Online Learning and Exploiting Relational Models in Reinforcement Learning

Tom Croonenborghs, Jan Ramon, Hendrik Blockeel and **Maurice Bruynooghe**
K.U.Leuven, Dept. of Computer Science
Celestijnenlaan 200A, B-3001 Leuven
{Tom.Croonenborghs, Jan.Ramon, Hendrik.Blockeel, Maurice.Bruynooghe}@cs.kuleuven.be

Abstract

In recent years, there has been a growing interest in using rich representations such as relational languages for reinforcement learning. However, while expressive languages have many advantages in terms of generalization and reasoning, extending existing approaches to such a relational setting is a non-trivial problem. In this paper, we present a first step towards the online learning and exploitation of relational models. We propose a representation for the transition and reward function that can be learned online and present a method that exploits these models by augmenting Relational Reinforcement Learning algorithms with planning techniques. The benefits and robustness of our approach are evaluated experimentally.

1 Introduction

In reinforcement learning, an agent can observe its world and perform actions in it. Its goal is to maximize the obtained reward. For small domains with a limited number of states, exact solution methods such as dynamic programming exist. However, these methods are unable to handle real-world domains with large state spaces. For such problems, structuring the world and generalization becomes essential. Recently, there is a strong interest in *Relational Reinforcement Learning* [Tadepalli *et al.*, 2004] , a framework not only providing an expressive language for describing the world and for generalization, but also able to handle "relational" state spaces which can not be easily represented using vector spaces.

For example, consider a blocks world where there is a set of blocks $\{b_1, b_2, \ldots b_n\}$. Every block b_i stands either on the floor (denoted $on(b_i, fl)$) or on some other block b_j (denoted $on(b_i, b_j)$) and on every block b_i there is either exactly one other block or it is clear (denoted $clear(b_i)$). Here, we describe a state by the set of facts that are true in that state, e.g. $\{clear(b1), on(b1, b2), on(b2, fl), clear(b3), on(b3, fl)\}$. The agent can take a clear block b_i and put it on another clear block b_j or on the floor (denoted $move(b_i, b_j)$). Move actions with other arguments (e.g., $move(fl, b1)$) are possible but have no effect. As the number of blocks is not limited in this world, it is clear that such states can not be easily represented by vectors. In contrast, in a relational setting concepts such as "on" and "clear" are intuitive to work with.

Several studies have shown that learning a model of the world (a transition function and the reward function) is often beneficial for the agent. Once a model has been learned, it can be used in several different ways. First, it can help to speed up the learning process by generating more training examples through simulation of actions in states, as happens in the Dyna architecture [Sutton, 1991]. Second, it allows the agent to reason about actions in a way similar to planning [Tesauro, 1995], which may allow it to achieve better rewards in exploitation mode and to make better estimates of Q-values in exploration mode by using lookahead (the TD-leaf method [Baxter *et al.*, 1998] is an example of the latter). Note that both options are complementary and can be combined.

Though in the relational setting most research so far has focused on a model-free setting, recently there is growing interest in extending methods for *learning and exploiting models of the world* to a relational setting (see [van Otterlo, 2005] for a recent overview). This is a non-trivial task as the use of a more expressive relational language inevitably implies that models of the world are more complex and harder to learn and apply. For instance, with the Dyna strategy, it is fairly easy to learn a model by keeping a probability distribution on states. In the relational case however a probability distribution on the large space of relational states is necessary, which is a lot harder, as shown in the field of statistical relational learning.

Our work investigates the feasibility and benefit of using relational learned models for lookahead. Moreover, we study whether such a strategy is still beneficial in complex worlds where it is not possible to learn a perfect model. We present MARLIE (*Model-Assisted Reinforcement Learning in Expressive languages*), the first system to learn a relational transition and reward function on-line. Our contribution is threefold. (1) We propose a representation for the transition function that facilitates its efficient and incremental learning. (2) We propose a learning and exploitation method. In contrast to earlier approaches learning relational models off-line (e.g. [Zettlemoyer *et al.*, 2005]), the partial model is exploited immediately (avoiding as much as possible an initial period where the agent gains poor rewards trying to learn a good model) and in contrast to work such as [Kersting *et al.*, 2004] this model does not need to be complete. Also note that we

are considering the full RL problem in that our technique does not require resets or generative models as e.g. in [Fern *et al.*, 2006]. (3) We experimentally evaluate the efficiency and benefits of our approach and examine the influence of the quality of the learned model on these results.

Organization. Section 2 presents some background and Section 3 shows how the transition function of a Relational MDP can be represented and learned online. Section 4 describes how using the model to look some steps ahead improves over standard Q-learning. The experimental evaluation is shown in Section 5. Section 6 discusses related work and we conclude in Section 7.

2 Reinforcement Learning and MDPs

Reinforcement Learning (RL) [Sutton and Barto, 1998] is often formulated in the formalism of *Markov Decision Processes* (MDPs). The need to model relational domains has led to different formalizations of *Relational MDPs (RMDPs)*, e.g. [Fern *et al.*, 2006; Kersting and De Raedt, 2004; Kersting *et al.*, 2004][1]. We use the following simple form:

Definition 1 *A Relational MDP (RMDP) is defined as the five-tuple $M = \langle P_S, P_A, C, T, R\rangle$, where P_S is a set of* state predicates*, P_A is a set of* action predicates *and C is a set of* constants*. A ground state (action) atom is of the form $p(c_1, \ldots, c_n)$ with $p/n \in P_S$ ($p/n \in P_A$) and $\forall i : c_i \in C\}$. A state in the state space $\mathbb{S}$ is a set of ground state atoms; an action in the action state $\mathbb{A}$ is a ground action atom.*

The transition function $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ defines a probability distribution over the possible next states: $T(s, a, s')$ denotes the probability of landing in state s' when executing action a in state s. The reward function $R : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$ defines the reward for executing a certain action in a certain state.

The task of reinforcement learning consists of finding an *optimal policy* for a certain MDP, which is (initially) unknown to the RL-agent. As usual, we define it as a function of the discounted, cumulative reward, i.e. find a policy $\pi : \mathbb{S} \rightarrow \mathbb{A}$ that maximizes the value function: $V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^i R(s_t, \pi(s_t))|s_0 = s, s_{t+1} = \pi(s_t)]$, where $0 \leq \gamma < 1$ is the *discount factor*, which indicates the relative importance of future rewards with respect to immediate rewards.

The RRL-system [Driessens, 2004] applies Q-Learning in relational domains, by using a relational regression algorithm to approximate the Q-function defined as

$$Q(s, a) \equiv R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') max_{a'} Q(s', a') \quad (1)$$

Knowing these Q-values, an optimal policy π^* of the MDP can be constructed as $\pi^*(s) = \text{argmax}_a\, Q(s, a)$.

3 Online Learning of Relational Models

In this section we propose a representation for the transition function and reward function that can be easily and efficiently learned in an on-line setting. This module learns these functions in the form of probability distributions $T'(s'|s, a)$ and $R'(r(s, a)|s, a)$, given P_S, P_A and C of the RMDP.

3.1 Representation of the World Model

As said above, a state is a set of ground state atoms; hence, using a binary random variable for every possible ground state atom at each time point t, a *Dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989] can represent the transition function. The action taken at time point t is represented by a random variable a_t (one for every t) that ranges over all atoms from $\mathbb{A}$ that represent valid actions. The reward at time t is represented by a real-valued random variable r_t.

The reward in the current state depends on the random variables representing the state and the action taken. The action taken depends on the current knowledge of the agent and the current state. Its conditional probability distribution (CPD) is not explicitly modeled as the chosen action is the result of the agent's reasoning process. The current state in turn depends on the previous state and the action taken in that state. This specifies a layered network structure which is a partial order over the random variables. There can still be dependencies between variables of the same state. We assume an expert provides an order on the random variables describing states such that a random variable only depends on those preceding it in this order. Hence we avoid the problem of learning the structure of the network, a problem that would be especially hard in the case of online learning because a revision of the structure would interfere with the learning of the conditional probability tables in uninvestigated ways.

The CPDs of state random variables can be compactly represented by *relational probability trees* [Neville *et al.*, 2003; Fierens *et al.*, 2005]. In our setting, the main idea is to have a single relational probability tree for every predicate symbol $p \in P_S$. This tree is used to model the conditional probability distribution $T'_p(x|s, a)$ that gives for every ground atom x with predicate symbol p the probability that it will be true in the next state given the current state s and action a. This allows for maximal generalizations by using the same tree for all atoms of the same predicate symbol, but avoids the problems arising when generalizing over predicates with a different number of arguments with different types and different semantics. As an example, Figure 1 shows a probability tree CPD for $clear(X)$ in the blocks world. Another relational probability tree is used to represent the CPD of the reward random variable. Note that the learned relational probability tree does not necessarily use all preceding random variables of the network as splits in internal nodes.

3.2 Learning the Model

All the uninstantiated parts of the model are CPD's represented by relational probability trees. Therefore, learning the model reduces to the online learning of a set of decision trees. In our implementation, we use an improved[2] version of the incremental relational tree learning algorithm TG [Driessens *et al.*, 2001].

[1] See [van Otterlo, 2005] for an overview.

[2] Which learns several trees in parallel and is much more space-efficient while only slightly less time-efficient.

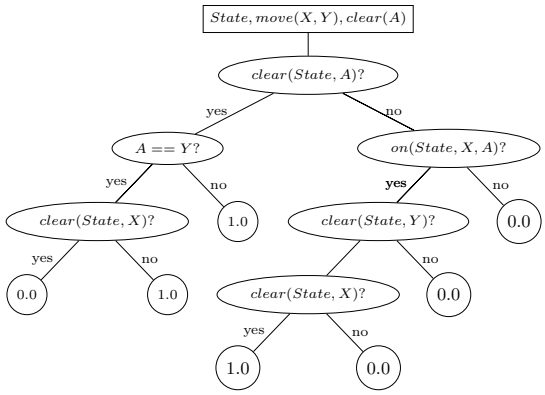

Figure 1: This probability tree shows the probability that block A will be clear when the action $move(X, Y)$ is executed in state $State$. The first node in the tree checks if block A was already clear (essentially the frame assumption). If this is the case, it can only become not clear when some other block is moved on top of it. If the block was not clear in the original state, it can only become clear in the afterstate if the block directly on top of it got moved to another block.

Learning examples can easily be created from the agents experience. Note that the number of possible facts in the next state may be very large, and therefore we do not generate all possible examples but apply a suitable sampling strategy.

4 Online Exploitation of Relational Models

The (partially correct) transition function $T'(s'|s, a)$ that is being learned enables the agent to predict future states. In this paper, we investigate the use of Q-learning with *lookahead trees* to give the agent more informed Q-values by looking some steps into the future when selecting an action. These lookahead trees are similar to the *sparse lookahead trees* used in [Kearns *et al.*, 2002] to obtain near-optimal policies for large MDPs.

Since the transition function can be stochastic or there may be uncertainty on the effect of a particular action (due to incomplete learning), an action needs to be sampled several times to obtain an accurate value. This sampling width SW is a parameter of the algorithm. Starting from the current state in the root node, we generate for every possible action, SW (directed) edges using the action as a label for that edge. The node at the tail of this edge represents the state obtained from executing that action in the head node. This can be continued until the tree reaches a certain depth. The Q-values of the deepest level can be estimated by the learned Q-function. By back-propagating these values to the top level, a policy can be constructed using these top level Q-values as in regular Q-learning. When back-propagating, the Q-values for the different samples of a certain action in a certain state are averaged and the Q-value of a higher level is determined using the Bellman equation (Eq. 1).

Several optimizations to this scheme are possible. Our implementation uses preconditions. This is especially useful in planning domains as, typically, the world remains unchanged if the agent tries an illegal action. While the transition function only indirectly states the preconditions of an action, the introduction and the learning of an extra binary random variable l_t (intended to be true if the state of the world changes) allows one to prune away actions predicted to be illegal.

Besides the sampling width, also other parameters can influence a lookahead tree. Currently, we are investigating the use of beam-like and randomized searches.

5 Empirical Evaluation

In this section we will present an empirical evaluation of our approach. First, we want to evaluate whether our incremental relational decision tree learner is able to build a model of the world. Second, we want to investigate how much the performance and speed of the agent improves by adding lookahead. Third, we want to evaluate the robustness of the approach, i.e., evaluate whether the lookahead is still beneficial when the learner is not able to build a complete model.

5.1 Domains Experimental setup

In all the following experiments, the RRL-TG[Driessens *et al.*, 2001] system is used to estimate the Q-values. Since the transition function for these domains are still learned rather easily, a sampling width of two is used. The agent also learns the function modeling the preconditions to prune the lookahead tree. The exploration policy consists of performing a single step lookahead. To eliminate the influence of a specific ordering on the random variables, independence is assumed between random variables in the same state. The figures show the average over a 5-fold run where each test run consists of 100 episodes and the average reward over this 100 episodes following a greedy policy, i.e., the percentage of episodes in which a reward is received, is used as a convergence measure.

Blocks World In the following experiments, we use the blocks world as described in the introduction with the $stack$-goal and the $on(A, B)$ goal (defined in the same way as in [Driessens, 2004][3]). In the $on(A, B)$-goal the agent only receives a reward iff block A is directly on top of block B. The objective of the $stack$-goal is to put all blocks in one and the same stack, i.e., if there is only one block on the floor. The results for the $unstack$-goal where the agent is rewarded iff all blocks are on the floor are not included as the behavior was comparable to the $stack$-goal.

During exploration, episodes have a maximum length of 25 steps above the ones needed by the optimal policy, during testing only optimal episodes are allowed. In the $stack$-problem the blocks world has seven blocks, for the $on(A, B)$-goal the number of blocks was varied for each episode between four and seven. The same language bias is used as in previous experiments with the RRL-TG algorithm in the Blocks World [Driessens, 2004].

Logistics The second domain is a logistics domain containing boxes, trucks and cities. The goal is to transport cer-

[3]The main difference is that here the agent can execute every possible action in a certain state instead of only the legal ones. This makes the problem more difficult.

tain boxes to certain cities[4]. The possible actions in this domain are $load\_box\_on\_truck/2$, which loads the specified box on the specified truck if they are both in the same city, the $unload\_box\_on\_truck/2$ which takes the box of the truck and moves it in the depot of the city where the truck is located. The third possible action is the $move/2$-action, which moves the truck to the specified city. The state space P_S consists of the following predicates: $box\_on\_truck/2$, $box\_in\_city/2$ and $truck\_in\_city/2$. These predicates also make up the language bias used in these experiments, i.e., the tree learning algorithm can test if a certain box is on a certain truck etc.

In the first setting there are two boxes, two cities and three trucks and the goal is to bring the two boxes to two specific cities. During exploration, 150 steps are allowed, but during testing the maximum length of an episode is 20 steps. For the second setting the domain is extended to four boxes, two cities and two trucks and the goal is to bring three specific boxes to certain locations within 50 time steps.

5.2 Experiments

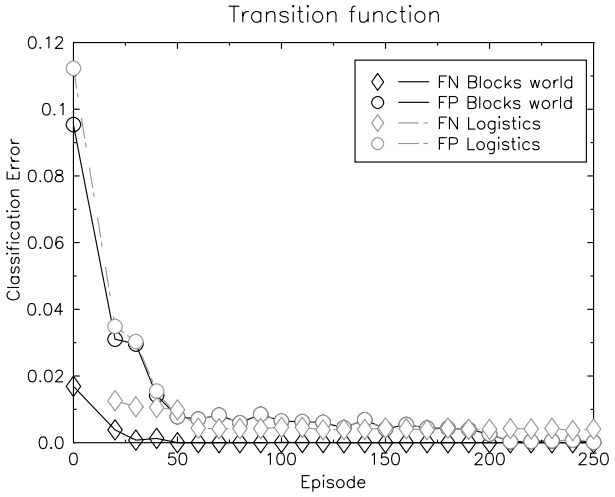

Figure 2: Errors made by the learned transition function

First, we test the quality of the transition function. To achieve this, we apply the learned model as a classifier. We randomly generate facts and predict if they will be true in the resulting next state, given some state and action. Figure 2 shows the percentage of errors made per step, False Positives (FP) and False Negatives (FN) indicate the number of atoms that were incorrectly predicted as true and false respectively. For the blocks world with seven blocks the transition function is learned rather rapidly and is optimal after about 200 episodes. For the logistics domain with five boxes, trucks and cities, it appears that a reasonable transition function is learned rapidly, but it never reaches optimal performance.

Next, we evaluate the improvement obtained by lookahead planning. Therefore, we ran both experiments with standard RRL-TG and experiments with different amounts of lookahead using the learned model. Figure 3 and Figure 4 show the results for the $on(a, b)$ and $stack$ goals of the blocks world respectively and for the logistics domain the results are plotted in Figure 5.

[4]Specific instantiations vary from episode to episode.

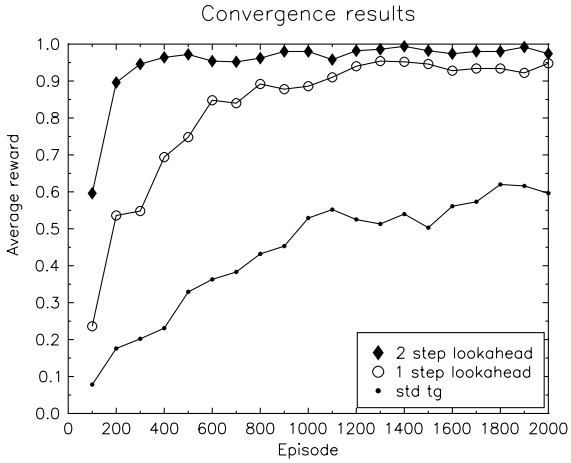

Figure 3: Blocks world with $on(A, B)$ goal

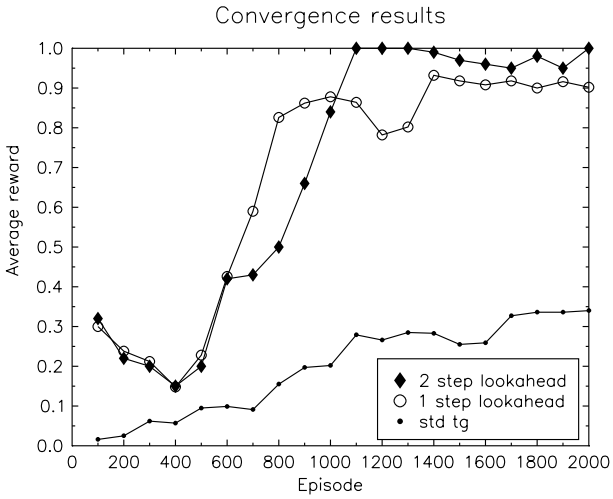

Figure 4: Blocks world with $stack$ goal

These experiments show that the number of episodes needed to obtain a good policy is much smaller when looking ahead. However, learning and lookahead takes additional computation time. Therefore, for worlds where performing actions is not expensive, it is also useful to evaluate how the total time needed to obtain a good policy compares with and without learning. Figure 6 shows the average reward in function of the time needed to learn the policy for the on(a,b) goal (the other settings produce similar results, omitted due to lack of space). Due to the computational costs of learning the model, using no lookahead performs better than single step lookahead. Indeed, in our current implementation, learning the model of the world is the computational bottleneck for small levels of lookahead. This is mainly due to the sampling techniques to create the learning examples, something we will improve in future work. However, performing two steps of lookahead still outperforms the standard version without lookahead.

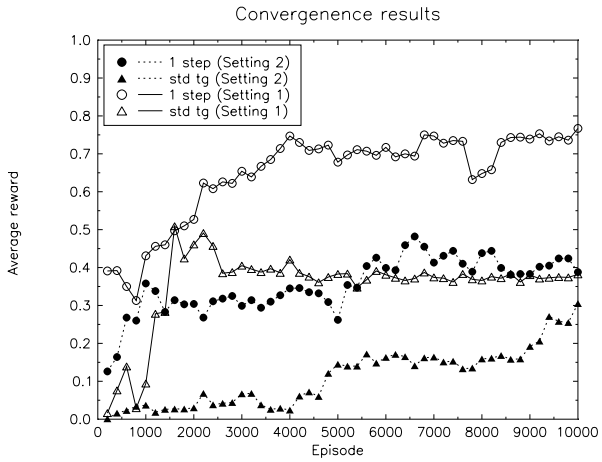

Figure 5: Logistics

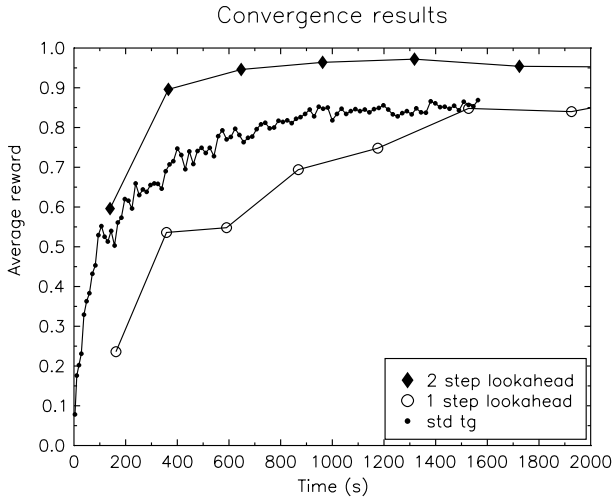

Figure 6: Blocks world with the $on(A, B)$ goal

Finally, we evaluate the robustness of our approach. We set up this experiment by generating less training examples for the transition function per step, extending the time to learn a good transition function. Every 30 episodes, we test both the quality of the transition function (the number of classification errors) and the average reward obtained by the learned policy using single step lookahead. In Figure 7 we plotted these points for two different learning rates, showing the obtained reward in function of the quality of the model. The horizontal line shows the performance of a learning agent after 2000 episodes using no lookahead. The figure shows that even when the learned model is less accurate, it is still beneficial to use lookahead. Only when the model performs very inaccurately, performance can drop.

6 Related Work

An important part of related work are the indirect or model-based approaches in the propositional setting. Most of these fit the Dyna architecture [Sutton, 1991] as mentioned in the introduction. There the agent learns a model of the world

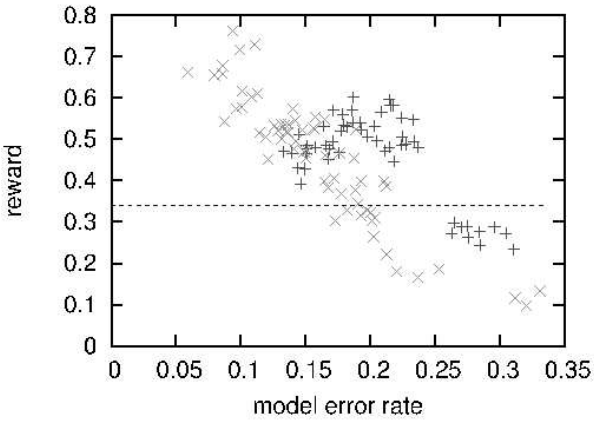

Figure 7: Reward versus the quality of the model

and uses these approximations of the transition and reward function to perform hypothetical actions to generate extra updates for the Q- or value function. Algorithms such as prioritized sweeping [Moore and Atkeson, 1993] (and extensions of these) focus the hypothetical actions on interesting parts of the state space. As mentioned earlier, these approaches are orthogonal to our approach and will be explored in our future work.

Learning a model of the environment becomes a non-trivial task in the relational setting. One method that focuses specifically on learning transition functions with relational structure is [Zettlemoyer *et al.*, 2005]. They present a method for learning probabilistic relational rules when given a dataset about state transitions that are applicable in large noisy stochastic worlds. The main difference with (the learning part of) our approach is that this method is not directly applicable to Reinforcement Learning, since it does not work incrementally and it is limited to domains where actions only have a small number of effects.

The emerging field of Statistical Relational Learning has seen a lot of work on relational upgrades of Bayesian networks. More specifically, [Sanghai *et al.*, 2005] defines Relational Dynamic Bayesian Networks (RDBNs) as a way to model relational stochastic processes that vary over time.

Combining search and RL has shown to be successful in the past, for instance in the context of game playing [Baxter *et al.*, 1998]. In [Davies *et al.*, 1998], online search is used with a (very) approximate value function to improve performance in continuous-state domains. Our approach can be seen as an instance of the *Learning Local Search (LLS)* algorithm described there.

Many new RRL algorithms have been proposed lately, but to our knowledge this is the first indirect RRL approach. The most related is [Sanner, 2005], where a ground relational naive Bayes Net is learned as an estimation of the Value function. The major difference however is that this work does not consider the aspects of time since they consider game playing and hence restrict themselves to undiscounted, finite-horizon domains that only have a single terminal reward for failure or success.

7 Conclusions and Future Work

In this paper we presented MARLIE, the first reinforcement learning system learning and exploiting a relational model of the world on-line. We argued that this system has several advantages when compared to earlier work. Compared to earlier on-line learning approaches, our approach is fully relational, allowing for a wider range of problems. Compared to earlier relational approaches, we argue that learning a complete model may be difficult. MARLIE builds probabilistic models allowing for partial models and uncertainty about the dynamics of the world, while at the same time exploiting knowledge as soon as it is available.

There are several directions for future work. First, one could improve the sampling strategies for looking ahead; e.g. by using iterative deepening strategies or sampling some random variables only lazily. Second, self-evaluation may be beneficial. In this way once it is shown that (parts of) the learned transition function is correct or good enough, no more resources need to be invested in it (but it may stay useful to check it at regular times). On the other hand, if it is known that the learned model or certain parts of it are not satisfactory yet, this can be taken into account in the decision making process. Finally, it would also be interesting to investigate how to use poor or partial models in combination with traditional planning reasoning strategies.

Acknowledgments

Tom Croonenborghs is supported by the Flemish Institute for the Promotion of Science and Technological Research in Industry (IWT). Jan Ramon and Hendrik Blockeel are postdoctoral fellows of the Fund for Scientific Research of Flanders (FWO-Vlaanderen).

References

[Baxter *et al.*, 1998] J. Baxter, A. Tridgell, and L. Weaver. Knightcap: A chess program that learns by combining td(λ) with game-tree search. In *Proc. of the 15th Int. Conf. on Machine Learning*, pages 28–36, 1998.

[Davies *et al.*, 1998] S. Davies, A. Ng, and A. Moore. Applying online search techniques to continuous-state Reinforcement Learning. In *Proc. of the 15th National Conf. on Artificial Intelligence*, pages 753–760, 1998.

[Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):33–58, 1989.

[Driessens *et al.*, 2001] K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *Proc. of the 12th European Conf. on Machine Learning*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 97–108, 2001.

[Driessens, 2004] K. Driessens. *Relational Reinforcement Learning*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 2004.

[Fern *et al.*, 2006] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:85–118, 2006.

[Fierens *et al.*, 2005] D. Fierens, J. Ramon, H. Blockeel, and M. Bruynooghe. A comparison of approaches for learning probability trees. In *Proc. of 16th European Conf. on Machine Learning*, volume 3720 of *Lecture Notes in Artificial Intelligence*, pages 556–563, 2005.

[Kearns *et al.*, 2002] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. *Machine Learning*, 49(2-3):193–208, 2002.

[Kersting and De Raedt, 2004] K. Kersting and L. De Raedt. Logical Markov Decision Programs and the convergence of logical TD(λ). In *Proc. of the 14th International Conf. on Inductive Logic Programming*, pages 180–197, 2004.

[Kersting *et al.*, 2004] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *Proc. of the 21th International Conf. on Machine Learning (ICML-2004)*, pages 465–472, Canada, 2004.

[Moore and Atkeson, 1993] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[Neville *et al.*, 2003] J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proc. of the 9th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, 2003.

[Sanghai *et al.*, 2005] S. Sanghai, P. Domingos, and D. Weld. Relational Dynamic Bayesian Networks. *Journal of Artificial Intelligence Research*, 24:759–797, 2005.

[Sanner, 2005] S. Sanner. Simultaneous learning of structure and value in relational reinforcement learning. In *Proc. of the ICML 2005 Workshop on Rich Representations for Reinforcement Learning*, 2005.

[Sutton and Barto, 1998] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

[Sutton, 1991] R. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, 1991.

[Tadepalli *et al.*, 2004] P. Tadepalli, R. Givan, and K. Driessens. Relational reinfocement learning: An overview. In *Proceedings of the ICML'04 Workshop on Relational Reinfocement Learning*, 2004.

[Tesauro, 1995] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–67, March 1995.

[van Otterlo, 2005] M. van Otterlo. A survey of reinforcement learning in relational domains. Technical Report TR-CTIT-05-31, University of Twente, 2005. ISBN=ISSN 1381-3625.

[Zettlemoyer *et al.*, 2005] L. Zettlemoyer, H. Pasula, and L. Kaelbling. Learning planning rules in noisy stochastic worlds. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 911–918, 2005.