# Integrating Experimentation and Guidance in Relational Reinforcement Learning

**Kurt Driessens** KURT.DRIESSENS@CS.KULEUVEN.AC.BE
Department of Computer Science, K.U.Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium

**Sašo Džeroski** SASO.DZEROSKI@IJS.SI
Department of Intelligent Systems, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

## 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 generalisation 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. The problem of sparse rewards has not been addressed for RRL. This paper presents a solution based on the use of "reasonable policies" to provide guidance. Experimental results in several domains show the merit of the approach and indicate pitfalls to be avoided.

## 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 greatest 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 characterised 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 generalises 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 & Tsitsiklis, 1996), nearest neighbour methods (Smart & Kaelbling, 2000) and regression trees (Chapman & Kaelbling, 1991).

A relational learner is employed by (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) 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 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 (Digger, Tetris).

In structural domains, the state space is typically very large, and while a relational learner can provide the right level of abstraction to learn in such a domain, there remains the problem that rewards may be distributed very sparsely in this state space. Using random exploration through the search space, rewards may simply never be encountered. In some of the application domains mentioned above this prohibits RRL from finding a good solution.

While plenty of exploration strategies exist (Wiering, 1999), few deal with the problems of exploration at the start of the learning process. It is exactly this problem that we are faced with in our RRL setting. There is, however, an approach which has been followed with success, and which consists of guiding the Q-learner with examples of "reasonable" strategies, provided by a teacher (Smart & Kaelbling, 2000). Thus a mix

between the classical unsupervised Q-learning and (supervised) behavioural cloning is obtained. It is the suitability of this approach in the context of RRL that we explore in this paper.

In Section 2, we discuss how guidance can be incorporated in a Q-learning approach, and how this was done in our RRL algorithm. In Section 3 we present a number of structural domains and their specific difficulties. In Section 4 we present experimental results on these domains. We discuss related work in Section 5 and conclude in Section 6.

## 2. Guidance and Exploration

This section describes how our approach integrates guidance and exploration within RRL. The idea is to provide guidance to begin with, then continue with experimentation as is usual in reinforcement learning. Figure 1 shows a more formal description of the resulting algorithm, called GRRL (guided RRL).

### 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 Using Reasonable Policies

While 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 non-optimal policies. These non-optimal policies are often easy (or easier) to implement or obtain than optimal ones. One 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

---

Initialise $\hat{Q}_0$ to assign 0 to all $(state, action)$ pairs
**for** $(i = 0; i < k; i++)$ {
  transform $trace_i$ into $(state, action, qvalue)$ triplets
  process generated triplets with TG algorithm
    transforming $\hat{Q}_i$ into $\hat{Q}_i + 1$
  }
run normal RRL-TG starting with $\hat{Q}_k$ as the
  initial Q-function hypothesis

---

*Figure 1.* The GRRL-algorithm: This is the RRL-TG algorithm with integrated guidance (k example traces).

$(state, action, q\text{-}value)$ triplets and then presented to the RRL algorithm. The TG algorithm can use these triplets to build a partial Q-function.

Since Q-learning is exploration insensitive — i.e., the Q-values will converge to the optimal values, independent of the exploration strategy used (Kaelbling et al., 1996) — the non-optimality of the used policy will have no negative effect on the convergence of the Q-function, while still helping the learning system to reach non-obvious rewards.

### 2.3 Further Exploration

To expand on the Q-function learned through the use of the example-traces, ordinary episodes are used to explore the state-space further. The partial Q-tree, built from the traces, will give some indication to the agent about which actions to prefer over others. It will not represent the correct Q-function, nor will it cover the entire state-action space, but it might be correct enough to guide RRL towards more rewards with the use of statistically founded exploration.

The RRL algorithm explores the state space autonomously. Boltzmann exploration (Kaelbling et al., 1996) based on the values predicted by the partial Q-tree is used: this makes a compromise between exploration and exploitation of the partial Q-tree.

## 3. The Domains

We illustrate the proposed approach on three example applications: the Blocks World and two computer games, Digger and Tetris.

### 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 2 (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. 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.

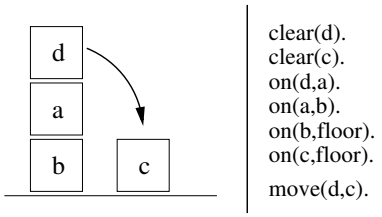


*Figure 2.* Example state and action in the blocks-world.

We study three different goals in the blocks world: stacking all blocks, unstacking all blocks (i.e., putting all blocks on the floor) and putting a specific block on top of another specific block. Note however that RRL — through the use of variables in its Q-function representation — learns the last goal without relying on specific identities of blocks (Džeroski et al., 2001). In a blocks world with 10 blocks, there are 3.5 million states which satisfy the stacking goal, 1.5 million states that satisfy a specific on(A,B) goal and one state only that satisfies the unstacking 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 isn't.

### 3.2 The Digger Game

In the Digger[1] game, the player controls a digging machine or "Digger" in an environment that contains emeralds, bags of gold, two kinds of monsters (nobbins and hobbins) and tunnels. The object of the game is to collect as many emeralds and gold as possible while avoiding or shooting monsters. In our tests we removed the hobbins and the bags of gold from the game. Hobbins are more dangerous than nobbins for human players because they can dig their own tunnels and reach Digger faster. However, they are less interesting for learning purposes, because they reduce the implicit penalty for digging new tunnels (and thereby increasing the mobility of the monsters). The bags of gold we removed to reduce the complexity of the game.

We use a lossless representation of the state of the Digger game and present RRL with predicates such as: *emerald*/2, *nearestEmerald*/2, *distanceTo*/2, *monster*/2, *visibleMonster*/2, *getDirection*/2 and *lineOfFire*/1 to use in the construction of the Q-tree.

[1] Digger was created in 1983, by Windmill Software.

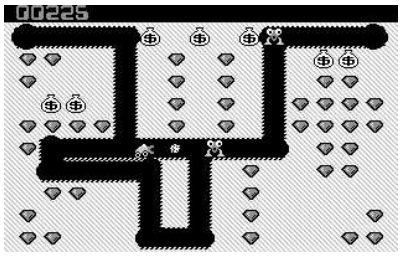

*Figure 3.* The DIGGER Game.

The actions available to RRL are $moveOne(X)$ and $shoot(X)$ with $X \in \{up, down, left, right\}$. These actions are implemented in such a way that Digger moves an entire row or column. (As a human player, you have the option to return half way.) The rewards in the Digger-game are distributed as follows: 25 points for eating an emerald (and an extra 250 points for eating 8 emeralds in a row), 250 points for shooting a monster and -200 points for dying. RRL is trained and tested on all 8 standard Digger-levels.

### 3.3 The Tetris Game

Tetris[2] 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.

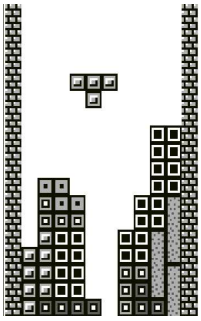

*Figure 4.* The TETRIS Game.

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

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

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

### 3.4 Comments

All these 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 all of them can be expected to benefit from the added guidance. The unstacking goal in the blocks world would almost never be reached by random exploration. Not only is there a single goal state out of 59 million states, but the number of possible action in a state increases as we get closer to the goal state: in a state from which a single action leads to the goal state, there are 73 actions possible. Since the unstacking goal is an extreme case, we expect both RRL and GRRL to generate unusual and interesting behaviour.

In the Digger Game, some rewards are easily reached. Often the playing field is filled with emeralds, so it is easy for the agent to locate the rewards these emeralds supply. However, discovering the reward for eating 8 emerald in a row, or given the limitations on the rate of fire of Digger, discovering the reward for shooting a monster is a lot harder to do with random exploration.

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 both the Digger and the Tetris games, optimal policies are very hard — and may be impossible — to construct.

## 4. The Experiments

The guidance added to exploration should have two-fold effect on learning performance. In terms of learning speed, we expect the guidance to help the Q-learner to discover higher 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 generalisation 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 — in the available time.

To test these effects, we compare RRL with GRRL in the following setup: First we run RRL in it's natural form, giving 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. To compare with GRRL, we substitute a number of episodes at the beginning of learning with generated traces, and then follow the same course. These traces are generated by either a hand-coded policy, a previously learned policy or a human controller. After these traces, GRRL is allowed to explore the state-space further on its own. Note that on the performance graphs, the traces presented to GRRL will count as episodes.

### 4.1 The Blocks World

For the blocks world it is easy to write optimal policies for the three goals we look at. Thus it is easy to supply RRL with a large amount of optimal example traces.

#### 4.1.1 Stacking

Reaching a state that satisfies the Stacking goal is not really all that hard, even with 10 blocks and random exploration: approximately one of every 17 states is a goal state. Even so, some improvement — a small decrease of the number of episodes needed to obtain a certain performance level — can already be seen in Figure 5 for small amounts of help (5 or 20 traces). The graph shows the average reward obtained by the current policy over 2000 randomly generated test-problems. However, the most interesting feature of the performance graph is the performance of the experiment where we supplied GRRL with 100 optimal traces in the beginning of the learning experiment. Not only does this experiment take longer to converge to the optimal policy, but during the first 100 episodes, there is no improvement at all.

This behaviour becomes worse when we supply GRRL with even more optimal traces. In Figure 6, we show the learning curve when we supply GRRL with 500 optimal traces. The reason for GRRL's failing to learn anything during the first part of the experiment (i.e., when being supplied with optimal traces) can be found in the generalisation engine. Trying to connect the correct Q-values with the corresponding state-action pairs, the generalisation engine tries to discover significant differences between state-action pairs with differing Q-values. In the ideal case, the TG -engine is able to distinguish between states that are at different distances from a reward producing state, and between optimal and non-optimal actions in these states.

However, when we supply GRRL with only optimal traces, overgeneralisation occurs. The generalisation

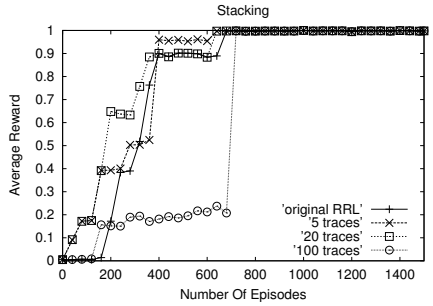


*Figure 5.* The Learning Curves of RRL and GRRL for the Stacking Goal.

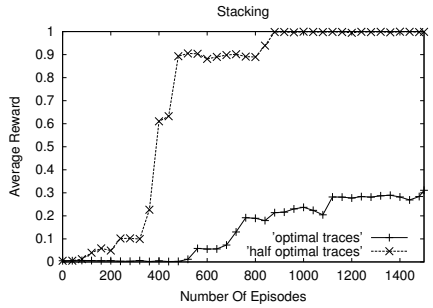


*Figure 6.* GRRL Learning from Optimal and Non-Optimal Traces for the Stacking Goal.

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 generalise 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.

To illustrate this behaviour, we supplied GRRL with 500 half-optimal traces in which the used policy alternates between a random and an optimal action. Figure 6 shows that, in this case, GRRL does learn during the guided traces. This experiment (although artificial) shows that GRRL can be useful even in domains where it is easy to hand-code a reasonable policy. GRRL will use the experience created by that policy to construct a better (possibly optimal) one.

The sudden leaps in performance are characteristic for RRL-TG : whenever a new (well chosen) test is added to the Q-tree, the performance jumps to a higher level.

#### 4.1.2 UNSTACKING

As already stated, in a world with 10 blocks, it is almost impossible to reach a state satisfying the Unstacking-goal at random. This is illustrated by the

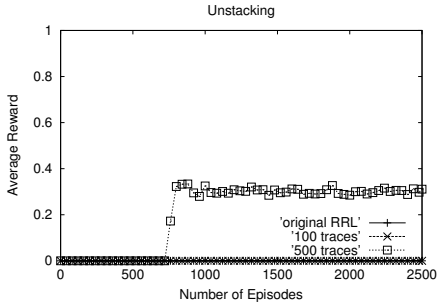


*Figure 7.* Learning Curves of RRL and GRRL for the Unstacking Goal.

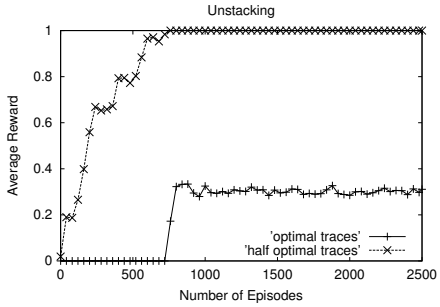


*Figure 8.* GRRL Learning from Optimal and Non-Optimal Traces for the Unstacking Goal.

graph in Figure 7. It shows the average reward obtained over 2000 randomly generated test-problems. RRL never learns anything useful, because it never reaches a single reward. Even if we supply GRRL with 100 optimal traces, nothing useful is learned. When we supply GRRL with 500 optimal traces, it has collected enough examples to learn something useful, but the difficulties with exploring such a huge state-space with so little reward still shows in the fact that GRRL is able to do very little extra with this information.

Things change when we supply GRRL with half optimal traces (where each optimal action is followed by a random one). Even though it is not trivial to reach the goal state when using a half optimal policy, it happens often enough for GRRL to learn a correct policy. Figure 8 shows that GRRL is able to learn quite a lot during the 500 supplied traces and then is able to reach the optimal policy after some extra exploration.

#### 4.1.3 ON(A,B)

The on(A,B) goal has always been a hard problem for RRL (Džeroski et al., 2001; Driessens et al., 2001). Figure 9 shows the learning curves for RRL when we supply it with 0, 5, 20 and 100 optimal traces. Every data point is the average reward over 10 000 randomly generated test cases. Although the optimal policy is never reached, the graph clearly shows the improvement that is generated by supplying RRL with varying amounts of guidance.

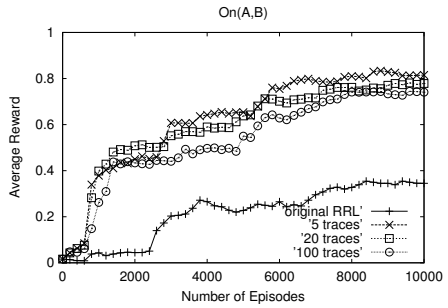


*Figure 9.* Learning Curves of RRL and GRRL for the on(a,b) Goal.

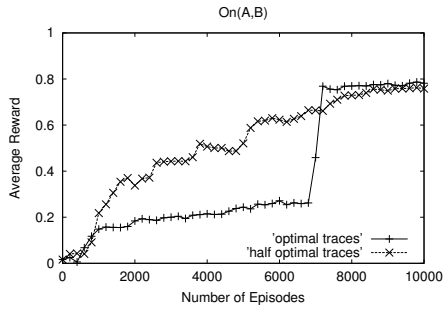


*Figure 10.* GRRL Learning from Optimal and Non-Optimal Traces for the on(a,b) Goal.

Figure 10 shows the learning behaviour of GRRL supplied with 500 optimal or 500 half-optimal traces for the On(A,B) goal. The sudden leap in performance for the optimal case can be explained by the fact that in GRRL, TG first builds a tree dividing the state-space according to the distance to the goal, but after enough further exploration adds tests to separate optimal from non-optimal actions as well.

### 4.2 The Digger Game

Because it is hard to write a policy for the Digger Game, we used a policy generated in earlier work (Driessens & Blockeel, 2001) by RRL. This strategy could be used in a more general case when RRL becomes stuck at a certain level of performance. The learning experiment could then be restarted using the previously generated policy as a starting point, giving the TG algorithm the opportunity to construct a possibly smaller and more accurate Q-tree.

Figure 11 shows the average reward obtained by the learned strategies over 640 digger test-games divided over the 8 different Digger levels. It shows that GRRL is indeed able to improve on the policy learned by RRL. Although the speed of convergence isn't improved, GRRL reaches a significantly higher level of overall performance. The graph also shows that more examples traces result in a higher performance. This is probably due to the fact that the policy used to generate these traces is non-optimal.

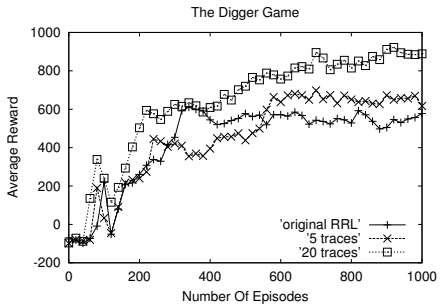


*Figure 11.* Learning Curves for RRL and GRRL for the Digger Game.

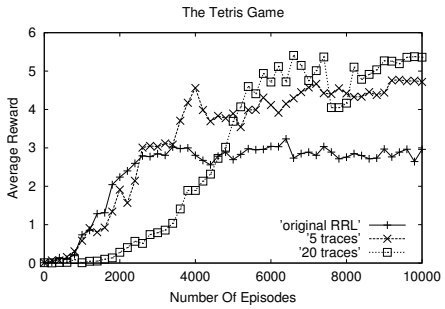


*Figure 12.* Learning Curves for RRL and GRRL for the Tetris Game.

### 4.3 The Tetris Game

Like with the Digger Game, it is very hard (if not impossible) to generate an optimal or even "reasonable" strategy for the Tetris game. This time, we opted to supply GRRL with traces of non-optimal playing behaviour from a human player.

The overall results for learning Tetris with RRL and GRRL are below our expectations. We believe that this is due to the fact that the future reward in Tetris is very hard to predict, especially by a regression technique that needs to discretize these rewards like the TG -algorithm. Although Tetris is a relational domain, we believe that RRL in its present form is not yet suitable to tackle the problem.

However, the added guidance in the beginning of the learning experiment still has its effects on the overall performance. Figure 12 shows the learning curves for RRL and GRRL supplied with 5 or 20 manually generated traces. The data points are the average number of deleted lines per game, calculated over 500 played test games. As with the Digger case, since we are not starting from an optimal policy, supplying GRRL with more traces does not seem to harm performance as it did in the blocks world.

## 5. Related Work

The idea of incorporating guidance in automated learning of control is not new. In (Chambers & Michie, 1969) three kinds of cooperative learning are discussed.

In the first, the learning system just accepts the offered advice. In thes second, the expert has the option of not offering any advice. In the third, some criterion decides whether the learner has enough experience to override the human decision. Roughly speaking, the first corresponds to behavioural cloning, the second to reinforcement learning and the third to guided reinforcement learning.

The link between this work and behavioural cloning (Bain & Sammut, 1995; Urbancic et al., 1996) is not very hard to make. If we would supply the TG generalisation algorithm with low q-values for unseen state-action pairs, RRL would learn to imitate the behaviour of the supplied traces. Because of this similarity of the techniques, it is not surprising that we run into similar problems as one encounters in behavioural cloning.

(Scheffer et al., 1997) discusses some of these problems. The differences between learning by experimentation and learning with "perfect guidance" (behavioural cloning) and the problems and benefits of both the approaches are highlighted. Behavioural cloning seems to have the advantage, as it sees precisely the optimal actions to take. However, this is all that it is given. Learning by experimentation, on the other hand, receives imperfect information about a wide range of state-action pairs. While some of the problems Scheffer mentions are solved by the combination of the two approaches as we suggest in this paper, other problems resurface in our experiments. Scheffer states that learning from guidance will experience difficulties when confronted with memory constraints so that it can not simply memorise the ideal sequence of actions but has to store associations instead. This is very closely related to the problems our generalisation engine has when it is supplied with only perfect state-action pairs.

Wang combines observation and practice in the OBSERVER learning system in (Wang, 1995) which learns STRIPS-like planning operators (Fikes & Nilsson, 1971). The system starts with learning from "perfect guidance" and improves on the planning operators (pre- and postconditions) through practice. There is no reinforcement learning involved.

The work of (Smart & Kaelbling, 2000) is closely related to ours in terms of combining guidance and experimentation. While we deal with relational problems, their work deals with continuous state-spaces. The authors use a nearest neighbour (NN) approach for generalisation and use example training runs to bootstrap the Q-function approximation. The use of nearest neighbour and convex hulls successfully prevents overgeneralization, whereas our approach which uses regression trees has overgeneralization problems. However, NN approaches for relational representations are not nearly as developed as for continuous spaces and would likely not perform in RRL in their current state of development.

Other approaches to speed up reinforcement learning by supplying it with non-optimal strategies include the work of (Shapiro et al., 2001). There the authors embed hierarchical reinforcement learning within an agent architecture. The agent is supplied with a "reasonable policy" and learns the best options for this policy through experience. This approach is complementary to and can be combined with our work on GRRL.

## 6. Conclusions and Future Work

In this paper, we have addressed the problem of integrating guidance and experimentation in reinforcement learning, and in particular relational reinforcement learning (RRL). The use of a more expressive representation in RRL allows for larger and more complex learning tasks to be addressed, where the problem of finding rewards that are sparsely distributed is more severe. We show that providing guidance at the beginning of the learning process (before experimentation starts) does help improve performance in such cases and that this approach can even be useful in domains where it is easy to hand-code a reasonable but non-optimal policy.

We demonstrate this through experiments in three domains, the blocks world and two computer games (Digger and Tetris). Three different forms of guidance are considered: traces (action sequences) generated by hand-coded policies, traces generated by policies learned by RRL and traces of a human performing the task at hand. In all three cases, the use of guidance followed by experimentation improves performance over using experimentation only, either in terms of the overall performance level achieved or the convergence speed.

One has to be careful about supplying the learning algorithm with too much "perfect guidance" right at the start. Coupled with the use of a generalisation engine, providing optimal actions only does not allow to learn to distinguish between optimal and non-optimal actions. Good guidance will show the learning algorithm both optimal and non-optimal actions in a great variety of states. Both restricting the visited states during guidance and limiting the guidance policy to take only correct actions will have a negative influence on the effectiveness of the offered guidance. The variety in the visited states is probably the hardest to achieve when constructing guidance traces.

This already points to some directions for further work. The above problem with over-generalisation could be alleviated by improving the generalisation engine of RRL to deal with positive only examples. An alternative is to find better ways to integrate guidance and experimentation. One could, for example, interleave the example traces with learning episodes (one trace, one episode, one trace, one episode, ...) or start with experimentation right away and supply (re-supply) RRL with new example traces whenever it gets stuck at a non-optimal performance level. Also, the fact that in huge state-spaces, it is likely that there is no overlap between part of the state-space visited within the traces and episodes calls for another direction of further work: active GRRL, where the learner is allowed to ask for traces starting from states encountered during experimentation or for episodes starting from states encountered during guidance.

Another route of investigation that could yield interesting results would be to have a closer look at the relations of our approach to the human learning process. In analogy to human learner–teacher interaction, one could have a teacher look at the behaviour of RRL or — given the declarative nature of the policies and Q-functions that are generated by RRL — at the policy that RRL has constructed itself and adjust the advice it wants to give. In the long run, because RRL uses an ILP approach to generalise its Q-function and policies, this advice doesn't have to be limited to traces, but could include feedback on which part of the constructed tree is useless and has to be rebuilt, or even constraints that the learned policy has to satisfy.

**Acknowledgements.** The authors would like to thank Tanja Urbančič and Dorian Suč for the interesting discussions and Hendrik Blockeel for his help on the text.

## References

Bain, M., & Sammut, C. (1995). A framework for behavioral cloning. In S. Muggleton, K. Furukawa and D. Michie (Eds.), *Machine intelligence 15*. Oxford University Press.

Bertsekas, D.P., & Tsitsiklis, J.N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Chambers, R. A., & Michie, D. (1969). Man-machine co-operation on a learning task. *Computer Graphics: Techniques and Applications*, 179–186.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. (pp. 726–731. ).

Driessens, K., & Blockeel, H. (2001). Learing digger using hierarchical reinforcement learning for concurrent goals. *Proc. 5th European Workshop on Reinforcement Learning* (pp. 11–12). Onderwijsinstituut CKI, University of Utrecht.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proc. 13th European Conference on Machine Learning* (pp. 97–108). Springer.

Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. *Proc. 15th International Conference on Machine Learning* (pp. 136–143). Morgan Kaufmann.

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

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application for theorem proving to problem solving. *Advance Papers of the 2nd International Joint Conference on Artificial Intelligence* (pp. 608–620). Edinburgh, Scotland.

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

Kramer, S. (1996). Structural regression trees. *Proc. 13th National Conference on Artificial Intelligence* (pp. 812–819). Cambridge/Menlo Park: AAAI Press/MIT Press.

Scheffer, T., Greiner, R., & Darken, C. (1997). Why experimentation can be better than "perfect guidance". *Proc. 14th International Conference on Machine Learning* (pp. 331–339). Morgan Kaufmann.

Shapiro, D., Langley, P., & Shachter, R. (2001). Using background knowledge to speed reinforcement learning in physical agents. *Proc. 5th International Conference on Autonomous Agents*. Association for Computing Machinery.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. *Proc. 17th International Conference on Machine Learning* (pp. 903–910). Morgan Kaufmann.

Urbancic, T., Bratko, I., & Sammut, C. (1996). Learning models of control skills: Phenomena, results and problems. *Proc. 13th Triennial World Congress of the International Federation of Automatic Control* (pp. 391–396). IFAC.

Wang, X. (1995). Learning by observation and practice: An incremental approach for planning operator acquisition. *Proc. 12th International Conference on Machine Learning* (pp. 549–557).

Watkins, C. (1989). *Learning from delayed rewards*. Doctoral dissertation, King's College, Cambridge.

Wiering, M. (1999). *Explorations in efficient reinforcement learning*. Doctoral dissertation, University of Amsterdam.