Building Relational World Models for Reinforcement Learning

Trevor Walker[1], Lisa Torrey[1],
Jude Shavlik[1], and Richard Maclin[2]

[1] University of Wisconsin, Madison WI 53706, USA
[2] University of Minnesota, Duluth, MN 55812, USA

Abstract. Many reinforcement learning domains are highly relational. While traditional temporal-difference methods can be applied to these domains, they are limited in their capacity to exploit the relational nature of the domain. Our algorithm, AMBIL, constructs relational world models in the form of relational Markov decision processes (MDPs). AMBIL works backwards from collections of high-reward states, utilizing inductive logic programming to learn their *preimage*, logical definitions of the region of state space that leads to the high-reward states via some action. These learned preimages are chained together to form an MDP that abstractly represents the domain. AMBIL estimates the reward and transition probabilities of this MDP from past experience. Since our MDPs are small, AMBIL uses value-iteration to quickly estimate the Q-values of each action in the induced states and determine a policy. AMBIL is able to employ complex background knowledge and supports relational representations. Empirical evaluation on both synthetic domains and a sub-task of the RoboCup soccer domain shows significant performance gains compared to standard Q-learning.

1 Introduction

We present a relational reinforcement learning (RL) method, AMBIL (Abstract Model Building via ILP), that can handle complex relational reinforcement learning domains with real-valued, high-dimensional feature spaces and sparse reward structures. Via its novel method for partitioning an environment into useful states, AMBIL attempts to find good policies by building a model of a domain's state-transition structure in the form of an abstract Markov decision process (MDP). AMBIL uses inductive logic programming (ILP) extensively to learn the states in this MDP, treating each state as a separate ILP learning problem. The use of ILP techniques provides power and generality to our models that would otherwise be unattainable with other approaches.

Traditional RL methods, such as model-free, temporal-difference (TD) learning algorithms [15], find optimal or near-optimal policies to maximize the reward received while acting within the domain. However, as the domain becomes more complex, these methods often rely upon function approximation for effective generalization of training data.

In complex domains, generalization by function approximators can pose significant difficulties and typically requires a large number of examples. AMBIL attempts to improve generalization by focusing on the state transitions seen in the training data. AMBIL uses ILP to partition the state space and, since these partitions are based directly upon the state transitions of actions, can generalize the examples more effectively. AMBIL also exploits the relational aspects of RL domains; similar domain actions can be abstracted and learned together rather than one at a time. AMBIL abstracts objects in the domain, learning rules that apply to whole classes of objects.

The partitions learned by AMBIL form a relational MDP, with the transition and reward function estimated from previously observed data. Existing techniques, such as value-iteration [15], provide estimates of the expected reward for each MDP state. AMBIL's MDP both determines the policy and provides an explicit, easy-to-analyze representation of the domain.

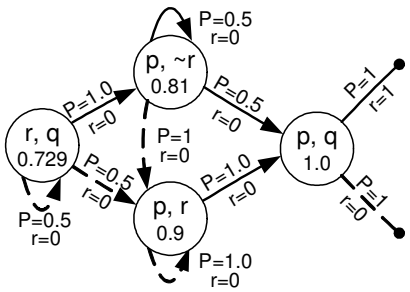

Figure 1. Sample MDP produced by AMBIL. First-order logical rules define each state. Arcs represent action transitions for two actions (one action with solid and one with dotted lines). Arcs show the probability of transitions, given the action, and immediate rewards. Value-iteration, with γ=0.9, calculates the Q-value of each state-action pair (not shown). The maximum Q-value from each state (shown inside each state) determines the policy for the state.

AMBIL is not the first method to address relational RL. Several techniques exist that apply relational function approximation to traditional Q-learning or dispense with function approximation altogether. We compare AMBIL to these methods in the related-work section.

We present empirical results in a synthetic domain and in the challenging Breakaway subtask of the RoboCup soccer domain [8], demonstrating faster learning and higher asymptotes at convergence than traditional Q-learning reinforcement learning algorithms.

2 Background

In reinforcement learning [15], an agent performs actions in some domain. After each action, the agent receives feedback in the form of numeric rewards. The agent attempts to learn a policy π that maps domain states to actions to maximize the sum of rewards received.

A reinforcement learning domain can be thought of as a Markov Decision Process (MDP). An MDP is a five-tuple $\langle S,A,P,R,\gamma \rangle$ where S is a finite set of states; A is a finite set of actions; P is a transition function denoting the next-state distribution after taking action a in state s; R is a bounded reward function denoting the expected immediate reward received for taking action a in state s; and $\gamma \in [0,1)$ is a discount factor.

In many cases, the underlying MDP for a domain is unknown. In these situations, the agent must interact with the domain in order to learn a policy. Either the agent can learn a policy directly, without learning the underlying MDP, or the agent can attempt to learn the underlying structure in some form and then create a policy based upon the learned structure. The former approach is called model-free or direct reinforcement learning and the latter is called model-learning.

Q-learning [16], a common model-free approach used in RL when the underlying MDP is not known, makes few assumptions or restrictions concerning the RL domain and performs competitively with many model-learning approaches. It works by learning a function $Q(s,a)$ that represents the expected reward for taking action a in state s. Q-learning determines policy by choosing the action with the highest Q-value for a given state. The Q-function itself is often modeled using a function approximator, allowing Q-learning some ability to scale to complex domains with real-valued, high-dimensional feature spaces.

3 Building World Models

AMBIL, a model-building technique, partitions the state space and creates an MDP from the partitions. First-order logical rules define the portion of state space each partition covers. The MDP in turn leads to a policy intended to maximize the rewards received. AMBIL then exploits this policy to collect more examples from the domain. AMBIL is a batch algorithm. It iterates between building models and utilizing those models in the domain. Figure 1 depicts a sample MDP produced by AMBIL. The following sections explain the model-building process.

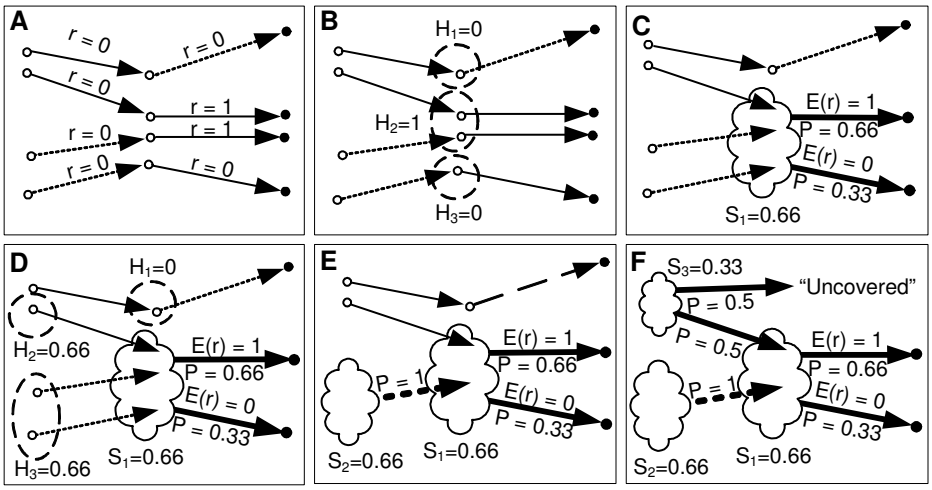

Figure 2. Sample MDP being built in a 2D continuous state space. **A**. Example states (open dots), reached by two actions (solid and dotted arcs). Actions reaching filled dots terminated the episode. **B**. Initial terminating preimages considered for learning, with their heuristic scores H_i shown, based upon the rewards in the example data. **C**. An MDP state learned based upon preimage H_2. Hote the generalized state S_1 could cover more example states than intended. Action arcs show the aggregate reward and transition functions for S_1. All calculations use γ=1. **D**. Next stage of preimage selection and scoring. **E**. MDP extended by generalizing preimage H_3 into MDP state S_2. **F**. Final MDP after all preimages have been generalized. Note some transitions, such as the top one from S_3, may not lead to a learned state and are placed in a special "uncovered" state, with a score of zero.

3.1 Terminology

AMBIL operates on a collection of examples containing sequences of {*state, action, reward*} tuples from the RL domain we are attempting to solve. We will refer to these as *example states*. Additionally, as we partition the example state space, the partitions become states in an MDP. We will refer to these states as *MDP states*.

AMBIL bases its partitioning upon *preimages* of the current, partially-built MDP. The preimage of an MDP state *s* for action *a* is the set of example states *s'* that lead to *s* via action *a*. A *terminating preimage* is the set of example states *s'* in which action *a* was taken, resulting in the termination of an episode with immediate reward of $r \pm \delta$. For example, the terminating preimage of shots scored would be all example states in which the agent shot, resulting in a score and an immediate reward of +1, ending the episode.

3.2 Algorithm Overview

Table 1 presents the AMBIL algorithm and Figure 2 depicts several stages of the algorithm in operation.

When partitioning example space, AMBIL begins with an empty MDP model. It then examines all terminating preimages and greedily selects one that scores the highest ac-

Table 1. AMBIL's MDP Model Building Algorithm

Input: *E*: set of example states
BK: background knowledge expressed in first-order predicate calculus

Do:

1. Initialize MDP to empty model
2. Score possible preimages and greedily select best one to learn
3. Learn rule(s) via inductive logic programming based upon selected preimage, *E*, and *BK*
4. Add learned rule(s) to MDP as new MDP state
5. Estimate MDP's reward functions and transition probabilities
6. Calculate *Q*-values for all states in MDP
7. If examples states sufficiently covered, stop. Else go to step 2

Return: MDP

cording to a heuristic (see Figure 2B). Once a preimage is selected, AMBIL uses ILP to generalize the example states in the preimage into one or more first-order logical rules, which may be include some negative examples in domains that are stochastic. Each rule becomes a single MDP state, covering all of the example states that match its logical rule. Each time AMBIL adds an MDP state, it calculates the transition and reward functions for that state by examining the example states covered by the rule (see Figure 2C). Then, AMBIL applies value-iteration to obtain the score for each state currently in the MDP. This process repeats, greedily selecting from all previously unlearned terminating preimages and MDP state preimages (see Figure 2D-F), until the MDP covers at least a user-specified fraction of the example states.

3.3 Preimage Selection

When constructing an MDP, AMBIL greedily selects preimages to define the MDP. These preimages consist of example states not currently covered by some MDP state and lead to a previously learned MDP state or a terminating preimage.

A simple heuristic scores each eligible preimage by an *optimistic* Q-value using Bellman-backups [1], according to Equation 1. The optimistic Q-value represents the expected Q-value of the preimage's constituent example states. Thus, the value is the expected reward of a transition from an example state in the preimage to the destination state of the MDP via action a. This value is optimistic since it assumes AMBIL can learn the preimage exactly and that the example data is an i.i.d. sample of all possible example states in the preimage. In RL domains, this will not be true, since the distribution of example states visited depends upon the policy of the learner. In addition, the generalized rule(s) learned to represent the preimage will often cover parts of example space that were not part of the preimage. Even though the optimistic Q-value is inaccurate, it serves as a good heuristic to guide preimage selection.

$$Q_{opt}(\text{Preimage}(\text{S}, a)) = \frac{\sum_{s \in \text{Preimage}(\text{S},a)} r(s,a) + \gamma Q(\text{S})}{\left|\text{Preimage}(\text{S}, a)\right|} \quad (1)$$

Given the current MDP, only a subset of the possible preimages are eligible for learning. AMBIL ignores preimages learned previously. Each preimage must contain a minimum number of example states, guaranteeing that enough data will be available for the rule-learning stage. Preimages must also obtain a minimum optimistic Q-value score. This eliminates preimages unlikely to result in an increase in performance.

AMBIL exploits the relational nature of the domain during preimage selection. It considers preimages with multiple actions whenever the user indicates that two or more actions share similar behavior. In these cases, the preimage will be parameterized appropriately for each action. For example, two shooting actions, such as shoot(left) and shoot(right) used in Section 4's experiments, might be considered together as shoot(GoalPart), where GoalPart is a variable parameterizing the portion of the goal the shot was aimed at. This grouping allows AMBIL to exploit the relational nature of some actions. Even when these groupings exist, AMBIL also considers the single-action preimages since parameterized concepts may be more difficult to learn or specialized versions of actions might be needed.

When AMBIL first creates an MDP, the MDP will contain no states to use as a basis for preimages. Thus, AMBIL currently uses terminating preimages to initiate the MDP-building process. In the domains we have focused on, clearly defined terminating preimages exist (such as shots resulting in a scored goal). AMBIL could also use a domain's reward-structure information, if available, to determine the initial preimages. In non-sparse or infinite-horizon domains with no user-provided objective, AMBIL could cluster the sampled rewards to determine initial concepts.

3.4 Learning Concepts via ILP

For each preimage selected, AMBIL uses inductive logic programming to generate first-order rules that describe the set of states that the preimage covers. Given an example

```
shot_score(GoalPart) :-
   x_distance_wrt_kick_at_goal (GoalPart) > 6.0
   y_distance_wrt_kick_at_goal(GoalPart) > 2.0
   angle_between_kick_&_goalie(GoalPart) < 129.
```

Example 1. Learned rule for shot_scored preimage, with a parameter to represent both the shoot(left) and shoot(right) actions. The variable GoalPart allows this rule to be applied to shooting at either side of the goal, both during learning and problem solving.

state, the learned rules classify whether or not it is in the preimage, i.e. whether it leads to the relevant MDP state via the indicated action or not. Although AMBIL could use simpler propositional methods to learn the preimage classification, ILP allows the relational aspect of the domain to be exploited: similar actions can be parameterized and learned as a single concept, similar domain objects can be generalized, and extensive background knowledge can be utilized to aid in describing the preimages. Currently, AMBIL uses the ILP system Aleph [12].

AMBIL selects the positive and negative examples based upon the preimage being learned. For a preimage(*s,a*), the example states that transitions to MDP state *s* via action *a*, AMBIL collects, as positive examples, all example states where the action *a* was taken and the following example state is covered by MDP state *s*. The negative examples are the example states where action *a* was taken and the following state is not covered by *s*. Example states in which the action *a* was not taken are ignored. There are generally many more negatives than positives. AMBIL uniformly subsamples the positive and negative examples to reduce Aleph's runtime. Typically, we subsample down to 500 total examples.

The positive and negative examples AMBIL provides to Aleph are in the form of sets of first-order predicates. As such, the examples can be parameterized to contain additional information. As mentioned in Section 3.3, this allows AMBIL to learn multiple actions as a single preimage. For example, the preimage "shots that score a goal" can be parameterized to include the shot destination as an argument, such as shot_scored(GoalPart). The user must specify which actions are similar and what parameters they require, but once that is done, AMBIL handles all of the parameterizations automatically.

Example 1 shows a parameterized rule learned for a soccer domain involving two shoot actions, shoot(left) and shoot(right). In this example, the positives contained all examples that shot left or right and scored. The negative set contained all examples that shot left or right and did not score.

3.5 Building the MDP

Given a learned preimage classification theory from ILP, AMBIL extends the MDP model by adding states, one for each rule in the theory. We treat the separate rules from a theory independently since they may represent different aspects of the learned preimage and doing so increases the specificity of the final model.

While AMBIL treats the learned model as an MDP, the model might not actually be one. Often, the created model violates the Markovian assumptions required by MDPs and might be better characterized as a partially observable Markov decision process. However, in our experiments, even when the Markovian assumptions were clearly violated, treating the model as an MDP still yielded good results and made the calculation of the value function much faster.

In a proper MDP, states are discrete and disjoint. However, many of the rules generated by Aleph overlap with either other rules within a single preimage's theory or other states previously added to the MDP. In order to enforce disjointness among MDP states, AMBIL orders the states by their creation order, essentially creating an IF-ELSEIF-ELSE structure used to determine in which MDP state an example state belongs. When AMBIL

adds multiple rules from a single preimage's theory, the created states are ordered according to their accuracy on the Aleph training set.

After adding states to the MDP, AMBIL calculates the MDP's state-transition probabilities and reward functions. AMBIL computes the transition probability from MDP state *s* to MDP state *s'* via action *a* by counting the number of example states that are covered by *s* and lead to *s'* via action *a*, normalizing these counts by the number of times action *a* was taken in state s. Similarly, it calculates the expected reward for action *a* from state *s* by averaging over the rewards seen in the training data. AMBIL employs *m*-estimates to condition the transition probabilities. We use m=5 in our experiments. In some cases, example states will exist that are not covered by any MDP state. AMBIL assigns these states to a special default state called the *uncovered* MDP state.

After it calculates the transition probabilities and rewards, AMBIL performs value-iteration [15] for all states in the MDP, except the special uncovered MDP state. The uncovered state's *Q*-value is some domain-dependent "background" score (e.g., zero). This discourages actions that would lead to the uncovered state. Although not necessary in our experiments, AMBIL could utilize refinements of the standard value-iteration algorithm, such as prioritized sweeping.

When adding states to the MDP, AMBIL must address several additional considerations. If two or more states overlap, states added later may not have adequate data. If the amount of data available to an MDP state is below some minimum threshold (currently, five example states), AMBIL discards that particular state.

On occasion, when AMBIL adds a state, the policy action it recommends (the argmax *Q* over all actions for this state) is not the same as the action of the original preimage. When the action does not match the preimage, this is indicative of one of two things: either a state does not have adequate data or Aleph improperly generalized the preimage. In these cases, AMBIL could discard the MDP state. However, in our experiments, this occurs infrequently and discarding the states is unnecessary.

3.6 The RL Learning Cycle

The previous sections described the process AMBIL uses to build a single MDP (and the associated policy). In the complete learning cycle for a given RL domain, AMBIL first gathers some initial data by interacting with the domain. It then repeatedly generates an MDP with a corresponding policy and interacts with the domain following the new policy to gather more data. In each iteration, AMBIL attempts to generate a new, higher-scoring policy.

Since the AMBIL model-building process is computationally expensive, we may not want to rebuild a full model whenever new data is available. However, updating the reward and transition functions for the MDP between full builds is computationally feasible and does result in some improvement of the policy. We use this approach in the empirical results section below.

To gather initial data AMBIL either explores the domain randomly or uses another RL learning algorithm (e.g.) standard model-free Q-learning as a bootstrap. In domains with very sparse rewards under a random policy, the bootstrapping process is preferred since it is more likely to obtain informative data.

After an MDP exists, when interacting with the RL domain, given an example state, AMBIL determines the action to perform by matching the example state against the states in the MDP model. The argmax of the Q-values for all actions for a given state determines the policy for that state. Additionally, as done by standard RL algorithms, AMBIL performs a small fraction of exploratory actions.

4 Empirical Results

We present empirical results in two domains: a synthetic RL domain and the Breakaway RL domain [8], a subtask of the RoboCup soccer domain. We compare AMBIL with the standard SARSA(λ) [15] algorithm and with a model-learning approach based upon a

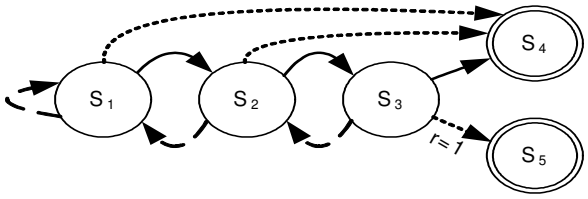

Figure 3. Synthetic domain MDP. Arcs represent state-transitions for three separate actions. All actions are deterministic. S_4 and S_5 are terminating state. All rewards are zero, except for the single action leading from S_3 to S_5.

Dyna-Q architecture [14]. We choose Dyna-Q as a control because it is an established approach for creating models of an environment in order to speed up RL.

4.1 Domains

The synthetic domain, shown in Figure 3, is a simple five-state, three-action, non-deterministic MDP, with two numeric features (drawn from overlapping uniform distributions), and a sparse reward structure, with the only reward occurring in one of two terminating states. The feature values for each state purposely overlap to simulate uncertainty in determining the underlying MDP state. We provided the learners no direct knowledge of the underlying MDP. They must interact with the domain to obtain information.

The Breakaway domain is a 2-on-1 soccer end-game task based upon the RoboCup soccer simulator. In Breakaway, *M* attackers attempt to score on *N* defenders, including a goalie. Only the attacker with the ball makes policy decisions, while the attackers without the ball and the defenders follow hard-coded policies. The reward structure for this domain is very sparse, with a reward of one when a goal is scored and zero all other times. Each Breakaway episode is limited to 10 seconds, after which the episode ends with a zero reward. The Breakaway domain is highly non-deterministic with many real-valued features (for 2-on-1, there are 27 features).

Our Breakaway state representation is that of Maclin *et al.* [8]. For the *Q*-learner, we discretize these features into 32 overlapping intervals called *tiles*, each of which becomes a Boolean feature. Stone and Sutton [13] used this enhancement in RoboCup; tiling allows linear function approximators to represent non-linear concepts.

For Breakaway, the AMBIL background knowledge consists of *feature_less_than*, *feature_greater_than*, *feature_in_range*, and *feature_not_in_range* predicates. Additionally, the background includes predicates that provide information relative to passes and kicks. For example, *x_distance_wrt_kick* measures the distance from the kicker to an object along the direction of the kick. The background knowledge was designed to allow Aleph to discretize the base features, rather than add additional high-level domain knowledge.

4.2 Learning Algorithms

All three learning algorithms, AMBIL, SARSA(λ), and Dyna-Q, are implemented as batch learners and, as much as possible, we used the same tuning parameters on the standard *Q*-learner, giving the benefit of doubt to the experimental control. Parameters were tuned on the Q-learning base line and used for the other learners. Each learner "batch learns" every 25 games. All use an exploration rate of 1% and a discount factor of 0.97. For the *Q*-learner and Dyna-Q we used a λ setting, with λ=1 for recent example states decaying to λ=0 after a fixed number of games (200 for Breakaway, 50 for the synthetic MDP).

Both AMBIL and Dyna-Q use the standard Q-learning algorithm until enough data is available to start the respective algorithm. Fifty and 250 games are played prior to Dyna/AMBIL running for the synthetic and Breakaway domains, respectively.

Dyna-Q creates models that predict the feature values in the next state and the reward function of the domain directly from the data and then uses the models to train *Q*-functions. We modeled the reward function using a C4.5 decision tree [10]. Next state

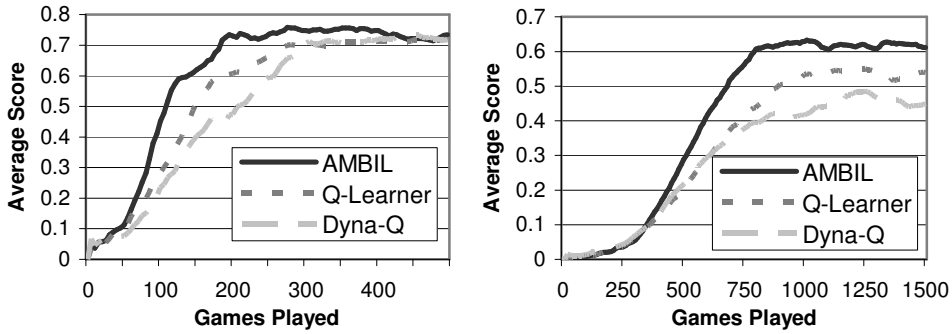

Figure 4. Empirical Results. (Left) Synthetic Domain averaged reward received per game, averaged over previous 50 games. (Right) 2-on-1 Breakaway average reward received per game, averaged over previous 250 games.

feature values are modeled independently of each other using support vector regression. We attempted to make these models as accurate as possible, although modeling high-dimensional, real-valued environments is known to be difficult. The next state models created by Dyna-Q are used to create synthetic examples. These examples are then utilized in the same manner as real examples. We created enough synthetic data to maintain a 4-1 ratio of actual examples to synthetic examples.

For the Breakaway domain, we also attempted to implement an RRL algorithm [5], a combination of traditional Q-learning with a relational TILDE-RT [2] function approximator. We were unable to obtain results better than random walks with this approach.

4.3 Results

Figure 4 shows the average reward per game for the synthetic domain and for the 2-on-1 Breakaway domain. For both domains, we performed 10 runs of each algorithm and averaged the results. The reward per game is averaged over the previous 50 games for the synthetic domain and previous 250 games for the Breakaway domain.

In both domains, AMBIL outperformed both the *Q*-learner and the Dyna-Q algorithms, both in terms of early learning rate and asymptotic performance.

Several factors contribute to AMBIL's early performance gains over the Q-learner. The background knowledge we provide to AMBIL offers it an advantage. We attempted to provide propositionalized versions of the background knowledge to the standard Q-learner, but this resulted in worse performance, presumably due to overfitting allowed by the greatly increased number of features.

Beyond the background knowledge, AMBIL's models, by construction, focus on reaching high-reward states immediately and generalizing accurately. In Q-learning, on the other hand, reward information propagates slowly through the model by the means of SARSA(λ) backups and generalization performed by function approximation can be inaccurate, especially early in the learning curve.

The Dyna-Q implementation performed poorly in both domains. This was due to the difficulty of modeling the underlying domain directly. It is the difficulty of this type of modeling that motivated AMBIL's approach.

5 Related Work

Reinforcement learning using TD methods has been studied extensively. Sutton and Barto [15] provide an excellent summary of the basic techniques.

Dietterich and Flann [3] introduced the concept of using chains of preimages in their explanation-based reinforcement learning. Their action chaining approach shares some basic similarities with AMBIL. However, their approach requires an accurate definition of the action consequences.

Kersting *et al.* [7] create abstract relational MDPs with many similarities to our own models. However, their approach to learning abstract MDPs requires the underlying MDP. We essentially provide a learning method capable of learning a similar abstract MDP in complex domains without knowledge of the underlying MDP.

Morales' [9] rQ-learning provides a state-abstraction approach to reinforcement learning, although their approach does not use an MDP representation. Unlike AMBIL, which learns the abstract states, Morales' approach requires user-provided abstractions. However, rQ-learning supports STRIPs-like operators with more richness then AMBIL's actions and provides a learning algorithm for refining these operators.

Van Otterlo's [11] CARCASS system provides a relational MDP representation, similar to AMBIL's, and provides methods to score and use the resulting MDP based upon interaction with the domain. Like rQ-learning, Van Otterlo's methods assume user-provided abstractions.

As an alternative approach to building an MDP, Džeroski *et al.* [5] proposed using a relational decision tree, such as TILDE [2], during Q-learning. Like AMBIL, this allows for both the integration of background knowledge and the exploitation of the relational aspects of actions and objects in the domain. However, like Q-learning, their approach represents only the long-term expected reward and does not model the immediate reward or transition information, while AMBIL does. Furthermore, they are still performing function approximation, which can be difficult for RL. Lecoeuche [6] and Driessens *et al.* [4], among others, further refined this approach.

Another recent approach to relational reinforcement learning, by Zettlemoyer *et al.* [17], also models the domain without building an MDP. Instead, they learn probabilistic STRIPs-like rules. A probabilistic planner uses these rules to solve the domain. Like AMBIL, they focus on the state transitions resulting from observed actions, although the rule learning process and final model does not resemble AMBIL's.

6 Conclusions and Future Work

Models of reinforcement learning domains allow faster learning than model-free Q-learning methods, as demonstrated in our empirical study, and provide information about the structure of the domain. Our algorithm, AMBIL, builds MDPs via inductive logic programming techniques, focusing on areas of high reward to guide search. The use of ILP techniques allows our models to represent relational domains, with abstraction of both objects and actions, and allows the incorporation of background knowledge.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. This research is supported by DARPA IPTO under contract FA8650-06-C-7606.

References

1. Bellman, R. E.: Dynamic Programming. Princeton University Press, Princeton, New Jersey (1957)
2. Blockeel, H. and De Raedt, L.: Top-down induction of first-order locical decision trees. Artificial Intelligence (June 1998)
3. Dietterich, T., Flann, N.: Explanation-based learning and reinforcement learning: A unified view. In: Proceedings of the International Conference on Machine Learning (1995)
4. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In: Proceedings of the European Conference on Machine Learning (2001)
5. Džeroski, S., De Raedt, L., Blockeel, H.: Relational reinforcement learning. In: Proceedings of the International Conference on Machine Learning (1998)

6. Lecoeuche, R.: Learning optimal dialog management rules by using reinforcement learning and inductive logic programming. In: Proceedings of the North American Chapter of the Association of Computational Linquistic (June 2001)
7. Kersting, K., Van Otterlo, M., De Raedt, L.: Bellman goes relational. In: Proceedings of the International Conference on Machine Learning (2004)
8. Maclin, R., Shavlik, J., Torrey, L., Walker, T., Wild, E.: Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression, In: Proceedings of the Twentieth Conference on Artificial Intelligence, (2005)
9. Morales, E. F.: Scaling up reinforcement learning with a relational representation. In: Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC'03 (2003)
10. Quinlan,J. R.: C4.5: Programs for Machine Learning. Morgan Kauffman (1993)
11. Van Otterlo, M.: Efficient reinforcement learning using relational aggregation. In: Proceedings of the Sixth European Workshop on Reinforcement Learning (2003)
12. Srinivasan, A.: The Aleph Manual (2001)
13. Stone, P., Sutton, R.: Scaling reinforcement learning toward RoboCup soccer. In: Proceedings of the International Conference on Machine Learning (2001)
14. Sutton, R.: Integrated modeling and control based on reinforcement learning and dynamic programming. In: Advances in Neural Information Processing Systems 3 (1991)
15. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, (1998)
16. Watkins, C. J. C. H.: Learning from Delayed Rewards. Ph.D. thesis, Cambridge University (1989)
17. Zettlemoyer, L. S., Pasula, H. M., Kaelbling, L. P.: Learning Planning Rules in Noisy Stochastic Worlds. In: Proceedings of the Twentieth Conference on Artificial Intelligence (2005)