# Learning Planning Operators in Real-World, Partially Observable Environments

**Matthew D. Schmill, Tim Oates, and Paul R. Cohen**
Computer Science Department, LGRC
University of Massachusetts, Box 34610
Amherst, MA 01003-4610
{schmill,oates,cohen}@cs.umass.edu

### Abstract

We are interested in the development of activities in situated, embodied agents such as mobile robots. Central to our theory of development is means-ends analysis planning, and as such, we must rely on operator models that can express the effects of a robot's action in a dynamic, partially-observable environment. This paper presents a two-step process which employs clustering and decision tree induction to perform unsupervised learning of operator models from simple interactions between an agent and its environment. We report our findings with an implementation of this system on a Pioneer-1 mobile robot.

## Introduction

We are developing a theory of conceptual development in intelligent agents. In our theory, activity plays a critical role in the development of many high-level cognitive structures, in particular classes, concepts, and language. It is our goal to derive and implement a theory of the emergence of complex activity in intelligent agents and implement it using the Pioneer-1 mobile robot.

In our theory, means-ends planning plays a central role in the development of activities. Development begins with the agent exploring its native actions (*move* and *turn*, plus raising and lowering a gripper, for the Pioneer-1), and building operator models for them. With these, the agent can begin creating simple sequences of native actions using means-ends planning. Successful plans can be cached and taken as activities themselves. These activities are in turn modeled, and feed the next wave of activity planning, resulting in a hierarchy of increasingly sophisticated activity. It is on this foundation that we will base our investigation into the acquisition of classes, concepts, and language.

Our theory hinges on the idea that the native actions can be modeled in such a way that they will support planning for a mobile robot. In particular, a mobile robot operating in a complex, partially observable environment creates the following requirements of operator models:



- Operator models must express effects of actions that as they are sensed by the robot. Sensor readings are real-valued and actions have temporal extent. Pushing an object is sensed by a Pioneer-1 through a variety of continuous sensors. Its sonars report the distance, in millimeters, to the object in front of it. Its velocity encoders report the robot's velocities before and after it makes contact with the object. Likewise, its vision system and bump sensors produce continuous valued readings 10 times a second throughout the duration of the activity. Operator models must encode how these sensor readings change over the course of activity.
- Operator models must express the multiplicity of outcomes for a single action. Executing an action may result in many different outcomes with qualitatively different sensor patterns associated with them. The MOVE-FORWARD action, for example, may lead to pushing a small object, being impeded by a large object, or narrowly missing and passing by an object. An agent must learn to distinguish the qualitatively different outcomes of activity.
- Operator models must include a predictive component (like the preconditions of classical planning) which relate actions and sensory patterns to possible outcomes of that action. That is, if there is a large object a short distance ahead of the robot, a MOVE-FORWARD action is more likely to result in the robot crashing than the robot moving without obstruction. The possiblity of crashing should be predicted when sensors are reporting a large object ahead.

The remainder of this paper describes our unsupervised algorithms for learning probabilistic operator models for a robot acting in a complex, continuous environment. We treat this as a two part problem, one of learning the effects (roughly speaking, the postconditions) of the robot's actions, and one of learning sensory features, which we call *initial conditions*, that can be used to predict the effects of an action. In the next section, we consider a clustering scheme that distinguishes between the many outcomes a single action might produce. Clusters built by this scheme form the basis of operator models, with dynamic representations of out-

come. In the section that follows, we discuss the induction of initial conditions that are associated with the various outcomes of the robot's actions, and the challenges that a partially-observable environment presents to this task. We conclude with a discussion of our operator models and our preliminary work in planning using the learned operator models.

## Clustering by Dynamics

To ground our discussion of learning planning operators, consider the Pioneer-1 mobile robot. Its sensors include, among others, a bump switch on the end of each gripper paddle that indicates when its gripper hits an object, an infrared break beam between the gripper paddles that indicates when an object enters the gripper, and wheel encoders that measure the rate at which the wheels are spinning. Also included are visual sensors for tracking colored objects. The vision system can track up to 3 objects, reporting their locations and sizes in the visual field.

Suppose the robot is moving forward at a fixed velocity. The values returned by the sensors mentioned above can be used to discriminate many different experiences of the robot. For example, if the robot runs into a large immovable object, such as a wall, the bump sensors go high and the wheel velocities abruptly drop to zero. If it bumps into a trash can, which is large but movable, the bump sensors go high and the wheel velocities remain constant. If it comes across an object that can be grasped, the break beam goes high when the object enters the gripper and there is no change in wheel velocity. As observers of the robot's actions, we make these discriminations easily. For our own purposes, we give names to these outcomes, and invoke them as needed to achieve our goals. For the Pioneer, though, there is only one action, *move*, to which it has access. To be successful in planning, the Pioneer must itself discriminate these outcomes, and use them as the basis for a set of operator models based around the *move* action.

Let $E$ denote an *experience*, a multivariate time series containing $n$ measurements from a set of sensors recorded over the course of engaging in a single action such that $E = \{e_t | 1 \leq t \leq n\}$. The $e_i$ are vectors of values containing one element for each sensor. Given a set of $m$ experiences, we want to obtain, in an unsupervised manner, a partition into subsets of experiences such that each subset corresponds to a qualitatively different type of experience.

If an appropriate measure of the similarity of two time series is available, clustering is a suitable unsupervised learning method for this problem. Finding such a measure of similarity is difficult because experiences that are qualitatively the same may be quantitatively different in at least two ways. First, they may be of different lengths, making it difficult or impossible to embed the time series in a metric space and use, for example, Euclidean distance to determine similarity. Second, within a single time series, the rate at which progress is made can vary non-linearly. For example, the robot may move slowly or quickly toward a wall, leading to either a slow or rapid decrease in the distance returned by its forward-pointing sonar. In each case, though, then end result is the same, the robot bumps into the wall. Such differences in rate make similarity measures such as cross-correlation unusable.

The measure of similarity that we use is Dynamic Time Warping (DTW) (SK83). It is ideally suited for the time series generated by a robot's sensors. DTW is a generalization of classical algorithms for comparing discrete sequences (e.g. minimum string edit distance (CLR90)) to sequences of continuous values. It was used extensively in speech recognition, a domain in which the time series are notoriously complex and noisy, until the advent of Hidden Markov Models offered a unified probabilistic framework for the entire recognition process (Jel97).

Given two experiences, $E_1$ and $E_2$ (more generally, two continuous multivariate time series), DTW finds a warping of the time dimension in $E_1$ that minimizes the difference between the two experiences. Warping consists of *stretches* or *compressions* of intervals in the time dimension of one experience so that it more closely resembles the second.

Consider the univariate time series $E_1$ and $E_2$ of the leftmost column in figure 1. Many warpings (denoted $E_1'$) of $E_1$ are possible, such as the one in the middle column of the figure, in which the interval from point 3 to 7 are compressed and the preceding interval is stretched. This one, however, certainly does not minimize the shaded area between $E_1$ and $E_2$. The rightmost column shows how the intervals from 1 to 3 and 4 through 7 can be compressed, stretching the interval between them, to produce a warping which minimizes the area between the time series. In general, there are exponentially many ways to warp the time dimension of $E_1$. DTW uses dynamic programming to find the warping that minimizes the area between the curve in time that is a low order polynomial of the lengths of $E_1$ and $E_2$, i.e. $O(|E_1||E_2|)$.

DTW returns the optimal warping of $E_1$, the one that minimizes the area between $E_1'$ and $E_2$, and the area associated with that warping. The area is used as a measure of similarity between the two time series. Note that this measure of similarity handles nonlinearities in the rates at which experiences progress and is not affected by differences in the lengths of experiences. In general, the area between $E_1'$ and $E_2$ may not be the same as the area between $E_2'$ into $E_1$. We use a symmetrized version of DTW that essentially computes the average of those two areas based on a single warping (KL83).

Given $m$ experiences, we can construct a complete pairwise distance matrix by invoking DTW $m(m-1)/2$ times (the factor of 2 is due to the use of symmetrized DTW). We then apply a standard hierarchical, agglomerative clustering algorithm that starts with one cluster for each experience and merges the pair of clusters with the minimum average intercluster distance

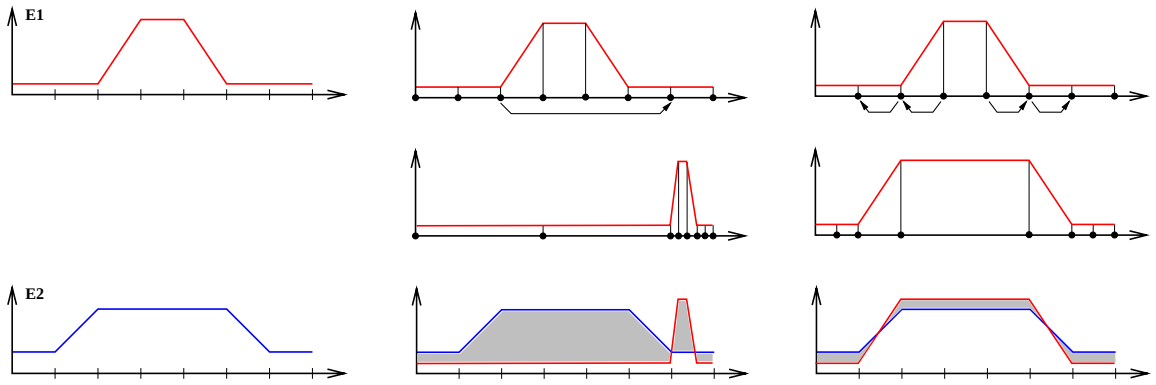


Figure 1: Two time series, $E_1$ and $E_2$, (the leftmost column) and two possible warpings of $E_1$ into $E_2$ (the middle and rightmost columns).

(Eve93). Without a stopping criterion, merging will continue until there is a single cluster containing all $m$ experiences. To avoid that situation, we do not merge clusters for which the mean intercluster distance is significantly different from the mean intracluster distance as measured by a t-test.

Clusters then become the basis for operator models. Each corresponds to a qualitatively different outcome: pushing, crashing, etc., and can be described by a cluster prototype. The cluster prototype may be the average of all the cluster members' time series, or simply the centroid experience, and contains the sensory patterns that are characteristic of that outcome. These time series, then, are analog to the *postconditions* of classical planning operator models. They express what the Pioneer can expect to happen given the action it is about to take fits into some cluster. What is missing is the predictive component of operator models: sensory conditions that inform the Pioneer of which cluster outcomes are likely to unfold given its choice of action to take.

## Initial Condition Induction

For our operator models to be useful for generative planning, we must be able to identify what aspects of the state of the world cause one outcome cluster to unfold instead of another. What causes the Pioneer to be impeded during a move as opposed to moving freely? One might call the features of the environment that answer this question the *preconditions* to crashing or moving freely.

When the Pioneer is considering which action to take, it has only its sensors to guide it. When it considers what will happen if it executes the *move* action, it can access its vision system, its sonars, and its internal state sensors. It does not have the benefit of rich representations of the world to tell it that it is facing a wall as opposed to a trash can. If we are to understand preconditions to have *causal* interpretations, then the Pioneer cannot truly generate preconditions. It can only associate sensory states with outcomes: a small red object in the visual field is most closely associated with cluster $c_1$ unfolding, while a large blue object is associated with cluster $c_5$ unfolding. We call these sensory states *initial conditions* to distinguish them from preconditions. In this section, we describe our system for inducing the initial conditions associated with cluster outcomes.

Our approach to initial condition induction is as follows. We have a set of experiences $\mathcal{E}$, and a partitioning of these experiences into outcome clusters. Included in each experience are sensor readings for one second leading up to the onset of activity. We call this interval the *precursor* to the actual experience, and the remainder of the experience the *successor*. The precursor describes the context in which the action was taken. We can use the precursor sensor readings, then, as features in a classification task where we want to predict cluster membership of the successor based on precursor sensor readings.

For the purposes of this paper, we make the simplifying assumption that in all of our experiences, the Pioneer is at rest during the precursor. Under this simplifying assumption, we can take the mean values (some sensors are noisy) over the precursor interval for each sensor. Thus, if there are $n$ sensors, and $m$ experiences, we have a training set of $m$ instances, each instance described by $n$ features, and labeled by the number of the cluster the experience belongs to. In future work, we intend to lift this assumption, using regression lines over the precursor instead of the mean.

A variety of algorithms are suitable for this classification task: artificial neural networks, the simple Bayes classifier, and decision tree induction have all been shown to perform well. Since we are not only interested in being able to predict the cluster that will unfold, but also in the sensory features that matter to the predictor, our choice of classifier is narrowed to algorithms that maintain declarative, easily interpreted structures. We chose decision tree induction for its balance between performance and ease of interpretation. Of the many decision tree induction algorithms, we use TBA, a system designed to prevent overfitting

which is present in most induction algorithms (JC96; JS97), since in the context of our system, overfitting equates to producing unnecessarily restrictive initial conditions.

Inducing initial conditions in this way is a simple task. Experiences are converted to training instances and fed to TBA, which produces an *initial condition tree* for each action. The initial conditions for cluster $c_i$ built on action $a$ are extracted by searching the leaves of $a$'s tree for occurrences of $c_i$. For each leaf containing an instance of $c_i$, a clause is formed by tracing a path back to the root and conjoining a clause for each decision node. The initial condition set is the disjunction of all the conjunctive clauses formed in this way.

Consider the sample tree of figure 2. At the leaves are outcome clusters, which have been hand-assigned descriptive titles after generation (for our own benefit), and their observed frequency. If we are extracting the initial conditions for the *premature halt* outcome of the turn action, in which the robot comes to a stop before it intended to (usually due to a blocked path), we note that it is included in two leaves. The path to the first occurs when the direction of the turn is clockwise. The second occurs when the direction is counter-clockwise and the gripper beam is not broken in the precursor interval. A first pass at the initial conditions for *premature halt* would be

$$\begin{array}{c}(\text{direction} = \textit{clockwise}) \vee \\ (\text{direction} = \textit{counter-clockwise} \wedge \text{beam} = \text{off})\end{array}$$

It is worth noting, though, that between the two leaves, *premature halt* is found less frequently in the first one. The percentages shown in the figure represent the frequency of that outcome in the leaf. In the clockwise leaf, this outcome was observed in 16% of cases, while only 8% in the counter-clockwise leaf. While this is most likely a spurious condition due to the fact that this outcome is infrequent, one can easily imagine situations where relative frequencies between leaves would be important and useful information to a planner. Suppose that *premature halt* was also found in the third leaf of figure 2, in which the break beam was on, with an observed frequency of 40%. It would follow that the outcome would be most likely if the break beam were broken, a useful heuristic to guide the planner's search. Therefore, we include these frequencies in our initial condition specifications and denote them as follows:

$$\begin{array}{c}(\text{direction} = \textit{clockwise})[16\%] \vee \\ (\text{direction} = \textit{counter-clockwise} \wedge \text{beam} = \text{off})[8\%]\end{array}$$

With the addition of initial conditions, our operator models now have the requisite properties for means-ends planning. Before moving on to an outline of how the operator models are utilized by a planner, we present our experimental results with the clustering and initial condition induction schemes.

## Experimental Results

Our experiments with operator models evaluate each component of the system individually. We started by collecting data for 4 sets of experiences: 102 experiences with the robot moving in a straight line while collecting data from its velocity encoders, break beams, and gripper bumper (which we will call the *tactile* sensors), the same 102 move experiences collecting data from the Pioneer's vision system, including the X and Y location, area, and distance to a single visible object being tracked (which we will call the *visual* sensors), 50 experiences with the robot turning in place collecting tactile data, and those same 50 turn experiences collecting visual data. In each experience, the robot attempted to move or turn for a duration between 2 and 8 seconds in a laboratory environment. Among the experiments were examples of the robot moving backward and forward at 4 different speeds and turning clockwise or counter-clockwise while pushing various objects, passing by them, becoming blocked, and with objects entering and leaving the visual field.

### Clustering

Our goal in clustering is for the clusters to map to action outcomes for the purposes of planning in service of our theories of conceptual development. Roughly, we would like our agents to produce ontologies of activity that are in accordance with those a human might produce. As such, our primary means of evaluating cluster quality is to compare them against clusters generated manually by the experimenter who designed the experiences they comprise.

We evaluate the clusters generated by DTW and agglomerative clustering with a $2 \times 2$ contingency table called an *accordance table*. Consider the following table:

| | $t_e$ | $\neg t_e$ |
|---|---|---|
| $t_t$ | $n_1$ | $n_2$ |
| $\neg t_t$ | $n_3$ | $n_4$ |

We calculate the cells of this table by considering all pairs of experiences $e_j$ and $e_k$, and their relationships in the target (hand-built) and evaluation (DTW) clusterings. If $e_j$ and $e_k$ reside in the same cluster in the target clustering (denoted by $t_t$), and $e_j$ and $e_k$ also reside in the same cluster in the evaluation clustering (denoted by $t_e$), then cell $n_1$ is incremented. The other cells of the table are incremented when either the target or evaluation clusterings places the experiences in different clusters ($\neg t_t$ and $\neg t_e$, respectively).

Cells $n_1$ and $n_4$ of this table represent the number of experience pairs in which the clustering algorithms are in accordance. We call $n_1 + n_4$ the number of *agreements* and $n_2 + n_3$ the number of *disagreements*. The *accordance ratios* that we are interested in are $\frac{n_1}{n_1+n_2}$, accordance with respect to $t_t$, and $\frac{n_4}{n_3+n_4}$, accordance with respect to $\neg t_t$.

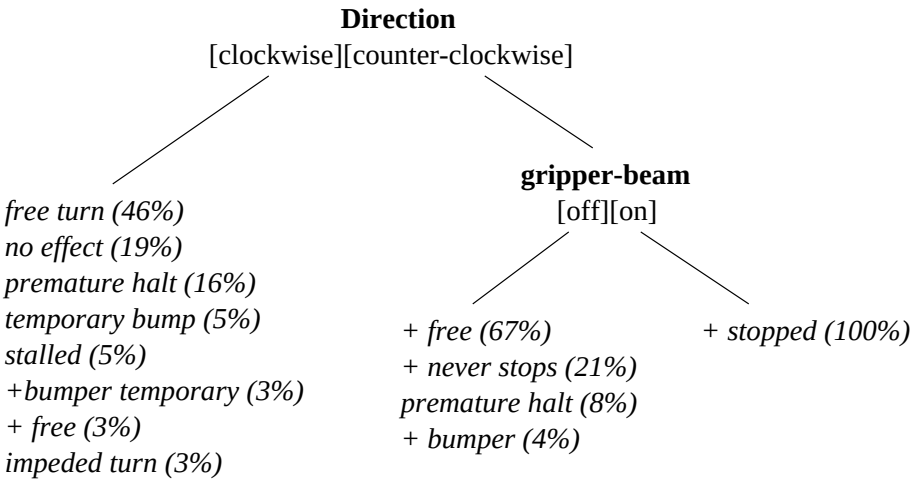


Figure 2: An initial condition tree built on clusters describing the effects of *turn* actions on the Pioneer's bump, beam, and velocity sensors.

| | # | $t_t$ | $t_t \wedge t_e$ | % | $\neg t_t$ | $\neg t_t \wedge \neg t_e$ | % | Agree | Disagree | % |
|---|---|---|---|---|---|---|---|---|---|---|
| Move visual | -5 | 876 | 720 | **82.2** | 4275 | 4125 | **96.4** | 4845 | 306 | **94.0** |
| Move tactile | -7 | 443 | 378 | **85.3** | 4708 | 4468 | **95.0** | 4846 | 305 | **94.0** |
| Turn visual | -5 | 262 | 262 | **100.0** | 599 | 571 | **95.3** | 833 | 28 | **96.7** |
| Turn tactile | -6 | 163 | 163 | **100.0** | 698 | 593 | **85.0** | 756 | 105 | **87.8** |

Figure 3: Accordance statistics for automated clustering against the hand built clustering.

Table 3 shows the breakdown of accordance for the combination of dynamic time warping and agglomerative clustering versus the ideal clustering built by hand. The column labeled "#" indicates the difference between the number of hand-built and automated clusters. In each problem, the automated algorithm clustered more aggressively, resulting in fewer clusters. The columns that follow present the accordance ratios for experiences grouped together, apart, and the total number of agreements and disagreements.

The table shows very high levels of accordance. Ratios ranged from a minimum of 82.2% for experiences clustered together ($t_t$) in the move/visual set to 100% for experiences clustered together in the turn problems. For the turn problems, the aggressive clustering may account for the high $t_t$ accuracy, causing slightly lower accuracy in the $\neg t_t$ case.

## Initial Conditions

Using the hand-built clusters generated during the previous evaluation, we built initial condition trees for each of the four datasets. We had three goals in evaluating these trees: to examine their accuracy in predicting the class label of new experiences, to ensure that they are encoding relevant contingencies in the environment, and to ensure that the structure of the trees will eventually stabilize.

The trees generated for the turn/tactile and move/visual sets are shown in figure 2 and figure 4, respectively. It is readily apparent from the trees that they do indeed uncover some of the structure of the environment. In figure 2, the gripper beam being broken is an indication that something is within the gripper's grasp. This is one of the few instances in which a Pioneer can easily be prevented from turning, confirmed by the 100% observed frequency of the outcome *stopped*. The tree in 4 is much richer in structure. It first splits on the x location of the object being tracked on the visual field, as this location determines whether moving will bring the Pioneer into contact with the object, or whether it will pass on the left or the right. The tree also splits on the object's apparent width when it is toward the center of the visual field, as small object tend to disappear under the camera and into the gripper, while large ones end up filling the visual field. Finally, the extreme right range of X (a value of 140) indicates that there is no visible object being tracked. In this case, it is unlikely an object will come into view moving forward, but more likely (37%) that one will come into view when moving backwards.

Tree stability (robustness against drastic changes in tree structure when new training data becomes available) is important for two reasons. First, stability is a sign that additional experiences are not uncovering any new, hidden structure of the environment[1]. We would

[1]The converse is not necessarily true, as many induction algorithms continue to grow their representations even after the true structure of the domain is learned (OJ97).

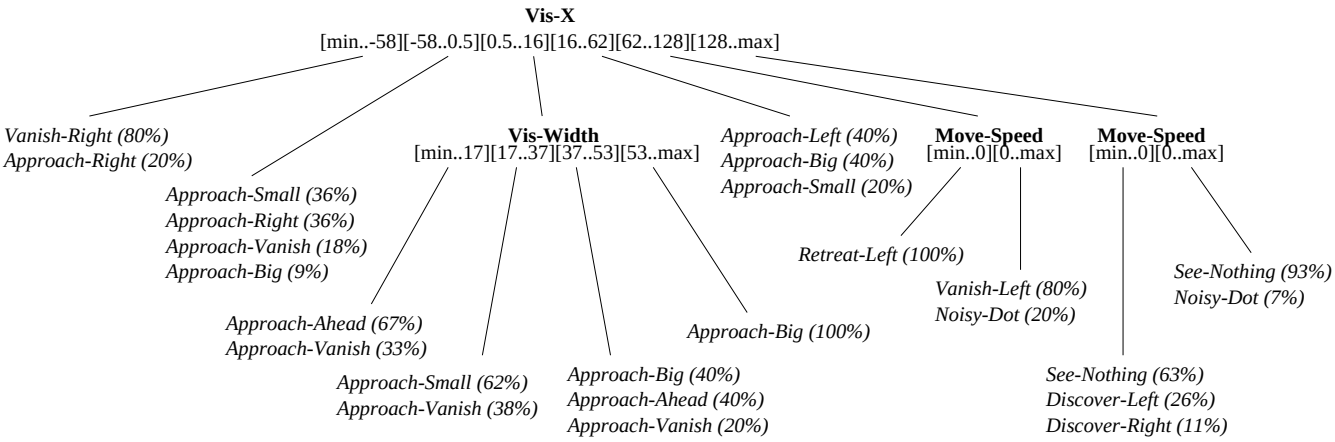


Figure 4: An initial condition tree built on clusters describing the effects of *move* actions on the Pioneer's visual sensors.

like our trees to stabilize early to show that our system needs relatively few examples to cover most of the contingencies. Second, since we use the structure of the trees to support planning, we would like stable trees so our operator models are not constantly changing, thus preventing effective plan reuse. To test for tree stability we generated 50 new experiences with the Pioneer, randomly, and used DTW to assign cluster labels to them, rebuilding the trees after adding the first 25, then the full 50 to the original data.

The initial condition trees showed only minor changes when additional experiences were added to the training set. Of the four trees, one tree (turn/tactile) did not change. In another (turn/visual), two nodes changed after adding 50 instances. In the third tree (move/tactile), one decision node was collapsed, and the last tree (move/visual) underwent significant changes. So, after roughly 75 turns, the turn trees have appeared to achieve a moderate degree of stability, and after 125 moves, the visual tree was still unstable, and the tactile tree was relatively stable. Most telling, though, were the types of changes that occurred in the trees. Many of the significant changes occurred when a decision node was replaced in a tree by what we call a *surrogate node*. In some cases, one sensor can convey the same information as another. For example, the visual sensors of an object's height and it's Y location on the visual field are often correlated: a taller object's center of mass is higher on the visual field. When the tree building algorithm evaluates a decision node and its surrogate, they may appear equal statistically, though they may affect the structure of the tree below considerably. We are considering ways to improve the worst-case stability of our tree induction algorithm.

Finally, we were interested in the accuracy of the trees. The typical evaluation metric for decision tree induction is a winner-take-all task, in which trees are generated on a test set, and accuracy is tested on a separate test set. It is not surprising, then, that our trees did not test well. Consider the distributions of class labels in some of the leaves of figures 2 and 4. In many cases, the majority class has an observed frequency of 60% or less; one should not expect better performance on a test set. Using the 50 randomly generated experiences from the stability test, we produced accuracy scores for the 4 initial condition trees, and indeed did not produce high accuracy scores. On the *move* sets, error rates of 80% to 90% were found, and on the *turn* sets, error rates 40% to 50% were recorded.

In a related experiment, we trained backpropogation networks to perform the same classification task. Again, the classifier scored poorly on the winner-take-all task (Sch99), achieving scoring well on training data, but achieving error rates indistinguishable from those of the decision trees on test data.

The key to understanding why our classification schemes perform so poorly is identifying the extent to which the environment is not fully observable through the Pioneer's sensors. Consider the case in which the Pioneer is considering turning. If there is nothing in its visual field, how can it predict whether something will appear or not? It can't. It can only narrow down the possibilities, and be left with a probability distribution over the likely outcomes. Consider the case in which the robot sits before a red object. If it moves forward, it will surely come in contact with the object, and can predict this, but will it succeed in pushing it, or will it crash? Since the Pioneer cannot sense features beyond location and size, it also cannot distinguish these outcomes, either. It is, in a sense, unfair to expect the Pioneer to score well on a winner-take-all accuracy test, since its sensors do not supply enough information to predict with absolute accuracy.

Despite our problems evaluating the initial condition induction, we remain encouraged. Our initial condition trees are uncovering the structure of the Pioneer's environment, and are allowing the Pioneer to make *estimates* of cluster outcome, even if it cannot pinpoint

a single outcome, and the trees are reasonably stable, even with as few as 75 data points to build on.

## Planning

Our planner is still in its preliminary stages, and as such, our goal in this section is to describe how our learned operator models are to be used in planning. The planner is a standard backward chaining, state based, means-ends analysis planner. Like the STRIPS planner (FN71) (and its many derivatives), planning here boils down to a search through the space of operator sequences. Due to nature of our operator models, and the environment we are working in, we must diverge from the simplicity of STRIPS-style planning in three situations.

*Goal crossings* are used as the basis for goal-matching.

*Active planning* can be used for the purpose of simultaneously refining planning operators and relaxing overly constrained plan spaces.

Planning operators must be instantiated as closed-loop controllers before they can be executed in plans on the Pioneer-1.

Recall the basic operation of a STRIPS-style planner. The agent begins with a goal, described in terms of a desired sensory state, and computes the difference between its current state and the goal state. The planner then considers any activity that can resolve some or all of those differences. Since our operator models are made up of prototype time series sensor data, and not discrete propositions over the state space, our planner selects cadidate activities when their sensory prototypes exhibit *goal crossings*. Goal crossings are simply points in the prototype at which goal sensor values are achieved. Generally, the activity with the most desirable goal crossing is then added to the tail of the plan, and its initial conditions are added to the goal stack.

Unfortunately, we cannot count on the initial condition induction algorithm to always produce perfect initial condition sets. In particular, induction algorithms are prone to overfitting, which will result in spurious initial conditions, which in turn overconstrains planning. In some cases, overconstrained planning will result in a planner failure when a solution does exist. We call the process of attempting to execute a plan in which most, but not all of a component operator's intial conditions can be met *active planning*. Active planning serves a dual purpose. First, it allows successful plans to be found and executed when spurious intial conditions are included in operator models. Second, it allows for those spurious conditions to be indentified; if a plan produces the same rate of success without satisfying a particular initial condition as it does with that condition satisfied, then it can be concluded that the condition is not necessary in the model.

Once a sequence of operator models has been selected to achieve the goal state, the sequence must be converted into series of closed-loop controllers. We call the process of generating a controller from an operator model *operator instantiation*. Operator instantiation works as follows. First, determine how the sensor values change *after* the action being instantiated is deactivated. On a robot, activities take a non-trivial amount of time to deactivate, such as the time it takes to come to a halt when a MOVE-FORWARD action is deactivated. We call the effects of an action that occur after it has been deactivated the *deactivation effects*. Next, compute when the operator should be deactivated by subtracting the deactivation effects from the goal crossing which the controller is attempting to achieve. Finally, a control program is generated using ACL (Sch98), a language we have developed for operator instantiation on the Pioneer.

A sample planner trace is shown in figure 5a, with its corresponding ACL control program shown in figure 5b. This plan is designed to satisfy the sensor goal of foveating on an object. In terms of sensor values, the robot wants to achieve an X location of a visible object between -10 and 10, or (vis-x $\in [-10 \ldots 10]$).

The plan trace includes two paths. One, which attempts to achieve the goal by moving forward using the operator "noisy-tiny-dot" (an outcome in which noise in the vision system leads to a variety of spurious sensor readings in the absence of any visible object), leads the planner down a fruitless path. Indeed, moving forward is not the best way for the Pioneer to foveate on an object. The second path through the plan space, which uses the turn operator "pass-left-to-right", is much simpler and has a much greater chance of success. This operator model has no initial conditions, and thus the planner returns the controller in figure 5b, which states that the Pioneer should turn counter-clockwise until an object enters the range $[-16.35 \ldots 8.35]$.

## Conclusions

We presented a system for the induction of planning operators from simple, sensorimotor interaction between an agent and a real-world, partially observable environment. Our system first uses dynamic time warping and agglomerative clustering to partition the experiences of an agent with its primitive actions into qualitatively different outcome classes. We evaluated this learning component by comparing it against hand-built clusterings, and found high levels of accordance between the automated system and the human expert, indicating that the automated system is keying on many of the same dynamic features that the human is.

Once our system has clustered its experiences by the dynamics of their outcomes, it attempts to learn sensory features that are associated with those outcomes. We make a distinction between preconditions, which have causal connotations, and initial conditions, which associate sensory features with likely outcomes, and describe an approach to inducing initial conditions based on decision tree induction. We found that the initial condition trees do indeed encode interesting features of

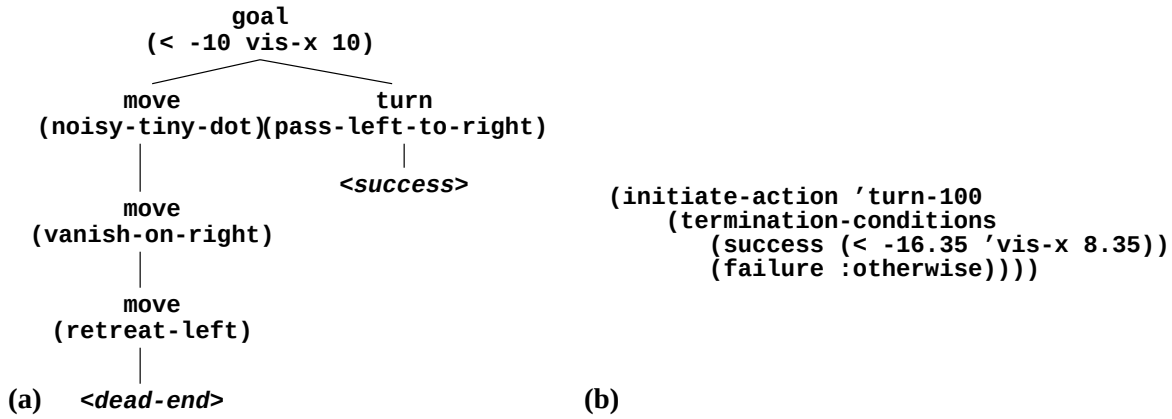


Figure 5: (a) A sample plan trace that achieves a simple goal with learned operators. (b) A closed-loop controller that implements the plan.

the environment associated with the outcomes of actions, and we have reason to believe that the trees will stabilize quickly, though we will continue to push for greater stability. Finally, the Pioneer's sensory apparatus allows it only partial observability, which has a considerable effect on the accuracy of initial condition trees in a winner-take-all prediction task.

This work suggests that researchers choose one of two courses for planning in realistic, partially observable domains. On one hand, providing additional learning mechanisms that will allow an agent to create richer representations of its world (in particular, the locations and features of objects not within sensor range) solves the problem of partial observability by making the world wholly observable. On the other hand, one might emphasize the importance of planning with contingencies to monitor and handle cases in which the expected outcome fails to occur. In future work, we hope to show that the latter is a viable option.

## Acknowledgements

This research is supported by DARPA/AFOSR contract #F49620-97-1-0485 and DARPA contract #N66001-96-C-8504. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the DARPA, AFOSR or the U.S. Government.

## References

T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

Brian Everitt. *Cluster Analysis.* John Wiley & Sons, Inc., 1993.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(2):189–208, 1971.

David Jensen and Paul R. Cohen. Overfitting in inductive learning algorithms: Why it occurs and how to correct it. Submitted to the *Thirteenth International Conference on Machine Learning*, 1996.

Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1997.

David Jensen and Matthew D. Schmill. Adjusting for multiple comparisons in decision tree pruning. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 195–198, 1997.

Joseph B. Kruskall and Mark Liberman. The symmetric time warping problem: From continuous to discrete. In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.

Tim Oates and David Jensen. The effects of training set size on decision tree complexity. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

Matthew D. Schmill. The eksl saphira-lisp system and acl user's guide. EKSL Memorandum #98-32, 1998.

Matthew D. Schmill. Predicting outcome classes for the experiences of a mobile robot. EKSL Memorandum #99-33, February 1999.

David Sankoff and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: Theory and Practice of Sequence Comparisons.* Addison-Wesley Publishing Company, Reading, MA, 1983.