Instance-Based Online Learning of Deterministic Relational 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

We present an instance-based, online method for learning action models in unanticipated, relational domains. Our algorithm memorizes pre- and post-states of transitions an agent encounters while experiencing the environment, and makes predictions by using analogy to map the recorded transitions to novel situations. Our algorithm is implemented in the Soar cognitive architecture, integrating its task-independent episodic memory module and analogical reasoning implemented in procedural memory. We evaluate this algorithm's prediction performance in a modified version of the blocks world domain and the taxi domain. We also present a reinforcement learning agent that uses our model learning algorithm to significantly speed up its convergence to an optimal policy in the modified blocks world domain.

Introduction

Having a model of the effects of one's actions in the environment allows one to internally simulate the outcomes of possible action trajectories, saving time and avoiding undesirable outcomes. We call these models *action models*. More precisely, an action model is a function from state-action pairs to after states.

Our research focuses on how a persistent agent can incrementally learn an action model while it experiences its environment. One straightforward way of doing this is to maintain a probability distribution of after states for each transition separately. Many model-based reinforcement learning (Sutton & Barto 1998) techniques use this model learning method (Barto et. al. 1995; Kearns & Singh 2002; Sutton 1990). While this method is applicable to any Markovian environment, its lack of generalization across states precludes the learned models from making predictions about unexperienced transitions. Imagine a robot that learns that driving off a particular table results in breaking its camera. Without the ability to generalize its model, the robot will repeat the error for every other table it encounters.

In this paper, we present a novel method for incrementally learning generalizing action models of deterministic environments expressed in relational representations. Our method involves the integration of many disparate functional modules in a general cognitive agent including procedural memory, episodic memory, planning, RL and analogy. We demonstrate that the learning algorithm performs well without modification in two different domains and that it can be integrated with Q-learning, intermixing model learning and policy learning, which significantly improves overall policy learning in certain types of domains.

Assumptions

We are interested in agents with continual existence, so learning must be incremental and integrated with performance. We restrict our attention to Markovian, fully observable, deterministic domains. Domain states are specified by a set of object symbols O, a mapping of object symbols to type symbols T, and a set of relations with fixed arity $R = \{R_1^k \subseteq O^k, R_2^l \subseteq O^l, R_3^m \subseteq O^m, \ldots\}$ that apply over the objects. Each relation $R_i^k \in R$ is the set of all k-tuples for which the relation is true. As an example, consider a state in the well known blocks world domain with three blocks A, B, and C, where A is on top of B, B is on top of C, and C is on the table. In our notation, the state can be described as

$$O = \{A, B, C\}$$
$$T = \{A \to block, B \to block, C \to block\}$$
$$R = \{ontop, ontable\}$$
$$ontop = \{\langle A, B\rangle, \langle B, C\rangle\}$$
$$ontable = \{\langle C\rangle\}$$

Actions are represented as a name symbol and a tuple of objects that are parameters, such as $MoveToTable(A)$. Performing an action causes a single state transition. State transitions can involve adding or removing objects from O or adding or removing tuples from any of the relations. Continuing the previous example, if we take the action $MoveToTable(A)$, the after state would be

$$O = \{A, B, C\}$$
$$T = \{A \to block, B \to block, C \to block\}$$
$$R = \{ontop, ontable\}$$
$$ontop = \{\langle B, C\rangle\}$$
$$ontable = \{\langle A\rangle, \langle C\rangle\}$$

We assume that changes in a domain arise only from an agent's actions, and that the outcomes of actions depend only on the relations between objects and not the objects' symbols, which are arbitrary. We also assume that an agent gains experience in a domain only by taking actions in it.

Thus, the training instances for learning action models are local to the agent's trajectory through the environment, rather than being a uniform random sampling of all possible transitions. Therefore, the model learning algorithm must cope with biased training examples, and must modify the model as new examples are collected. Finally, we assume that an agent knows which actions are valid to execute in every state, but cannot directly reason about the action preconditions.

Related Work

There are several existing pieces of work on learning action models, none of which covers all the criteria discussed above: being online, incremental, relational, and generalizing. Wang's system (Wang 1995) learns STRIPS operators from observing multiple expert planning traces, creating the most general operators that cover all traces. This method requires representative training instances and access to a simulator for the domain, violating our assumption that the agent has no information about its environment beyond its own experiences.

TRAIL (Benson 1995) has a learning component that uses inductive logic programming (ILP) to learn preconditions for actions. Although training examples are obtained online from plan execution, the algorithm is not incremental in that it must wait for a sufficient number of positive and negative instances before attempting ILP. It is unclear from the paper how this number is determined and how the system deals with unrepresentative training data inherent to online sampling.

Pasula et. al. (2007) developed an algorithm that generates a set of relational rules from training examples that describe the distribution of outcomes of probabilistic actions. It does this by searching the space of possible rules for a set that balances training data consistency with complexity. This approach also requires the training data to be a representative sampling of the entire space of transitions because it cannot dynamically incorporate new training instances, which violates our requirement that the agent collect experiences online.

Diuk et al. (2008) introduce a formalism called Deterministic Object-Oriented MDPs and an algorithm for learning deterministic action models within this formalism. The algorithm starts with specific rules based on single transition observations and moves to more general rules as more transitions are experienced and observed effects are combined. The algorithm is incremental and online, but it does not use relational representations, so it unable to generalize across object instances.

Algorithm Description

Our algorithm is implemented in Soar (Laird 2008), a cognitive architecture that incorporates the necessary learning, memory, and control mechanisms to create agents with the capabilities described above. Communication

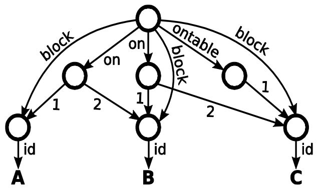

Figure 1. Working memory encoding of blocks world state

between all mechanisms occurs via Soar's shared working memory, which represents data as a labeled graph. Soar has an episodic memory module (Derbinsky 2009; Nuxoll 2007) that effectively takes snapshots of its working memory state, called episodes, at regular intervals and stores them in chronological order. An episode can be retrieved by matching its contents to a cue or by next and previous temporal relationships. In cue based retrieval, a working memory graph is provided as a cue, and the episode with the highest degree of structural match to the cue is retrieved. The degree of structural match is calculated as the number of vertices shared by the episode and cue that have identical contexts. An episode that has a subgraph isomorphism to the cue represents the best possible match. Ties are broken by episode recency. Derbinsky & Laird (2009) present evidence that cue-based retrieval is tractable even with one million stored episodes. Soar also has a reinforcement learning module that tunes agent behavior with Q-learning (Nason & Laird 2004).

Soar's episodic memory provides the basis for learning an action model by automatically storing all state transitions experienced by the agent. However, to support predictions in novel situations, the agent must also be able to generalize stored transitions with analogy. We encode the relational representation of the task in working memory, as shown in Figure 1. To make a prediction for taking action a in state s, the agent first retrieves a previously encountered state s_1' that is similar to s, in which it also performed a, and the following state s_2'. It then compares those states to determine how s_1' changed due to it taking action a. s_1' is analogically mapped onto s to obtain a feature correspondence, which is then used to map the changes calculated for s_1' onto s to obtain the prediction. These four major steps: retrieval, finding changes, feature mapping, and mapping changes, make up the subsections below. Figure 2 shows an example in blocks world where the agent must predict what happens when block B is moved from block A to C. We will ground the description of each step in this example.

To simplify the algorithm description, we introduce some terminology. Call s the target state, a the target action, s_1' the source state, and s_2' the after state. Let OS, OT, and OA be the set of objects in the source, target, and after states, and RS, RT, and RA the set of relations in the source, target, and after states, respectively. Let $OM: OS \rightarrow OT$ be a mapping of objects from source to target state, and $RM_i: RS_i \rightarrow RT_i$ be a mapping of the elements of the i^{th} relation from source to target state.

Retrieval. The first step of the algorithm retrieves the

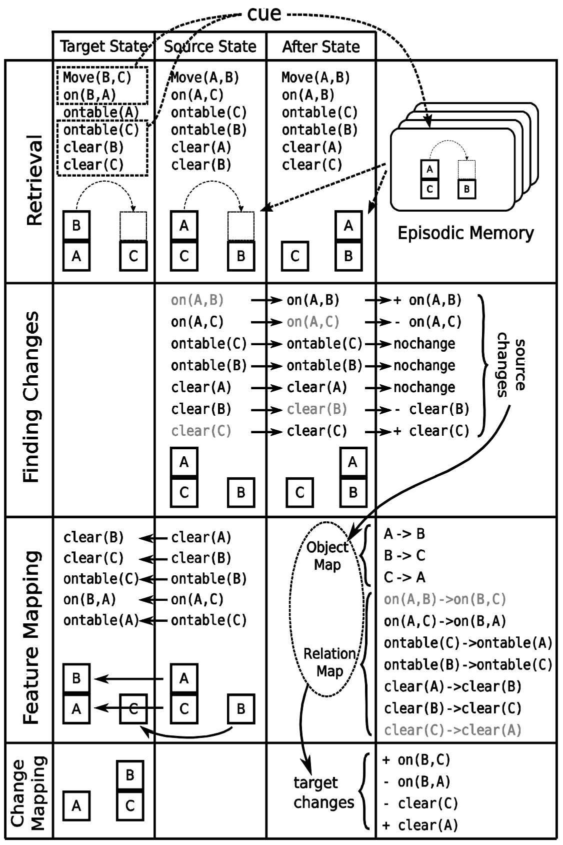

Figure 2. Example prediction algorithm run

transition that is most analogous to the one being predicted from episodic memory. This is done by performing a cue-based retrieval using only the relation tuples from the target state and action that are relevant in determining the outcome of the action. Object identities are ignored because we assume actions are conditioned only on relations. This alone achieves some generalization that would not be possible without a relational representation.

The agent does not know *a priori* which relation tuples are relevant to predicting the outcome of a specific action. We use a proximity heuristic that chooses tuples within a chosen order of syntactic connectivity to the action parameters. First order tuples are those involving action parameters, second order those that share parameters with first order tuples, etc. We use this heuristic because causality is usually local to the objects being acted upon, and syntactic connectivity reflects this locality. In the example, the cue consists of first order tuples with B and C as parameters, while ontable(A) is a second order tuple and thus is not included.

After the cue based retrieval returns the source state, the agent performs a *next* retrieval to obtain the after state. In the example, the agent retrieves a source state in which it moved A from C to B. It then retrieves the state after the action occurred, with A stacked on B.

Finding Changes. After retrieving the source and after states, the system determines what changes occurred by performing simple set comparisons between the two states. The algorithm computes the added and removed objects, and the added and removed relation tuples:

$$\Delta_{O+} \leftarrow OA \setminus OS$$
$$\Delta_{O-} \leftarrow OS \setminus OA$$
$$\Delta_{R+}^{i} \leftarrow RA_i \setminus RS_i$$
$$\Delta_{R-}^{i} \leftarrow RS_i \setminus RA_i$$

In the example, the on relation between A and C is removed while a new on relation between A and B is added. Also, C is now clear whereas B is no longer clear.

Feature Mapping. Next, the system determines the correspondence between the objects and relation tuples in the source state and those in the target state. Our algorithm attempts to map state features to maximize the structural correspondence between the source and target states. It is inspired by the Structure Mapping theory and its implementation, the Structure Mapping Engine (Falkenheiner et. al. 1989). Our mapping algorithm is simpler than SME, and we are only concerned with within-domain mappings whereas they are also concerned with cross-domain mappings.

The mapping algorithm is iterative and maintains a set of candidate source-target relation tuple mappings at each step. A source relation tuple $s \in RS_i$ and a target relation tuple $t \in RT_i$ make up a candidate mapping $s \rightarrow t$ to be included into RM_i if at least one of their corresponding parameters have already been mapped to each other:

$$\exists j[s_j \rightarrow t_j \in OM]$$

The mapping algorithm starts with the parameters of the source actions mapped to those in the target action. At each step, it chooses one candidate relation mapping to be included into RM_i. There are usually many choices, and instead of performing a combinatorial search over the different possible orderings of commitments, we use simple heuristics to assign preferences to them. For example, the algorithm prefers to map relation tuples that have more parameters already mapped. Ties are broken randomly. Once it commits to a tuple mapping, all of their unmapped parameters are also mapped to each other:

$$[s \rightarrow t \in RM_i] \Rightarrow \forall j[s_j \rightarrow r_j \in OM]$$

The new object mappings in turn introduce more candidate tuple maps in the next iteration. This process continues until there are no more mapping candidates.

The algorithm also needs to map objects and relation tuples added in the after state to their analogues in the prediction. Each new object in the after state is mapped to a unique new object symbol.

$$OM \leftarrow OM \cup \{o \rightarrow x | o \in \Delta_{O+}\}$$

Each new relation tuple r in the after state is mapped to a relation tuple r' where

$$\forall i[r'_i = OM(r_i)]$$

In the example, A is first mapped to B and B to C since they are the action parameters. The clear and ontable relations for those objects are then mapped, followed by the on relations. The other parameters of the on relations, C and A, are then mapped, which then allows ontable(C) to be mapped to ontable(A). Lastly, the new relation tuples, indicated by gray text, are mapped.

Mapping Changes. Finally, the changes calculated for the source transition are mapped onto the target state using the mappings from the previous step to obtain the prediction.

$$OP \leftarrow (OT \cup \{OM(o) | o \in \Delta_{O+}\}) \setminus \{OM(o) | o \in \Delta_{O-}\}$$
$$RP_i \leftarrow \left(RT_i \cup \left\{RM_i(r) \middle| r \in \Delta^i_{R+}\right\}\right) \setminus \left\{RM_i(r) \middle| r \in \Delta^i_{R-}\right\}$$

In the example, mapping changes from the source state results in the prediction that B will be on C.

Finally, it is worth noting that cue-based retrieval returns the best match to the cue, which may not be an exact match. When making a prediction for an unexperienced transition, the algorithm retrieves the memorized transition that is structurally most similar to it, and then through analogical mapping attempts to adapt the outcome of the memorized transition to the novel one. This is how the model generalizes to novel situations.

Our approach makes predictions directly from stored transitions and is thus an instance-based learning (IBL) algorithm (Aha et al. 1991). As with IBL methods, it is incremental, anytime, and suitable for online training.

Experiments and Results

We evaluate this approach on two domains – the first designed specifically to stress action modeling, and the second used in previous RL research.

Breaking Blocks World

This domain is identical to ordinary blocks world except that the robot gripper can pick up any block, not just the blocks at the tops of stacks. If it tries to pick up a block from the middle of a stack, all blocks stacked above that block fall to the ground and break. Breaking blocks results in a large negative reward. The task ends when all unbroken blocks are stacked in a single column in order. This domain also has an extra relation, *above*, the transitive version of the *on* relation. This helps the agent determine the relevant context of *move* actions, but makes predicting the result of moves more difficult.

Breaking blocks world presents a significant challenge for action modeling. Predicting which blocks will break requires taking into account which blocks are above the one being moved. Predicting new *above* tuples requires taking into account the blocks below the destination.

Taxi Domain

In the Taxi domain (Dietterich 1998), the agent controls a taxi that navigates in a grid-based map with walls that the taxi cannot pass through. The goal is to pick up and deliver a passenger to a given destination. The taxi can move in the four cardinal directions, pick up the passenger when the taxi is in the same cell, drop off the passenger, and refill its fuel. The taxi uses up one unit of fuel for each move it makes. We use a 5x5 grid.

The challenges in modeling this domain include generalizing the effects of movement, which consumes fuel and changes the taxi and passenger's positions, and recognizing when movement is blocked by walls.

Experiment 1: Prediction Performance

Our first experiment tests the prediction performance of our algorithm by measuring the number of correct predictions it makes with respect to the size of its episodic memory. We do this by first recording a random trajectory through an environment, then measuring the number of transitions in that trajectory our action model correctly predicts given increasing length prefixes of the trajectory as its episodic store. This measures the contribution that additional episodes make to accuracy. We test prefix lengths at powers of 2 to show the increase in accuracy per doubling of experience.

This testing methodology is different from the traditional methodology for testing learning algorithms in which the training set and test set are independently generated. Our "training set" (episodic store) and test set come from the same trajectory. This is because unlike traditional learning algorithms that freeze their model after the designated training period, the training of our algorithm occurs online, simultaneously with performance, and test instances eventually become training instances.

Breaking Blocks World. Figure 3 shows the prediction accuracy averaged over 30 trials for two different feature connectivity orders used in the cue. The straight line shows the expected success rate if no generalization occurred and only seen transitions are correctly predicted.

The predictor generalizes over very few episodes to successfully predict many transitions. With just a single transition in memory, it can predict about a third of all transitions in the trajectory correctly. This is not surprising, considering that the MoveToBlock action is highly prototypical even in this modified version of blocks world. However, taking advantage of this regularity without explicit knowledge of the domain speaks for the strength of analogical mapping. With 32 episodes recorded, the model predicts about 88% of the transitions correctly. But even when episodic memory contains all the test episodes, the model still makes prediction errors. This is because the connectivity heuristic does not always include all the state features relevant to the action being predicted in the retrieval cue. This leads to the retrieval of more recent episodes that match the cue perfectly, even though the states are not the same in all relevant respects.

The figure also shows that using a low connectivity order in the cue gives better performance with fewer episodes stored in memory, but worse performance with more episodes. This is because low connectivity order encourages more generalization, but also discourages

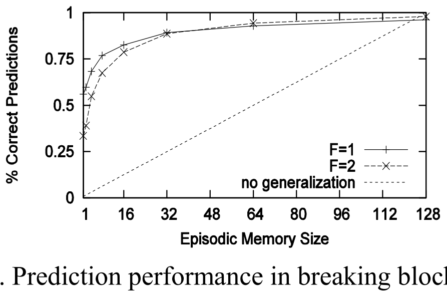

Figure 3. Prediction performance in breaking blocks world

discrimination between states that don't correctly generalize to each other.

Taxi Domain. The taxi domain has two major transition modes. When the passenger is in the taxi, his location changes when the agent performs move actions. When he is outside the taxi, his location does not change with move actions. The domain mechanics switch between these two modes whenever the agent performs getin and getout actions. Our algorithm can learn to correctly predict the consequences of movement in both these modes simultaneously. We obtain episodic memories for random trajectories 256 steps in length. The runs begin with the passenger outside the taxi. At the 128th step, we manually place the passenger into taxi. We do this instead of allowing the agent to pick up and drop off the passenger at random so that data across trials can be reasonably aggregated. Therefore, the first 128 episodes in memory are of outside taxi transitions, and the second 128 episodes are of inside taxi transitions. We test the agent's prediction performance using these 256 episode memories on both the outside taxi and inside taxi trajectories.

Figure 4 shows the prediction performance for both modes, and two connectivity order settings for each mode, averaged over 30 trials each. In this domain, using a low feature connectivity order leads to consistently lower prediction performance. One reason for this is that wall relations are 2 orders of connectivity removed from the taxi object, and not including those relations in the cue leads to over-generalization.

The agent generalizes the first eight episodes effectively to correctly predict almost 80% of the outside taxi transitions. Furthermore, with the inside taxi episodes (128-256) in its memory, the agent's performance on predicting the outside taxi transitions does not degrade.

As expected, the agent does not predict most of the inside taxi transitions correctly using only the first 128 outside taxi episodes. The correct predictions are of transitions where the taxi runs into walls and no locations change. Once it has access to inside taxi episodes, the agent again demonstrates good generalization and a fast increase in correct prediction rate. Thus, our algorithm recovers from unrepresentative training examples without extra computation, one of the advantages of IBL.

Experiment 2: Task Performance

In our second experiment, we evaluate whether our model learning algorithm can be usefully integrated into an RL agent, and whether it is more useful than a non-generalizing model learning technique, adaptive RTDP (ARTDP) (Barto 1995). It is possible that even though the agent's predictions improve with experience, the errors in prediction are significant enough to degrade performance.

This experiment is carried out in a 5-block breaking blocks world problem initialized with all 5 blocks on the table. The agent takes actions until all non-broken blocks are stacked in alphabetical order. A reward of -1 is given for each action, and a reward of -1000 is given for breaking a block. The agent knows this reward function *a priori*. The discount factor is 0.9. The optimal policy is 4 steps long and does not break any blocks.

Our agent uses a variant of the Dyna-Q algorithm (Sutton 1990). In every state, the agent uses its learned model to predict the outcome of each available action. Since the agent knows the reward function, each of these simulated look-aheads is used to perform a TD update. After these updates, the agent selects an action to take in the environment based on its exploration policy and the actions' updated Q-values. The actual transition that results is also used to perform a TD update, and is recorded in episodic memory.

The ARTDP agent also builds a model of the environment and uses that model to perform TD updates. Unlike our agent, the RTDP agent models each state transition independently. Therefore this agent cannot make any predictions when encountering a state for the first time, regardless of how many similar states it has learned to model. Both agents use a Boltzmann exploration policy with temperature set at 1. Figure 5 shows the average reward for both agents, and Figure 6 shows the average number of blocks broken.

The overall behavior we observe is that the model learned with our algorithm generalizes from just a few experiences of breaking blocks to correctly predict when blocks will be broken in new situations. Therefore, the

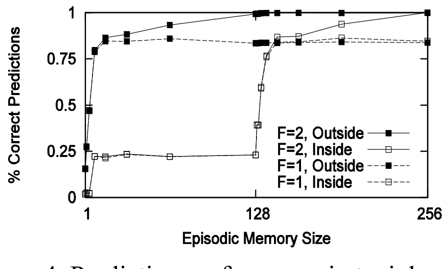

Figure 4. Prediction performance in taxi domain

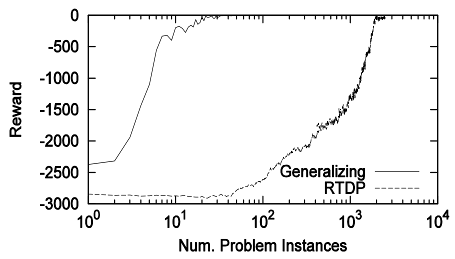

Figure 5. Reward obtained per problem instance

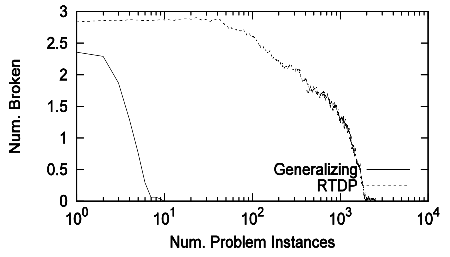

Figure 6. Blocks broken per problem instance

agent utilizing that model to perform look-aheads quickly learns to avoid taking block-breaking transitions, and converges to a near-optimal policy in very few trials.

In contrast, the RTDP agent, though it is learning a model of the task, cannot generalize its model. When the agent encounters a new state, the learned model is of no help in making predictions about the outcomes of its actions, and the agent must resort to trying them out in the environment. Hence, the agent must try almost all block-breaking transitions before learning to avoid them, resulting in much slower convergence to optimality.

Summary and Discussion

We have presented an instance-based algorithm for online and incremental action model learning. We evaluated our algorithm in two relational domains with very different structure, the taxi domain and the breaking blocks world domain, and showed that the algorithm successfully learns and generalizes action models in both domains. In the taxi domain, our algorithm recovers when the effects of actions suddenly change due to a modality shift in the environment. Moreover, a Q-learning agent using our model outperforms a non-generalizing ARTDP agent by an order of magnitude in the breaking blocks world domain.

Our algorithm should be applicable to any relational domain satisfying the assumptions discussed above, but its efficacy is sensitive to how the domain is encoded relative to the feature connectivity heuristic. The connectivity heuristic works adequately for our experimental domains because the features relevant to determining action effects in those domains have low connectivity order. We hypothesize that this property holds for many human-designed domains. For example, only two of the 22 PDDL domains used in the deterministic track of the last three International Planning Competitions (ICAPS 2010) violated this property, and in those cases, actions had global effects rather than high connectivity order.

A second shortcoming is that the algorithm cannot detect the need to test for the absence of state features in the cue. One possible solution to both these issues is for the agent to learn the relevant state features by experimentation (Gorski & Laird 2009). Another approach is to give the agent general background knowledge to reason about which state features are likely to be relevant. These are both directions for future research.

The cognitive architecture approach to integration is to develop a system with task-independent components that can realize general intelligent behavior and a flexible representation for sharing data across these components. The action modeling algorithm we presented in this paper is implemented solely using Soar's non-specialized episodic memory and procedural memory modules. Because it operates on data represented as working memory graphs, we were able to integrate this new functionality with Soar's existing subgoaling and reinforcement learning mechanisms without modification. Because the algorithm is online and incremental, it integrates with Soar's execution cycle without sacrificing the reactivity of the rest of the system.

Acknowledgements

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

References

Aha, D. W., Kibler, D., Albert, M.K. 1991. Instance-Based Learning Algorithms. *Machine Learning*, 6 (1): 37-66.

Barto, A. G., Bradtke, S. J., and Singh, S. P. 1995. Learning to Act using Real-Time Dynamic Programming. *Artificial Intelligence*, 72 (1-2): 81-138.

Benson, S. 1995. Inductive Learning of Reactive Action Models. In *Proceedings of ICML-95*, 47-54.

Derbinsky, N. and Laird, J. E. 2009. Efficiently Implementing Episodic Memory. In *ICCBR-09*, 403-417.

Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *Proceedings of ICML-98*.

Diuk, C., Cohen, A., and Littman, M. 2008. An Object-Oriented Representation for Efficient Reinforcement Learning. In *Proceedings of ICML-08*. Helsinki.

Falkenhainer, B., Forbus, K. D., and Gentner, D. 1989. The Structure-Mapping Engine: Algorithms and Examples. *Artificial Intelligence*, 41: 1-63.

Gorski, N. A., Laird, J. E. 2009. Learning to Use Episodic Memory. In *Proceedings of ICCM-09*. Manchester, UK.

ICAPS Competitions. 2010. Retrieved Jan 21, 2010 from http://ipc.icaps-conference.org.

Kearns, M. and Singh, S. 2002. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49: 209-232.

Laird, J. E. Extending the Soar Cognitive Architecture, 2008. In *First AGI Conference*: 224-235.

Nason, S. and Laird, J. E. 2004. Soar-RL: Integrating Reinforcement Learning with Soar. *Cognitive Systems Research*, 6 (1): 51-59.

Nuxoll, A. 2007. Enhancing Intelligent Agents with Episodic Memory. Ph.D. Thesis, University of Michigan, Ann Arbor, Michigan.

Sutton, R. S. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximate Dynamic Programming. In *Proceedings of ICML-1990*: 216-224.

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

Pasula, H., Zettlemoyer, L., and Kaelbling, L. 2007. Learning Symbolic Models of Stochastic World Domains. *Journal of Artificial Intelligence Research*, 29: 309-352.

Wang, X. 1995. Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In *Proceedings of ICML-95*: 549-557.