Model-Assisted Approaches for Relational Reinforcement Learning: Some challenges for the SRL community

Tom Croonenborghs TOM.CROONENBORGHS@CS.KULEUVEN.AC.BE
Jan Ramon JAN.RAMON@CS.KULEUVEN.AC.BE
Hendrik Blockeel HENDRIK.BLOCKEEL@CS.KULEUVEN.AC.BE
Maurice Bruynooghe MAURICE.BRUYNOOGHE@CS.KULEUVEN.AC.BE
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

For a relational reinforcement learning (RRL) agent, learning a model of the world can be very helpful. However, in many situations learning a perfect model is not possible. Therefore, only probabilistic methods capable of taking uncertainty into account can be used to exploit the collected knowledge. It is clear then that RRL offers an interesting testbed for statistical relational learning methods. In this paper, we describe an algorithm taking a middle ground between model-free and model-based (Relational) Reinforcement Learning. A model of the world dynamics in the form of a relational Dynamic Bayesian Network (DBN) is learned incrementally. Empirical results show that sampling the partially learned model outperforms traditional RRL Q-learners.

We also focus on a number of open problems. First, it is clear that other SRL techniques, besides the one we are using, could be used just as well. It might be interesting to see what their strengths and weaknesses are in the specific RRL context. In addition, it is typical for our approach that chunks of partial knowledge are obtained, and little is known about how to combine, evaluate and exploit this partial knowledge more efficiently.

1. Introduction

Relational Reinforcement Learning (Džeroski et al., 2001) has received a lot of attention over the last few years. The focus has merely been on the model-free setting, where the agent learns the policy (or value function) directly by sampling the environment, but without first learning information about this environment. Recently, various kinds of relational MDPs (RMDPs) have been formalized with different methods that deduce an (optimal) policy, given the full specification of this RMDP as input.

When this RMDP is not given, it can be very useful for an RRL-agent to learn about his environment by e.g., learning a transition function. So far, little research has been on using the agent's experience in gathering knowledge about the world dynamics, although we have shown in (Croonenborghs et al., 2004) that (partial) models about these dynamics can be useful for Relational Reinforcement Learning. However, learning probabilistic (transition) models with relational structure is known to be a hard problem.

Furthermore, in a real reinforcement learning setting, learning about the world dynamics and exploiting this knowledge can not be separated. To learn a good model of the world, it may be necessary to first develop a good policy that can reach all the important places of the world. On the other hand, to develop a good policy, some knowledge of the world dynamics may be essential. Moreover, since learning a complete and perfect model is often not possible, only probabilistic methods capable of taking uncertainty into account can be used to exploit this collected knowledge.

In this paper, a first model-assisted approach is presented by combining a model-free (Q-learning) with a model-based approach. The agent performs Q-Learning, but while he explores his environment, he simultaneously and incrementally learns a model of the world. This model is then used to increase the performance of Q-Learning by looking (a small number of) steps ahead, in order to obtain a better estimate of the Q-value.

We also discuss a number of open problems in this area. In particular, though research in recent years helped a lot to understand the fundamentals of relational probabilistic

Presented at the ICML Workshop on Open Problems in Statistical Relational Learning, Pittsburgh, PA, 2006.

modeling, little is known about how to combine, evaluate and exploit the collected partial knowledge more efficiently.

The remainder of this paper is organized as follows. First, Section 2 presents some background. Section 3 discusses related work. Section 4 describes a first indirect approach to Relational Reinforcement Learning. Before concluding in Section 6, we discuss a number of challenges and open problems in Section 5.

2. Background

Reinforcement Learning (RL) (Sutton & 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 & De Raedt, 2004; Kersting et al., 2004; Mausam & Weld, 2003) [1]. We present a simple form of a RMDP that will be used in this paper.

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 state in the state space* $\mathbb{S}$ *is a set of ground state atoms, i.e.,* $\mathbb{S} = \{p(c_1, \ldots, c_n)|p/n \in P_S$ *and* $\forall i : c_i \in C\}$*; an action in the action state is a ground action atom, i.e.,* $\mathbb{A} = \{p(c_1, \ldots, c_n)|p/n \in P_A$ *and* $\forall i : c_i \in C\}$*. The transition function* $T : \mathbb{S}\times\mathbb{A}\times\mathbb{S} \rightarrow [0, 1]$ *defines a probability distribution over the possible next states, i.e.* $T(s, a, s')$ *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. This optimality is often defined as a function of the discounted, cumulative reward, i.e. the goal is to find a policy $\pi : \mathbb{S} \rightarrow \mathbb{A}$ that maximizes: $V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^i R(s_t, \pi(s_t))|s_0 = s]$, 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 $\pi^*(s) = \text{argmax}_a\, Q(s, a)$.

[1] An overview of the different formalizations can be found in (van Otterlo, 2005).

3. Model-assisted approaches for Relational Reinforcement Learning

Most model-based approaches in the propositional setting fit the Dyna architecture (Sutton, 1991). The agent learns a model of the world 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 & Atkeson, 1993) (and extensions of these) focus the hypothetical actions on interesting parts of the state space. In these approaches table-based methods are used to approximate the transition and reward distributions, although there has been some work on learning more structured representations. An example is (Dearden, 2001), where tree-based representations of the conditional probabilities are learned to perform prioritized sweeping based on the *structured policy iteration* algorithm (Boutilier et al., 1995).

Although indirect RL has shown its advantages, little research has been on indirect RRL. However, it seems that knowing information about the environment is especially interesting in the relational setting, where one deals with complex domains that contain a lot of structure. Of course, when making the step to relational domains, learning a model of the world becomes more challenging. Moreover, learning a perfect model will often be impossible and handling this uncertainty will be an important aspect of indirect RRL.

One method that focusses specifically on learning transition functions with relational structure is Pasula et al. (Pasula et al., 2004). They present a method for learning probabilistic relational rules when given a dataset about state transitions. In (Zettlemoyer et al., 2005) they extend this method to show that it is applicable in large noisy stochastic worlds. This method is however not directly applicable to Reinforcement Learning, since it does not work incrementally and in practice the assumption that actions only have a small number of effects is often violated.

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.

In the next section, we present a first model-assisted approach. The transition function is learned online using logical decision trees, which can be seen as a Relational Dynamic Bayesian Network. This model is then used to augment regular Relational Q-Learning with lookahead planning trees. This lookahead will allow the agent to obtain more accurate Q-values and hence accelerate convergence. We believe that Q-learning is not always the optimal choice for RRL and we would like to investigate alternatives in the

future, although it would be interesting to upgrade certain of these Q-based indirect approaches to the relational case, e.g. (Boutilier et al., 1995). For this first study on model learning however, starting from and comparing with existing systems seems useful.

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.

To our knowledge, this is the first method that addresses the problem of indirect Relational Reinforcement Learning. A recent overview of different methods in the field of RRL can be found in (van Otterlo, 2005). 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.

4. Learning a Relational Dynamic Bayesian Network with an application in RRL

When learning a model of the world, a substantial part of the running time, no complete model is known. This excludes a number of possible ways to exploit the world model. E.g. planning techniques that construct plans to reach a goal and rely on preconditions and postconditions of actions are unlikely to succeed. Still, the models may be sufficiently good to predict parts of the next states correctly, and even if the predicted next states are partly wrong, they can be used to look ahead in what will come. Sampling possible states a few steps ahead in the future can greatly help to see the direction to (high) rewards. Our proposed algorithm consists of two parts: the first module incrementally learns a transition and reward function in the form of probability distributions $T'(s'|s,a)$ and $R'(r(s,a)|s,a)$. Sections 4.1 and 4.2 details this module. A second module, described in Section 4.4 exploits the knowledge in the learned model to determine the agent's policy.

4.1. Representation and learning of the transition function

First, we will describe how we model the transition function that needs to be learned, given P_S, P_A and C of the RMDP.

One can model an episode of an agent as a large dynamic Bayesian network (DBN) (Dean & Kanazawa, 1989) with in each layer the state of the world and the action of the agent at a particular time point. However, as we are working in a relational setting, this DBN is a relational one. As explained previously, a state is a set of ground state atoms. Hence, in the layer of the DBN representing a state there is a binary random variable for every ground state atom. For every time point t, there is also a random variable a_t ranging over all atoms of P_A and C, representing valid actions. The action taken depends on the current knowledge of the agent and the current state. We will not explicitly model its conditional probability distribution (CPD).

In this work we do not consider dependencies among random variables of the same state, i.e. we assume that the random variables describing the next state depend only on the current state. This assumption can easily be loosened by providing an ordering on the nodes and letting nodes only depend on smaller nodes in the same state. However, learning bayesian net structure is known to be a hard problem. In particular, in the case of online learning, a revision of the structure would interfere with the learning of the conditional probability tables in uninvestigated ways and will therefore only be considered in future work.

Hence, for every predicate symbol $p \in P_S$, we have a conditional probability distribution $T'_p(x|s,a)$ giving 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.

For every predicate, we will represent the CPD with a probability tree represented with logical decision trees (Blockeel & De Raedt, 1998; Fierens et al., 2005). An example of such a probability tree CPD for $clear(X)$ in the blocks world is given in Figure 1.

We would like to learn this model in an incremental way while the agent is doing Q-learning. As we do not consider structure learning, learning the transition function reduces to learning the (relational) conditional probability tables. We will use for this task an on-line relational regression tree learning algorithm, based on the TG algorithm (Driessens et al., 2001).

The learning examples for this algorithm simply have to state if a certain ground atom is present in the resulting state or not. These examples can easily be created from the agent's interaction with his environment. Note that the number of possible atoms in the next state may be very large, and therefore we do not generate all possible examples but apply a suitable sampling strategy.

4.2. Learning other models

Learning the reward function $R'(r(s,a)|s,a)$ is more straightforward in the sense that this is similar as learning the Q-function. The only difference is that the learning ex-

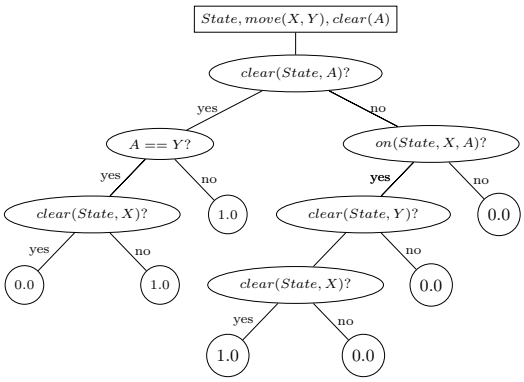

Figure 1. An example probability tree: It shows the probability that block A will be clear, given the fact that the action $move(X, Y)$ is executed in state $State$. The first node in the tree checks if block A was already clear (essentially the frame assumption). If this is the case, it can only become 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 resulting state if the block directly on top of it got moved to another block.

amples contain the reward received for a certain state and action instead of the calculated Q-value.

The transition function only indirectly states the preconditions of an action, it could however be convenient to learn a model that explicitly states the preconditions of a certain action. It is common to assume that if an illegal action is tried by the agent, the world remains unchanged. In this work a simple approach to learning preconditions is taken by learning a function $T'(s' = s|s, a)$ that predicts the probability that a (part of the) state will be changed when executing a certain action in it.

4.3. Predicting the resulting state

If a (partially correct) transition function $T'(s'|s, a)$ is available, it becomes possible to predict the probability distribution over the possible next states. At the moment, a sampling approach is used to predict a possible next state. It iterates over all possible ground atoms in a state and decides for each atom individually if it is part of the resulting state, depending on the predicted conditional probability.

4.4. Q-Learning with Lookahead

Being able to predict the next state, allows us to look several steps ahead to get a more informed Q-value. The idea is to build a *lookahead planning tree*, much in the same way as a *sparse lookahead tree* is build in (Kearns et al., 2002) to obtain near-optimal policies for large MDPs.

Since the transition function can be stochastic or partially incorrect, 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. Using the Bellman equation (Eq. 1) and the learned reward function these values can then be back-propagated to their parents by averaging over the different edges (samples), until estimates are obtained for the top level.

When the function $T'(s' = s|s, a)$ is learned as a model for the preconditions, it can be used to prune the number of actions in the tree.

An important benefit of being able to predict the next state is that when doing single step lookahead, instead of learning the $Q(s, a)$-function a Q-function can be learned over the (State, Action, NextState)-triplets. This is not only useful because in a lot of domains, a Q-value can be expressed more easily in function of the next state, but as shown in (Croonenborghs et al., 2006) (R)RL-agents can gain from multiple (possibly redundant) sources. The same applies for the reward function, therefore we use these functions to predict the Q-values and rewards of the last level. For higher levels the lookahead algorithm described above is used.

Although the Q-values are determined in a slightly different way, the same strategy can be used to derive policies for the agent as in regular Q-learning.

4.5. Empirical evaluation

4.5.1. 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 modelling the preconditions to prune the lookahead tree. The exploration policy consists of performing a single step lookahead. The figures show the average over a 10-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 As a first test domain, we used the blocks world as it is defined in a lot of work on Relational Reinforcement Learning(Driessens, 2004)[2]. The world con-

[2]The main difference is that here the agent can execute every possible action in a certain state instead of only the legal ones.

tains five blocks and the standard goals, i.e. the $stack$-goal, the $unstack$-goal and the $on(A, B)$-goal are used. 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. And in the $unstack$-goal the agent is rewarded iff all blocks are on the floor, i.e. there is no block on top of another block.

During exploration, episodes have a maximum length of 20 steps above the ones needed by the optimal policy, during testing only optimal episodes are allowed. For the $on(A, B)$ goal, we also tested a more difficult setting with 10 blocks. For this setting, the agent was allowed to take 30 extra steps when exploring and averages are shown over a 5-fold run. 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 certain boxes to certain cities[3]. 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 timesteps.

4.5.2. Preliminary experiments

In this section we present some preliminary experiments to show the potential benefits of indirect Relational Reinforcement Learning. Currently, more experiments are performed to compare different sampling strategies in different environments. For the blocks world with five blocks we compared three different settings: a regular Q-learning agent (std tg), an agent doing single step lookahead[4] and an agent that looks two steps ahead. For the $on(A, B)$-goal[5]

[3]Specific instantiations vary from episode to episode.

[4]i.e. by learning a $Q(S, A, S')$-function

[5]The most difficult goal for standard TG.

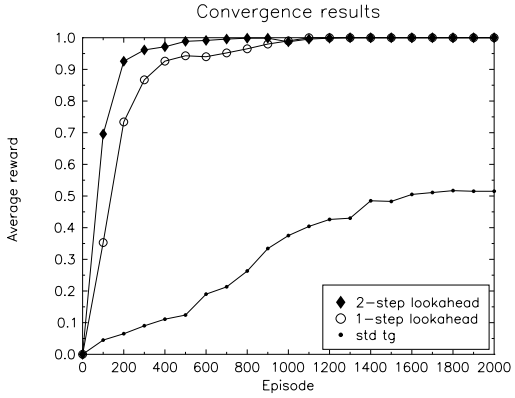

Figure 2. Episode $on(A, B)$

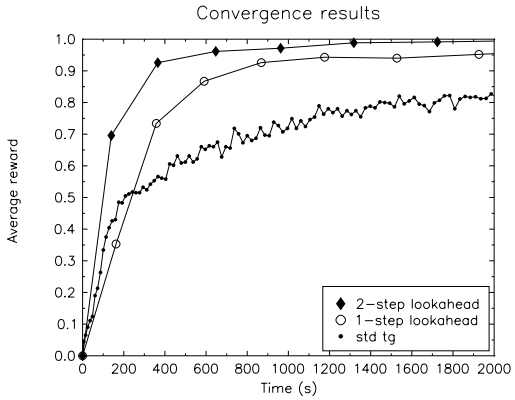

Figure 3. Timings $on(A, B)$

(Figure 2) using lookahead clearly outperforms the standard setting. Doing a two step lookahead further improves performance, but the differences are small due to the fact that they both convergence rather fast. Since learning the transition function is computationally expensive, we also show the convergence in function of the time needed to learn the policy in Figure 3. Considering this as a convergence measure, using lookahead still significantly improves over using regular Q-learning. Since the time to execute a policy is practically neglectable to the time to learn the different models, looking two steps ahead also outperforms the single step lookahead under this convergence measure.

For the $stack$ (Figure 4) and $unstack$ goal (Figure 5) single step lookahead only improves in the beginning of learning, but after about a 1000 episodes it does not really gain over the standard setting. The main reasons for this are that it is more difficult to identify the goal-states (in the next state), something which might be improved with more background knowledge and a richer language bias. On the other hand, there is in a sense more redundancy between the current and the afterstate for modelling the Q-function.

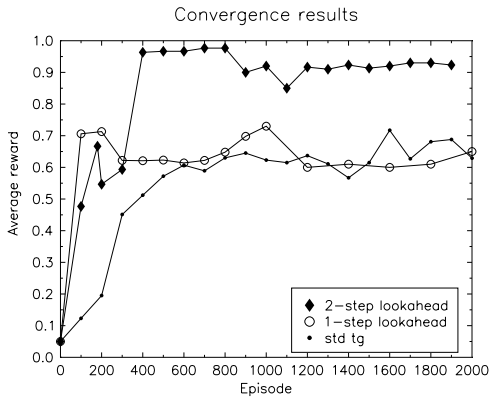

Figure 4. Stack goal

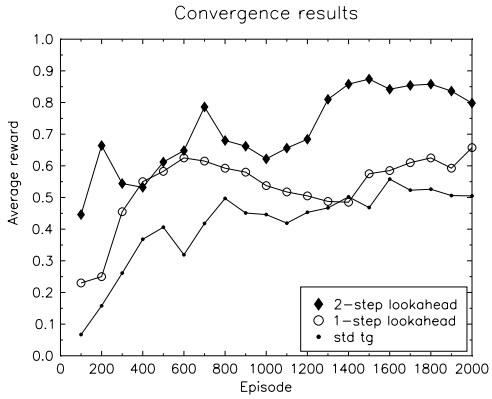

Figure 5. Unstack goal

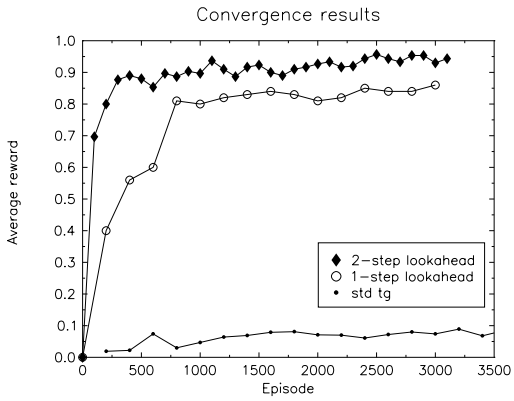

Figure 6. Blocks world with 10 blocks

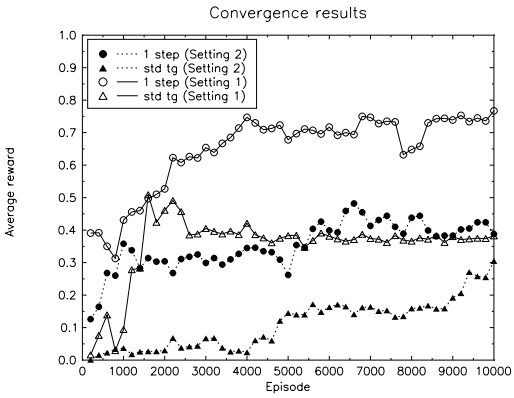

Figure 7. Logistics

It also shows that performing an extra step of lookahead helps for both problems.

In a bigger blocks world (Figure 6), it is hard for the standard Q-learning to learn a reasonable policy, after 10000 episodes a reward of about 0.3 is reached and looking ahead significantly improves the learning behavior. In the logistics domain, the variance is rather high, partly due to the limited background knowledge and language bias. Therefor, in general no optimal policy is learned because during some test runs no reasonable policy will be learned. It is clear however that using single step lookahead significantly outperforms regular Q-learning (Figure 7).

The quality of the transition function is also tested separately by applying it as a classification task, i.e. predict if a certain atom will be part of the afterstate, given a certain state and action.

Results are show when averaged over 10 samples per step and 10 testepisodes, False Positives (FP) indicate the number of atoms that were predicted as true but are not in the real afterstate and False Negatives (FN) ,i.e. atoms that were predicted as not being part of the afterstate, but are in reality. In the blocks world with five blocks (Figure 8), the transition function is learned rather rapidly and is optimal after about 200 episodes. Figure 9 shows the result of a logistics domain with five boxes, trucks and cities for the first 100 episodes, but on average it is not able to learn the transition function perfectly. It is interesting to see that despite the poor transition function looking ahead helps.

5. Challenges and Open Problems

In the previous sections, we showed how one can incrementally learn a model of the world and that exploiting this model, even if it is incomplete, can be beneficial for a reinforcement learning agent. However, several open problems remain, mainly in the statistical relational learning domain.

First, we believe that it would be advantageous to evaluate the different components of the learned model. In a first stage, the learned model will be very inaccurate, and it will make little sense to look ahead far, as the predicted next

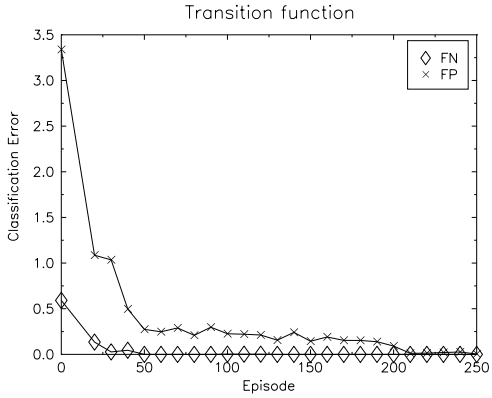

Figure 8. Transition function (blocks world)

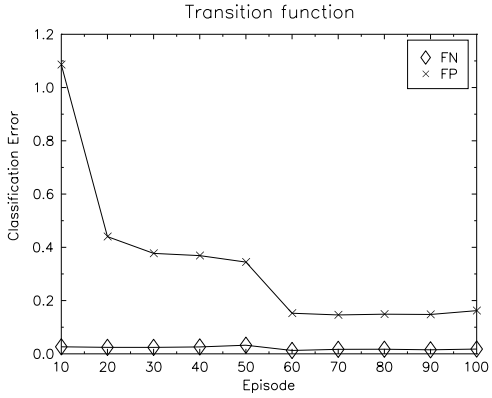

Figure 9. Transition function (logistics domain)

states will be too noisy. However, it is unclear when to decide when the model has improved sufficiently to look ahead further. Notice that a rather accurate evaluation of the learned model would be needed. A straightforward error measure such as the mean error is probably insufficient, as some parts of the world model may be harder to learn than others, and it makes a large difference whether the part which is difficult to learn is the part relevant for the agent (e.g. for predicting his future rewards) or concerns only a remote and unimportant process in the world. Being able to evaluate the model quality would also help in decision making on the meta learning level when several alternative learning strategies are available (e.g. learning a $Q(s, a, s')$ function vs. learning a $Q(s, a)$ and reward function and combining them).

A second important issue is the sampling strategy. It is a non-trivial task to come up with an efficient strategy to sample next states and look ahead efficiently. On a global level, one can imagine different search strategies, e.g. we intend to try an iterative deepening search, examining all actions at the current state but only the most promosing ones at deeper levels. On a local level, a single sample of a next state given the current state and the action should be generated. (Sanghai et al., 2005) presents some initial work on efficient state monitoring for Relational Dynamic Bayesian Networks, the methods proposed there are however only applicable in restricted settings. Most frameworks for representing probabilistic relational knowledge assign a single constant value to every fact that can be true (in the next state). However, as (Poole, 2003) points out, things often get difficult when reasoning over populations, e.g. probability distributions over all blocks are needed. Only a few representation frameworks allow to represent such probability distributions elegantly.

In the future, we also want to reason backwards and apply planning techniques. An important step into this direction is provided by (Kersting et al., 2004)). However, this approach assumes a closed world and does not deal with uncertain information as in the setting discussed in this paper. Finally, as already mentioned, there are the general questions of how other SRL approaches than the one described here would fare on learning a partial model of the world in parallel with learning an optimal policy, and in what other ways results from SRL may be applied to further the state of the art in RRL.

6. Conclusions

In this paper we presented a first model-assisted method in which a RRL-agent incrementally learns a model of the world dynamics in the form of a Dynamic Bayesian Network. Experiments show that the performance of a standard Q-learning agent can be improved when using this model to predict the afterstate. We also discussed a number of open problems for the SRL community in this area.

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 post-doctoral fellows of the Fund for Scientific Research (FWO) of Flanders. The authors would like to thank Daan Fierens for helpful comments and discussion.

References

Baxter, J., Tridgell, A., & Weaver, L. (1998). Knightcap: A chess program that learns by combining td(λ) with game-tree search. *Proceedings of the 15th International Conference on Machine Learning* (pp. 28–36). Morgan Kaufmann.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelli-*

gence, *101*, 285–297.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1104–1111). San Francisco: Morgan Kaufmann.

Croonenborghs, T., Ramon, J., & Bruynooghe, M. (2004). Towards informed reinforcement learning. *Proceedings of the ICML2004 Workshop on Relational Reinforcement Learning* (pp. 21–26). Banff, Canada.

Croonenborghs, T., Tuyls, K., Ramon, J., & Bruynooghe, M. (2006). Multi-agent relational reinforcement learning. Explorations in multi-state coordination tasks. *Learning and Adaptation in Multi Agent Systems: First International Workshop , LAMAS 2005, Revised Selected Papers* (pp. 192–206). Springer Berlin / Heidelberg. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41977.

Davies, S., Ng, A., & Moore, A. (1998). Applying online search techniques to continuous-state Reinforcement Learning. *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 753–760).

Dean, T., & Kanazawa, K. (1989). A Model For Reasoning About Persistence and Causation. *Computational Intelligence*, *5*, 33–58.

Dearden, R. (2001). Structured prioritized sweeping. *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 82–89).

Driessens, K. (2004). *Relational reinforcement learning*. Doctoral dissertation, Department of Computer Science, Katholieke Universiteit Leuven.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proceedings of the 12th European Conference on Machine Learning* (pp. 97–108). Springer-Verlag.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, *25*, 85–118.

Fierens, D., Ramon, J., Blockeel, H., & Bruynooghe, M. (2005). A comparison of approaches for learning probability trees. *Proceedings of 16th European Conference on Machine Learning, Porto, Portugal* (pp. 556–563).

Kearns, M., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, *49*, 193–208.

Kersting, K., & De Raedt, L. (2004). Logical Markov Decision Programs and the Convergence of Logical TD(λ). *Proceedings of the 14th International Conference on Inductive Logic Programming* (pp. 180–197). Springer-Verlag.

Kersting, K., Van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)* (pp. 465–472). Banff, Canada.

Mausam, & Weld, D. S. (2003). Solving relational mdps with first-order machine learning. *Workshop on Planning under Uncertainty and Incomplete Information, at ICAPS*.

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

Pasula, H., Zettlemoyer, L. S., & Kaelbling, L. P. (2004). Learning probabilistic relational planning rules. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)* (pp. 73–82).

Poole, D. (2003). First-order probabilistic inference. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (pp. 985–991). Acapulco, Mexico: Morgan Kaufmann.

Sanghai, S., Domingos, P., & Weld, D. (2005). Relatioal Dynamic Bayesian Networks. *Journal of Artificial Intelligence Research*, *24*, 759–797.

Sanner, S. (2005). Simultaneous learning of structure and value in relational reinforcement learning. *Proceedings of the ICML 2005 Workshop on Rich Representations for Reinforcement Learning*.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: The MIT Press.

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

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

Zettlemoyer, L. S., Pasula, H., & Kaelbling, L. P. (2005). Learning planning rules in noisy stochastic worlds. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)* (pp. 911–918).