Building Relational World Models for Reinforcement Learning

Lisa Torrey, Trevor Walker, Jude Shavlik
{ltorrey,twalker,shavlik}@cs.wisc.edu
Computer Science Department
University of Wisconsin – Madison

Richard Maclin
rmaclin@d.umn.edu
Computer Science Department
University of Minnesota - Duluth

This research is supported by DARPA IPTO under contract FA8650-06-C-7606.

Abstract

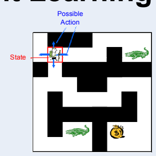
Many reinforcement learning domains are highly relational. While traditional temporal-difference methods can be applied to these domains, they are limited in their capacity to exploit the relational nature of the domain. Our algorithm, AMBIL, constructs relational world models in the form of relational Markov decision processes (MDP). 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.

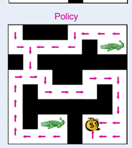
Reinforcement Learning

Given a Task environment
- States of the world
- Actions that can be performed
- Reinforcements (feedback)
  - +100 – get money
  - \- 100 – eaten by alligator
  - -1 – run info wall
  - 0 – otherwise

Do
- Learn policy to maximize total future reward by exploring environment
- Learn Q(s,a) function – the expected future reward for performing action *a* in state *s*

Policy(s) = argmax Q(s,a)





Why use ILP?

Action Abstraction

- Similar actions learned as single concept
- Reduces the number of concepts to learn
- Increases data available for each concept

shoot_at_goal_left, shoot_at_goal_center, shoot_at_goal_right } shoot_at(GoalPart)

Object Abstraction

- Objects in domain can be abstracted into classes

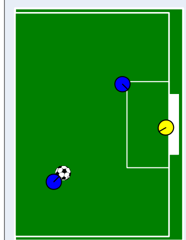
opponent1, opponent2, opponent3 } Opponent class

Background Knowledge

- Background knowledge can be easily utilized
- Complex relationships can be expressed

Learning Domain

2-on-1 Breakaway Soccer



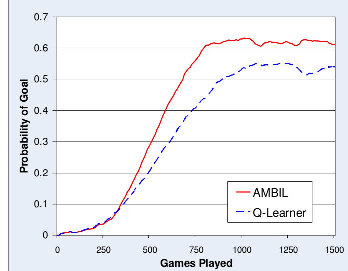
Object: blue team attempts to shoot goal in fixed amount of time

Learn: player will ball learns to move, pass, and shoot at appropriate times

Reinforcement: +1 Goal
0 Other outcome

Maclin et al. (2004) demonstrated RL can be used on this task

Results



Q-Learning vs AMBIL for Reinforcement Learning

Traditional Q-Learning Process

Initialize random starting policy → Play games using current policy → Calculate Q-values → Generate next policy → (back to Play games using current policy)

Q-Values are calculated using standard Bellman backups, based upon the observed rewards, next state, and previous policy.

$$Q_{new}(s_t,a)=\alpha\cdot Q_{old}(s_t,a)+(1-\alpha)\left(r(s_t)+\gamma \operatorname*{argmax}_{a'\in Action} Q_{old}(s_{t+1},a')\right)$$

Over time, the policy will be refined, with *α* controlling the learning rate.

In Q-learning, policies are composed of multiple Q-functions, one for each action. Typically, an agent takes the action with the highest Q-value for the state.

Q-functions are often estimated using function approximation, which can be difficult in complex domains.

AMBIL Learning Process

Initialize random starting policy → Play games using current policy → Build and solve world model → Use solved world model as next policy → (back to Play games using current policy)

Building a world model completely replaces the Q-value learning mechanism from traditional Q-learning.

AMBIL partitions state space into a relational Markov Decision Process (MDP). Each abstract state of the MDP contains estimates for the expected rewards, transition probabilities, and Q-values for each action.

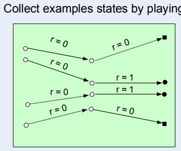
AMBIL uses the world model directly to determine policy.

For any given abstract state in the world model, the recommended action is determined by the action with the maximum Q-value for the state.

AMBIL does not utilize function approximation when building a model or determining policy.

Building a Relation World Model

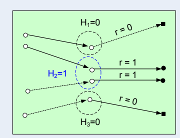
Collect examples states by playing



- ○ Example states
- ● Game ended with score
- ■ Game ended without score
- ⟶ Action A taken
- ⇢ Action B taken

Rewards observed after performing an action are shown along the transition arcs.

Select concept to learn

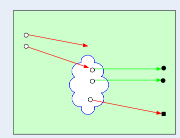


For each action, groups states that have similar outcomes, such as goal was scored.

Score each possible concept according to the expected discounted reward sum for performing the appropriate action.

Greedily select the concept to learn. H_2 would be selected above.

Learn concept using ILP

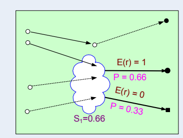


Generalize the selected concept using ILP.

Transitions with same action as target concept become positive and negative training examples, depending on whether they are in the selected concept.

In non-deterministic domains, the learned ILP rule(s) may cover negative examples.

Create MDP States

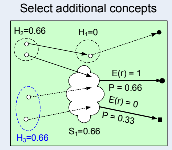


Each ILP rule becomes an MDP state.

For each outcome an MDP state leads to, the probability and expected reward is estimated from the constituent example states.

A state score is calculated using Bellman backups.

Select additional concepts

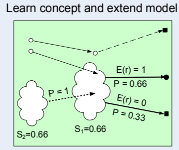


Extend the MDP by selecting an additional concept to learn.

Consider all of the original concepts, plus new concepts leading to the MDP state created in the previous step.

Greedily select the concept to learn. H_3 would be selected above.

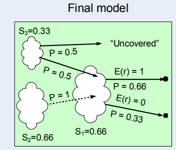
Learn concept and extend model



Another concept is learned via ILP and the MDP is extended.

The process of concept selection and extending the MDP is repeated until the state space is covered.

Final model



This is the final model for the example.

A special "uncovered" state provides a catch all for example states that couldn't be covered. This occurs both while building the model and when there isn't enough data left to create additional states.