update Predictive Model

1. Update confidence of (s, a) according to (3.10).
2. Determine a cluster in $\hat{Sa}_i \in \hat{Sa}$ that satisfies (3.12), using (4.14) as the κ function.
3. If no such cluster exists, create a new cluster with s as the only member.
4. Else add s to the $\hat{Sa}_i$ found.
5. Update confidence of the cluster s was added to by multiplying $C(s, a)$ to $C(\hat{Sa}_i)$ or just setting $C(\hat{Sa}_k = \{s\}) = C(s, a)$ as appropriate.
6. If s was already in any one of the other clusters $\hat{Sa}_{old}$, remove s from that cluster; update confidence of $\hat{Sa}_{old}$ by dividing $C(\hat{Sa}_{old})$ by previous value of $C(s, a)$.
7. Retrain Ca.

Algorithm 4.3: Updating the Models when using the PM

In step 2 in **Update Predictive Model**, in the beginning we may not know the $\max\min$ term in k in (4.14). So to fix this we will need to start off with an estimate of the term, and then update it based on observation. As the estimate changes, we may need to rebuild our predictive model periodically. During this, the old clusters are removed from the PM and new clusters are created by adding observed states to the cluster using the usual method using the currently known state transition distributions. In addition, the $\max\min$ term might become extremely high, resulting in situations where the only viable choice for ϵ_{Sa_j} may be 0, which is an unattractive option. However, if the domain is such that we can form effective plans by only considering pairwise costs (see end of section 4.2.2), then the $\max\min$ term will be small. We may also replace the $\max$ operator in k with the average operator over the values observed so far. In future work we will need to investigate how we may modify the k term in (4.14) for different domains and how we may modify the predictive model without rebuilding it from scratch.

Now consider the new graph construction procedure, which is a modification of the procedure given in section 4.2.1. In that procedure, we construct nodes for each state and then add an edge between each pair of nodes corresponding to states s_i and s_j with cost equal to $\min_{a \in A} ES_a(s_i, s_j)$. During planning without using the PM, this cost is obtained from the observed model. When planning with the PM and computing the $\min$, we use values from either the OM or the PM depending on which one has the greater confidence for the particular action and s_i.

New Construct Graph

1. Form nodes with the states
2. Form directed edges between each pair of states.
3. For each directed edge (s_x, s_y) going from node/state s_x to s_y,

(a) If $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ already computed, go to next edge

(b) Else compute $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ using **Compute** $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ **with** γ

(c) To the edge assign the computed cost $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$

Compute $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ **with** γ'

1. For each $a \in A$
   (a) Determine cluster $\hat{Sa}_i$ that s_x belongs to using classifier Ca.
   (b) If $C(\hat{Sa}_i) \leq C(s_x, a)$ use the OM to obtain estimate $\hat{ES}_a(s_x, s_y)$ of $ES_a(s_x, s_y)$.
   (c) Else use the PM to compute estimate $\hat{ES}_a(s_x, s_y)$ of $ES_a(s_x, s_y)$ using (4.12); use $P(\Delta_a(s_j, a(s_j)))$ as an estimate $\hat{P}(\Delta(s_x, (s_x)))$ of $P(\Delta_a(s_x, a(s_x)))$. Note that during this step we may need to recursively call **Compute** $min_{a \in A} \hat{ES}_a(s, s')$ **with** $\gamma \cdot \gamma'$ for the ESs inside the sum in (4.12).
2. Compute $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ using the values obtained from previous step.

Algorithm 4.4: Planning when using the Predictive Model

Note that in step 1c we may end up performing too many recursions , so we need some way to make sure we do not recurse too much. We may have to rely on setting the γ parameter judiciously, or we may stop recursing once the γ' parameter reaches below some threshold and return only the $d_{s,a,s'}$ value as the ES value in (4.12). We may also avoid recursing completely by using some statistic of known ES values if appropriate for the domain (because of prior knowledge). We will focus more on this in future work.

4.4.2 Computing and Using Values for $d_{s,a,s'}$

As mentioned earlier, we need some way to compute the values for $d_{s,a,s'}$ for the states we are predicting the ES values for. To compute these values we can do one of the following. We can either assume that these values are known to the user. Or we can use the current average/max/min (or any statistic as appropriate for the domain) of all the $d_{s,a,s'}$ observed as the standard value. Or we can assume that the cost signals depend on the $\Delta_{s,a,s'}$ - i.e. there is a mapping from $\mathcal{A}_\mathcal{U}$ to $\{d_{s,a,s'}\}$, where, as described in section 3.2, $\mathcal{A}_\mathcal{U}$ is the range of Δ and denotes the set of most compact algorithms that cover all possible 'change induced' by all the actions. Since the novel states share the same Δ with non-novel states (see figure 3), we can then learn this mapping and thus compute the $d_{s,a,s'}$ values for the novel states. We believe the last assumption is actually reasonable for many interesting domains - e.g. in a robot navigation domain the damage incurred to the robot on reaching the first floor depends on whether it leaped off top floor or used the elevator - i.e. the damage depends on what change was induced. We will investigate the implications of each of these options in detail in future work.

In all of these cases, we need to incorporate the computed values of $d_{s,a,s'}$ in our cluster construction process in the predictive model. We need to modify the error function (4.13) to take into account the $d_{s,a,s'}$:

$$
\begin{aligned}
|\hat{ES}_a(s_n, s_g) - ES_a(s_n, s_g)| &\leq \gamma\epsilon_{Sa_j} \max_{s' \in R(s,a)} \min_{a' \in A} ES_{a'}(s', s_g) \\
&\quad \sum_{s' \in S} |P(\Delta_a(s_j, s'))d_{s_n,a,s'} - P(\Delta_a(s_n, s'))d_{s_j,a,s'}| \\
&\leq \epsilon_{Sa_j} k \max_{a,s'} |d_{s_n,a,s'} - d_{s_j,a,s'}| \\
&\quad \sum_{s' \in S} |P(\Delta_a(s_j, s'))d_{s_n,a,s'} - P(\Delta_a(s_n, s'))d_{s_j,a,s'}| \\
&\leq \epsilon_{Sa_j} k(\epsilon_{Sa_j} | \max_{s,a,s'} d_{s,a,s'} - \min_{s,a,s'} d_{s,a,s'}|) \\
&= (\epsilon_{Sa_j}{}^2)k' \qquad (4.15)
\end{aligned}
$$

where the first line and the k term in the second line follows from (4.14) and the last line is a rewrite of the third. As before, this is an upper bound on the κ function from section 3.3.1. We can thus set the η parameter from (3.7) to $\min_{s,a,s'} d_{s,a,s'}$ ensuring the expected error is less than the minimum cost of particular transition. Any other value may also be chosen depending on the application.

Finally, to compute the $d_{s,a,s'}$, in addition to the methods described earlier, we can think of constructing a predictive model for predicting the $d_{s,a,s'}$. The PM for $d_{s,a,s'}$ has the same components as in the PM from section 3.3.1. Thus, in this PM, each cluster $Sa_i^d \in Sa^d$ contains states which are functionally similar in terms of $d_{s,a,s'}$ with respect to action a. So if $s_k, s_j \in Sa_i^d$, then

$$
\sum_{s' \in S} |P(s'|s_k, a)d_{s_k,a,s'} - P(s'|s_j, a)d_{s_j,a,s'}| \leq \epsilon_{Sa_i^d} \qquad (4.16)
$$

for some constant $\epsilon_{Sa_i^d}$. Again as in section 3.3.1, $\epsilon_{Sa_i^d}$ is selected so that it satisfies:

$$
\begin{aligned}
&C(\hat{Sa^d})(1 - P(e|Ca^d, \hat{Sa^d}_i))\kappa^d(\epsilon_{Sa_i^d}) + \\
&(1 - C(\hat{Sa^d}))P(e|Ca_d, \hat{Sa^d}_i))\kappa^d(2) < \eta^d \qquad (4.17)
\end{aligned}
$$

where Ca^d, as before, is the classifier for Sa^d, and $\eta^d \geq 0$ and C is the confidence term from section 3.3.2. Thus, if we do use a predictive model, we can use η^d in place of (4.15) in place of the $d_{s,a,s'}$ to get

$$
estErr_{Sa_i}{}' = \gamma\epsilon_{Sa_i} \max_{s' \in R(s,a)} \min_{a' \in A} ES_{a'}(s', s_g)\eta^d \qquad (4.18)
$$

Thus from the appropriate value for η^d should be determined by the value for η. Note that this final method can be used in reinforcement learning problems with non task-independent cost signals.

4.5 ES and Reinforcement Learning

In this section we contrast the ES method and two other Reinforcement Learning methods that also operate on goal directed MDPs. The first method we consider is Real Time Dynamic Programming ([Barto *et al.*, 1995]). The basic idea of RTDP is to use typical reinforcement learning methods to find the minimum cost path to the goal state. During behaving (i.e. trying to reach the goal state) RTDP uses equations 4.6 to back up costs (i.e. update the ES value) for one or more states (the states to back up perhaps being determined by some method like prioritized sweeping - see [Sutton and Barto, 1998]). Then the agent simply selects the best action at each state s - i.e. the action corresponding

to $\arg\min_{a \in A} ES_a(s, s_g)$ where s_g is the goal state. The obvious advantage of the ES method over this is that the ES method, due to the graph search aspect of it, can look ahead and form a feasible plan even before the cost to the goal state s_g has been backed up to the current state s. This also implies that, when there are multiple goal states, RTDP has to be able to back up the cost to go for the new goal state before it can form a plan while the ES is able to form a plan to the new goal state immediately - provided, of course, that the agent has seen the goal state at least once. At the same time, recalling the comments at the end of section 4.2.2, the ES method may not do well if there are many states with reachable next states that have wildly varying cost signals to other states. On the other hand, RTDP will be forced to take random actions so it will not demonstrably perform better, and will probably perform worse because we are assuming that there are many ways to select a bad state in the pathological case . When used in conjunction with the PM, the ES method may be able to form plans even before visiting the goal state (for an example of this, see experiments in section 5) and is able to behave in novel states more intelligently (i.e. can do better than taking random actions).

The second RL method for solving goal directed MDPS that we contrast ES with is Multi-Value Functions ([Moore *et al.*, 1999]). This algorithm addresses the problem that RTDP faces in a goal directed MDP with multiple goal states. It efficiently (both in terms of time and space) computes a 'good' policy to get to all goal state from any particular state. This is accomplished by constructing a hierarchy of states where states in one level knows how to get to states in other levels. The algorithm expects as input some knowledge of the distributions $P(s'|s, a)$ and the cost signals to start building the hierarchy. The ES method can be viewed as an online version of MVF, in the sense that it uses knowledge of which states are reachable from each state online to find a path from the current state to the goal state. So the ES method (and ES augmented with the predictive model - see section 4.4) may be used with MVF in a complementary fashion, with ES being used to plan when sufficient information about the domain is not available to build the hierarchy, and then using the MVF to efficiently store the policies when sufficient information is available. When combined with the PM, ES is able to behave even more effectively in domains with multiple goal states whose solution plans contain similar states. We can also think of improving the time required to plan in ES using techniques from sophisticated extensions to classical planning such as [Veloso *et al.*, 1995]

Of course, as we mentioned in the section 1, a main reason for developing ES was that it allows easy integration with the predictive model for solving cumulative learning problems and therefore serves as an interesting application of the PM.

5 Experiments

In this section we present the results of applying our methods in a simple domain which serves to demonstrate in a very stark manner the advantage of our method. First we describe the experiment setup, and then we describe the results and discuss their implications. We are deliberately using a simple domain to demonstrate basic efficacy of our method. Our future work will focus on more interesting and complex problems.

5.1 Experiment Setup

The simple 25×25 gridworld domain for our task is given in figure 4. Each cell in the grid corresponds to a state, and at each state the agent is given the (x, y) grid location and the type of the cell (BLOCKED or FREE) to the north, south, east and west of the state. In this domain we consider two tasks, where the first task consists of getting to state G1 and the second task consists of getting to state G2 - the start state is for both the tasks is S (see figure 4). First the agent learns the task G1 and then it learns the task G2. To solve the tasks the agent has four action at its disposal - MOVE-<DIRECTION> where <DIRECTION> can be North, South, East or West. Each action takes the agent by one cell in the expected direction with probability $2/3$ and randomly takes the agent in any of the other directions by one cell with probability $1/9$. If the cell that the action selects for the agent is blocked,

then the agent remains in its current cell.

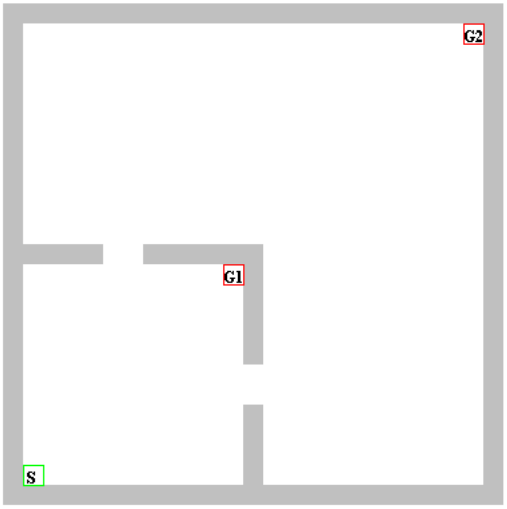

Figure 4: **Experiment Domain.**

We compared the performance of an agent using only the ES method (referred to as the No-PM agent subsequently), an agent using a Predictive Model with the ES model (referred to as the PM agent) and a reinforcement learning agent. We allowed all three to learn the task G1 for at least 100 episodes, where an episode consists of the agent reaching the goal state for a particular task from the start state, and then let the agent learn the G2 task. The reward table for the RL agent was reinitialized at the start of the G2 task because otherwise it would pose an unnecessary burden on the agent of having to unlearn the reward table for task G1. We closed the two doorways in the domain while learning the G1 task. This was done so that the first task G1 is learned quickly by not allowing the agent to wonder off accidentally into the larger room. After learning the G1 task, the agent was then allowed to learn the G2 task by opening the doorways. The agent learned the G2 task for 200 episodes. The aim of the experiments is to investigate transfer of knowledge gathered in task G1 to task G2.

We used a neural network using Backpropagation as our classifier for the PM agent. The various parameters were set as follows. We set the γ parameter for (4.1) and the γ' parameter from the algorithms in section 4.4 to 0.98 initially and decremented them by 0.099 every trial. We set an exploration parameter ϵ to 0.2 initially and multiplied it by 0.98 every trial. This parameter, as usual, determines how often the agent should perform a random action for the sake of exploration. Finally, the η parameter was set to 1 which represents the cost of going from one state to the next - this is as per the discussion in section 4.4. The $\max \min$ term in k in (4.14) was set to 10 and periodically estimated during the experiment. The predictive model was rebuilt when the estimate changed appreciably. Finally, the possible values for Δ for this problem are of the form: $(\delta x, \delta y, \delta N, \delta S, \delta E, \delta W)$, where

- δx represents the change in x value,
- δy represents the change in y value,
- δN represents how the cell to the North of the current cell changes (and similarly for $\delta S, \delta E$ and δW).

Thus the possible values for δx and δy are $+1$ and -1 and the possible values for δN etc. are FREE $\rightarrow$ BLOCKED, BLOCKED $\rightarrow$ FREE, FREE $\rightarrow$ FREE, BLOCKED $\rightarrow$ BLOCKED.

5.2 Expected Results

In both the tasks, we expect the vanilla reinforcement learning agent to not do as well as the PM or non-PM agents. This is because the RL agent needs to propagate rewards back to the start state before it can start acting optimally while the PM and the non-PM agent can start acting intelligently right from the start using the ES values and the Predictive Model in the case of the PM agent. We, however, expect more advanced methods like Dyna ([Sutton and Barto, 1998]) to perform similarly to the non-PM agent (but not the PM agent for reasons given below) because it can reduce number of episodes required to learn a task by off-line training (also see section 6.2.2).

In addition, for the first task (G1), we do not expect there to be much difference between the PM agent and the non-PM agent. This is because due to the simplicity/small size of the task, the non-PM agent learns the task quickly and the advantage of using a Predictive Model by the PM agent is not seen.

For the second task, goal state G2, we expect the PM agent to dominate the non-PM agent in a very striking manner. This is because in the G2 task all the states required to get from the start state to the to state G2 are similar to states encountered during learning task G1. More specifically, all the states in the smaller room (i.e. task G1) that do not have walls in any direction (north south east or west) are $\epsilon = 0$ similar to all such states in the larger room (i.e. states encountered task G2). The same is true for all states that have wall(s) only to the north, to the south etc. As a result, the PM agent can use the predictive model to predict what happens when the different actions are taken in the larger room - i.e. the agent can act as if it has already experienced the states in the larger room and thus form a plan to reach the goal state using the graph search procedure right from the very beginning. On the other hand the non-PM agent will actually need to experience the new states outside of the small room sufficiently often before being able to reach the goal state with any degree of consistency. It is in this task we hope to show the power of our method.

It should be noted that, we expect the PM agent to learn the G2 task quickly only because of the presence of similar states across the tasks, and not because both the G1 and G2 states are located on the north-east corner. In fact G2 could have been located anywhere in the bigger room and our expected results would still be the same.

5.3 Results

The results were obtained by averaging over 5 separate trials and are given in figures 5 - 8. The results agree with our expectations quite well. We now discuss each of the experiments in the following.

Figure 5 shows the performance for the three agents for the G1 task. As can be seen from the figure, there is little appreciable difference between PM and non-PM agents - both converging to the minimum no. of steps (around 30) by 5 episodes. Vanilla RL on the other-hand requires many more episodes to converge (about 150 episodes - not shown here) to the optimal value of around 30 steps. On the other hand, PM and non-PM agents took somewhat longer computational time to complete the 100 episodes task (around 3 seconds for RL, opposed to 10 seconds for PM and non-PM). This result agrees with our expectations.

Figure 6 shows the performance of the three different agents for the G2 task. The RL agent converges to the optimal of around 65 steps around episode 250, whereas both the PM and non-PM converges to the same value earlier (around episodes 3 and 60 respectively, we compare these two in more detail below). As before RL takes much less computational time to converge (4 seconds as opposed to 2 minutes or so).

Figure 7 shows the performance of the PM and the non-PM agents, "magnified" in the sense that the first episode is excluded in which non-PM takes about 3700 steps. The most striking aspect of this figure is that the PM agents gets almost to the optimal from the very first episode (75 steps) and then rapidly gets to the optimal in about 2 more episodes.

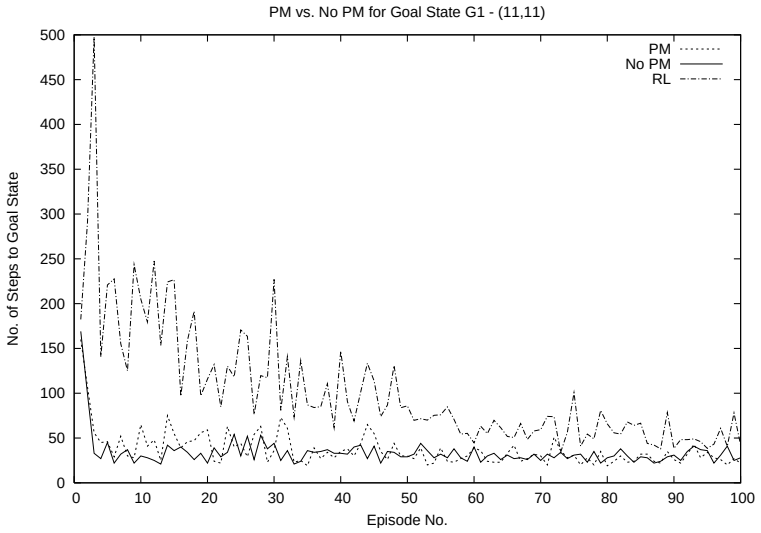

Figure 5: **Results for Goal G1.**

During the same time the non-PM agent expends around 3700 steps in its first episode, and then an average of over 100 steps for the next 15 episodes. So the PM agent saves a lot of steps during the very first episodes. This is illustrated in figure 8, which shows the number of steps taken by each agent in the first 3 episodes, with each episode ending in reaching the goal state. Finally, the PM agent always finds a straight line path to the goal state while the non-PM agent sometime takes small unnecessary detours as a result the interpolating line of the non-PM agent curve is slightly (couple of steps) above that of the PM agent.

6 Discussion and Related Work

In this section first we discuss the advantages and problems with our method. We also discuss the avenues of possible research that the above implies. Then we discuss prior work that is related to our method.

6.1 Discussion and Future Work

The results illustrate that there are certain domains where the predictive model will be very useful - namely domains where there are many states that are ϵ similar to each other for sufficiently small values of ϵ, and where training or obtaining training examples is far more expensive than computational cost. This is actually the situation in many real life applications, and certainly in the 'tasks' that human beings perform everyday. So this is definitely an advantage of our method as it reduces number of examples required significantly when there are many tasks that involve similar states. Thus future work will focus on quantifying similarity between tasks and describing more precisely the relationship between similarity across tasks and decrease in number of examples required.

A big problem of our system is that it is quite computationally intensive - the chief computational cost coming from maintaining and using the classifiers for the predictive model and replanning on every failure. We need to address these two issues. But we also note that sophisticated extensions to classical planning (e.g. [Veloso *et al.*, 1995]) and the multi-value function approach ([Moore *et al.*, 1999]) mentioned in section 3.3.4 can help reduce cost due to planning and replanning respectively. In addition the current system can deal only with finite number of countable-valued features. Although in *principle* this is sufficient for any problem that can be solved using a computer, for domains with continuous valued features (or even features with large number of values), our system will probably have prob-

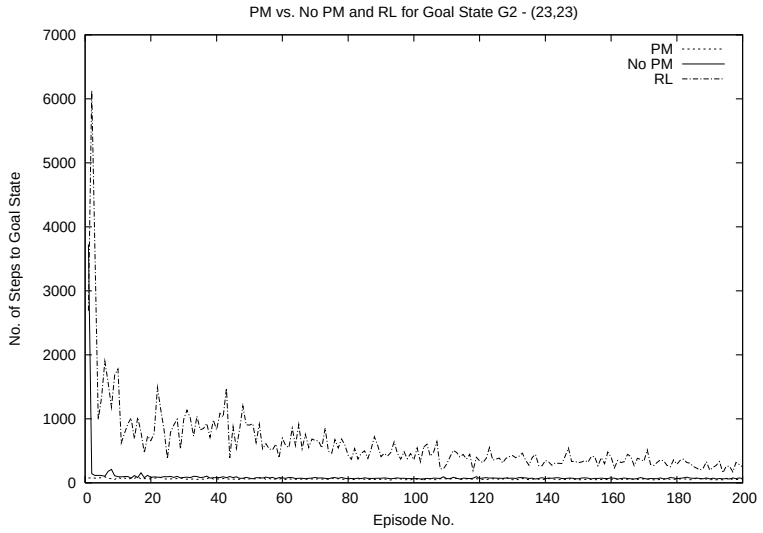

Figure 6: **Results for Goal G2.**

lems scaling, so we need to address this problem too. We also need to determine a way to deal with continuous/parameterized and temporally extended actions. We believe ideas in [Sutton *et al.*, 1999] and [Bertsekas and Tsitsiklis,] will be useful for the above two problems. We will also need to address the various extensions we identified for the system in the previous sections, namely, learning the Δ function from experience, dealing more effectively with overlapping feature functions, relationship between L_a functions (section 3.3.1) and the predictive model, behavior of the ES method during early stages of solving a task, efficiently rebuilding the PM when using it with the PM, obtaining tighter bounds on (4.14) and (4.15) for different domains, ensuring the algorithm **Compute** $\min_{a' \in A} \hat{ES}_{a'}(s_x, s_y)$ **with** γ' does not recurse too much, and investigating the various options for learning $d_{s,a,s'}$ values. We also wish to introduce a component to our model which would allow the agent to behave intelligently in the presence of other active entities in the world. We will, of course, also need to apply our method to problem domains far more interesting than the one described here.

Furthermore, as mentioned in the philosophical aside in section 3.1, we speculate that we can in fact develop a useful non-phenomenal semantic network based on the idea of functional similarity. Of course for this to work in complex and interesting domains, we would need to be able to deal with multi-modal and continuous inputs/states and actions. Such a semantic network can then form the central component to acquire, store and reuse knowledge in a Cumulative Learning System (as mentioned in the introduction). This avenue of research definitely needs to be explored.

6.2 Related Work

In section 3.3.4 we compared the Predictive Model to other Forward Models and in section 4.5 we compared ES+PM to two related methods. In this section, we talk about three areas very closely related to our method. These are Lifelong Learning, Model Based Reinforcement Learning and Hierarchical Reinforcement Learning.

6.2.1 Lifelong Learning

The general research area that our method is most closely related to is, as mentioned in the Introduction, Lifelong Learning ([Thrun and Pratt, 1998],[Thrun, 1995], [Caruana, 1997],[Drummond, 2002], [Mitchell and Thrun, 1993]). In these methods, the aim is to

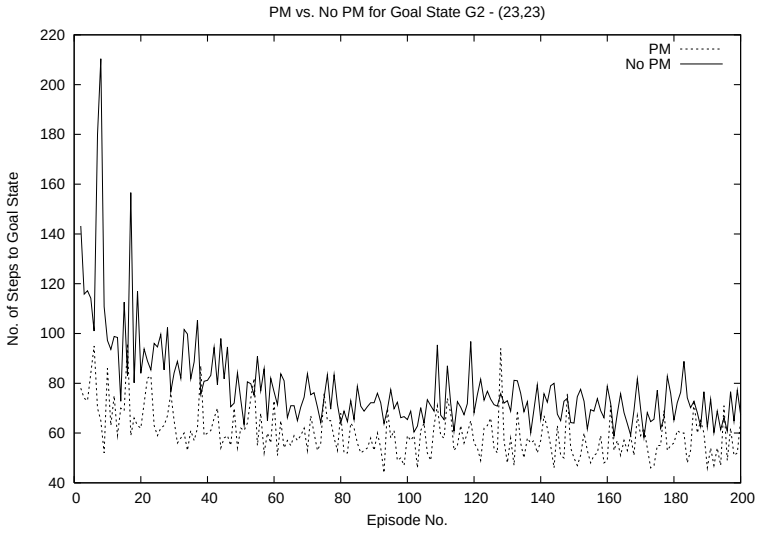

Figure 7: **Magnified result for Goal G2.**

have an agent that lives for a long time and gradually accumulates knowledge. It then uses this knowledge to make itself better. The difference between these methods and ours is the use of functional similarity to acquire knowledge and a focus on a cognitive architecture. The method presented in this paper is most closely related to [Mitchell and Thrun, 1993]. In that paper, the authors learned a neural network to predict what the next reward/state (somewhat of a kludge, since it combines these two disparate things) is going to be after taking some action in the current state. Our method may be viewed as a finer grained and more principled extension of that work.

6.2.2 Model Based Reinforcement Learning

The group of methods closely related to the ES method is model based reinforcement learning. Most of the leading methods in this area are nicely summarized in [Kaelbling *et al.*, 1996]. Other paper which deals with similar topics are [Moore *et al.*, 1999] and [Koenig, 2001]. The basic principle of these methods is to augment basic Reinforcement Learning methods by learning the model of the world (i.e. state transition probabilities and the reward model) in addition to the value functions and Q-functions. Then at each update, the model is used to update other states/state-action pairs in addition to the state or state-action pairs observed during the current transition - thus the algorithm converges much faster. The difference between the various algorithms lie in how exactly the other states to be updated are selected. These model based methods are computationally slower than the Q-learning method. It is fairly obvious that using something like our Predictive Model will greatly benefit these methods. For goal directed tasks, we expect our combination of ES and the PM will outperform these methods.

6.2.3 Hierarchical Reinforcement Learning

Another interesting research area is Hierarchical Reinforcement learning [Barto and Mahadevan, 2003]. The basic philosophy of these methods is to have a hierarchical system where the higher level systems learns and plans with higher level representations while the lower levels deal with lower level actions. This way the system is able exploit high level commonalities across different tasks. It would be interesting to introduce hierarchical elements into our system, thus allowing the agent to exploit “abstract” high level commonalities across tasks.

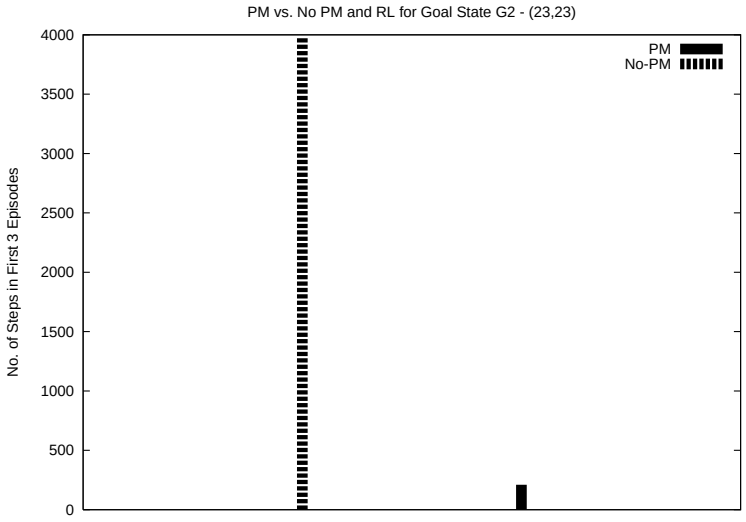

Figure 8: **Comparison of No. of Steps Expended in First 3 Episodes Goal G2.**

7 Conclusion

The most important notion that we introduced in this paper is that of functional similarity between states, wherein two states are similar with respect to an action if they behave in the same way when that action is applied to the states. We described a predictive model that learns which states are similar with respect to different actions and uses that to predict how novel states will behave when those actions are applied. We also presented a planning mechanism that applies the philosophy of basic motion planning/classical planning to solve goal directed MDPs. Then we described a way to combine the PM with the ES method and performed experiments to show the efficacy of the method.

We talked about possible future research directions, which primarily address the computation-intensiveness of our method, problems dealing with continuous domains, various extensions to the current method and possibly developing our method into a full cognitive architecture. Finally, we also discussed several related fields of research.

Appendix A: Proofs

Definition 1. *We define the discounted expected number of steps required to get from state s to s_g when action a is taken as follows:*

$$ES_a(s, s_g) = \sum_{s' \in S} P(s'|s, a) \left(d_{s,a,s'} + \gamma \min_{a \in A} ES_a(s', s) \right)$$

where $d_{s,a,s'}$ is the damage cost associated with action the transition.

Theorem 1. *Given the correct values of $d_{s,a,s'}$, and the state transition distributions, the update rule,*

$$\hat{ES}_a(s, s_g)_n = \sum_{s' \in S} P(s'|s, a) \left(d_{s,a,s'} + \gamma \min_{a' \in A} \hat{ES}_{a'}(s', s)_{n-1} \right) \tag{A.1}$$

,where $\hat{ES}$ denotes the estimate of ES, and $\hat{ES}_a(s, s')_n$ denotes the value of $\hat{ES}_a(s, s')$ after the n^{th} update converges to the actual $ES_a(s, s_g), \forall a, s, s_g$ as $n \to \infty$ and given also that $ES_a(s, s')$ for each triple (s, a, s') is also updated infinitely often.

Proof. Consider the error in step n for any $ES(s, a, s_g)$:

$$\begin{aligned}
|\hat{ES}_a(s, s_g)_n - ES_a(s, s_g)| &= |\sum_{s' \in S} P(s'|s, a) \left(d_{s,a,s'} + \gamma \min_{a \in A} \hat{ES}_a(s', s_g)_{n-1} \right) \\
&\quad - \sum_{s' \in S} P(s'|s, a) \left(d_{s,a,s'} + \gamma \min_{a \in A} ES_a(s', s_g) \right)| \\
&\leq \gamma \sum_{s' \in S} P(s'|s, a) \left| \min_{a \in A} \hat{ES}_a(s', s_g)_{n-1} - \min_{a \in A} ES_a(s', s_g) \right| \\
&= \gamma \sum_{s' \in S} P(s'|s, a) \left| \max_{a \in A}(-\hat{ES}_a(s', s_g)_{n-1}) - \max_{a \in A}(-ES_a(s', s_g)) \right| \\
&\leq \gamma \sum_{s' \in S} P(s'|s, a) \max_{a \in A} \left| (-\hat{ES}_a(s', s_g)_{n-1}) - (-ES_a(s', s_g)) \right| \\
&= \gamma \sum_{s' \in S} P(s'|s, a) \max_{a \in A} \left| ES_a(s', s_g) - \hat{ES}_a(s', s_g)_{n-1} \right| \\
&\leq \gamma \sum_{s' \in S} P(s'|s, a) \max_{a \in A, s_t, s_g \in S} \left| ES_a(s_t, s_g) - \hat{ES}_a(s_t, s_g)_{n-1} \right| \\
&= \gamma \max_{a \in A, s_t, s_g \in S} \left| ES_a(s_t, s_g) - \hat{ES}_a(s_t, s_g)_{n-1} \right| \qquad \text{(A.2)}
\end{aligned}$$

where the second inequality follows from the property of functions that

$$|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|$$

Since the last line is the maximum error in step $n-1$, the ES values converge to the actual values as $n \to \infty$ and we can ensure that each triple is updated infinitely often.

We can ensure that each triple is updated infinitely often as $n \to \infty$ by using the familiar technique of traversing a 2D square grid, with natural number coordinates, starting from coordinate $(1, 1)$, in a diagonal fashion (figure 9). Here each point in each axis corresponds

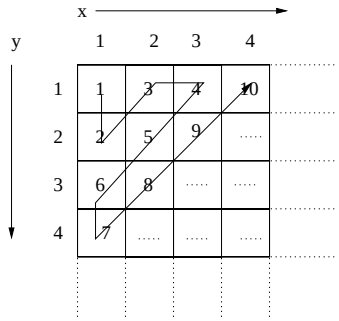

Figure 9: **Method for Updating** ES**.**

to a state for some enumeration of the states - so a particular cell corresponds to a pair of states. At the n^{th} update, we update the ES value corresponding to the triple s_x, a_i, s_y where x and y are, respectively the x and y coordinate of the cell m, where $m = n \bmod |A|$ and a_i is the i^{th} action with $i = n - m|A|$. The x and y coordinates can be calculated from m as follows. Set $k = \lceil(-1 + \sqrt{(1 + 8m)})/2\rceil$, then

$$x = k - \frac{(1+(-1)^k)}{2}\left(\frac{k(k+1)}{2} - m\right) - \frac{(1-(-1)^k)}{2}\left(m - \frac{k(k-1)}{2} - 1\right) \quad \text{(A.3)}$$

$$y = k - \frac{(1+(-1)^k)}{2}\left(m - \frac{k(k-1)}{2} - 1\right) - \frac{(1-(-1)^k)}{2}\left(\frac{k(k+1)}{2} - m\right) \quad \text{(A.4)}$$

□

Theorem 2. *The following inequality holds for all states* s*,* s_{mid} *and* s_{goal}*,*

$$\min_{a_g \in A} ES_{a_g}(s, s_{goal}) \leq \min_{a_m \in A} ES_{a_m}(s, s_{mid}) + \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) \quad \text{(A.5)}$$

Proof. We prove this by, for any particular goal state s_{goal}, first constructing a new domain W_N from our W given domain and show that the inequality in (A.5) holds. Then we convert W_N back to W state by state and show that the inequality holds in each step and thus holds in W.

Given a domain W consisting of a set of states S and actions A and the associated state transition probabilities, we perform the following operation to get the new domain. We introduce a new special state s_{sp} which has special 0 cost actions to all states in the domain. Then we sort the states in ascending order in terms of $\min_{a \in A} ES_a(s, s_{goal})$, breaking ties arbitrarily. We index the states in terms of their position in the sorted list so that s_{goal} is denoted by s_0, the state closest to s_{goal} is state s_1 and so on. For each state s_i, action a, and and all states s_j, such that $i < j$, and $P(s_j|s_i, a) > 0$ (i.e. all states that are reachable from state s_i using action a) we set $P(s_{sp}|s_i, a) = \sum_{\{s_j\}} P(s_j|s_i, a)$ and set $P(s_j|s_i, a) = 0$ - i.e. switch all the "connections"/single-action-paths to the higher-indexed state to the new state s_{sp}. Its an easy induction on the index of the state to show that (A.5) holds in W_N.

Basis $n = 0$ **:** Since $n = 0$, this implies $s = s_{goal}$ in equation (A.5). So the equation evidently.

Inductive Step $n \leq i$ **:** We assume that (A.5) holds for all state $s_n, n \leq i$. We now show by contradiction that (A.5) holds for s_{i+1}. i.e. assume for any state s_{mid},

$$\min_{a_g \in A} ES_{a_g}(s_{i+1}, s_{goal}) \quad > \quad \min_{a_{am} \in A} ES_{a_{am}}(s, s_{mid}) - \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal})$$

where s_{mid} is any state. Note that since there is no path from s_{i+1} to $s_k, k > i+1$, we do not need to consider the case when $s_{mid} = s_k, k > i+1$ since (A.5) holds in this case. Now, letting

$$a_{nm} = \arg \min_{a_{nm} \in A} ES_{a_{nm}}(s_{i+1}, s_{mid})$$

because of the $\min$ on the left hand side above, we can most certainly write,

$$ES_{a_{nm}}(s_{i+1}, s_{goal}) > ES_{a_{nm}}(s_{i+1}, s_{mid}) + \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) \quad \text{(A.6)}$$

$$ES_{a_{nm}}(s_{i+1}, s_{goal}) - ES_{a_{nm}}(s_{i+1}, s_{mid}) - \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) > 0 \quad \text{(A.7)}$$

Then expanding out the above using the definition of ES and canceling out the $d_{s_{i+1},a,s'}$ terms in the definition, we can write

$$\gamma \sum_{s' \in S_{W_N}} P(s'|s_{i+1}, a_{nm})(\min_{a \in A} ES_a(s', s_{goal}) - \min_{a \in A} ES_{a'}(s', s_{mid})) - \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) \quad > \quad 0 \quad \text{(A.8)}$$

Using the fact that $\sum_{s' \in S_{WN}} P(s'|s, a) = 1$

$$\gamma \sum_{s' \in S_{WN}} P(s'|s_{i+1}, a_{nm})(\min_{a \in A} ES_a(s', s_{goal}) - \min_{a' \in A} ES_{a'}(s', s_{mid}) - \frac{1}{\gamma} \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal})) > 0 \quad \text{(A.9)}$$

In the sum above, each term is < 0, since each s' is either s_{sp} or a state $s_k, k < i+1$ (A.5) holds for all s'. Therefore the whole sum is also < 0 and this concludes the inductive case.

Now we change back to our original domain W by restoring each $P(s_j|s_i, a)$, $i < j$, and for all $s_i \in S$, $a \in A$ to its original value. Then we update the ES values using the rule from theorem 1 and show that after each update (A.5) holds and thus this ensures that the equation holds in W. Before we do so, we need to set $ES_a(s_i, s_j)$, $i < j$ for all $s_i \in S$, $a \in A$ to some value (since the probabilities have been restored), and we set these to 0. As far as (A.5) goes, this only affects cases when $s_i = s$ and $s_j = s_{mid}$, as all other ES values remain the same. (A.5) holds in this instance because, as by definition $ES_a(s_i, s_g) \leq ES_a(s_j, s_g)$, it implies $ES_a(s, s_{mid}) \leq ES_a(s_{mid}, s_{goal})$. So the ES values in the initial form in the reconstituted W satisfy (A.5). Now let us update the ES values using the rule from theorem 1. Specifically, let the the update at step 1 be for some state s_i selected according theorem 1, and this is given by

$$ES_a(s_i, s)_1 = \sum_{s' \in S_W} P(s'|s_i, a) \left(d_{s_i,a,s'} + \gamma \min_{a \in A} ES_a(s', s)_0 \right) \quad \text{(A.10)}$$

Now we show that as this update is propagated using theorem 1, (A.5) holds for all s, s_{mid} by induction on the n, the number of updates.

Basis $n = 1$ **:** Since $n = 1$, this refers to the very first update given in (A.10). After the very first update the only value that could have changed is $ES_a(s_i, s)$. We need to consider the case when $s = s_{mid}$ in (A.5) and show that the following equation holds after the update:

$$
\begin{aligned}
ES_a(s_i, s_{goal}) - ES_a(s_i, s_{mid}) - \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) &\leq 0 \\
\Rightarrow \gamma \sum_{s' \in S_W} P(s'|s_i, a)(\min_{a \in A} ES_a(s', s_{goal}) - \min_{a' \in A} ES_{a'}(s', s_{mid}) & \\
-\frac{1}{\gamma} \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal})) &\leq 0 \quad \text{(A.11)}
\end{aligned}
$$

where the second line derives from the steps used to derive (A.9). We need to show this because if the above holds true for the action a, it certainly holds true for a_m where

$$
a_m = \arg\min_{a \in A} ES_a(s_i, s_{mid}) \tag{A.12}
$$

and as shown previously this is the action we need to consider when proving (A.5) holds. Now (A.11) holds (i.e. left hand side is ≤ 0) because (A.5) holds for all s' by the comments preceding the basis case.

Inductive Case $n > n'$ **:** We assume that (A.5) holds for all s, s_{mid} after n'^{th} update. Let us assume that in the $n' + 1^{th}$ update we update the entry $ES_a(s_k, s)$ as before using:

$$
ES_a(s_k, s)_{n'+1} = \sum_{s' \in S_W} P(s'|s_k, a) \left(d_{s_k, a, s'} + \gamma \min_{a \in A} ES_a(s', s)_{n'} \right)
$$

As in the base case, when $s = s_{mid}$ we have to show that the following equation holds after the update:

$$
\begin{aligned}
ES_a(s_k, s_{goal}) - ES_a(s_k, s_{mid}) - \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal}) &\leq 0 \\
\Rightarrow \gamma \sum_{s' \in S_W} P(s'|s_k, a)(\min_{a \in A} ES_a(s', s_{goal}) - \min_{a' \in A} ES_{a'}(s', s_{mid}) & \\
-\frac{1}{\gamma} \min_{a_{mg} \in A} ES_{a_{mg}}(s_{mid}, s_{goal})) &\leq 0 \quad \text{(A.13)}
\end{aligned}
$$

and again, (A.5) holds for all s' either because the relevant ES values *did not* change, or by the inductive hypothesis. In the case when $s = s_{goal}$, the exact same argument applies. This concludes our proof.

□

Recomputation of error in (4.14)

We want to have the error as follows:

$$
|\min_{a \in A} ES_a(s, s') - \min_{a \in A} \hat{ES}_a(s, s')| \leq \eta
$$

The new error when we take into account the error in estimating the correct values for the $\min_{a \in A} ES_a(s, s')$ terms in the right hand side of (4.13) is:

$$
|ES_a(s_n, s_g) - \hat{ES}_a(s_n, s_g)| \quad \leq \quad \gamma \epsilon_{Sa_j} \max_{s, s' \in S} \min_{a \in A} \hat{ES}_a(s, s') + \gamma \eta
$$

where the *new* $\gamma\eta$ term comes from the error in estimating the $\min_{a \in A} ES_a(s, s')$ terms. However, the η term in $\gamma\eta$, also has a hidden $\gamma\eta$ term, just like the error in the above equation. So now we get,

$$|ES_a(s_n, s_g) - \hat{ES}_a(s_n, s_g)| \quad \leq \quad \epsilon_{Sa_j} k + \sum_{i=1}^{\infty} \gamma^i \eta$$

where $k = \gamma \max_{s,s' \in S} \min_{a \in A} \hat{ES}_a(s, s')$. The new term is a geomteric series with rate = γ and therefore the new version of (4.14) is

$$|ES_a(s_n, s_g) - \hat{ES}_a(s_n, s_g)| \quad \leq \quad \epsilon_{Sa_j} k + \frac{\eta}{1-\gamma} \tag{A.14}$$

Therefore, to get an error of less than η' from our PM when using it with ES, we set the η parameter so that even with the new $\frac{\eta}{1-\gamma}$ term, the error is less than η' in (3.12). Letting $\nu = C(\hat{Sa}_i)(1 - P(e|Ca, \hat{Sa}_i))$, from equality in (3.12) we have

$$\begin{aligned} \nu(\epsilon_{Sa_j} k) + (1-\nu)2k &= \eta \\ \epsilon_{Sa_j} &= \frac{\eta - (1-\nu)2k}{k\nu} \end{aligned}$$

now to have the error less than η' we need to set η so that,

$$\nu(\epsilon_{Sa_j} k + \frac{\eta}{1-\gamma}) + (1-\nu)(2k + \frac{\eta}{1-\gamma}) \leq \eta'$$

, plugging in the value for ϵ_{Sa_j} in (A.15) the relationship between η and η' is given by,

$$\eta \leq \frac{(\eta' - (1-\nu)2k)(1-\gamma)k}{(k+1-\gamma) - (1-\gamma)(1-\nu)2k} \tag{A.15}$$

Of course we would need to adjust the ν parameter to ensure that the above produces a large enough and positive value for η.

References

[Barto and Mahadevan, 2003] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems Journal*, 13:41–77, 2003.

[Barto *et al.*, 1995] A.G. Barto, S.J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Bertsekas and Tsitsiklis,] D. P. Bertsekas and J.N. Tsitsiklis. *Learning to act using real-time dynamic programming*. Athena Scientfic, Belmont, MA.

[Caruana, 1997] R. Caruana. Multitask learning. *Machine Learning*, 28 (1):41–75, 1997.

[Cover and Thomas, 1991] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[Drummond, 2002] C. Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence*, 16:59–104, 2002.

[Flanagan and Wing, 1997] J. R. Flanagan and A. M. Wing. The role of internal models in motion planning and control: Evidence from grip force adjustments during movements of hand-held loads. *Journal of Neuroscience*, 17:1519–1528, 1997.

[Jordan and Rumelhart, 1992] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[Kaelbling *et al.*, 1996] L. Kaelbling, M. L. Littman M.L., and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Karniel, 2002] A. Karniel. Three creatures named 'forward model'. *Neural Network*, 15:305–307, 2002.

[Kawato, 1999] M. Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9:718–727, 1999.

[Koenig, 2001] S. Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129:165–197, 2001.

[Minsky, 1988] Marvin L. Minsky. *The Society of Mind*. First Touchstone Edition, 1988.

[Mitchell and Thrun, 1993] T. M. Mitchell and S. B. Thrun. Explanation-based neural network learning for robot control. In *Adavances in Neural Information Processing Systems*, pages 287–294, San Mateo, CA, 1993. Morgan Kaufmann Press.

[Moore *et al.*, 1999] A. W. Moore, L. C. Baird, and L. Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, 1999.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 1999] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[Thrun and Pratt, 1998] S. Thrun and L. Y. Pratt, editors. *Learning To Learn*. Kluwer Academic Publishers, Boston, MA, 1998.

[Thrun, 1995] S. Thrun. Lifelong learning: A case study. Technical Report CMU-CS-95-208, Computer Science Department, Carnegie Mellon University, 1995.

[Veloso *et al.*, 1995] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, and E. Fink. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.