Machine Learning, 57, 271–304, 2004

Integrating Guidance into Relational Reinforcement Learning

KURT DRIESSENS kurt.driessens@cs.kuleuven.ac.be
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

SAŠO DŽEROSKI saso.dzeroski@ijs.si
Department of Intelligent Systems, Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia

Editor: Stan Matwin

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. 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. Different types of policies and different strategies to supply guidance through these policies are discussed and evaluated experimentally in several relational domains to show the merits of the approach.

Keywords: reinforcement learning, relational learning, guided exploration

1. Introduction

In reinforcement learning (for an excellent introduction see the book by Sutton and Barto, 1998), an agent tries to learn a policy, i.e., how to select an action in a given state of the environment, so that it maximizes the total amount of reward it receives when interacting with the environment.

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 then 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 (Rumelhart and McClelland, 1986), nearest neighbor methods (Smart and Kaelbling, 2000) and regression trees (Chapman and Kaelbling, 1991).

A relational learner is employed by Džeroski, De Raedt, and Blockeel (1998), hence the name "relational reinforcement learning" or RRL. This relational learner uses first order representations for states and actions, and uses a first order regression algorithm 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 that will be described in more detail further in this paper, include learning to solve simple planning tasks in a blocks world, or learning to play certain computer games (Tetris, Digger).

In structural domains, the state space is typically very large, and although a relational learner can provide the right level of abstraction to learn in such a domain, the problem remains 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 and Kaelbling, 2000). Thus a mix between the classical unsupervised Q-learning and (supervised) behavioral cloning is obtained. It is the suitability of this approach in the context of RRL that we explore in this paper.

2. Reinforcement learning and relational learning

2.1. Reinforcement learning

This section gives an overview of reinforcement learning ideas relevant to relational reinforcement learning. For an extensive treatise on reinforcement learning, we refer the reader to Sutton and Barto (1998). We first state the task of reinforcement learning, then briefly describe the Q-learning approach to reinforcement learning. In its basic variant, Q-learning is tabular: this is unsuitable for problems with large state spaces, where generalization over states and actions is needed.

2.1.1. Task definition. The typical reinforcement learning task using discounted rewards can be formulated as follows:

Given

- a set of possible states S.
- a set of possible actions A.
- an **unknown** transition function δ: $S \times A \rightarrow S$.
- an **unknown** real-valued reward function $r : S \times A \rightarrow \mathbb{R}$.

Find a policy $\pi^* : S \rightarrow A$ that maximizes

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

for all s_t where $0 \leq \gamma < 1$.

At each point in time, the reinforcement learning agent can be in one of the states s_t of S and selects an action $a_t = \pi(s_t) \in A$ to execute according to its policy π. Executing an action a_t in a state s_t will put the agent in a new state $s_{t+1} = \delta(s_t, a_t)$. The agent also receives a reward $r_t = r(s_t, a_t)$. The function $V^{\pi}(s)$ denotes the value (expected return; discounted cumulative reward) of state s under policy π. The factor γ, the discount factor, specifies the relative importance of future rewards compared to immediate rewards.

The agent does not necessarily know what effect its actions will have, i.e., what state it will end up in after executing an action. This means that the function δ is unknown to the agent. In fact, it may even be stochastic: executing the same action in the same state on different occasions may yield different successor states. We also assume that the agent does not know the reward function r. The task of learning is then to find an optimal policy, i.e., a policy that will maximize the discounted sum of the rewards. We will assume episodic learning, where a sequence of actions ends in a terminal state.

2.1.2. Tabular Q-learning. Here we summarize Q-learning, one of the most common approaches to reinforcement learning, which assigns values to state-action pairs and thus implicitly represents policies. The optimal policy π^* will always select the action that maximizes the sum of the immediate reward and the value of the immediate successor state, i.e.,

$$\pi^*(s) = \arg\max_a (r(s, a) + \gamma V^{\pi^*}(\delta(s, a)))$$

The Q-function for policy π is defined as follows:

$$Q^{\pi}(s, a) = r(s, a) + \gamma V^{\pi}(\delta(s, a))$$

Knowing Q^*, the Q-function for the optimal policy, allows us to rewrite the definition of π^* as follows

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

An approximation to the Q^*-function, Q, in the form of a look-up table, is learned by Algorithm 1.

Algorithm 1 The Q-learning algorithm.

for each $s \in S, a \in A$
  initialize table entry $Q(s, a)$
end for
repeat {for each episode}
  generate a starting state s_0
  $i \leftarrow 0$
  repeat {for each step i of episode}
    select an action a_i using the policy derived from Q
    take action a_i, observe r_i and s_{i+1}
    $Q(s_i, a_i) \leftarrow r_i + \gamma \max_{a \in A} Q(s_{i+1}, a)$
    $i \leftarrow i + 1$
  until s_i is terminal
until no more episodes

The agent learns through continuous interaction with the environment, during which it exploits what it has learned so far, but it also explores. In practice, this means that the current approximation Q is used to select an action most of the time. However, in a small fraction of cases an action is selected randomly from the available choices, so that unseen (*state*, *action*) pairs can be explored.

For smoother learning and to be able to deal with stochastic environments, an update of the form

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big]$$

would be used, where $\alpha = 1/\text{numberOfVisits}(s, a)$ causes the Q-values to gradually shift to their expected values. This is a special case of temporal-difference learning, to which algorithms such as SARSA (Sutton, 1996) also belong. Instead of considering all possible actions a in state s_{t+1} and taking the maximum $Q(s_{t+1}, a)$, SARSA only considers the action a_{t+1} actually chosen in state s' during the current episode. The update rule is thus $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ where $\pi(s_{t+1}) = a_{t+1}$. For Algorithm 1, the learned action-value function Q directly approximates Q^*, regardless of the policy being followed.

2.2. *Relational learning*

Relational learning is concerned with learning from examples and background knowledge represented in a relational formalism, such as relational or first-order logic (or subsets thereof, such as in logic programming). Needless to say, all inductive hypotheses generated

are also represented in a relational formalism. Examples are typically specific facts or sets thereof, whereas background knowledge consists of specific facts or general rules defining concepts relevant to the examples.

Many techniques for relational learning come from the field of inductive logic programming (Muggleton and De Raedt, 1994; Lavrač and Džeroski, 1994). Situated at the intersection of machine learning and logic programming, ILP has been concerned with finding patterns expressed as logic programs. Initially, ILP focussed on automated program synthesis from examples, formulated as a binary classification task. In recent years, however, the scope of ILP has broadened to cover the whole spectrum of machine learning tasks, ranging from classification and regression, to clustering and reinforcement learning.

Many common approaches and algorithms in machine learning have now been adapted to work in relational settings. We thus have algorithms for first order logical decision tree induction (Blockeel and De Raedt, 1998), relational distance-based clustering and prediction (Kirsten, Wrobel, and Horvath, 2001), and relational reinforcement learning (Džeroski, De Raedt, and Blockeel, 1998; Džeroski, De Raedt, and Driessens, 2001), to name a few. There is even a generic recipe for upgrading propositional learning approaches to relational settings (Van Laer and De Raedt, 2001), which involves upgrading the key notions of the propositional learner. For example, to obtain a relational distance-based approach, we need to upgrade a propositional distance measure to a relational one.

2.2.1. Examples and background knowledge. Examples in relational learning most commonly consist of specific (or ground) facts in the setting of learning from entailment; De Raedt (1997) or sets of facts or interpretations in the setting of learning from interpretations (De Raedt and Džeroski, 1994). For example, in the blocks world domain, which is often used in planning, states that satisfy a certain property (e.g., have all blocks stacked in one stack) may be positive examples, and the classification task would be to recognize states where this property holds. States in the blocks world are naturally represented relationally, with the relation $on(A, B)$ specifying that block A is on block B. An example state with three blocks stacked on each other is represented by the set of facts $\{clear(a), on(a, b), on(b, c), on(c, floor)\}$.

In the blocks world the learning system can take actions of the type $move(A, B)$, moving block A onto B, which can be another block or the floor. Examples might also be sets of facts comprising a state description and an action taken in that state, and the learning task might be to predict the number of steps remaining to reach the state where all blocks are on the floor. If one is to use relational learning for generalization in Q-learning, the regression task would be to predict the Q-value of such state/action examples.

The use of background knowledge is a typical feature for relational learning. Background knowledge consists of specific facts or general rules defining concepts relevant to the examples. For the blocks world, the predicate $above(A, B)$ defines when block A is above block B in terms of the predicate $on(A, B)$. This knowledge holds over all states in the blocks world. Similarly, the predicate $numberOfBlocksOn(A, N)$ specifies that there are exactly N blocks above block A. Predicates in the background knowledge can be used in the learned models for predicting the target property (be it continuous or discrete).

```
on(A,B) ?
+--yes : on(B,C) ?
         +--yes : stacked
         +--no : unstacked
+--no : unstacked
```

Figure 1. A logical classification tree in the blocks world.

2.2.2. Logical decision trees. Much like ordinary decision trees, logical decision trees consist of internal nodes and leaves. The leaves contain predictions for the target property, either discrete values in the case of classification or real numbers (in the case of regression). The difference is in the internal nodes, where tests in the propositional case are of the form Attribute = value and in the relational case conjunctions of literals (predicates). Example tests that we might find in a logical tree in the blocks world include $on(A, B)$; $on(A, floor)$; and $numberofblockson(A, N)$, $N > 3$.

An example decision tree, predicting whether the blocks in a 3-blocks world are in one stack is given in figure 1. Note that variables may be shared between internal nodes, i.e., variables introduced in one test can be referred to in another. Variables can only be referred to in the 'yes' branch of the internal node that introduced them.

A crucial difference with propositional decision trees is that the tests considered in each node can be quite different. They depend on which variables have been introduced in nodes along the path from the root to the current node. The possible tests are obtained by applying predicates from the background knowledge to the 'referrable' variables, taking into account declarative bias information (e.g. argument types and input/output modes).

Several learning systems exist now for building relational decision trees (sometimes called structural, sometimes logical). State-of the art representatives are S-CART (Kramer, 1996) and TILDE (Blockeel and De Raedt, 1998). S-CART upgrades the CART (Breiman et al., 1984) approach to constructing classification and regression trees, while TILDE upgrades C4.5 (Quinlan, 1993).

2.2.3. Relational instance-based learning. Instance-based prediction methods, such as the nearest neighbor method or instance-based learning (Aha, Kibler, and Albert, 1991), are popular because of their simplicity, good performance and robustness. They are also very modular as far as the representation of the examples is concerned. One doesn't really need to change the nearest neighbor algorithm to be able to deal with relational examples: all one needs is a distance on relational examples. Once we have the distance, we can use the nearest neighbor approach essentially unchanged, as well as many distance-based clustering approaches.

Designing a distance on relational examples, however, is a nontrivial matter. Most such distances are closely related to distances on structured objects and sets. Proposals for relational distances include those by Emde and Wettschereck (1996) and Ramon and Bruynooghe (2001).

The RIBL distance measure (Emde and Wettschereck, 1996) calculates the distance between objects by first calculating distances along their elementary properties, then finding related objects and taking into account distances between them (a depth bound prevents infinite recursion). Elementary properties include discrete and real-valued arguments, as well as list and term valued ones. The two objects can each have many related objects of different types: the related object are grouped per type, then distances between the corresponding pairs of groups (sets) are calculated.

3. Relational reinforcement learning

Relational reinforcement learning or RRL (Džeroski, De Raedt, and Blockeel, 1998) is a learning technique that combines Q-learning with relational representations for the encountered states, actions and the resulting Q-function.

Algorithm 2 The Relational Reinforcement Learning Algorithm

initialize the Q-function hypothesis $\hat{Q}_0$
$e \leftarrow 0$
repeat {for each episode e}
  Examples $\leftarrow \emptyset$
  generate a starting state s_0
  $i \leftarrow 0$
  repeat {for each step i of episode e}
    choose a_i for s_i using a policy derived from the current hypothesis $\hat{Q}_e$
    take action a_i, observe r_i and s_{i+1}
    $i \leftarrow i + 1$
  until s_i is terminal
  for $j = i - 1$ to 0 **do**
    generate example $x = (s_j, a_j, \hat{q}_j)$ where $\hat{q}_j \leftarrow r_j + \gamma \max_{a \in A} \hat{Q}_e(s_{j+1}, a)$
    Examples $\leftarrow$ *Examples* $\cup \{x\}$
  end for
  Update $\hat{Q}_e$ using *Examples* and an incremental relational regression algorithm to produce $\hat{Q}_{e+1}$
  $e \leftarrow e + 1$
until no more episodes

The RRL-system learns through exploration of the state-space in a way that is similar to normal Q-learning algorithms. It starts with running a normal episode but uses the encountered states, chosen actions and the received awards to generate a set of examples that can then be used to build a Q-function generalization. RRL differs from other generalizing Q-learning techniques because it uses a relational representation for the encountered states

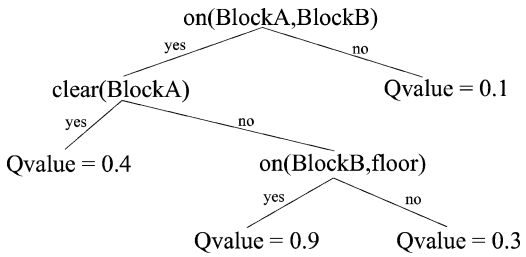

Figure 2. A first-order regression tree predicting Q-values.

and the chosen actions. See figure 3 in Section 4.1 for an example of this notation in the blocks world.

To build the generalized Q-function, RRL applies an incremental first-order logic regression algorithm to the constructed example set. The resulting Q-function is then used to generate the policy for the next episodes and updated by the new experiences that result from these episodes. A more detailed description of the approach is given in Algorithm 2. Currently, two regression-algorithms for RRL have been developed.

*3.1. The RRL-*TG*-algorithm*

The TG-algorithm (Driessens, Ramon, and Blockeel, 2001) is an incremental first-order regression tree building algorithm that is based on the G-tree algorithm of Chapman and Kaelbling (Chapman and Kaelbling, 1991).

Figure 2 gives an example of a first-order regression tree. The test in an internal node should be read as the existentially quantified conjunction of all literals in the nodes in the path from the root of the tree to that node.

On a high level (see Algorithm 3), the TG-algorithm (as well as the original G-tree algorithm) stores the current regression tree, and for each leaf node statistics for all tests that could be used to split that leaf further. Each time an example is inserted, it is sorted down the decision tree according to the tests in the internal nodes, and in the leaf the statistics of the tests are updated.

Algorithm 3 The TG-algorithm

```
create a leaf with empty statistics
for each data point that becomes available do
   classify data point down to a leaf
   update statistics in this leaf and discard data point
   if split needed in updated leaf then
      use best test to construct node
      grow two new leafs with empty statistics
   end if
end for
```

In contrast to the G-tree algorithm, TG uses a relational representation language for describing the examples (i.e., the (*state*, *action*) pairs) and the tests that can be used in the regression tree.

While this use of relational representations has the advantage that the regression tree can use objects, the properties of objects and the relation between them when describing the Q-function, it complicates the tree building itself. For example, keeping track of the candidate-tests (the refinements of a query) is a non-trivial task. In the propositional case the set of candidate queries consists of the set of all features minus the features that are already tested higher in the tree. In the first-order case, the set of candidate queries consists of all possible ways to extend a query. The longer a query is and the more variables it contains, the larger is the number of possible ways to bind the variables and the larger is the set of candidate tests. One consequence is that it is not feasible to store all the statistics necessary for tree restructuring as done by Utgoff, Berkman, and Clouse (1997). This means that choices made by TG in the beginning of learning can not be unmade when they turn out to be wrong.

3.2. *The RRL-*RIB*-algorithm*

RRL-RIB (Driessens and Ramon, 2003) uses a relational instance-based regression algorithm. It uses c-nearest-neighbor prediction as a regression technique, i.e. the predicted Q-value $\hat{q}_i$ is calculated as follows:

$$\hat{q}_i = \frac{\sum_j \frac{q_j}{d_{ij}}}{\sum_j \frac{1}{d_{ij}}} \tag{1}$$

with d_{ij} the distance between example i and example j. To prevent division by 0, a small amount δ can be added to this distance.

To deal with the relational setting, RRL-RIB must be supplied with a relational distance defined on the chosen representation of (*state*, *action*) pairs. We refer to the work of Ramon (2002) and Ramon and Bruynooghe (2001) for more information on first-order distances.

The relational setting we work in imposes some constraints on the available instance based techniques. First of all, the time needed for the calculation of a true first-order distance between examples is not neglectable. This, together with the larger memory requirements of data-log, which is used to represent stored examples, compared to less expressive data formats, force RRL-RIB to limit the number of examples that are stored in memory.

Therefor, RRL-RIB uses several techniques to limit the number of examples stored in memory. It limits the inflow of new examples by only storing examples which introduce new knowledge in the database. These are examples that are predicted with a too large error or reside in an area of the state-action space of which we have few other examples.

When these inflow-filters are not sufficient RRL-RIB needs to decide which examples to throw out of the data-base. Therefor it can compute a score for each stored example that indicates the usefulness of that example. Two scoring methods are implemented in RRL-RIB. The "error-contribution"-score calculates the total prediction error with and without the example to be scored. The example who's removal causes the least total error is removed.

The "error-proximity"-score is based on the assumption that examples which are close to prediction errors are also causing this error. When using this score to select examples, the example closest to large prediction errors is removed.

Algorithm 4 The RIB data selection algorithm

```
for each data point that becomes available do
  try to predict the Q-value of the new data point
  if prediction error too large then
    store the new example in the data base
    remove the stored examples i for which there exists
      an example j in the data base such that:
      q_i < q_j − M · d_ij
  end if
end for
```

Another method of keeping the number of stored examples low is based on the "maximum deviation per distance unit" of the Q-function. If the user of the RRL-system can specify a maximum value M, such that:

$$\frac{|q_i - q_j|}{d_{ij}} < M \tag{2}$$

for all examples i and j, then this value can be used to eliminate examples from the database. A more detailed description of how and when this technique can be used is found in Driessens and Ramon (2003). It is this elimination technique that is used in the RRL-RIB experiments in this paper. When a new data point is presented to the RIB regression engine, it is treated according to Algorithm 4. In the experiments, the value of M is set to 0.1, the largest difference between to distinct Q-values.

4. Some relational worlds

In this section, we introduce three domains where using a relational representation of states is natural. Each of the domains involves objects and relations between them. The number of possible states in all three domains is very large. The three domains are: the blocks world, the Digger computer game, and the Tetris computer game.

4.1. The blocks world

In the blocks world, blocks can be on the floor or can be stacked on each other. Each state can be described by a set (list) of facts, e.g., $s = \{clear(c), clear(d), on(d, a), on(a, b), on(b, floor), on(c, floor)\}$ represents the state in figure 3. The available actions are then $move(X, Y)$ where $X \neq Y$, X is a block and Y is a block or the floor. The number

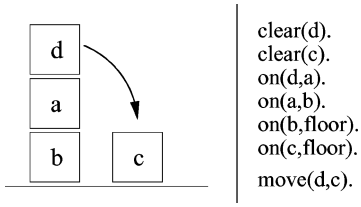

Figure 3. Example state and action in the blocks-world.

of states in the blocks world grows rapidly with the number of blocks. With 10 blocks, there are close to 59 million possible states.

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. 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 (where A and B are bound to specific blocks) and one state only that satisfies the unstacking goal. A reward of 1 is given in case a goal state is reached in the minimal number of steps; the episode ends with a reward of 0 if it is not.

In addition to the state and action information, the RRL-TG algorithm was supplied 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).

For the instance-based RRL-RIB we used the following distance definition for two (*state*, *action*) pairs in the blocks world:

1. Try to rename the blocks so that block-names that appear in the action (and possibly in the goal) match between the two (*state–action*) pairs. If this is not possible, add a penalty to your distance for each mismatch. Rename each block that does not appear in the goal or the action to the same name.
2. To calculate the distance between the two states, regard each state (with renamed blocks) as a set of stacks and calculate the distance between these two sets using the matching-distance between sets (Ramon and Bruynooghe, 2001) based on the distance between the stacks of blocks.
3. To compute the distance between two stacks of blocks, transform each stack into a string by reading the names of the blocks from the top of the stack to the bottom, and compute the edit distance (Wagner and Fischer, 1974) between the resulting strings.

For the M-value of Eq. (2), we choose the maximum difference between two different Q-values in this setting, i.e. 0.9.

4.2. The Tetris game

Tetris[1] is a widespread puzzle-video game played on a two-dimensional grid (see figure 4). Differently shaped blocks fall from the top of the game field and fill up the grid. The object

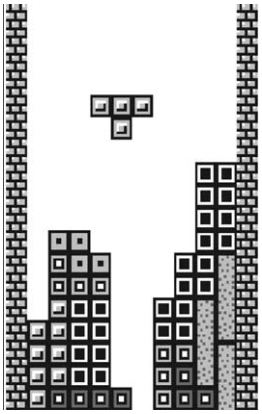

Figure 4. A snapshot of the TETRIS game.

of the game is to score points while keeping 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, find the optimal orientation and location of the dropping 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 (relational) reinforcement learning.) We represent the full state of the Tetris Game, the type of the next dropping block included in a term with 3 arguments:

1. Information about the game field ordered by column;
2. Information about the shape, orientation, row- and column-position of the falling block;
3. Information about the shape of the next block

This state representation was not presented to RRL-TG directly. Instead, RRL-TG was allowed to use the following predicates:

– *blockwidth/2*, *blockheight/2*: which specify the width and height of the falling block respectively
– *topBlock/2*: which returns the height of the wall at a given column
– *holeCovered/1*: whether there is a hole in the wall at a given column
– *holeDepth/2*: which returns the depth of the topmost hole in the wall at a given column
– *fits/2*: whether the falling block fits at a given location with a given orientation
– *increasesHeight/2*: whether dropping the falling block at a given location with a given orientation increases the overall height of the wall
– *fillsRow/2* and *fillsDouble/2*: whether the falling block completes a line (or two lines) at a given location with a given orientation.

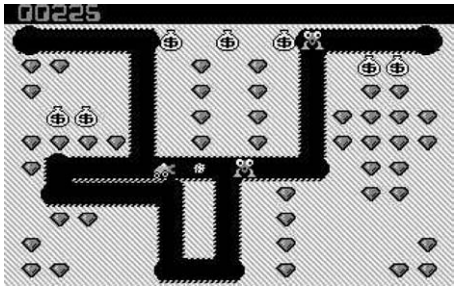

Figure 5. A snapshot of the DIGGER game.

On top of these, RRL was given a number of selectors to select individual rows, columns and different blockshapes.

4.3. The Digger game

Digger[2] is a computer game created in 1983, by Windmill Software. It is one of the few old computer games which still hold a fair amount of popularity. In this 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 (see figure 5). The object of the game is to collect as many emeralds and as much 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, as well as increase the mobility of the nobbins. 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) when trying to reach certain rewards. We removed the bags of gold from the game to reduce the complexity.

A state representation consists of the following components:

– the coordinates of digger, e.g., *digPos(6,9)*
– information on digger itself, supplied in the format:
`digInf(digger_dead,time_to_reload,level_done,pts_scored,steps_taken)`,
e.g., *digInf(false, 63, false, 0, 17)*,
– information on tunnels as seen by digger (range of view in each direction, e.g., *tunnel(4,0,2,0)*; information on the tunnel is relative to the digger; there is only one digger, so there is no need for a digger index argument)
– list of emeralds (e.g., [*em(14,9), em(14,8), em(14,5), . . .*]),
– list of monsters (e.g., [*mon(10,1,down), mon(10,9,down) . . .*]), and
– information on the fireball fired by digger (x-coordinate, y-coordinate, travelling direction, e.g., *fb(7,9,right)*).

The actions are of the form *moveOne*(X) and *shoot*(Y), where X and Y are in '*up,down, left,right*'.

In addition to the state and action representation described above, predicates such as *emerald/2, nearestEmerald/2, monster/2, visibleMonster/2, distanceTo/2, getDirection/2, lineOfFire/1*, etc., were provided as background knowledge.

5. Adding guidance to reinforcement learning

5.1. The need for guidance

In the early stages of learning, the exploration strategy used in Q-learning is pretty much random and causes the learning system to perform poorly. Only when enough information about the environment is discovered, i.e. when sufficient knowledge about the reward function is gathered, can better exploration strategies be used. Gathering knowledge about the reward function can be hard when rewards are sparse and especially if these rewards are hard to reach using a random strategy. A lot of time is usually spent exploring regions of the state-action space without learning anything because no rewards (or only similar rewards) are encountered.

Relational applications suffer from this problem often because they deal with very large state-spaces when compared to attribute-value problems. First, the size of the state-space grows exponentially with regard to the number of objects in the world, the number of properties of each object and the number of possible relations between objects. Second, when actions are related to objects—such as moving one object to another—the number of actions grows equally fast.

The three domains mentioned in the previous section all have large state-spaces and are hard to solve with ordinary Q-learning. To illustrate this, figure 6 shows the success-rate of random policies in the blocks world. The agent with the random policy is started form a randomly generated state (which is not a goal state) and is allowed to take at most 10 actions. For each of the three goals (i.e. stacking, unstacking and $on(A, B)$) the left graph shows the percentage of trials that end in a goal state and therefore with a reward, with

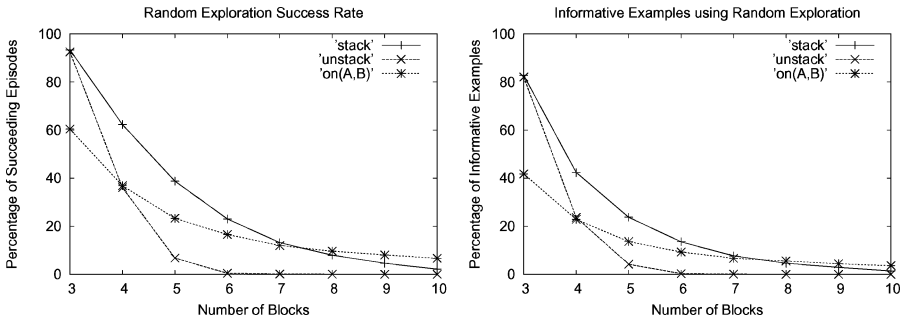

Figure 6. Success rate and information percentage in blocks world with random policy.

respect to the number of blocks in the world. As shown in the graph, the unstacking goal in the blocks world with 10 blocks 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 actions 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. The graph in the right of figure 6 shows the percentage of learning examples with a non zero Q-value that is presented to the regression algorithm. Since all examples with a zero Q-value can be regarded as noise for the regression algorithm, it is clear that learning the correct Q-function from these examples is very hard.

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 shapes of the blocks following the next one are unknown) makes it a very hard application for Q-learning.

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 emeralds 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 with random exploration.

5.2. Using "reasonable" policies for guidance

Although random policies can have a hard time reaching sparsely spread rewards in a large world, it is often relatively easy to reach these rewards 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 choose the actions instead of a policy derived from the current Q-function hypothesis (which will not be informative in the early stages of learning). The episodes created in this way can be used in exactly the same way as normal episodes in the RRL-algorithm to create a set of examples which is presented to the relational regression algorithm. In case of a human controller, one could log the normal operation of a system together with the corresponding rewards and generate the learning examples from this log.

Since tabled Q-learning is exploration insensitive—i.e., the Q-values will converge to the optimal values, independent of the exploration strategy used (Kaelbling, Littman, and Moore, 1996)—the non-optimality of the used policy will have no negative effect on the convergence of the Q-table. While Q-learning with generalization is not exploration insensitive, we hope (and will demonstrate) that the "guiding policy" will help the learning system to reach non-obvious rewards and that this results in a two-fold improvement in 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 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—in the available time.

While the idea of supplying guidance or another initialization procedure to increase the performance of a tabula rasa algorithm such as reinforcement learning is not new (see the discussion of related work in Section 8), it is underutilized. With the emergence of new reinforcement learning approaches, such as RRL, that are able to tackle larger problems, this idea is gaining importance and could provide the leverage necessary to solve really hard problems.

5.3. Different strategies for supplying guidance

When we supply guidance by creating episodes and presenting the resulting learning examples to the used regression engine, we can use different strategies to decide when to supply this guidance.

One option that we will investigate is supplying the guidance at the beginning of learning, when the reinforcement learning agent is forced to use a random policy to explore the state-space. This strategy also makes sense when using guidance from a human expert. After logging the normal operations of a human controlling the system, one can translate these logs into a set of learning examples and present this set to the regression algorithm. This will allow the regression engine to build a partial Q-function which can later be used to guide the further exploration of the state-space. This Q-function approximation 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 Q-function based exploration. The RRL algorithm explores the state space using Boltzmann exploration (Kaelbling, Littman, and Moore, 1996) based on the values predicted by the partial Q-function. This makes a compromise between exploration and exploitation of the partial Q-function.

Another strategy is to supply the guidance interleaved with normal exploration episodes. In analogy with human learning, this mixture of perfect and more or less random examples can make it easier for the regression engine to distinguish beneficial actions from bad ones. We will compare the influence of guidance when it is supplied with different frequencies.

One benefit of interleaving guided traces with exploration episodes is that the reinforcement learning system can remember the episodes or starting states for that it failed to reach a reward. It can then ask for guidance starting from the states in which it failed. This will allow the guidance to zoom in on areas of the state-space which are not yet covered correctly by the regression algorithm.

6. Experiments

6.1. Experimental setup

Our experiments are aimed at evaluating the influence of guidance on the performance of relational reinforcement learning. The guidance added to exploration should have a 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 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—for a given amount of learning experience.

To test these effects, we compare RRL (without guidance) with G-RRL (with guidance) 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 G-RRL, we substitute some of the exploration with guided traces. These traces are generated by either a hand-coded policy, a previously learned policy or a human controller. In between these traces, G-RRL is allowed to explore the state-space further on its own. Note that in the performance graphs, the traces presented to G-RRL will count as episodes.

We conduct experiments in the three domains described in Section 4: the blocks world and two computer games (Tetris and Digger). Each of these three application domains is characterized by a huge state space and hard to reach rewards. In these domains, we investigate the effects of using guidance in a number of settings, characterized along several dimensions. The dimensions include the strategy/mode of providing guidance (as described in Section 5.3) and the generalization engine used within relational reinforcement learning (as described in Section 3).

We use several types of policies for guidance. In the blocks world, we use optimal (hand-coded) policies and policies where optimal actions are interleaved with random actions (we call the latter "half-optimal"). In the Tetris game, we use performance traces from a human player, as well as hand-coded inoptimal policies. Finally, in the Digger policies, we use a policy learned by relational reinforcement learning to provide traces for guidance, thus bootstrapping the RRL process.

6.2. 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.

We will test the influence of guided traces on the two different regression algorithms discussed in Section 3. Both have different strategies for dealing with incoming learning examples and as such will react differently to the presented guidance.

The TG-algorithm needs a higher number of learning examples when compared to RIB. On the other hand, the TG-implementation is a lot more efficient than the RIB-implementation, so TG is able to handle more training episodes for a given amount of computation time. Since we are trying to investigate the influence of guidance for the two systems and not trying to compare their performance, we supply the TG-algorithm with a lot of training episodes (as it has little difficulty handling them) and the RIB algorithm with less training episodes (given the fact that it usually doesn't need them).

6.2.1. Guidance at the start of learning. 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 can be obtained by using a small

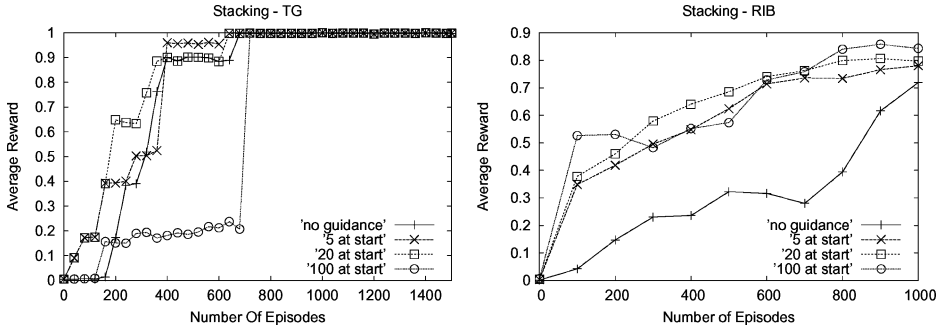

Figure 7. Guidance at start for the Stacking goal in the blocks world.

amount of guidance as shown in figure 7. The TG based algorithm is quite good at learning a close to optimal policy by itself. However, the added help from the guided traces helps it to decrease of the number of episodes needed to obtain a certain performance level. RRL-RIB has a harder time with this goal. It doesn't come close to reaching the optimal policy, but the help it receives from the guided traces do allow it to both reach better performance earlier in the learning episode as well as reach a higher level of performance overall. The graphs shows the average performance of RRL over 10 testruns.

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 graph in figure 8. It also shows the average performance over 10 testruns. RRL never learns anything useful on its own, because it never reaches a single reward. Even if we supply RRL-TG with 100 optimal traces, nothing useful is learned. When we supply RRL-TG 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 RRL-TG is able to do very little extra with this information. This is caused by the fact that RRL-TG throws out the statistics it collected when it chooses a suitable splitting criterion and generates two new and

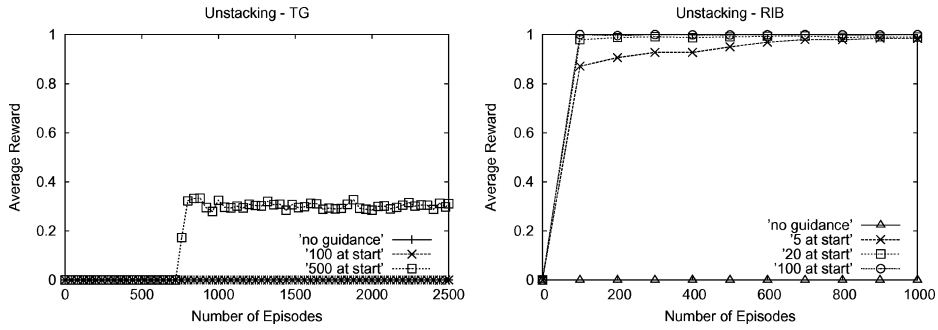

Figure 8. Guidance at start for the Unstacking goal in the blocks world.

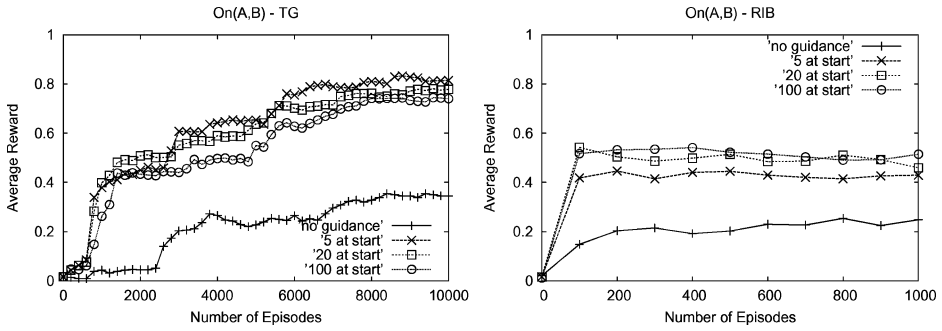

Figure 9. Guidance at start for the On(A,B) goal in the blocks world.

empty leaves. The RIB algorithm is very good at remembering high scoring (*state*, *action*) pairs. Once an optimal Q-value is encountered, it will never be forgotten. This makes RRL-RIB perform very well on the Unstacking-goal, reaching a close to optimal strategy with little guidance. Figure 8 does show that even 5 guided traces can be sufficient, although a little extra guidance helps to reach high performance sooner.

The on(A,B) goal has always been a hard problem for RRL (Džeroski, De Raedt, and Driessens, 2001; Driessens, Ramon, and Blockeel, 2001). Figure 9 shows the learning curves for both RRL-TG and RRL-RIB when we supply them with 0, 5, 20 and 100 optimal traces. Every data point is the average reward over 10 000 randomly generated test cases in the case of RRL-TG, 1 000 in the case of RRL-RIB both collected over 10 separate test runs. Although the optimal policy is never reached, the graph clearly shows the improvement that is generated by supplying RRL with varying amounts of guidance.

6.2.2. A learning delay. An interesting feature of the performance graphs of RRL-TG is the performance of the experiment with the stacking goal where we supplied it with 100 (or more) optimal traces in the beginning of the learning experiment. Not only does this experiment take longer to converge to a high performance policy, but during the first 100 episodes, there is no improvement at all. RRL-RIB does not seem to suffer from this at all.

This behavior becomes worse when we supply G-RRL with even more optimal traces. In figure 10, we show the learning curves when we supply G-RRL with 500 optimal traces. The reason for RRL-TG'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 generalization engine. Trying to connect the correct Q-values with the corresponding state-action pairs, the generalization 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 RRL-TG 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

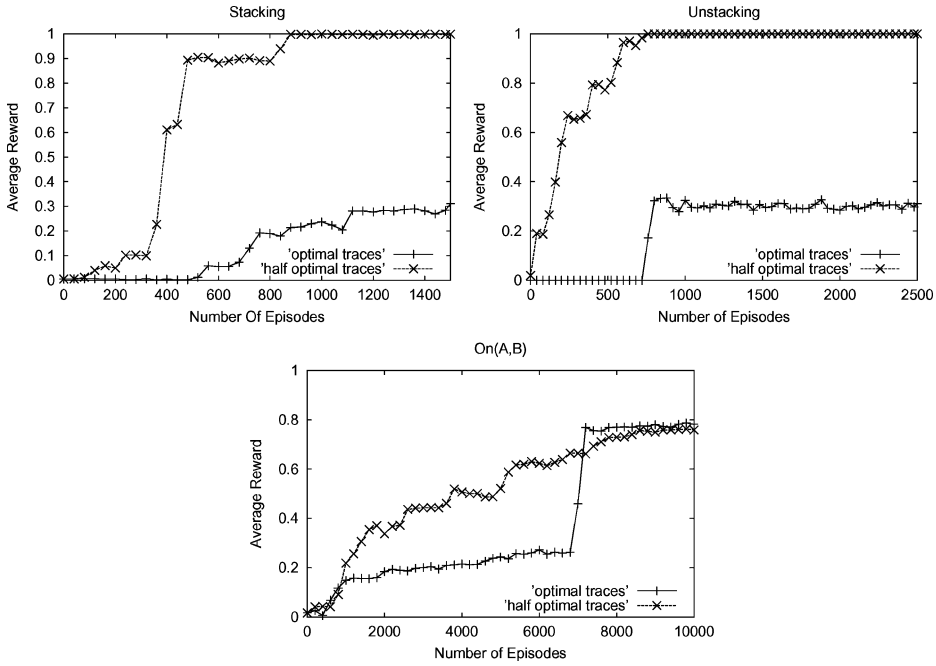

Figure 10. Half optimal guidance in blocks world for TG.

which are at different distances from the goal-state. Later, during exploration, it will expand this tree to account for optimal and non-optimal actions in these states. These trees are usually larger than they should be, because RRL is often able to generalize in one leaf of its tree over both non-optimal actions in states that are close to the goal and optimal actions in states that are a little further from the goal.

To illustrate this behavior, we supplied RRL-TG with 500 half-optimal guidance traces in which the used policy alternates between a random and an optimal action. Figure 10 shows that, in this case, G-RRL does learn during the guided traces. Most noticeable is the behavior of RRL-TG with half optimal guidance when it has to deal with the unstacking goal. Even though it is not trivial to reach the goal state when using a half optimal policy, it is reached often enough for G-RRL to learn a correct policy. Figure 10 shows that G-RRL 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.

This experiment (although artificial) shows that G-RRL can be useful even in domains where it is easy to hand-code a reasonable policy. G-RRL 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.

RRL-RIB does not suffer from this overgeneralization. Since the Q-value estimation is the result of a weighted average of neighboring examples, the RIB algorithm is able to

make more subtle differences between (*state*, *action*) pairs. Since the used weights in the average calculation are based on the distance between two (*state*, *action*) pairs, and since this distance has to include information about the resemblance of the two actions, there is almost no chance of overgeneralization of the Q-values over different actions in the same state.

6.2.3. Spreading the guidance. Spreading the guidance through the entire learning experiment instead of presenting an equal amount of guidance all in the beginning of learning avoids the overgeneralization problem that occurred when using TG. The left side of figure 11 clearly shows that RRL-TG does not suffer the same learning delay.

RRL-RIB did not suffer from overgeneralization and as a consequence, there is little difference between the obtained results with initial and spread guidance. RIB is designed to select and store examples with high Q-values. It does not matter when during the learning experiment these examples are encountered.

Figure 12 shows the influence of spreading the guidance for the unstacking goal. Again, there is little influence on the performance of RRL-RIB, except for the fact that the performance progression becomes somewhat slower but smoother. For RRL-TG, there is a large difference. With initial (and optimal) guidance, TG was not able to learn any useful

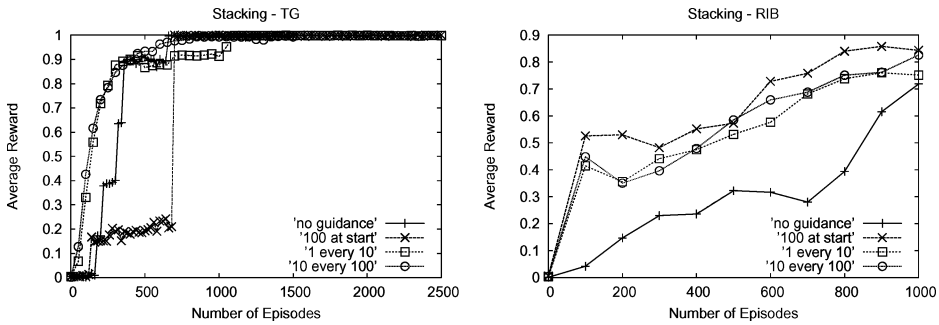

Figure 11. Guidance at start and spread for the Stacking goal in the blocks world.

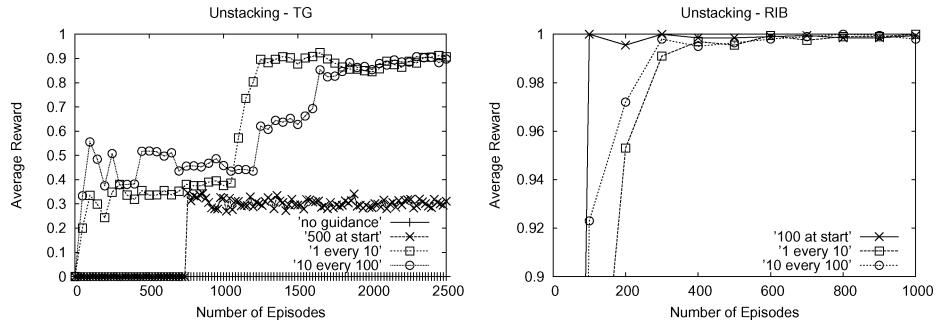

Figure 12. Guidance at start and spread for the Unstacking goal in the blocks world.

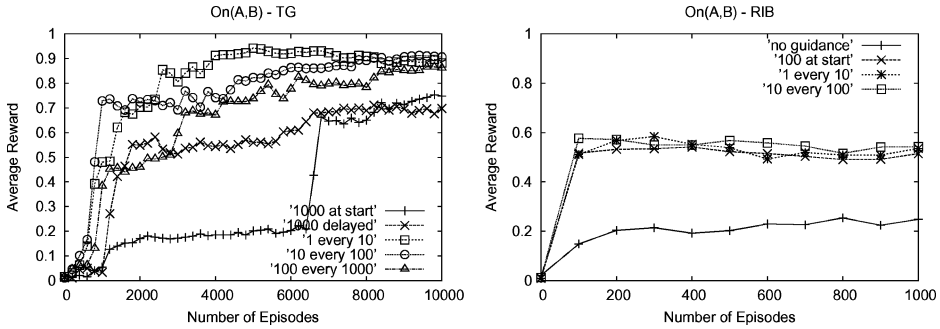

Figure 13. Guidance at start and spread for the On(A,B) goal in the blocks world.

strategy. In this case however, the mix of guided traces and explorative traces allows TG to build a well performing Q-tree. It still is less likely to find the optimal policy then with the (artificial) half-optimal guidance but performs reasonably well.

For the On(A,B) goal, the influence of the spread guidance on the performance of RRL-TG is large, both in terms of learning speed as the overall level of performance reached as shown in figure 13. Again, for RRL-RIB, there is little difference.

All graphs, but especially the left side of figure 13 show the influence of different frequencies used to provide guidance. Note that in all cases, an equal amount of guidance was used. Although the results show little difference, there is a small advantage for thinly spread guidance. 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.

Another possibility for dealing with scarce guidance is to provide all the guidance after RRL has had some time to explore the environment. Although figure 13 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 of the learning experiment.

6.2.4. 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. In a planning problem like the ones of the blocks world, this comes down to the fact whether RRL received a reward of 1 or not. We can give RRL the opportunity to ask for guided traces starting from some of these starting states were it failed. This will allow RRL to explore parts of the state space where it does not yet have enough knowledge and to supply the generalization algorithm with examples which are not yet correctly predicted.

Figure 14 shows the results of this active guidance. Two kinds of behavior can be distinguished. In the first, G-RRL succeeds in finding an almost optimal strategy, and the active guidance succeeds in pushing G-RRL to even better performance at the end of the learning

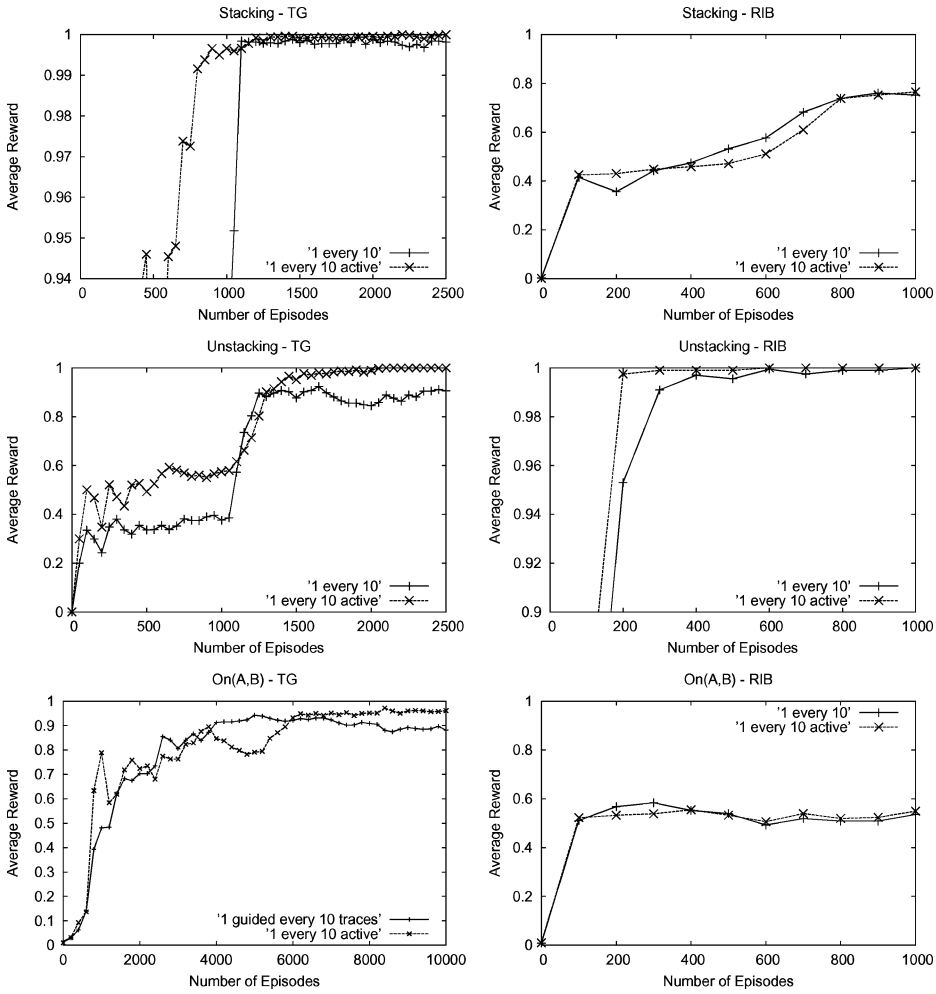

Figure 14. Active guidance in blocks world.

experiment. This is the case for all goals with RRL-TG and for the Unstacking goal with RRL-RIB. For example, the percentage of cases where RRL-TG does not reach the goal state in the On(A,B) experiment is reduced from 11% to 3.9%. For Stacking and Unstacking, active guidance helps RRL-TG to find an optimal policy: in these cases regular guidance improves performance but is not sufficient to find an optimal policy.

The above behavior is completely consistent with our expectations. In the beginning, both modes of guidance provide enough new examples to increase the accuracy of the learned Q-functions. However, when a large part of the state-space is already covered by the Q-functions, the specific examples provided by active guidance allow for the Q-function to be extended to improve its coverage of the outer reaches of the state-space.

In the second kind of behavior, G-RRL does not succeed in reaching a sufficiently high level of performance. This happens for RRL-RIB on the tasks of Stacking and On(A,B): there is little difference here between the help provided by normal and active guidance. Active guidance is not able to focus onto critical regions of the state-space and improve upon the examples provided by regular guidance.

6.2.5. Comparing the generalization engines. Our experiments in the blocks world were not designed specifically to compare the relative performance of the two generalization engines within RRL (RRL-TG and RRL-RIB). Still, several observations and comparisons can be made. Some of the differences in performance arise from the general characteristics of the learning algorithm families (decision trees and instance-based), while others stem from the interaction of the specific learning algorithms and the use of guidance as well as the mode the guidance is provided in.

In general, decision trees can process large amounts of data, i.e., numbers of examples, fast. They thrive on large numbers of examples and generalize well when examples abound. However, when a small number of examples is given, as well as in other cases where cautious generalization is needed (e.g., when the distribution of examples is highly skewed), decision trees can perform poorly. Instance-based methods, on the other hand, are very well suited for cautious generalization and perform well even when a small number of examples is provided. However, processing large numbers of examples becomes a problem and can drastically increase the time complexity of learning and especially of prediction.

Overall, given a large number of episodes, RRL-TG achieves higher performance than RRL-RIB. This is true regardless of whether guidance is provided or not. The only case where the performance of RRL-TG stays well under that of RRL-RIB is in the Unstacking problem where all guidance is provided at the start: given the extreme sparseness of rewards (and skewedness of the distribution of examples) for Unstacking, cautious generalization is really needed. In addition, the combination of the RRL-TG algorithm and providing guidance at the start is inappropriate (as discussed below).

For a small number of episodes, RRL-RIB typically achieves higher performance than RRL-TG. This is due to the fact that cautious generalization is needed when a small number of examples is given. Similar relative performance (for a small number of episodes) is observed by Driessens and Ramon (2003) in a slightly different experimental setting for the blocks world, where RRL explores worlds with 3, 4, and 5 blocks and is given guided traces in a world with 10 blocks.

The performance of G-RRL is clearly affected by the interaction between the generalization engine used and the mode the guidance is provided in. RRL-RIB performs better when all guidance is provided at the start: its ability to generalize cautiously is crucial in this respect. The more guidance is provided at the start, the better.

For RRL-TG, providing all guidance at the start causes problems for two reasons. The first has to do with decision trees and their proneness to overgeneralization, as discussed above: from optimal traces only, one cannot distinguish between optimal and suboptimal actions. The second reason is specifically related to the TG algorithm: TG essentially forgets the statistics collected from the data after a split in the decision tree is made. In the extreme case, this means that all guidance traces are lost after the first split in the decision tree is

made (this is exactly what happens in the case of providing all guided traces for Unstacking at the beginning).

6.3. The Tetris game

It is quite hard (if not impossible) to generate an optimal or even "human level" strategy for the Tetris game. So at first, we opted to supply G-RRL with traces of non-optimal playing behavior from a human player.

The overall results for learning Tetris with RRL and G-RRL are below our expectations. However, the added guidance in the beginning of the learning experiment still has its effects on the overall performance. Figure 15 shows the learning curves for RRL and G-RRL 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.

For the experiments with the spread guidance where we needed more guided traces, we opted for a simple hand coded strategy consisting of 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.

Figure 16 shows the improvement of RRL with spread guidance. Since the problem of overgeneralization is more apparent in the Tetris experiments—the experiment with 1000 starting traces (an equal amount of guidance as in the spread case) 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. However, the improvement received from spreading the

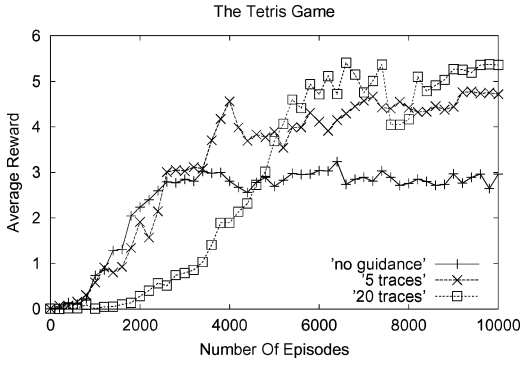

Figure 15. Guidance at start in Tetris.

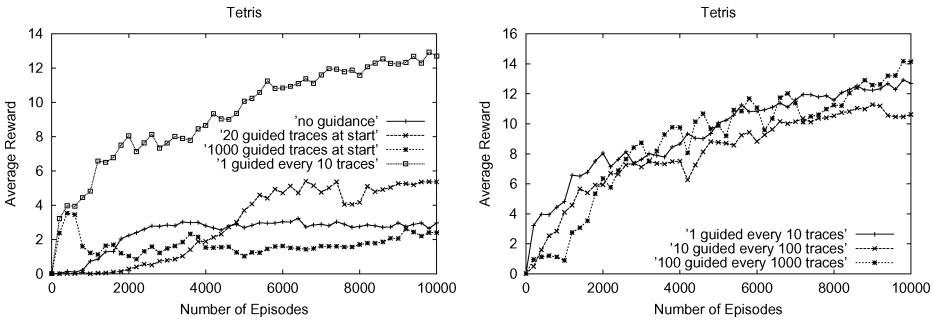

Figure 16. Spread guidance in Tetris.

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.

The right side of figure 16 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 experiment will slow down the progress made by the RRL-system during the initial stages of the experiment, there is little or no difference between the performances later on in the experiment. This behavior can be observed again in the Tetris case were there is a short slow down at the start of the learning curve for 100 guided traces every 1000 learning episodes.

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 17. The graph shows that there is a significant increase in performance for

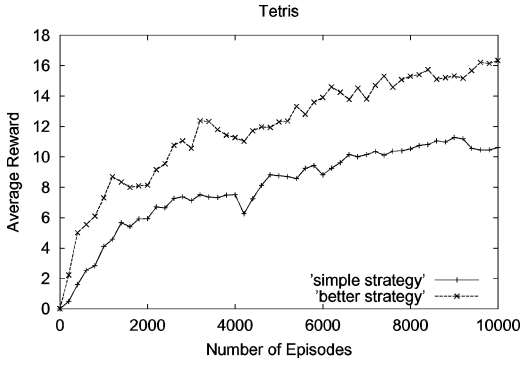

Figure 17. Different guidance-policies in Tetris.

using the better policy to guide RRL. This also illustrates that the exploration insensitivity of table-based Q-learning (Kaelbling, Littman, and Moore, 1996) does not carry over to Q-learning with generalization. However, one should notice that although the "guidance strategy" improved by approximately 10 lines per game, the improvement of the resulting strategy learned by RRL is smaller.

On this problem, RRL performs a magnitude worse when compared to the results reported in Bertsekas and Tsitsiklis (1996) and Lagoudakis, Parr, and Littman (2002); both approaches use approximate policy iteration on the Tetris application. We believe that the bad performance of RRL-TG is due to the fact that Q-learning is unsuitable for the Tetris domain. Given the stochastic nature of the game, 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). Thus, although Tetris is a relational domain, we believe that RRL-TG is not suitable to tackle the problem. We suspect that the better performance of the approaches considerd in Bertsekas and Tsitsiklis (1996) and Lagoudakis, Parr, and Littman (2002) is derived from the learning paradigm (approximate policy iteration), which apparently is of much higher importance than the representation. To test this hypothesis, one should investigate the use of an approximate policy iteration approach in a relational setting.

6.4. The Digger game

Because it is hard to write a policy for the Digger Game, we used a policy generated in earlier work (Driessens and 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 generalization algorithm the opportunity to construct a possibly smaller and more accurate Q-function approximation. Again, only tests using RRL-TG were done.

Figure 18 shows the average reward obtained by the learned strategies over 640 Digger test-games divided over the 8 different Digger levels. It shows that G-RRL is indeed able to improve on the policy learned by RRL. Although the speed of convergence isn't improved, G-RRL reaches a significantly higher level of overall performance. The graph also shows

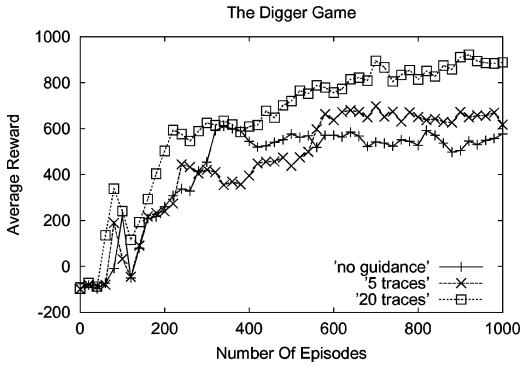

Figure 18. Guidance at start in Digger.

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.

7. Discussion

In this section, we summarize the experimental results from the previous sections and draw some general conclusions. This takes the form of several questions that were addressed by the experiments and the answers obtained from the experimental results.

- *Does guidance improve the performance of RRL for problems with large state spaces and sparse rewards?* The answer to this question is an equivocal yes. Guidance always improved the performance of RRL, in terms of the level of performance achieved, the speed of convergence, or both. This proved true across a range of experimental setups, where the problem addressed was varied, as well as the guidance policies, the mode of providing guidance and the generalization engine used within RRL. The only exception is that RRL-TG may converge more slowly towards the optimal policy or even achieve a lower performance level if all guidance is supplied at the start.
- *Does more guidance mean better results?* Providing more guided traces can improve the performance of RRL, in particular when they are provided to RRL-RIB or we are talking about a relatively small number of guided traces, which are provided at the start. Providing RRL-TG with a large number of guided traces at the start can even reduce performance due to overgeneralization problems.
- *What mode of providing guidance works best?* This depends on the generalization engine used. For RRL-TG, guidance should be spread, i.e., guided traces should be interleaved with exploration traces. For RRL-RIB, providing all the guidance at the start works best. If a relatively high level of performance is achieved, it can be further improved by using active guidance.
- *Which generalization engine works better within RRL*? RRL-TG is more efficient and achieves higher levels of performance, provided a large number of episodes are available and guidance is spread. RRL-RIB converges faster and achieves better performance levels when a smaller number of episodes is available and all evidence is given at the start.
- *How do the different guidance policies affect the performance of G-RRL*? When the guidance policy yields high rewards or is even optimal, G-RRL often, but not always, approaches the performance of the guidance policies. When the guidance policy is clearly not optimal, G-RRL may perform much better than the guidance policy. In such cases, improving the guidance policy is likely to result in improved performance in G-RRL. However, it is crucial not to provide too much good guidance to RRL-TG at the start, due to overgeneralization problems. In such cases, guidance from suboptimal policies may avoid overgeneralization and result in better performance.

8. Related work

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

system just accepts the offered advice. In the 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 behavioral cloning, the second to reinforcement learning and the third to guided reinforcement learning.

The link between this work and behavioral cloning (Bain and Sammut, 1995; Urbancic, Bratko, and Sammut, 1996) is not very hard to make. If we would supply the TG generalization algorithm with low Q-values for unseen state-action pairs, RRL would learn to imitate the behavior 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 behavioral cloning.

Scheffer, Greiner, and Darken (1997) discusses some of these problems. The differences between learning by experimentation and learning with "perfect guidance" (behavioral cloning) and the problems and benefits of both approaches are highlighted. Behavioral 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 memorize the ideal sequence of actions but has to store associations instead. This is very closely related to the problems our generalization 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 and 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.

Lin's work on reinforcement learning, planning and teaching (Lin, 1992) and the work of Smart and Kaelbling on reinforcement learning in continuous state-spaces (Smart and Kaelbling, 2000) is closely related to ours in terms of combining guidance and experimentation. Lin uses a neural network approach for generalization and uses a human strategy to teach the agent. The reinforcement learning agent is then allowed to replay each teaching episode to increase the amount of information gained from a single lesson. However, the number of times that one lesson can be replayed has to be restricted to prevent over-learning. This behavior is strongly related to the over-generalization behavior of TG when only perfect guidance is presented.

Smart's work deals with continuous state-spaces uses a nearest neighbor approach for generalization and use example training runs to bootstrap the Q-function approximation. The use of nearest neighbor and convex hulls successfully prevents overgeneralization. It is not clear how to translate the convex hull approach to the relational setting.

Another technique that is based on the same principles as our approach is used by Dixon, Malak, and Khosla (2000). They incorporate prior knowledge into the reinforcement learning agent by building an off-policy exploration module in which they include the prior knowledge. They use artificial neural networks as a generalization engine.

Other approaches to speed up reinforcement learning by supplying it with non-optimal strategies include the work of Shapiro, Langley, and Shachter (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 G-RRL.

Recently, there has been a large increase in the work on policy generation for relational domains. Most closely related to the approach used in the RRL system is the work on relational Markov decision processes. Kersting and De Raedt (2003) introduce Logical Markov Decision Processes as a compact representation of relational MDPs. By the use of abstract states (i.e. a conjunction of first order literals) and abstract actions that are defined in a STRIPS-like manner (Fikes and Nilsson, 1971) they greatly reduce the number of possible (*state*, *action*) pairs and thus the number of Q-values that need to be learned. Independent of this, Morales (2003) introduced rQ-learning, i.e. Q-learning in *R-Space*. R-Space consists of *r-states* and *r-actions* which are very comparable to the abstract states and abstract actions in the work of Kersting and De Raedt. The rQ-learning algorithms tries to calculate the Q-values of the (r-state, r-action) pairs. Van Otterlo (2004) defines the CARCASS representation that consists of pairs of abstract states and the set of abstract actions that can be performed in the given abstract state. While this is again comparable to the two previously discussed approaches, van Otterlo not only defines a Q-learning algorithm for his representation but also suggests learning a model of the relational MDP defined by the CARCASS to allow prioritized sweeping to be used as a solution method.

Fern, Yoon, and Givan (2003) use an approximate variant of policy iteration to handle large state spaces (Fern, Yoon, and Givan, 2003). A policy language bias is used to enable the learning system to build a policy from a sampled set of Q-values. Just like in standard policy iteration (Sutton and Barto, 1998), approximate policy iteration interleaves policy evaluation and policy improvement steps. Yoon, Fern, and Givan (2002) present an approach for translating abstract policies from small relational MDPs to larger problems.

9. Conclusions

In this paper, we address 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 to the reinforcement learning agent does help improve performance in such cases. Guidance in our case takes the form of traces of execution of a "reasonable policy" that provides sufficiently dense rewards.

We demonstrate the utility of guidance through experiments in three domains: the blocks world and two computer games (Tetris and Digger). Each of these three application domains is characterized by a huge state space and hard to reach rewards. We investigate the effect of using guidance in a number of settings, characterized along several dimensions. The dimensions include the mode of providing guidance, the form of the guidance policy used, and the generalization engine used within relational reinforcement learning.

We investigate two modes of using guidance: providing all guidance at the start and spreading guidance, i.e., providing some guided episodes followed by several exploration episodes, and repeating this. A variation on the latter mode is active learning, where the agent

asks for guided traces starting from initial states that it selects itself rather than receiving guided traces from randomly chosen initial states. Three different forms of guidance policies are considered: hand-coded—possibly optimal—policies, policies already learned by RRL in a previous experiment and the policy of a human performing the task at hand. We use two different generalization engines within RRL, induction of relational regression trees (TG) and relational instance-based regression (RIB).

Overall, the use of guidance in addition to experimentation improves performance over using experimentation only, for all considered combinations of the dimensions mentioned above. We observed improvements in terms of the overall performance level achieved, the convergence speed, or both. The improvements result from using the best of both worlds: guidance provides perfect or reasonably good information about the optimal action to take in a narrow range of situations, while experimentation can obtain imperfect information about the optimal action to take in a wide range of situations. The actual magnitudes of performance improvement does depend on the considered combination of the mode of providing guidance and generalization engine.

While both guidance at the start and spread guidance improve the performance of RRL, spread guidance often yields higher, but more importantly never lower performance. This is especially the case if regression engine is vulnerable to over-generalization such as the relational regression trees (TG-trees). Providing all the guidance up front doesn't quite work well in this case for several reasons. Namely, making a split on "perfect" guidance generated examples only distinguishes between the regions of equal state-values, but not the actions that move you between them. This can be corrected by splits further down the tree, but this requires lots of extra examples, and therefore more learning episodes. This problem is aggravated by the fact that after making a split, the guidance received so far is lost. These problems do not appear when instance-based regression (RIB) is used as a generalization engine as the RIB-algorithm is designed to remember high yielding examples and the bias towards action difference of the used distance prevents the over-generalization that occurs with TG.

Active learning with spread guidance helps improve performance in the later stages of learning, by enabling fine tuning of the learned Q-function by focusing on problematic regions of the state space. This often results in a significant reduction of the cases where RRL exhibits non-optimal behavior. Experiments show that a sufficiently high level of performance has to be reached by G-RRL for the active guidance to have any effect. If performance is too low to allow fine-tuning, active guidance does not improve on normal guidance.

Guidance from different sources can be successfully used within the proposed approach. Whether we used traces generated by hand-programmed policies, policies learned by RRL or a human performing the task at hand, we always obtained higher performance as compared to using no guidance. The quality of the policy used to generate guidance does matter, as illustrated by the experiment on the Tetris domain, where using a better policy to provide guidance yielded significantly better performance.

10. Further work

One possible direction for further work is the tighter integration of the use of guidance and the generalization engine used. For example, when dealing with a model building regression

algorithm like TG, one could supply more and possibly specific guidance when the algorithm is bound to make an important decision. In the case of TG, this would be when TG is ready to choose a new test to split a leaf. Even when this guidance is not case specific, it could be used to check whether a reasonable policy contradicts 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.

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 behavior of RRL or—given the declarative nature of the policies and Q-functions that are generated by RRL-TG—at the policy that RRL has constructed itself and adjust the advice it wants to give. In the long run, because RRL uses an inductive logic programming approach to generalize 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 Q-function is useless and has to be rebuilt, or even constraints that the learned policy has to satisfy.

Although the 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 stochastic applications one could try to remember all the stochastic elements and try to recreate the episode. For the Tetris game this would include the entire sequence of blocks and asking the guidance strategy for a game with the given sequence. However, given the large difference in the Tetris state as a consequence of only a few different actions, we anticipate the effect of this approach to be 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.

One direction of future work that will surely be investigated is the application of other first order regression algorithms as Q-function approximations and the interaction of these generalization algorithms with different types and modes of guidance. In addition, reinforcement learing approaches other than Q-learning (such as direct policy search and approximate policy iteration) in a relational setting should be investigated in more detail.

Notes

1. Tetris was invented by Alexey Pazhitnov and is owned by The Tetris Company and Blue Planet Software.
2. http://www.digger.org.

References

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6:1, 37–66.

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

Bertsekas, & Tsitsiklis (1996). *Neuro-dynamic programming*. Athena Scientific.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101:1/2, 285–297.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont: Wadsworth.

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

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 726–731).

De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence*, 95, 187–201.

De Raedt, L., & Džeroski, S. (1994). First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70, 375–392.

Dixon, K., Malak, R., & Khosla, P. (2000). Incorporating prior knowledge and previously learned information into reinforcement learning agents. Technical report, Institute for Complex Engineered Systems, Carnegie Mellon University.

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

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. In *Submitted to ICML 2003*.

Driessens, K., Ramon, J., & Blockeel, H. (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* (pp. 97–108). Springer-Verlag.

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

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

Emde, W., & Wettschereck, D. (1996). Relational instance-based learning. In L. Saitta (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 122–130). Morgan Kaufmann.

Fern, A., Yoon, S., & Givan, R. (2003). Approximate policy iteration with a policy language bias. In T. S., L. Saul, & B. Bernhard Schlkopf (Eds.), *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems*. The MIT Press.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application for theorem proving to problem solving. In *Advance Papers of the Second 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.

Kersting, K., & De Raedt, L. (2003). Logical markov decision programs. In *Proceedings of the IJCAI'03 Workshop on Learning Statistical Models of Relational Data* (pp. 63–70).

Kirsten, M., Wrobel, S., & Horvath, T. (2001). Distance based approaches to relational learning and clustering. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining* (pp. 213–232). Springer-Verlag.

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

Lagoudakis, M., Parr, R., & Littman, M. (2002). Least-squares methods in reinforcement learning for control. In *Proceedings of the 2nd Hellenic Conference on Artificial Intelligence (SETN-02)* (pp. 249–260), Springer.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. Ellis Horwood.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8, 293–321.

Morales, E. (2003). Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-agent Systems* (pp. 15–26).

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20, 629–679.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann series in machine learning. Morgan Kaufmann.

Ramon, J. (2002). Clustering and instance based learning in first order logic. Ph.D. thesis, Department of Computer Science, K.U. Leuven.

Ramon, J., & Bruynooghe, M. (2001). A polynomial time computable metric between point sets. *Acta Informatica*, 37, 765–780.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*: *Foundations* (ed. w/ PDP Research Group). vol. 1, MA: MIT Press Cambridge.

Scheffer, T., Greiner, R., & Darken, C. (1997). Why experimentation can be better than "Perfect Guidance". In *Proceedings of the 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. In *Proceedings of the 5th International Conference on Autonomous Agents*. Association for Computing Machinery.

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

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceeding of the 8th Conference on Advances in Neural Information Processing Systems* (pp. 1038–1044). Cambridge, MA: The MIT Press.

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

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

Utgoff, P., Berkman, N., & Clouse, J. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:1, 5–44.

Van Laer, W., & De Raedt, L. (2001). How to upgrade propositional learners to first order logic: A case study. In S. Džeroski, & N. Lavrač (Eds.), *Relational Data Mining* (pp. 235–261). Springer-Verlag.

van Otterlo, M. (2004). Reinforcement learning for relational MDPs. In *Proceedings of the Machine Learning Conference of Belgium and the Netherlands 2004*.

Wagner, R., & Fischer, M. (1974). The string to string correction problem. *Journal of the ACM*, 21(1), 168–173.

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

Watkins, C. (1989). Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge.

Wiering, M. (1999). Explorations in efficient reinforcement learning. Ph.D. thesis, University of Amsterdam.

Yoon, S., Fern, A., & Givan, R. (2002). Inductive policy selection for first order MDPs. In *Proceedings of UAI'02*.

Received March 21, 2003
Revised June 13, 2004
Accepted June 14, 2004
Final manuscript June 16, 2004