Combining Learned Discrete and Continuous Action Models

Joseph Z. Xu and John E. Laird
Department of Computer Science and Engineering, University of Michigan
2260 Hayward Street, Ann Arbor, MI 48109-2121 USA
{jzxu, laird}@umich.edu

Abstract

Action modeling is an important skill for agents that must perform tasks in novel domains. Previous work on action modeling has focused on learning STRIPS operators in discrete, relational domains. There has also been a separate vein of work in continuous function approximation for use in optimal control in robotics. Most real world domains are grounded in continuous dynamics but also exhibit emergent regularities at an abstract relational level of description. These two levels of regularity are often difficult to capture using a single action representation and learning method. In this paper we describe a system that combines discrete and continuous action modeling techniques in the Soar cognitive architecture. Our system accepts a continuous state representation from the environment and derives a relational state on top of it using spatial relations. The dynamics over each representation is learned separately using two simple instance-based algorithms. The predictions from the individual models are then combined in a way that takes advantage of the information captured by each representation. We empirically show that this combined model is more accurate and generalizable than each of the individual models in a spatial navigation domain.

Introduction

An intelligent agent that wishes to plan in an environment must have an internal model of how the environment changes in response to its actions. We call such a model an action model. Traditionally, action models have been provided to the agent a priori using hand-coded representations such as STRIPS operators. However, if the agent needs to plan in a novel environment, then it must learn the action model from experience. We call this learning process *action modeling*.

Traditionally, action modeling has been studied in the context of learning STRIPS operators (Carbonell & Gil 1990, Wang 1995, Pasula et. al. 2007, Xu & Laird 2010). These methods involved learning the pre- and post-condition lists of a given set of STRIPS operators by observing examples of their execution. These methods are confined to relational domains.

There has also been work in robotics on learning models for continuous environments. Most of these methods represent the environment state as a vector of real numbers and use function approximators to learn a continuous function $f(x,u) \rightarrow y$, where x is the current state, u is the action taken, y is the resultant state. Many general function approximation methods such as radial basis functions, neural networks, and regression trees have been applied to this problem. In this paper we will use locally weighted learning (Atkeson et. al. 1997) for its simplicity and power.

Realistic problems exhibit a combination of discrete and continuous dynamics. For example, the trajectory of a ball traveling through the air follows a smooth and continuous function, but undergoes a sudden qualitative change when with the ball hits the ground. While discrete and continuous action modeling can individually address these domains to some extent, it is not natural or easy. A complex scene consisting of many objects can be approximately described by a large number of spatial predicates such as left-of, on-top, etc., and changes to the scene can be encoded as predicate value changes. However the Poverty Conjecture (Forbus et al. 1991) argues that no single general set of spatial relations can fully capture all the information encoded in a continuous scene. As an example, consider a fast-moving car turning a corner. It is difficult to use a set of spatial predicates to distinguish between cases where the car will successfully make the turn and cases where it will not.

Continuous models can accurately represent such continuous dynamics, but are weak at capturing relational dynamics. Although they are capable of generalization by smoothing the learned function around the neighborhoods of training examples, continuous models have difficulty generalizing across higher-order regularities such as how the trajectories of two balls change in a collision unless relevant metrics such as the distance between the two balls are explicitly included in the state vector.

Recent work in robotics (Plaku & Hager 2010, Choi & Amir 2009) has explored combining robot motion planning

techniques with classical planning techniques in mixed discrete/continuous domains. However, these methods assume that the agents have hand-coded models of both the continuous and relational dynamics of the environment. Hybrid automata are a popular approach to modeling these domains in the controls literature, but we have not found any work on learning hybrid automata structure from experience.

In this paper, we integrate a learning algorithm for continuous action modeling with a learning algorithm for relational action modeling. Although the two algorithms operate independently with different representations and on different time scales, they are embedded together in the Soar cognitive architecture (Laird 2009). The combination outperforms the individual approaches in an example domain that has both continuous and discrete dynamics.

The Rooms Environment

The test environment that we use throughout the paper is called the Rooms environment. This environment consists of an agent embodied as a moving square navigating among a set of connected rooms. Each room has one switch in the form of a floor tile and one door that leads to another room. Initially all doors are closed. When the agent moves over a switch, the door leading to the next room slides open. The environment's state is represented as a vector of real numbers (Figure 1.a). The vector contains the *x, y* coordinates of the agent, rooms, doors, and switches. The agent's output is a vector of two numbers representing its vertical and horizontal velocities. The environment has discrete time and changes in lock step with the agent. At every time step, the environment sends its state vector to the agent, who must then specify its horizontal and vertical velocities for the next time step. The environment then updates its state based on those inputs and repeats the loop.

System

The agent is implemented in the Soar cognitive architecture. Soar combines multiple memory, learning, and decision mechanisms. Soar's working memory and long-term procedural memory mediate the interactions between all other components. Working memory encodes information as a labeled, directed graph structure. Production rules test for the presence of specific substructures in the working memory graph and fire when those structures are present, making changes to working memory. Productions can initiate action in and receive responses from other Soar modules by testing for and creating structures in designated parts of working memory that are monitored by the components.

Soar also has an episodic memory (Nuxoll & Laird 2007, Derbinsky & Laird 2009) that automatically records the state of working memory at fixed intervals, a semantic memory for storing long-term declarative facts, and the Spatial Visual System (SVS; Wintermute 2010) for reasoning about continuous spatial scenes. In this paper we are only concerned with episodic memory and SVS.

Episodic Memory

The episodic memory mechanism stores periodic snapshots of working memory in an independent long-term memory. Each snapshot is called an episode. Stored episodes can be retrieved into working memory by querying episodic memory with a cue. The cue specifies a subset of the episode that the agent wants to retrieve. The episodic memory mechanism returns the episode that shares the most common substructure with the cue. We describe later how we use this structure-matching mechanism to implement a relational action model learner.

SVS

SVS mediates between a continuous environment and Soar's symbolic working memory. The environment deposits its state in SVS as a vector of continuous numbers (Figure 1.a). The numbers represent the centroid positions (x, y, z), rotations (roll, pitch, yaw), and scaling factors (sx, sy, sz) of all objects in the environment. Associated with each object is also a 3D geometry defined as a convex hull over a point cloud. For the Rooms environment, we use only two dimensional coordinates and ignore rotation and scaling.

SVS encodes the environmental state in a continuous spatial scene buffer (Figure 1.b). The agent reasons about the continuous scene by querying for the truth values of spatial literals. For example, the agent can ask SVS whether the convex hulls of two objects intersect. Currently SVS contains a fixed set of basic innate spatial predicates, including intersection, containment, and alignment. The result of a query is placed into Soar's working memory. By using an appropriate set of queries, the agent can create a relational representation of the spatial scene in working memory. For the Rooms environment, the agent queries for containment relationships between rooms and the agent, doors, and switches, and also intersection relationships between the agent and switches and between rooms. The relational state is encoded as a bipartite graph with a set of literal vertices and a set of object vertices (Figure 1.c). Edges connect each literal vertex with the object vertices corresponding to its arguments. Each literal vertex is labeled as either true or false with the result of the corresponding query.

One of the contributions of this paper is the addition of continuous model learning and continuous control

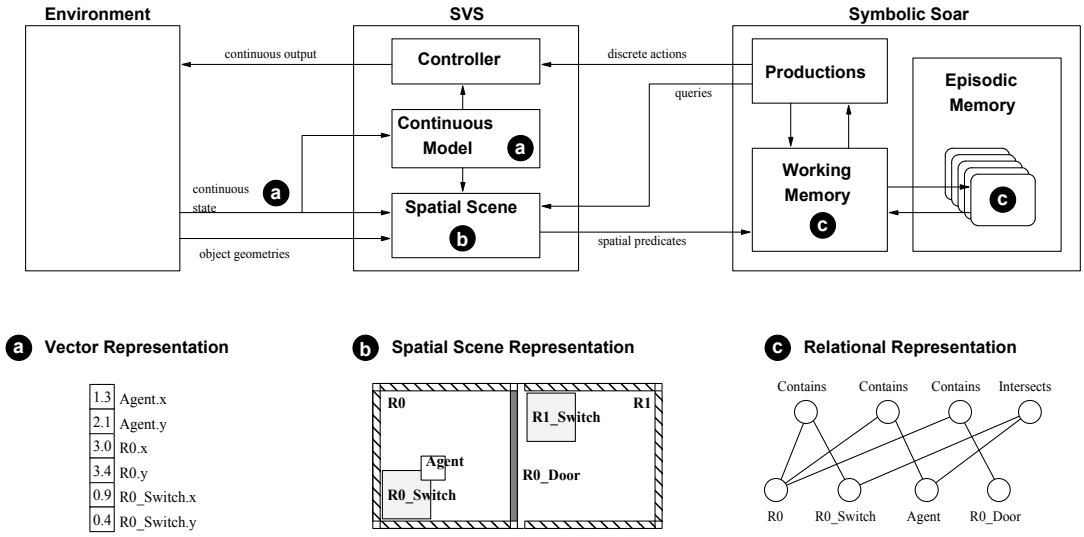

Figure 1. System and representations overview.

subcomponents to SVS. The model learning component is described in detail below.

Continuous Controller

The controller mediates between the discrete actions generated by production rules and the continuous outputs to the environment. Discrete actions are defined as desired changes to predicate values. For example, when the agent is not on a switch, it can instruct the controller to make the literal *intersect(agent, r0_switch)* true, i.e., it wants to move to the switch. In order to translate this command into continuous outputs, the controller needs to know what outputs will take the environment towards a state where the predicate is true.

We associate each spatial literal with an objective function over continuous states that has a global minimum at a state where the literal is true. For example, we take the objective function for the literal *intersect(a, b)* to be the squared Euclidean distance between the centroids of *a* and *b*. Given a command to change the value of a certain literal, the continuous controller follows the gradient of the objective function until either the desired literal is achieved or a local minimum is reached. It does this by sampling the range of possible outputs at each time step, predicting the next state resulting from using that sample with the learned continuous model, and calculating the value of the objective function at that state. It then chooses the output that results in the lowest next objective value, and repeats. Hence the controller implements a greedy search.

Even though the agent specifies only one literal to change per discrete action, it is often the case that other literals will change while executing the action. For example, if the agent is in room *r1* and moves to make *intersect(agent, r2_switch)* true where *r2_switch* is located in room *r2*, then it will also have made *contains(r2, agent)* true and *contains(r1, agent)* false in the process. Furthermore, the controller can reach local minimums before achieving the desired literal. For example, if the agent wishes to enter a room whose door is closed, then the controller will become trapped in a local minimum next to the door. The agent must then take a different discrete action to make progress.

Learning Action Models

In the context of our system, we define the action model to predict the literal changes in the relational state that results from taking a discrete action. Even though our system is also capable of making predictions about changes to the continuous state, we are mainly concerned with relational predictions because goals are usually given to the agent as relational conditions. For example, a goal in the Rooms environment may be for the agent to move into a certain room *rg*, which corresponds to making *contains(rg, agent)* true.

Learning Continuous Action Models

Given a vector state x and a motor output u, the purpose of the continuous model is to predict the resultant vector state y. In other words it must approximate the environment function $f(x,u) \rightarrow y$. We use locally weighted learning to learn the model. Locally weighted learning is an instance-based learning technique that combines nearest neighbor and linear regression. For each time step elapsed in the environment, the model stores the observed tuple (x,u,y) in a database. To make a prediction for a state and output (x',u'), the model first chooses k closest training samples to (x',u') and performs a linear regression on them. The

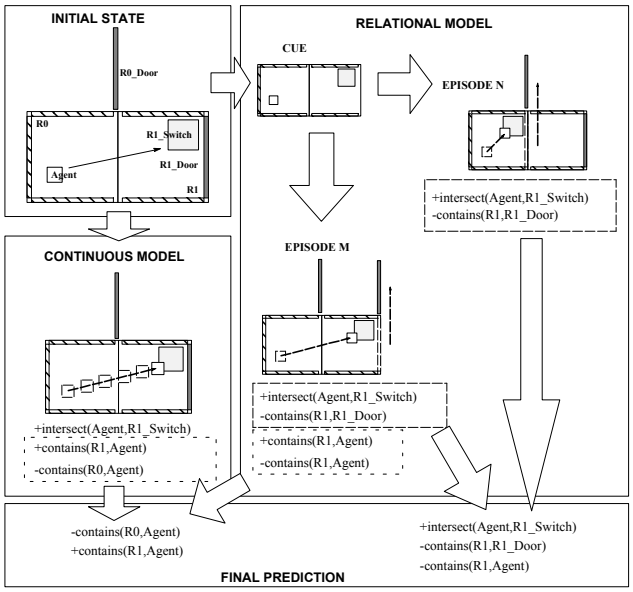

Figure 2. Example walkthrough of the combined model making a prediction. Two predictions are made by the relational model, one that predicts the agent will change rooms (*contains(R0, Agent)* is made false and *contains(R1, Agent)* is made true) and another that does not. Both agree on the opening of the door (*contains(R1, R1_Door)* is made false). The continuous model is used to resolve the disagreement in favor of the room change. The final prediction includes both the room change and the door opening.

resulting local linear model is then used to make the prediction. This approach is online and incremental, and has been shown to provide good generalization as well as being able to fit arbitrarily complex functions. We set k at 20 for the experiments in this paper.

To make a prediction about which literals change as a result of performing some discrete action, SVS simulates the controller's trajectory using the continuous model rather than sending it to the environment. At each step, the model updates the spatial scene based on its prediction. These changes to the scene appear in working memory as if they were caused by the environment. This kind of transparency allows the same set of production rules to control both actions taken in the environment as well as in simulation.

Learning Relational Action Models

The relational model makes direct predictions about how the relational state changes as a result of discrete actions issued to the controller. This is similar to the problem of learning STRIPS operators from example transitions. We use an instance-based algorithm that is both online and incremental first described in (Xu & Laird 2010).

The agent stores each transition it experiences in its episodic memory as a pair of adjacent episodes. To make a prediction about which literals change from taking action a in state s, our algorithm first queries episodic memory for a state t that is similar to s where a was also performed. We retrieve t with a cue that includes the discrete action, a subset of the literal graph, and type information (i.e. containment literals can only map to containment literals and doors can only map to doors) but not the names of objects. Therefore, the algorithm generalizes training instances to all situations with similar relational structure modulo object names.

Next, the algorithm retrieves episode t' that immediately follows t. The two episodes are compared to find the literal changes that occurred, and then those changes are analogically mapped back into s. The analogical mapping algorithm is similar to the one used in the Structure Matching Engine (Falkenheiner 1989). The intuition is that since s and t are relationally similar, the changes that occur between t and t' will be relationally similar to the changes that will occur in s.

The Combined Model

We now describe how the two models are combined to make a final prediction. Figure 2 shows an example walkthrough of this process. The agent first makes multiple predictions with its relational model using different retrieved episodes. Because important spatial information may be missing from episodic memory or the retrieval cue, episodes with different spatial properties will structurally match the initial state equally well. In the example, the cue used does not specify the relationship between the agent's current room and the room containing the switch, so one of the retrieved episodes is of the agent moving to a switch in the same room (N) while the other is of the agent moving to a different room (M).

In order to determine which retrieval is correct, the agent makes a prediction using the continuous model. Since the continuous model's prediction is based on a simulation instead of an analogical mapping, it takes into account all the spatial information in the scene. In the example, the continuous model correctly predicts that the agent will enter another room when it tries to hit the switch. However, the continuous model is poor at generalizing over relational dynamics, and it doesn't predict that hitting the switch will open the door, because in the agent's training examples the switch was in a different location. To make the final prediction, we assume that the literal changes agreed upon by all the relational model predictions are valid, and for the literal changes that are not in agreement, we take the predictions from the continuous model. In the example, the agent combines the agreed upon predictions about the door opening with the predictions about the agent changing rooms from the continuous model to make the correct final prediction.

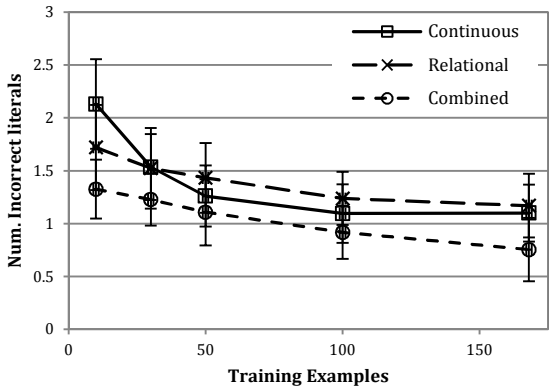

Figure 3. Average number of incorrect literals per prediction for the three types of models. Error bars indicate one standard deviation.

Experiments

To measure the advantage gained by combining the relational and continuous models, we tested the prediction accuracy of the continuous model alone, the relational model alone, and the combined model on random transitions in the Rooms environment.

To train the agent for each condition, we repeatedly instantiate the agent in randomly generated states in the Rooms environment. The state always consists of four rooms with one switch each and doors separating them. The doors are randomly chosen to be open or closed. For a single agent, the positions of the switches in the rooms are fixed. This prevents the continuous model from smoothing over training examples where the same switch is in different places and as a result incorrectly predicting that the switch moves.

The agent first wanders for ten steps, adding ten samples to its continuous model, and then attempts to change a randomly chosen literal. This results in the agent traversing a trajectory along the literal's gradient and further adds a number of samples to its continuous model. This entire training sequence adds exactly two episodes to the agent's episodic memory: an episode before attempting to change the literal and an episode after. We repeat this training procedure on all 168 combinations of door configuration and discrete actions for one configuration of switches.

In preliminary experiments we found that the continuous model trained in this way did not consistently learn that doors impeded movement. This is because when making a prediction for a location close to a door, the nearest neighbor search will return training examples of the agent on both sides of the door. While the training examples on one side of the door suggest that the agent will be blocked by the door, the examples on the other side suggest that the agent will move unimpeded. The weighted linear regression smoothes over these examples and predicts that the agent's movement will be slowed by the door but not stopped. Because allowing movement through doors is tantamount to ignoring all spatial aspects of the problem, we manually added knowledge to the agent to stop a simulation as if a local minimum was encountered whenever the agent intersected a door. As discussed before, not being able to accommodate abrupt changes in the dynamics of the world is a general problem with continuous function approximators. Since we are using the continuous model to primarily help in generalization of the relational model, this piece of a priori knowledge does not invalidate our results. However, it is a problem we will address in future work.

At the intervals where the agent has experienced 10, 30, 50, 100, and 168 training examples, we test the agent's prediction performance on a distinct test set of 50 randomly generated combinations of initial states and discrete actions. The same 50 test combinations are used at each interval. We measure the agent's performance by the average number of incorrect literals it predicts. 1.75 predicates changed in each test transition on average, with a maximum of 10. The results averaged over 24 random switch configurations are shown in Figure 3. As the plot shows, while the relational model alone initially performs better than the continuous model, presumably due to its ability to generalize better on few training examples, its performance plateaus faster and it ends performing slightly worse than the continuous model. The combined model both generalizes better and has better asymptotic performance than either of the individual models.

To better understand the performance of each model, Figure 4 breaks the error into two cases. The stacks in a group from left to right correspond to the continuous model, the relational model, and the combined model. The bottom bars in the stack show the number of missed literals, meaning the model predicted that a literal remains unchanged when in fact it changed. The top bars show the number of extra literals, meaning the model predicted that a literal changes when in fact it did not.

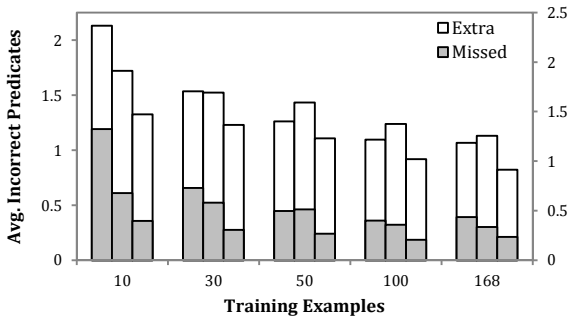

Figure 4. Average number of extra and missed predicates per prediction. The bars from left to right are continuous model only, relational model only, and combined model.

Most of the discrepancy between the performance of the continuous model and the combined model consists of missed literals. This is due to the fact that the continuous model cannot capture the causal link between stepping on a switch and opening a door. The continuous model only learns that a door opens (changes position) when the agent moves to the absolute location of the corresponding switch in the training room configuration. In the test configurations, the switch positions are randomly permuted, but the continuous model still associates the original position of the switches with door opening. The problem is that the relative distances between the agent and the switches are not considered by the distance metric of the nearest neighbor algorithm underlying the continuous model. The relational model can capture this dynamic because its distance measure is based on relational structure and can be conditioned on the *intersect(agent, switch)* literal which is invariant of the absolute positions of the agent and switch. Note that even though the continuous model does better overall than the relational model, it consistently misses more literals.

The cause of the discrepancy between the relational model and the combined model is more convoluted. By examining the traces of individual runs, we see that in some cases the relational model predicts that the agent cannot enter a room or step on a switch in a different room because the doors between the start and end positions were closed in the retrieved episodes. This results in missed literal changes. In other cases the exact opposite problem occurs and results in extra literal changes. The combined model is robust to the second case due to the predictions from the continuous model. However, in cases involving stepping on switches in other rooms, even though the continuous model successfully predicts that the agent can intersect the switch and the relational model successfully predicts that stepping on the switch opens a door, the combined model only takes the intersection of these predictions, resulting in it not predicting the door to open. This is a shortcoming of the way we are combining the predictions and should be improved in future work.

Conclusion

In this paper we have described an approach to integrating continuous and relational model learning techniques in a single agent architecture. We showed that by combining the predictions made by the two types of models, we can make predictions that both generalize better on small numbers of training examples and are more accurate in the limit. Although more sophisticated and specialized continuous modeling techniques can probably yield better results, our point is that a hybrid of relatively simple techniques can capture some of the same complex dynamics that the specialized algorithms are designed for.

As discussed previously, one of the shortcomings of locally weighted learning is that it incorrectly smoothes over training examples that are qualitatively different. This has not been addressed here because this paper is primarily concerned with improving relational predictions using the continuous model as additional knowledge. Our planned immediate future research direction is to explore ways to prevent these types of problems using knowledge encoded in the relational model.

Another interesting research direction concerns the choice of spatial queries used to build the relational state. This is an important factor in determining how regular and learnable the relational dynamics of the environment are. In this paper we designed the queries by hand to be sufficient to learn a good model over. It would be interesting to explore approaches for the agent to learn these through experience.

Acknowledgment

The authors acknowledge the funding support of the Office of Navy Research under grant number N00014-08-1-0099.

References

Atkeson, C., Moore, A., and Schaal, S. 1997. Locally Weighted Learning. *AI Review 11*. 11-73.

Carbonell, J. G. and Gil, Y. 1990. Learning by Experimentation: The Operator Refinement Method. *Machine Learning, An Artificial Intelligence Approach, vol 3*. Morgan Kaufmann, San Mateo, California.

Choi, J. and Amir, E. 2009. Combining Planning and Motion Planning. *ICRA 2009*.

Derbinsky, N. and Laird, J. E. 2009. Efficiently Implementing Episodic Memory. *Proc. of 8th ICCBR*.

Falkenhainer, B., Forbus, K. D., and Gentner, D. 1989. The Structure-Mapping Engine: Algorithms and Examples. *Artificial Intelligence 41*.

Forbus, K.D., Nielsen, P., & Faltings, B. 1991. Qualitative spatial reasoning: the CLOCK project. *Artificial Intelligence* 51(1-3)

Laird, J. E. (2008). Extending the Soar Cognitive Architecture. *Proc of the First Artificial General Intelligence Conference.*

Nuxoll, A. M. and Laird, J. E. 2007. Extending Cognitive Architecture with Episodic Memory. *Proc. 22nd AAAI.*

Pasula, H., Zettlemoyer, L., and Kaelbling, L. 2007. Learning Symbolic Models of Stochastic World Domains. *JAIR 29*. 309-352.

Plaku, E. and Hager, G. 2010. Sampling-based Motion and Symbolic Action Planning with Geometric and Differential Constraints. *ICRA 2010*.

Wang, X. 1995. Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. *Proc. 12th ICML.* 549-557.

Wintermute, S. 2010. Abstraction, Imagery, and Control in Cognitive Architecture. PhD. Thesis, University of Michigan.

Xu, J. and Laird, J. E. 2010. Instance-Based Online Learning of Deterministic Relational Action Models. *Proc. 24th AAAI.*