Learned Models for Continuous Planning

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 nature of *activity* – structured behavior of nontrivial duration – in intelligent agents. We believe that the development of activity is a continual process in which simpler activities are composed, via planning, to form more sophisticated ones in a hierarchical fashion. The success or failure of a planner depends on its models of the environment, and its ability to implement its plans in the world. We describe an approach to generating dynamical models of activity from real-world experiences and explain how they can be applied towards planning in a continuous state space.

1 Introduction

We are interested in the problem of how activity emerges in an intelligent agent. We believe that activity plays a critical role in the development of many high-level cognitive structures: classes, concepts, and language, to name a few. Thus, it is our goal to derive and implement a theory of the development of activity in intelligent agents and implement it using the Pioneer-1 mobile robot.

Our model of the development of activities can be broken down into two continuously ongoing processes: a modeling process, and a planning process. In the modeling process, the agent endeavors to represent the effects of its actions on the observable environment. The Pioneer begins the modeling process with some primitive actions and sensors. Its primitive actions move the robot forward and backward, turn it clockwise and counter-clockwise, and raise and lower its gripper. The Pioneer's 42 real-valued primitive sensors include sonars, velocity encoders, bump sensors, and various readings from a blob vision system. At the outset, then, the Pioneer attempts to model how the primitive actions affect the patterns observed in its primitive sensors.

As models are created and refined, the planning process can begin to compose the models into simple plans for achieving goals. Useful plans will themselves become activities, be modeled, and reused by the planner to create new activities that do more and achieve increasingly sophisticated goals. The result of this process is a hierarchy of activities that can serve as a basis for further conceptual development like language, concepts, and classes.

Our focus in this paper is to detail the modeling process that enables planning. In the next section, we describe the analyses and transformations that are used to model the effects of activity. We follow with a discussion of how these models can be instantiated to generate simple controllers that serve as the basis for planning in the continuous space of the Pioneer-1 mobile robot. We conclude with a short discussion of the state of the system and future work in planning.

2 Learning Dynamics from Experience

We would like our models of activity to explain the sensory patterns a robot can expect if it engages in an activity. For example, engaging in a move activity might result in the robot's velocity encoders ramping up and the distance to an object decreasing. On the other hand, the robot may be obstructed when it attempts to move. In this case, the velocity encoders will report no change, the robot's bump sensor will report contact, and its sonars will likely report no change. In fact, each of the primitive actions offer a handful of qualitatively different possible outcomes. The first step in modeling, then, is for the robot to differentiate between the multiple outcomes of its activities.

Time series sensor data resulting from the application of an activity (called an *experience*) indicates that typical outcomes have characteristic sensory patterns as-

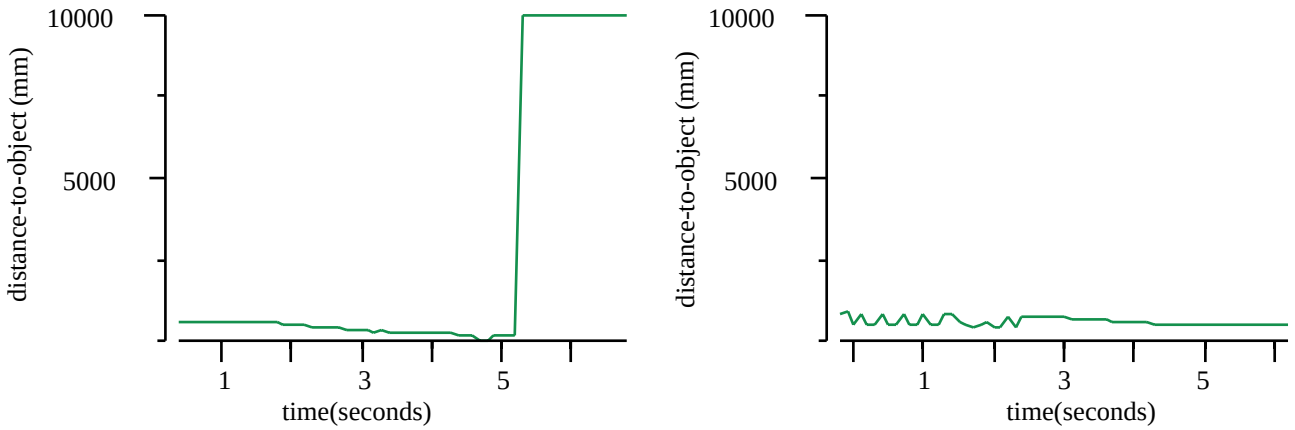

Figure 1: Traces of the distance between the agent and an object during two different experiences with the move activity. The data on the left is from an experience where the robot moves past the object without incident. On the right, data represents the robot pushing the object.

sociated with them. Figure 1 shows two characteristic sensory patterns for the move activity. In the plot on the left, the distance from the agent to the object slowly decreases, and then sharply increases to 10 meters. The increase to 10 meters is caused by the object disappearing from view, indicating that the robot passed by the object in this experience. In the right-hand plot, the distance to the object is close to zero, and constant. This pattern is characteristic of a movement in which the object is being pushed. Our approach to differentiating between different outcomes uses these signature patterns as a basis for clustering.

2.1 Clustering Experiences

The most common clustering algorithms rely on some kind of similarity metric, often called a *distance metric*, to judge whether or not two data should be kept together in a single cluster. For our system, the distance metric should make use of characteristic patterns in time series data to judge similarity.

A very simple distance metric for judging the distance between two time series would be to simply compute the mean square difference between the two time series. Under this metric, the patterns of figure 1 would have a large distance due to the sharp increase in the plot on the left.

This simple distance metric is not robust against differences in timing, though. Consider two experiences passing by an object, one in which the object is passed after 1 second, and another in which the object is passed after 5 seconds. The 4 second difference in timing causes 4 seconds of drastic mean square difference, most likely resulting in a distance between these two experience greater than the distance between the two plots of figure 1. A more robust distance metric would abstract away differences in timing and duration.

To perform this temporal abstraction, we utilize an algorithm called *dynamic time warping* [5]. The dynamic time warping algorithm uses *stretching* and *compressing* operations to create a temporal mapping between two time series that minimizes the mean square difference between them. The stretch operations extend the time scale over a region of the time series, mapping an interval of duration δ to an interval of duration $\mu\delta$, where $\mu > 1$, while preserving the temporal ordering of all data points. Conversely, the compress operation maps an interval of duration δ to an interval of duration $\mu\delta$, where $\mu < 1$.

In the previous example of the 1 second and 5 second pass experiences, dynamic time warping could simply compress the 5 seconds leading up to the object disappearing from sight into a 1 second interval. This new, *warped* time series has a very small mean square difference from the 1 second pass. On the other hand, no series of compressions or stretches can obscure the difference between the two time series of figure 1. As a result, the distance between these two time series is sure to be greater than the difference between the 1 and 5 second pass experiences.

The dynamic warping algorithm can be applied to the task of warping one experience to another to find the temporal mapping which minimizes the difference between arbitrary subsets of sensor time series. By using subsets of correlated sensors, rather than all 42 sensors at once, we relieve the algorithm of possible interference effects, whereby subtle patterns such as those found in the visual sensors may be overshadowed by wild or noisy patterns in other sensors, such as the sonars.

The minimized mean square difference between two experiences computed by the dynamic time warping algorithm is taken to be our distance metric for clustering. We employ a simple group-average, agglomerative clustering algorithm to collect together experiences whose warping distances are small. This type of clustering scheme begins with each experience as a separate cluster, and merges the two clusters with the lowest group-average distance between experiences in the two clusters. Merging continues until the distribution of within-cluster distances is judged significantly different than the distribution of between-cluster distances.

2.2 Clusters as Operator Models

The result of the clustering algorithm is a partitioning of the agent's experiences in which each cluster corresponds to one possible outcome of an activity. We would like clusters to serve as predictive operator models for planning, and to make them more practical for this purpose, we subject clusters to further transformation.

First, we represent each cluster by a *prototype experience*. A cluster's prototype experience is created by averaging the sensory time series of the experiences that belong to that cluster. This averaging serves to smooth out noisy sensor data as well reduce the volume of data that must be considered to make a prediction.

Finally, the prototype experience's time series are redescribed by a piecewise linear fit. Recall the variable nature of an activity's duration: an activity may last for 1 second, 5 seconds, or 2 hours. The crucial information from a planning perspective is the rate at which things change. A piecewise linear fit retains rate information while allowing the planner to project over variable durations. We will return to the issue of prediction in section 3.

Our piecewise fitting algorithm is a divide-and-conquer approach to line fitting that passes a window of size Δ over the data, calculating the difference between the slope of the least-squares fit on the left and right sides of the window. The t statistic is then computed for the most drastic change in slope over the data. If the t value is significant at some threshold α, the algorithm splits the time series about the most drastic slope change, and recursively invokes the line fitting algorithm on the two pieces. The algorithm continues until the most drastic slope change in every piece is no longer significant. Each piece is then described by its least-squares fit. Two screen shots of the piecewise fitting algorithm's results are shown in figure 2.

To recap, modeling entails 3 processes. First, the robot's experiences with its actions are clustered by their dynamics into groups of similar outcome. Each cluster is then represented by a prototype experience, and this experience's sensory traces are redescribed by a piecewise line fitting algorithm. With this representation, the robot is ready to use clusters for planning.

3 Continuous Space Planning

To generate activities by planning as we propose, we require a generative, means-ends analysis planner, similar to the STRIPS and ABSTRIPS planners [4]. Unfortunately, neither of these planners was designed to cope with a continuous environment. Systems such as the PRODIGY [1] planner adapt STRIPS-like planning to the domain of mobile robotics by discretizing the action and state spaces into discrete behaviors and predicates that can be reasoned about symbolically. Others, such as the Zeno planner[3], introduce rates of change and temporal quantification to introduce temporal constraints and continuous change to symbolic planning. Our approach is different in that it works directly at the sensor level, creating low-level controllers to achieve goals using learned operator models.

This process of creating low-level controllers out of learned operator models is called *operator instantiation*. We next describe a language for implementing simple controllers before detailing the operator instantiation process.

3.1 A Control Language

Our goal in designing a control language was to create a language that was expressive enough for a human engineer to implement hierarchical controllers of arbitrary complexity, but simple enough for a planner to generate the same type of controllers to achieve its own goals. The resulting language, ACL, possesses a simple LISP type syntax with very few primitives that are powerful enough to implement most conceivable behaviors.

The four basic constructs of ACL are INITIATE, TERMINATION-CONDITIONS, CONDITIONAL, and TERMINATE-ON. The first two are for the activation and deactivation of activity, the third provides conditional branching, and the last construct declares when a control program should terminate. The following control program is a useful illustration of the programming language:

```
(define-control-program grasp
  (run-to-completion foveate)
  (initiate (move+100)
     (termination-conditions
        (success (= gripper-beam 1))))
  (run-to-completion gripper-up)
  (terminate-on
```

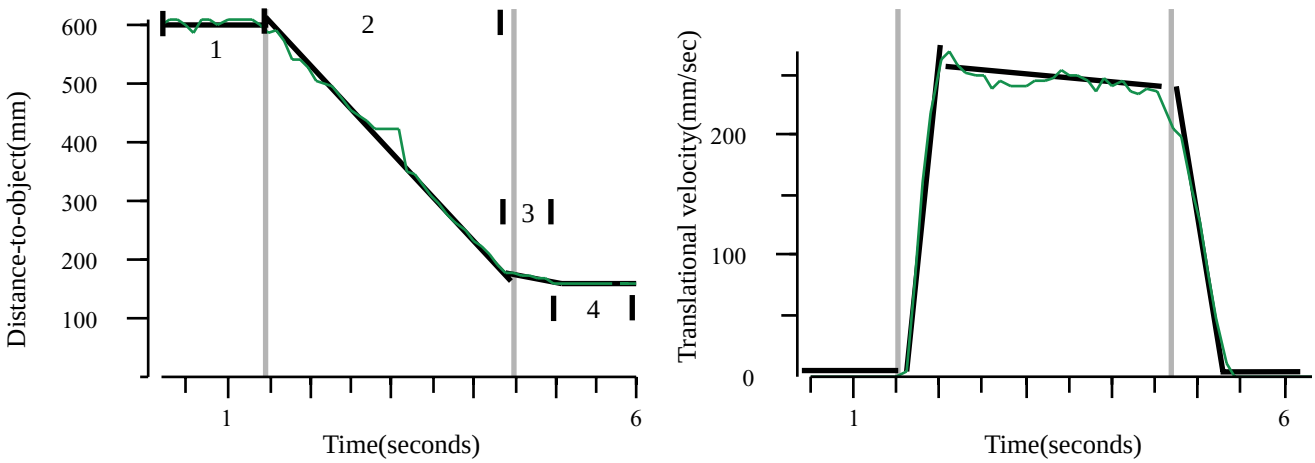

Figure 2: Two sensor traces from a move activity shown with their piecewise linear fittings. On the right is a trace of translational velocity. On the right is a trace of a visual sensor that records the distance to a blob being tracked by the vision system. The grey vertical bars indicate the activation and deactivation times of the move activity, while the number markers on the left graph denote the phases generated by the piecewise linear fit.

```
(success :success)
(failure :failure)))
```

The above program, named "grasp", finds an object, moves the agent towards it, and grasps it. The first line of the control program makes use of a macro RUN-TO-COMPLETION, which simply runs a subprogram until that subprogram completes. The next form initiates the MOVE+100 activity and specifies that it should be terminated when the gripper beams are broken, noting that this condition is to be called "success". The program next raises its gripper, and finishes with the TERMINATE-ON form, which terminates itself and indicates "success" or "failure" to its parent, depending on the status of the three subprograms that the grasp program initiated.

ACL provides the necessary language for specifying controllers for agents operating in a continuous space. The INITIATE/TERMINATION-CONDITIONS form is the elementary building block of these controllers, and the output of the operator instantiation process.

3.2 Operator Instantiation

Recall that operator instantiation is the process by which a known operator model is used to achieve a specific goal. In ACL terms, operator instantiation is the process by which an INITIATE/TERMINATION-CONDITIONS block is generated from an operator model intended to achieve a goal.

It will help in the discussion of operator instantiation to break operator models into three intervals. The first interval, called the *preactivation interval*, describes the sensory patterns observed before the operator is activated. The second interval, the *activation interval* represents experience during the lifetime of the activity. Finally, the *deactivation interval* describes what happens after an activity has been deactivated. These intervals are delineated by the vertical lines of figure 2.

The intervals of interest to operator instantiation are the activation and deactivation intervals. The activation interval represents the behavior that is under the control of the agent. During this interval, the agent can exercise limited control over what happens by deactivating the activity. The deactivation interval represents behavior that is not under control of the agent. It is what happens after the agent has deactivated the activity: it is what happens after the Pioneer-1 deactivates the move activity, for example.

The key to operator instantiation is in computing when to deactivate an activity such that the agent will come to rest at the goal. Consider the second step of the ACL program of section 3.1 in which an agent moves toward an object to be grasped. In this example program, the gripper beam is used as the trigger for deactivating the move activity. It is a simple but less than ideal stopping criteria, as the gripper beam is a mere 10 millimeters ahead of the back of the gripper. If it takes more than 10 millimeters for the robot to stop, then we can expect this activity to reliably bump into the object to be grasped as it comes to a stop. This may be unacceptable if the object can be tipped over.

A more reliable solution would be to use the Pioneer's vision system to measure the robot's distance to the object, and to account for the deceleration of the robot such that the movement step would end with the dis-

tance equal to zero. The task of operator instantiation is to compute the termination conditions for the move activity that will result in the goal being achieved after any effects of the deactivation phase have occurred.

The deactivation point is computed by starting with the goal value and working in reverse through the piecewise fits of the operator model's deactivation phase. Each piecewise fit that is not part of the activation phase is taken to be a fixed, unavoidable effect, and is added to the goal value. Once the cumulative effects of the deactivation phase have been computed, this new modified goal value is taken as the deactivation point of the activity.

Consider again the example of approaching an object to grasp it. The selected operator model's prototype for DISTANCE-TO-OBJECT is pictured in the left-hand graph of figure 2. Starting with the goal value of zero, the operator instantiation begins by adding the effects of phase 4. The slope of phase 4's linear fit is zero, and so this phase has no effect on the goal value. Phase 3 is next, and has a slope value of $-80.3\frac{mm}{sec}$. The duration of this phase is fixed, as the deceleration of the robot is constant, and is approximately one half second long. The effect is computed to be $-40.15mm$, and is subtracted from the original goal value. The next phase, phase 2, is the activation phase. Since this phase can be deactivated at will by the robot, the computation stops, and DISTANCE-TO-OBJECT=$40.15mm$ is taken to be the deactivation criteria for the move activity. The revised movement block of the grasp program would be:

```
(initiate (move+100)
  (termination-conditions
     (success (<= distance-to-object 40.15))))
```

4 Conclusions and Future Work

We propose an end to end system for explaining the development of structured activities in an intelligent agent. The system divides development into two processes: a modeling process whereby the agent learns about the dynamics of the activities that it already can perform, and a planning process whereby the agent composes existing activities, in a disciplined way, into new activities.

The modeling process must account for the fact that a single activity may have several possible outcomes, and that a single outcome may be of varying duration. We generate operator models by first clustering experiences by their outcome, and then redescribing these clusters with piecewise linear models. Currently, we have run the modeling system on some 200 experiences of the Pioneer's primitive actions and generated 102 clusters using different subsets of the Pioneer's primitive sensors. These clusters separate experiences into groups of categorically different outcomes: unobstructed versus obstructed moves, and passing an object on the left versus passing one on the right, for example.

We have implemented operator instantiation and tested it with our clusters on a small number of simple goal-operator pairs, such as the DISTANCE-TO-OBJECT example in the discussion of section 3.2. In these cases, the operators are instantiated by ACL code that correctly projects the termination criteria to arrive at the simple goals. Continued testing in this area is in order to test the reliability of the piecewise fits, and to ensure that treating the deactivation phases as fixed intervals is expressive enough to handle all activity outcomes.

Acknowledgments

This research is supported by the Defense Advanced Research Projects Agency, AFOSR, and USAF under contracts F49620-97-1-0485 and F30602-97-1-0289. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation herein. 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 DARPA, AFOSR, USAF, or the U.S. Government.

References

[1] Jaime Carbonell, Oren Etezioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. Prodigy: an integrated architecture for planning and learning. In *Working Notes of the AAAI Spring Symposium on Integrated Agent Architectures*, 1991.

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

[3] J. Penberthy and D. S. Weld. Temporal planning with continuous change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Menlo Park, CA, 1994. AAAI/MIT Press.

[4] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence Journal*, 5(2):115–135, September 1974.

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