On Using Guidance in Relational Reinforcement Learning

Kurt Driessens
Department of Computer Science
K.U.Leuven
Belgium
kurt.driessens@cs.kuleuven.ac.be

Sašo Džeroski
Department of Intelligent Systems
Jožef Stefan Institute
Slovenia
saso.dzeroski@ijs.si

Abstract

Reinforcement learning, and Q-learning in particular, encounter two major problems when dealing with large state spaces. First, learning the Q-function in tabular form may be infeasible because of the excessive amount of memory needed to store the table and because the Q-function only converges after each state has been visited multiple times. Second, rewards in the state space may be so sparse that with random exploration they will only be discovered extremely slowly. The first problem is often solved by learning a generalization of the encountered examples (e.g., using a neural net or decision tree). Relational reinforcement learning (RRL) is such an approach; it makes Q-learning feasible in structural domains by incorporating a relational learner into Q-learning. To solve the second problem a use of "reasonable policies" to provide guidance has been suggested. In this paper we investigate the best ways to provide guidance in two different domains.

1 Introduction

Q-learning (Watkins, 1989) is a form of reinforcement learning where the optimal policy is learned implicitly in the form of a Q-function, which takes a state-action pair as input and outputs the quality of the action in that state. The optimal action in a given state is the action with the largest Q-value.

One of the main limitations of standard Q-learning is related to the number of different state-action pairs that may exist. The Q-function can in principle be represented as a table with one entry for each state-action pair. When states and actions are characterized by parameters, the number of such pairs grows combinatorially in the number of parameters and thus can easily become very large, making it infeasible to represent the Q-function in tabular form, let alone learn it accurately (convergence of the Q-function only happens after each state-action pair has been visited many times). This problem is typically solved by integrating into the Q-learning algorithm an inductive learner, which learns a function that generalizes over given state-action pairs. Thus reasonable estimates of the Q-value of a state-action pair can be made without ever having visited it. Examples include neural networks (Bertsekas and Tsitsiklis, 1996), nearest neighbour methods (Smart and Kaelbling, 2000) and regression trees (Chapman and Kaelbling, 1991).

A relational learner is employed by Džeroski et al. in(Džeroski et al., 1998), hence the name "relational reinforcement learning" or RRL. RRL uses first order representations for states and actions, and learns a first order regression tree (Kramer, 1996; Blockeel et al., 1998) that maps these structural descriptions onto real numbers. The use of first order representations gives RRL a broader application domain than classical Q-learning approaches. Examples of such relatively complex applications are described in more detail in this paper, include learning to solve simple planning tasks in a blocks world, or learning to play certain computer games (such as Tetris).

When dealing with large state spaces, as is usually the case in structured domains, another problem encountered by Q-learning is that rewards may be distributed very sparsely throughout this space. In previous work (Driessens and Džeroski, 2002), we have suggested to supply guidance to the learning system at the start of the learning process to provide it with enough knowledge to explore the rest of the state-space on its own. In this work we will try to explore other ways of supplying guidance

to RRL.

The remainder of the paper is structured as follows. In Section 2 we discuss different ways in which guidance can be incorporated in a Q-learning approach. In Section 3, we present a number of structural domains and discuss specific difficulties that are encountered when solving tasks in these domain. In Section 4, we present experimental results in these domains, comparing different ways of providing guidance to RRL and we conclude in Section 5. For related work we would like to refer to previous work (Driessens and Džeroski, 2002).

2 The Approach

2.1 The RRL-TG System

We make use of the RRL-TG algorithm as described in (Driessens et al., 2001). The RRL system is a Q-learning system that uses a first order representation for the encountered states, actions and the resulting Q-function. RRL starts with running a normal episode just like a standard reinforcement learning algorithm but uses this episode to generate a set of examples for tree induction. At the end of each episode, a first order regression tree builder TG is supplied with the encountered $(state, action, q\text{-}value)$-triplets and incrementally builds a first order regression tree that represents the Q-function. In the next episode, the q-values predicted by the generated tree are used to calculate the q-values for the next set of examples.

2.2 On Providing Guidance

Although random policies can have a hard time reaching sparsely spread rewards in a large world, it is often possible to reach these rewards in a reasonable number of steps by using "reasonable" policies. While optimal policies are certainly "reasonable", non-optimal policies are often easy (or easier) to implement or generate than optimal ones. One obvious candidate for an often non-optimal but reasonable controller would be a human expert.

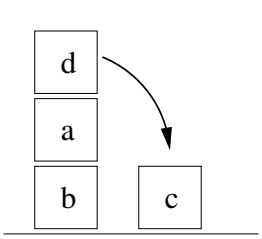
To integrate the guidance that these reasonable policies can supply with our relational reinforcement learning system, we use the given policy to generate a number of behaviour traces, together with the rewards that the actions in such a trace would receive. In case of a human controller, one can log the normal operation of a system together with the corresponding rewards. The generated traces are translated to state-action-qvalue triplets and then presented to the RRL algorithm. The generalisation algorithm responsible for the generation of the Q-function in RRL can use the state-action-qvalue triplets generated this way to build a partial Q-function.

In earlier work (Driessens and Džeroski, 2002), we supplied all of the guidance (traces) in the beginning of the learning experiment. Although this generally turned out to help RRL reach higher levels of performance, we also noticed the occurrence of overgeneralization in the regression algorithm TG. We propose in this work to counter this overgeneralization-effect caused by the supply of optimal state-action combinations only, by interleaving the guided traces with the exploration traces. We will compare different rates at which the guidance is supplied.

In the guided traces, the actions are chosen by a pre-defined (reasonable) policy and are only observed by RRL. In the exploration traces, actions are chosen autonomously by RRL, using the current estimate of the Q-function and the Boltzmann statistics (Kaelbling et al., 1996). For the generalization engine TG, there is no difference between the two types of traces. It will simply transform each encountered $(state, action, q\text{-}value)$ triplet into one training example to learn from. A more formal explanation of this approach can be found in (Driessens and Džeroski, 2002).

We will test the different suggestions on two example applications: the Blocks World and the computer game Tetris. While we will use an optimal policy to provide guidance in the Blocks World, it is very difficult — and maybe impossible — to construct an optimal policy for Tetris. We will therefore compare guidance given by two different — but still fairly simple — policies, of which one performs considerably better than the other.

In the case of the Blocks World we will try to go one step further in the kind of guidance we provide. Since we test the performance of the strategy generated by RRL at regular time intervals, we can tell RRL in which cases it failed. Next time RRL asks for some guidance, we can allow it to ask for help in one of the cases that it



clear(d).
clear(c).
on(d,a).
on(a,b).
on(b,floor).
on(c,floor).

move(d,c).

Figure 1: Example state and action in the blocks-world.

could not solve. We have named this approach "active" because of the similarity of this technique to "active learning" strategies in classification tasks (Cohn et al., 1994).

3 The Domains

3.1 The Blocks World

We use the blocks world with 10 blocks as our first testing environment. Blocks can be on the floor or can be stacked on each other. We represent the states and actions in the blocks-world as shown in Figure 1 (Prolog notation). The available actions are $move(x, y)$ where $x \neq y$, x is a block and y is a block or the floor. There is no limit on the number of blocks that can be on the floor. In addition, we supply the RRL algorithm with the number of blocks, the number of stacks and the following background predicates: $equal/2$, $above/2$, $height/2$ and $difference/3$ (an ordinary subtraction of two numerical values).

With 10 blocks, the blocks world becomes large enough to illustrate the effect of guidance in Q-learning, while remaining simple enough to explore different guidance strategies. With 10 blocks, there are close to 59 million possible states in the blocks world.

We study the goal of putting one specified block on top of another specified block. Please note that the use of RRL permits the use of variables in the description of the goal and the Q-function and thus allows us to learn to stack any two given blocks, without having to restart the learning process when changing the names of the blocks. In a blocks world with 10 blocks, there are 1.5 million states that satisfy a specific $on(A, B)$ goal. A reward of 1 is given in case a goal-state is reached in the optimal number of steps; the episode ends with a reward of 0 if it is not. This limits the number of steps allowed per episode.

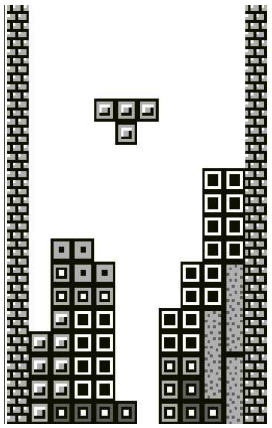


Figure 2: The TETRIS Game.

3.2 The Tetris Game

Tetris[1] is a well known puzzle-video game played on a two-dimensional grid. Differently shaped blocks fall from the top of the game field and fill up the grid. The object of the game is to keep the blocks from piling up to the top of the game field. To do this, one can move the dropping blocks right and left or rotate them as they fall. When one horizontal row is completely filled, that line disappears and the player scores points. When the blocks pile up to the top of the game field, the game ends.

In the tests presented, we only looked at the strategic part of the game, i.e., given the shape of the dropping and the next block, one has to decide on the optimal orientation and location of the block in the game-field. Using low level actions — turn, move left or move right — to reach such a subgoal is rather trivial and can easily be learned by RRL. However, the dropping of the block is instantaneous. It is impossible in our setup to slide a block into place sideways. We represent the full state of the Tetris game, the type of the next dropping block included. We allow RRL to use the following predicates (among others): $blockwidth/2$, $blockheight/2$, $rowSmaller/2$, $topBlock/2$, $holeDepth/2$, $holeCovered/1$, $fits/2$, $increasesHeight/2$, $fillsRow/2$ and

[1]Tetris was invented by Alexey Pazhitnov and is owned by The Tetris Company and Blue Planet Software.

$fillsDouble/2$. The system gets a reward after each action of 1 point for each (with that action) deleted line.

3.3 Comments

Both domains have large state spaces and are hard to solve with ordinary Q-learning. Although some of these problems can be tackled by using RRL, Q-learning in both of them has been shown to benefit from the added guidance (Driessens and Džeroski, 2002).

In Tetris, a good policy can supply a reward every 4 or 5 steps, which is quite frequent. However, a single bad action can have long lasting effects, thereby hiding obvious rewards from the learning algorithm. This, together with the fact that Tetris is a stochastic game (the shape of the next falling block is unknown) makes it a very hard application for Q-learning. For the Tetris game, an optimal policiy is hard — and may be impossible — to construct.

4 The Experiments

We expect that spreading the presented guidance will have two influences on the learning performance. In terms of learning speed, we expect the guidance to help the Q-learner to discover high yielding policies earlier in the learning experiment. Through the early discovery of important states and actions and the early availability of these state-action pairs to the generalization engine, we also expect that it is possible for the Q-learner to reach a higher level of performance — i.e., a higher average reward when testing the current strategy — in the available time.

To test these effects, we compare RRL with different guidance frequencies in the following setup: First we run RRL in its natural form and in the form where we supply all the guidance in the beginning of the learning episode (Driessens and Džeroski, 2002). We give it the possibility to train for a certain number of episodes; at regular time intervals we extract the learned policy from RRL and test it on a number of randomly generated test problems. For RRL with guidance, we replace a given number of episodes with traces from a hand-coded policy. We test the result in the same manner as the original RRL. Note that no matter what frequency we use to supply RRL with guidance, the total amount of guided traces will be the same in all cases.

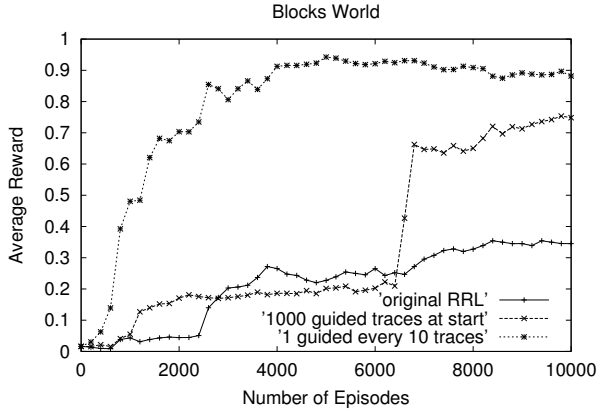


Figure 3: The Learning Curves for the Blocks World.

4.1 The Blocks World

As is clearly shown in Figure 3 the spreading of the guidance helps in terms of both learning speed and level of performance. Providing an equal amount of guidance in the beginning of learning gives a (relatively) comparable increase in level of performance, but not in terms of learning speed. This can be explained by the interaction of the TG-algorithm with the fact that only optimal traces are presented. When we supply RRL with only optimal traces, overgeneralization occurs. The generalization engine never encounters a non-optimal action and therefore, never learns to distinguish optimal from non-optimal actions. It will create a Q-tree that separates states which are at different distances from the goal-state and later, during exploration, expand this tree to account for optimal and non-optimal actions in these states. These trees are usually larger than they should be. RRL is often able to generalize over non-optimal actions in states that are close to the goal and optimal actions in states that are a little further from the goal in one leaf of its tree.

By spreading the guidance and supplying the optimal traces not all at once but interleaved with random — at least in the beginning of the learning process — traces avoids this overgeneralization. Of course, this problem (and as a consequence the solution as well) is strongly related to model building generaliz-

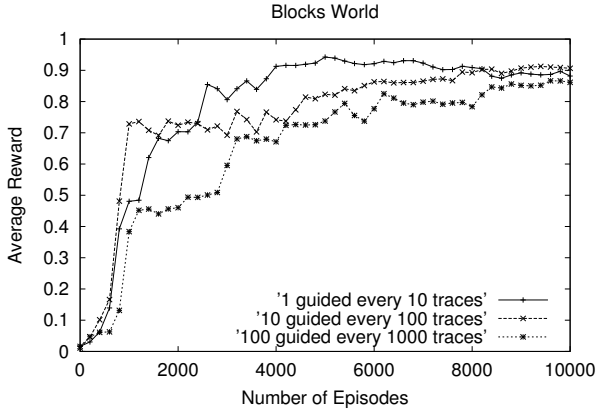


Figure 4: Comparing different Guidance Frequencies.

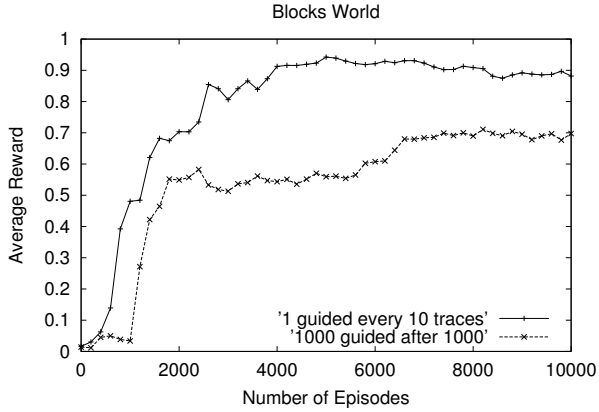


Figure 5: Comparing Spread Guidance to Delayed Batch Guidance.

ation algorithms such as tree-builders or rule-based systems. Other generalization techniques such as instance-based approaches might not suffer from this problem.

Figure 4 shows the influence of different frequencies used to provide guidance. Note that in all cases, an equal amount of guidance was used. Intuitively, it seems best to spread the available guidance as thin as possible and the performed experiments do not show any negative results of doing so. However, spreading out the guidance when there is only a small amount available (e.g. 1 guided trace every 10 000 episodes) might prevent the guidance from having any effect.

As a specific solution to the sparsity of guidance for the RRL-TG-algorithm we could let RRL ask for guidance episodes whenever it is bound to make a permanent decision. (In the case of the tree-learner TG this would be when it is about to make a new split in a leaf.) Even when this guidance is not case specific, it could be used to check whether a reasonable policy does not contradict the proposed split. Alternatively, one might decide to store (some of) the guided traces and re-use them: at present, all traces are forgotten once a split of the TG tree has been made.

When looking for a more general solution, one could try to provide a larger batch of guidance after RRL has had some time to try and explore the state-space on its own. This is related to a human teaching strategy, where providing the student with the perfect strategy at the start of learning is less effective than providing the student with that strategy after he or she has had some time to explore the systems behavior. Although Figure 5 shows inferior results for this approach when compared to the spread out guidance, this is probably due to the large size of the presented batch. Note also that learning here takes place faster than when all guidance is provided at the beginning.

4.2 The Tetris Game

The guidance strategy used in the Tetris experiment was very simple and included only the following rules:

1. Take an action that creates no new holes and does not increase the current height of the wall in the playing field.
2. If no action of type 1 can be found, take an action that does not increase the current height of the wall in the playing field.
3. If no action of type 1 or 2 can be taken, take an action that does not create a new hole.
4. If no action of type 1, 2 or 3 exists, take a random action.

This strategy scores an average of 6.3 lines per game.

Due to the lack of sufficient time to run more experiments, the shown graphs are performance curves of single learning experiments and as such show some erratic behavior. We will have better results for the final version.

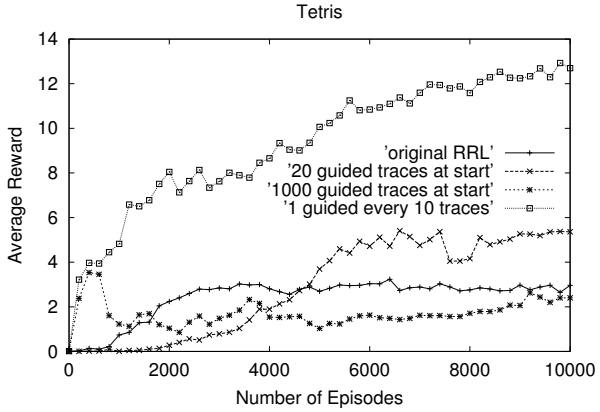


Figure 6: The Learning Curves for Tetris.

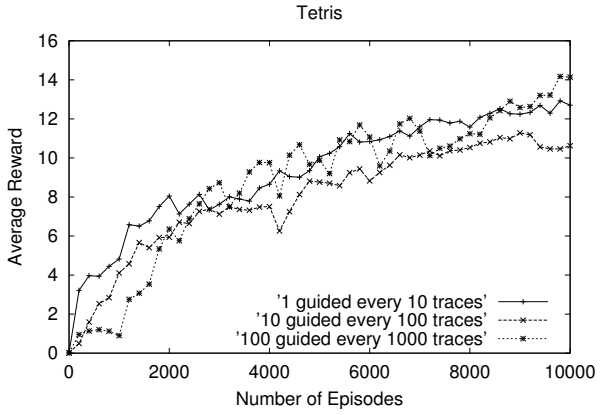


Figure 7: Comparing different Guidance Frequencies.

Figure 6 shows the improvement of RRL with guidance. Since the problem of overgeneralization is more apparent in the Tetris experiments — the experiment with 1000 starting traces does not perform better than the original RRL — we included an experiment where we gave RRL only 20 guided traces at the start of the experiment, to show that RRL does benefit from early guidance as well. However, the improvement received from spreading the guidance is even more apparent than in the Blocks World. Also note that the average number of lines deleted by RRL rises above 12 per game while the strategy used for guidance only reaches 6.3 lines per game.

Figure 7 shows the comparison of guidance frequencies. As we already noticed in the blocks-world experiments, although providing a lot of guided traces in the beginning of the

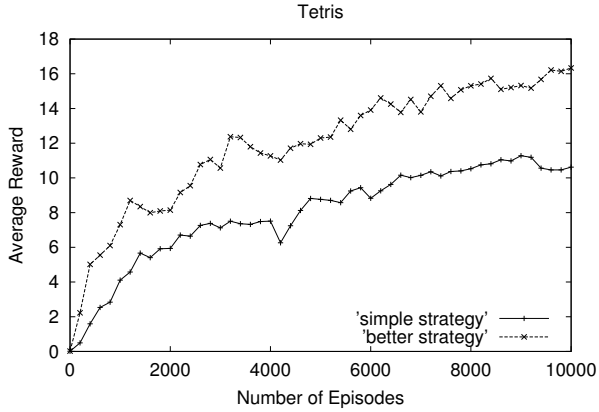


Figure 8: Comparing different Guiding Policies.

experiment will slow down the progress made by the RRL-system during the initial stages of the experiment, there is little to no difference between the performances later on in the experiment.

To test the influence of the performance of the policy used for guidance, we designed another (simple) strategy for Tetris. With the addition of a few more rules that tested the number of deleted lines a block would cause, we got the performance of the guidance strategy up to 16.7 lines per game on average. The results of using the two different strategies are shown in Figure 8. The graph shows that there is a significant increase in performance for using the better policy to guide RRL. This shows that the exploration insensitivity of table-based Q-learning (Kaelbling et al., 1996) does not carry over to Q-learning with generalization. However, one should notice that although the "guidance strategy" improved by approximatly 10 lines per game, the improvement of the resulting strategy learned by RRL is smaller.

4.3 "Active" Guidance

As stated at the end of Section 2, in the Blocks World where each episode is started from a randomly generated starting position, we can let RRL know in which cases it failed to reach the goal state, and give RRL the opportunity to ask for guided traces starting from some of these states. This will allow RRL to explore parts of the state space where it does not yet have enough knowledge and supply the TG-algorithm with examples which are not yet correctly predicted.

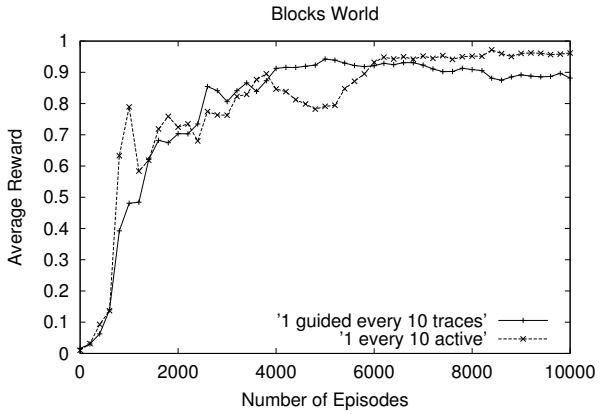


Figure 9: The Learning Curves for the Blocks World.

Figure 9 shows the results of this active guidance. Although there is little difference between the performance of the two approaches in the beginning of the learning experiments, the active guidance succeeds in pushing RRL to better performance at the end of the learning experiment. The percentage of cases where RRL does not reach the goal state is reduced from 11% to 3.9%. This is completely consistent with what one would expect. In the beginning, both systems will see more than enough new examples to both be able to increase the accuracy of their Q-function. However, when both the algorithms have a large part of the state-space covered by their Q-function, the specific examples provided by the active guidance allow for the Q-function to be extended to cover the outer reaches of the state-space.

Although this idea of "active" guidance seems very attractive both intuitively and in practice, it is not easy to extend this approach to applications with stochastic actions or a fixed starting state such as the Tetris game where the next block to be dropped is chosen randomly and the starting state is always an empty playing field. For the Tetris game one could imagine remembering the entire sequence of blocks and asking the guidance strategy for a game with the given sequence, however, given the large difference in the state provided by only a few different actions, we anticipate the effect of this approach to be very small. Another step towards active guidance in stochastic environments would be to keep track of actions (and states) with a large negative effect. For example in the Tetris game we could notice a large increase of the height of the wall of the playing field. We could then use these remembered states to ask for guidance. However, this approach requires not only a large amount of administration inside the learning system but also needs some a priori indication of bad and good results of actions.

5 Conclusions

In this paper, we explored several ways of including guidance into reinforcement learning. Although we have used relational reinforcement learning (RRL) in our experiments, we assume that the results carry over to all forms of Q-learning that use regression to approximate the Q-function. We have shown that spreading this guidance during the entire learning eposide can have a large influence on both the speed of learning and the overall level of performance that is obtained.

According to the results we obtained the approach does not suffer when the guidance is spread out thinly. We feel that spreading out the guidance as much as possible would yield the best results, provided there is enough guidance available to not lose its influence to noise elimination.

The influence of the performance of the strategy used for guidance seems small. Assuming the use of a "reasonable" policy, i.e. a policy that is able to discover the rewards in the state-space at a reasonable rate, the policy learned by RRL depends little on the performance of the policy used for guidance. The RRL-system is able to improve on the performance of the guidance policy.

The use of "active" guidance seems to help improve the performance of RRL even more in the case of deterministic applications. The specific examples provided by this active guidance helps the Q-function to cover a larger area of the state-space by allowing the learning algorithm to zoom in on unknown areas of the state-space. However, it seems hard to expand this approach to stochastic applications while maintaining the simplicity and elegance of the original idea.

Possible directions for further work include tighter integration of the use of guidance and the generalization engine used (RRL-TG), on one hand, and the investigation of the effects of using guidance with other types of learn-

ing algorithms (e.g., instance-based) within Q-learning, on the other hand.

References

Bertsekas and Tsitsiklis. 1996. *Neuro-Dynamic Programming.* Athena Scientific.

H. Blockeel, L. De Raedt, and J. Ramon. 1998. Top-down induction of clustering trees. pages 55–63.

D. Chapman and L. P. Kaelbling. 1991. Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. pages 726–731.

D. Cohn, L. Atlas, and R. Ladner. 1994. Improving generalization with active learning. 15:201–221.

K. Driessens and S. Džeroski. 2002. Integrating experimentation and guidance in relational reinforcement learning. In C. Sammut and A. Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 115–122. Morgan Kaufmann Publishers, Inc.

K. Driessens, J. Ramon, and H Blockeel. 2001. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *Proceedings of the 13th European Conference on Machine Learning*, pages 97–108. Springer-Verlag.

S. Džeroski, L. De Raedt, and H Blockeel. 1998. Relational reinforcement learning. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 136–143. Morgan Kaufmann.

L. Kaelbling, M. Littman, and A. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Stefan Kramer. 1996. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 812–819, Cambridge/Menlo Park. AAAI Press/MIT Press.

W. D. Smart and L. P. Kaelbling. 2000. Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th International Conference on Machine Learning*, pages 903–910. Morgan Kaufmann.

Christopher Watkins. 1989. *Learning from Delayed Rewards.* Ph.D. thesis, King's College, Cambridge.