Relational TD Reinforcement Learning

Christophe Rodrigues, Pierre Gérard, and Céline Rouveirol

LIPN (Computer Science lab of the Paris 13 University)
first.last@lipn.univ-paris13.fr

Abstract. Relational Reinforcement Learning (RRL) addresses the use of relational representations of states and actions in RL rather than the usual attribute-values. Most works in this field aims at improving relational function approximation, or at adapting advanced techniques to the relational framework. However, little has been done so far to investigate basic Temporal Difference in RRL. In this paper, we propose adaptations of Sarsa and regular Q-learning to the relational case, by using an incremental relational function approximator RIB. In the experimental study, we highlight how changing the RL algorithms impacts generalisation in relational regression.

1 Introduction

Most works on Reinforcement Learning (RL) [1] use propositional – feature based – representations to produce approximations of value-functions. If states and actions are represented by scalar vectors, the classical numerical approach to learn value-functions is to use regression algorithms. Recently, the field of Relational Reinforcement Learning (RRL) has emerged [2] aiming at extending Reinforcement learning to handle more complex – first order logic-based – representations for states and actions. Moving to a more complex language opens up possibilities beyond the reach of attribute-value learning systems, mainly thanks to the detection and exploitation of structural regularities in (state, action) pairs. Many works in RRL have focused on the development of relational regression algorithms, *ie*, they tackle the problem of approximating value-functions for relational representations of state and actions. Surprisingly, there has been yet little work to include basic Temporal Difference RL algorithms within such RRL systems. In this paper, building on the work of Driessens and colleagues [3], we plan to investigate the gain of upgrading the RL algorithm for a fixed relational regression algorithm, namely RIB [3].

In section 2, we introduce the RRL field by motivating the use of relational representations for RL, and sit our contribution in this context. In section 3, after a brief overview, we justify why we select RIB as the relational function approximator for our RRL system. In section 4, we detail how we adapt classical Temporal Difference algorithms to handle relational representations. Finally, section 5 describes several experiments and draw some lessons about the interactions between RL and ILP, before concluding on open research perspectives.

2 Relational representations in RL

Usually, RL uses propositional or attribute-value (AV) representations. States and actions are then represented by a vector of attributes, most of the time scalar. When value-based algorithms are used for RL, a value is associated to states or (state, action) pairs. This value enables to decide which state or (state, action) pair is most suitable among others. In order to produce value predictions, one can use numerical function approximators such as Neural Networks, as in [4] for instance.

Let us consider a simple and illustrative AI problem: the blocks world problem. Blocks are stacked on top of each other and an intelligent system can move blocks on each other in order to fulfil different goals, such as stacking all blocks in a single stack. Figure 1 shows a state example, where:

- blocks a, b and d are on $floor$;
- block c is on block b.

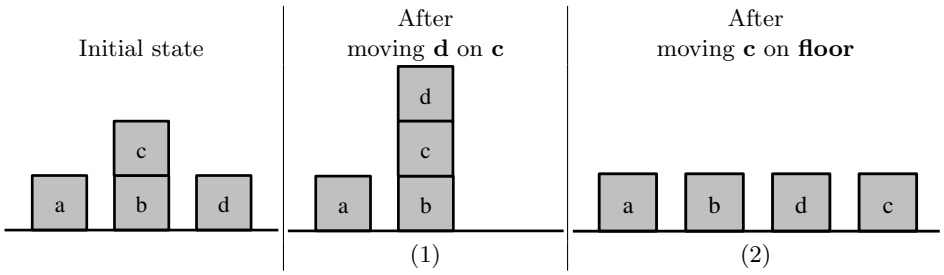

Fig. 1. Initial state and outcomes after action selection

Representing a state in such an environment is not straightforward for who wants to design a reasoning or a learning algorithm. A naive representation could use two attributes for each block: one for the column (from 0 to 3 in the example of figure 1) and the other for the height of each block in the stack (from 0 to 3 in the 4-blocks world of the example). With such a representation, the state in figure 1 is represented by the vector $(0, 0, 1, 0, 1, 1, 2, 0)$. Here, $(0, 0)$ corresponds to the "coordinates" of block a, $(1, 0)$ of block b, $(1, 1)$ of block c and $(2, 0)$ of block d. In order to represent an action in a propositional way, one could also use two attributes: one to specify the column of the block to move, and another one to specify the destination column. This way, moving block d (stack 2) on top of block c (stack 1) is represented by the action vector $(2, 1)$.

Figure 1 shows the resulting states s_{t+1} when different actions a_t are applied to the state s_t of figure 1. The corresponding (state, action) pairs can be represented in a propositional way as shown in table 1.

Option	(1)	(2)
Action	move **d** on **c**	move **c** on **floor**
s_t	$(0,0,1,0,1,1,2,0)$	$(0,0,1,0,1,1,2,0)$
a_t	$(2,1)$	$(1,4)$
s_{t+1}	$(0,0,1,0,1,1,1,2)$	$(0,0,1,0,2,0,3,0)$

Table 1. Examples of states, actions and outcomes with a propositional representation

RL algorithms have to discover regularities in the dynamics of the environment and to find out which possibilities are the most suitable ones among those considered (in our example, (1) should be preferred). For AV-based algorithms, numerical Q-values associated to (state, action) pairs enable to decide which action will be preferred in any given state. Rather than tabular algorithms, state-of-the-art RL algorithms use incremental regression in order to find accurate approximations of $Q : S \times A \rightarrow \mathbb{R}$ value functions with a limited number of parameters. Value-based RL methods use generalisation in order to produce a compact representation of the value function. How generalisation takes place is of course highly dependant on the chosen representation for states and actions.

In order to scale-up, a RL system should be able to use solutions learned on small problems as a starting point for large problems. By iterating this method, it becomes gradually possible to tackle complex problems, out of reach when directly addressed. For example, to deal with a 10 blocks problem[1], one could begin with a 5 blocks version of the problem[2] and then learning repeatedly with the solutions obtained at the previous iteration.

When regularities are identified on simpler versions of a problem and represented in a propositional way, it may be tricky to reuse them in order to tackle more complex versions of the problem. Thus scaling-up of AV RL algorithms to large problems is limited.

In order to implement such scaling-up capabilities, best is to shift to relational dedicated representations. In such representations, constants are associated to objects (blocks a, b, c and d for instance) which are linked thanks to relational predicates (like $on/2$). In our example, $on(a, b)$ denotes that block a is on top of block b). Assuming that we have a constant f representing the floor, the state on figure 1 can then be represented by $on(a, f), on(b, f), on(c, b), on(d, f)$ and the considered actions by (1) $move(d, c)$ and (2) $move(c, f)$. The corresponding outcomes are depicted by table 2.

[1] 58,941,091 states, difficult to handle from the outset
[2] 501 states only

Option	(1)	(2)
Action	move **d** on **c**	move **c** on **f**
s_t	$on(a,f), on(b,f), on(c,b), on(d,f)$	$on(a,f), on(b,f), on(c,b), on(d,f)$
a_t	$move(d,c)$	$move(c,f)$
s_{t+1}	$on(a,f), on(b,f), on(c,b), on(d,c)$	$on(a,f), on(b,f), on(c,f), on(d,f)$

Table 2. Examples of (state, action) pairs and outcomes with a relational representation

The first advantage of such a representation is its elicitation of relations between objects. More important is the new generalisation capabilities offered by such a language.

In order to bound the complexity of the value function representation, it is advisable to avoid tabular algorithms and rather shift to function approximation. As with propositional representations, a key point in value-based RL is to use incremental regression in order to find an approximation of the optimal value function. Since option (1) is more suitable than (2) in the above table, the function approximation should predict a greater value for

$$Q(on(a,f), on(b,f), on(c,b), on(d,f), move(d,c))$$

than for $Q(on(a,f), on(b,f), on(c,b), on(d,f), move(c,f))$, *eg* the less suitable (state, action) pair.

Moreover, the value of the former (state, action) pair should be identical to other pairs among which those depicted figure 2 and denoted as:

i on(c,f), on(c,f), on(**a**,d), on(**b**,f), move(**b**,**a**)
ii on(b,f), on(a,f), on(**d**,a), on(**c**,f), move(**c**,**d**)
iii on(d,f), on(c,f), on(**b**,c), on(**d**,f), move(**a**,**b**)

These states and actions are identical up to a permutation of constants a, b, c and d. When a system learns in one situation, it is suitable to generalise to the other ones. Considering generalisation issues, notice the matching of bold constants between the action and the state part. This kind of regularities between states

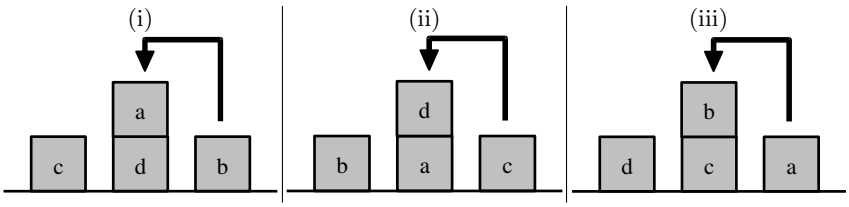

Fig. 2. Several (state, action) pairs with the same value

and actions is difficult to exhibit with AV RL. It is more straightforward with

relational representations: assuming that X, Y, Z and T are variables and f is still a constant denoting the floor, the (state, action) pairs of figure 2 can all be represented with the following first order description

$$on(X, f), on(Y, f), on(Z, Y), on(T, f), move(T, Z)$$

since variables X, Y, Z and T may instantiate to different constants depending on the (state, action) pair. In addition, such generalisations learned in small environments can be easily reused in more complex versions of the problem since the representation does neither rely on the number of constants nor on their order, thus providing the system with scaling-up capabilities.

3 Relational function approximation for RL

So as to draw expected benefits underlined in previous section, Relational Reinforcement Learning (RRL) addresses the development of RL algorithms operating on relational representations of states and actions, as informally described above.

Before defining RRL, we very briefly introduce basic notions of first order logics used in the remainder of the paper. Names of objects are called constants, their identifiers start with a lowercase character (a, b, f, ...). Variables, which are used to denote arbitrary objects have identifiers starting with an uppercase character (X, Y, Z, ...). A term here is either a constant or a variable. The arity o a function or predicate symbol is the number of terms to which the function or predicate symbol applies. A function or predicate symbol of arity n is referred to as f/n. Relations that link objects (*eg*, $on/2$) or actions (*eg*, $move/2$) are denoted by *predicate symbols*. Similarly, a predicate symbol of arity n is referred to as p/n. An *atom* is a predicate symbol applied to a number of terms (*eg*, $move(a, b)$). A ground atom is an atom without a variable. A *literal* is either an atom or a negated atom.

The relational Reinforcement Learning task can be defined as follows.

Definition 1. (Relational Reinforcement learning) *Given:*

- *a set of possible states $\mathcal{S}$, represented in a relational format,*
- *a set of possible actions $\mathcal{A}$, also represented in a relational format,*
- *an unknown transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$, (this function can be non-deterministic)*
- *an unknown real-valued reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$,*

the goal is to learn a policy for probabilistically selecting actions $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ that maximises the discounted return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ from any time step t. This return is the cumulative reward obtained in the future, starting in state s_t. Future rewards are weakened by using a discount factor $\gamma \in [0, 1]$. In value-based RL methods, the return is usually approximated thanks to a value function $V : \mathcal{S} \rightarrow \mathbb{R}$ or a Q-value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that $Q(s, a) = E\{R_t \mid s_t = s, a_t = a\}$.

States are relational interpretations, as used in the "learning from interpretations" setting [5, 6]. In this notation, each (state, action) pair is represented by a relational interpretation, *ie* a set of relational facts. The action is represented by an additional ground fact (see table 2).

Although there are numerous possible developments in RRL, the most studied one up to now is about learning to approximate value-functions for relational (state, action) pairs. [2, 7, 3, 8, 9] work on providing RRL with more and more efficient relational regression systems. This learning problem is a challenge for the Inductive Logic Programming (ILP) community, because it requires the development of incremental regression algorithms for rich representations.

The first relational regression system TILDE-RT, [10] coupled with a Q-learning-like system in the Q-RRL system [2], is not incremental: at the end of each episode, the system learns again from all (state, action) pairs already encountered.

The posterior work of Kurt Driessens and colleagues [7, 3, 8, 9] aimed at improving the relational regression system, focusing on the incrementality issue. TG [7] builds a regression tree as TILDE-RT does. TG mainly differs from TILDE-RT in the way it splits a node into several ones, if the test yielding this split is highly significant. TG is incremental, still the tree structure is quite rigid and does not allow to cope well with the concept drift inherent to RL : the quality of examples improves during learning and unnecessary nodes (nodes that do not belong to the optimal regression tree) may be introduced high in the tree, and cannot be suppressed later in the learning process.

More recently, the RIB system [3] adopts an instance based learning paradigm to approximate the Q-value function. RIB stores a number of prototypes each associated with a Q-value. These prototypes are employed to predict the Q-value of unseen examples, using a k-nearest-neighbour algorithm or a "maximum variance" estimation ([11], p80). Both rely on a relational distance adapted to the problem to solve (see [12] for a distance for the blocks world problem). RIB handles incrementality since it forgets prototypes that

i are not necessary to reach a good prediction performance or
ii have a bad prediction performance

KBR [9] uses a graph kernel defined on (state, action) pairs to produce highly accurate approximations of the value function. However, KBR is not incremental at all, yielding a poor efficiency for large problems. Given this state of the art, we have chosen to use RIB as relational regression system, as it is indeed incremental and stands for a quite good performance/efficiency compromise.

4 Relational Temporal Difference

In section 3, we have presented the state of the art concerning function approximation for RRL. All the results presented in [11] involve the same reinforcement learning algorithm, but coupled with a variety of relational regression algorithms.

In the RRL algorithm presented in [2], learning occurs only at the end of episodes, and not each time step. This matter is due to the fact that TILDE-RT was not fully incremental and thus, it appears more efficient providing the regression algorithm with large sets of samples, instead of learning each time an action is performed. Therefore, the RRL algorithm used by [2] stores full trajectories s_0, a_0, r_1, s_1, $\cdots$, s_{T-1}, a_{T-1}, r_T, s_T. Then, back-propagation of all the time-steps occurs at once only when reaching a terminal state, using the usual update rule:

$$Q(s_t, a_t) \xleftarrow{learn} r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q_(s_{t+1}, a)$$

Even if this batch-learning fits algorithms like TILDE-RT, it does not seem that justifiable for fully incremental ones like RIB, but the RL algorithm used in [7, 3, 8, 9] has not been upgraded. Because of the learning process occurring at the end of episodes, this algorithm can not really be considered as Temporal Difference (TD). Surprisingly, many developments have been achieved in RRL [13, 14] but little has been done yet to perform basic but regular TD.

This TD issue has been recently addressed by [15] but this work supposes that an action model[3] is available so that only state values have to be learned. In this paper, we propose adaptations of regular TD algorithms in a relational framework, without an action model, and using RIB for the relational regression part. We first propose the algorithm 1 as a straightforward adaptation of online TD.

Algorithm 1 Online TD RRL algorithm: Sarsa-RIB (RIB-S)

Require: state and action spaces $\langle \mathcal{S}, \mathcal{A} \rangle$, RIB regression system for Q_{RIB}
Ensure: approximation of the action-value function Q_{RIB}
  initialise Q
  loop
    choose randomly start state s for episode
    $a \leftarrow \pi^{\tau}_{Q_{RIB}}(s)$ (Boltzmann softmax)
    repeat
      perform a; get r and s' in return
      $a' \leftarrow \pi^{\tau}_{Q_{RIB}}(s')$ (Boltzmann softmax)
      if s' is NOT terminal **then**
        $Q_{RIB}(s, a) \xleftarrow{learn} r + \gamma Q_{RIB}(s', a')$
      else
        $Q_{RIB}(s, a) \xleftarrow{learn} r$
      end if
      $s \leftarrow s'$; $a \leftarrow a'$
    until s terminal
  end loop

[3] The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$

In this algorithm, $Q_{RIB}(s,a)$ stands for the RIB prediction for the (s,a) pair. $\pi^{\tau}_{Q_{RIB}}$ means that the action is chosen according to a policy π derived from the action values Q_{RIB}. The action is selected according to a Boltzmann distribution with a temperature τ [4].

There is no learning rate here because the RIB regression algorithm does not make full replacement of values. When a new sample is provided, only a part of the error is corrected.

In being online, RIB-S does not learn the optimal values and relies on exploration/exploitation tradeoff to converge towards an optimal behaviour. In order to learn optimal values even if a non-greedy policy is used, it is necessary to design off-line techniques like Q-learning [16]. The algorithm 2 is an adaptation of regular Q-learning for relational representations.

Algorithm 2 Off-line TD RRL algorithm: Q-learning-RIB (RIB-Q)

Require: state and action spaces $\langle \mathcal{S}, \mathcal{A} \rangle$, RIB regression system for Q_{RIB}
Ensure: approximation of the action-value function Q_{RIB}
  initialise Q
  loop
    choose randomly start state s for episode
    repeat
      $a \leftarrow \pi^{\tau}_{Q_{RIB}}(s)$ (Boltzmann softmax)
      perform a; get r and s' in return
      if s' is NOT terminal **then**
        $Q_{RIB}(s,a) \xleftarrow{learn} r + \gamma \max_{a' \in \mathcal{A}(s')} Q_{RIB}(s',a')$
      else
        $Q_{RIB}(s,a) \xleftarrow{learn} r$
      end if
      $s \leftarrow s'$
    until s terminal
  end loop

With these new algorithms, there is no need anymore to keep complete trajectories in memory. In addition, the value function is modified at each time step. As a consequence, action selection improves along an episode. Although we expect little performance gain from a strict RL perspective, from an ILP point of view, these algorithms take full advantage of the incrementality of RIB. This method changes both the presented samples and their order of presentation to the regression algorithm, resulting in a different generalisation of the Q-value function.

[4] The probability of choosing action a in state s is $\frac{e^{\frac{Q_{RIB}(a)}{\tau}}}{\sum_{b \in \mathcal{A}(s)} e^{\frac{Q_{RIB}(b)}{\tau}}}$.

5 Experimental study

5.1 Problem setup

In this section, we compare experimentally RIB-S and RIB-Q to each other, and we compare them to the former RIB-RL [8]. The experimental study uses the most common environment in RRL: the blocks world already used as an illustrative problem in section 2.

A blocks world environment is composed of several blocks that can be stacked on each other, or on the floor. A block on top of a stack is said "clear". One constant is defined for each block of the problem, plus one for the floor. Two predicates are required to describe the states :

- $on/2$: $on(A, B)$ is true *iff* the block B is on right on top of the block A ;
- $clear/1$: $clear(A)$ is true *iff* there is no block on top of the block A

A special constant f representing the floor is used in conjunction with the predicate $on/2$ to indicate which blocks are lying on the floor: $on(A, f)$ is true *iff* A lies on the floor.

Actions are defined thanks to the predicate $move/2$: $move(A, B)$ is true if block A is moved on top of block B. The action is possible *iff* both A and B are clear. $\mathcal{A}(s)$ denotes set of the actions allowed in state s. It is computed by considering clear blocks only. Actions are resolved as expected by computing the resulting state.

A reward of 1 is given when the system meets the assigned goal, and 0 otherwise. Goal states are terminal states. Three usual goals are considered :

- *stacking*: all the blocks are stacked on top of each other ;
- *unstacking*: every block is on the floor. This problem is more difficult because the single goal state may be difficult to reach at early episodes without devoted strategies. This experiment is irrelevant here since we do not aim at providing such dedicated mechanism ;
- *on(a,b)*: one specific block a is on top of another one b. This problem is also difficult because the value action function is more "rugged", thus generalisation is more difficult.

Each algorithm (RIB-S, RIB-Q and RIB-RL) is tested for 20 trials, and results are averaged. The trials are divided into episodes, each starting in a random state and ending depending on the task to solve (*stacking* or *on(a,b)*). The Q-value function is periodically evaluated during a trial. For each evaluation, 10 episodes of greedy exploitation without learning are performed, each starting randomly. Thus, each average plot of the following figures involves 20×10 test episodes. The period between two evaluations is shorter in earlier episodes of a trial.

In every experiment, the discount factor γ is set to 0.9. The Boltzmann temperature τ used in softmax action selection starts with a value of $\tau_{start} = 5$, a decay parameter of 0.95 is used, but the temperature is never less than $\tau_{min} = 1$. Each episode is interrupted after N time steps, depending of the number of blocks

in the environment : $N = (n_{blocks} - 1) \times 3$. In RIB, a parameter M is used in prediction to upper-bound the difference between the Q-values of different states, knowing their distance. In our experiments, $M = 0.1$.

In [7, 3, 11, 8, 9], experimental results show the evolution of the total reward collected during episodes (like on figure 5). In the blocks world problem, since reward is only given when the goal is fullfiled, this measure is actually 1 if the system succeeds to reach a terminal state and 0 if it fails. Thus, it only shows the proportion of successes within the allotted time, but not the quality of the behaviour. In this paper, we rather show the evolution – as time steps increase – of the number of time steps required to terminate test episodes. When this measure equals N, it means that the system did not succeed at all to reach a terminal state in alloted time steps.

In [11], experimental results show how RIB-RL performs in 3, 4 and 5 blocks problems. We have run experiments with 5, 6, 7 and 9 blocks.

5.2 Experimental results for task *stacking*

Figure 3 compares the different algorithms facing the *on(a,b)* problem with 5, 6, 7 and 9 blocks.

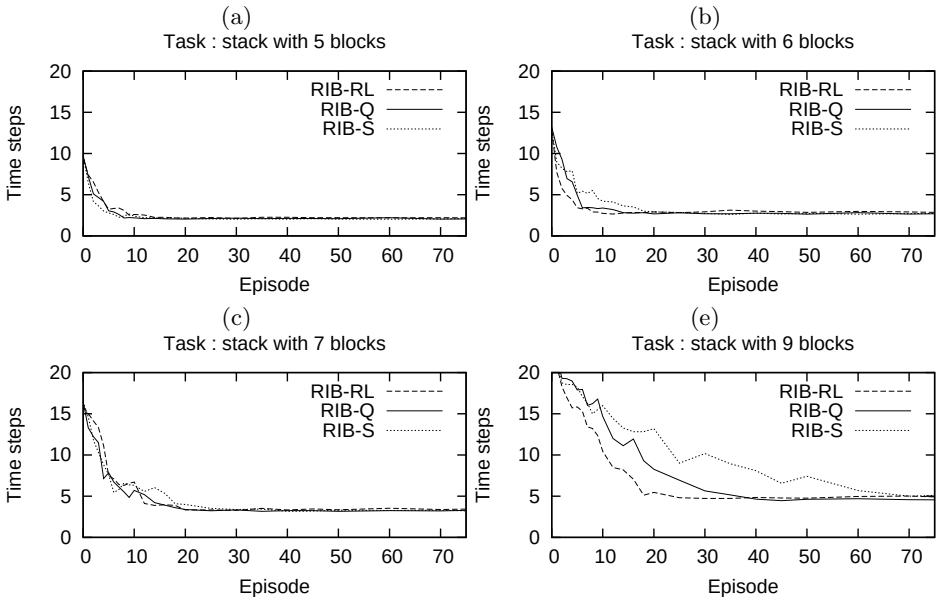

Fig. 3. Evolution of the number of time steps to complete an episode : (a) 5 blocks, (b) 6 blocks, (c) 7 blocks and (d) 9 blocks

Table 3 shows the average number of instances used by RIB after 1000 episodes (with standard deviation), indicating the complexity of the model obtained by each algorithm by each algorithm on the different problems.

stacking	5 blocks		6 blocks		7 blocks		9 blocks	
goal	Average	σ	Average	σ	Average	σ	Average	σ
RIB-RL	21	0	38	0	63	1.0	156	1.8
RIB-Q	19	0.2	32	0.6	51	1.9	94	3.6
RIB-S	19	0	32	0.5	50	1.3	88	4.8

Table 3. Number of prototypes used by RIB after 1000 episodes

5.3 Experimental results for task *on(a,b)*

Figure 4 compares the different algorithms facing the *stacking* problem with 5, 6, 7 and 9 blocks.

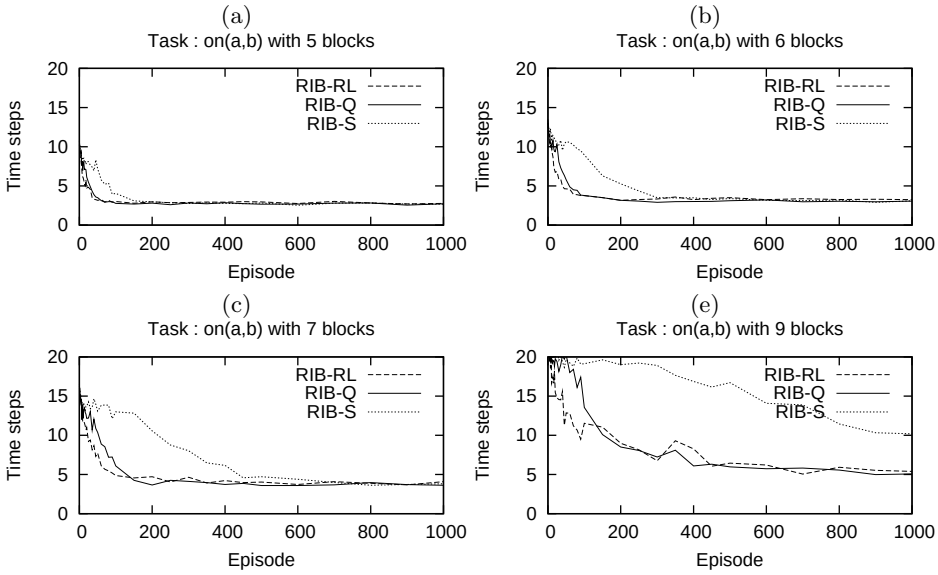

Fig. 4. Evolution of the number of time steps to complete an episode : (a) 5 blocks, (b) 6 blocks, (c) 7 blocks and (d) 9 blocks

Table 4 shows the average number of instances used by RIB after 1000 episodes (with standard deviation), indicating the complexity of the model obtained by each algorithm on the different problems. After 1000 episodes, RIB-S still doesn't reach an optimal behaviour, thus the 459±29.6 number of prototypes for RIB-S facing the 9 blocks $on(a, b)$ task is not representative.

$on(a,b)$	5 blocks		6 blocks		7 blocks		9 blocks	
goal	Average	σ	Average	σ	Average	σ	Average	σ
RIB-RL	325	3.0	757	18.4	1339	47.7	2593	87.1
RIB-Q	254	2.9	566	7.8	1063	18.2	2255	75.5
RIB-S	245	4.4	465	17.3	608	17.1	*459*	*29.6*

Table 4. Number of prototypes used by RIB after 1000 episodes

5.4 Discussion

The experimental results presented in sections 5.2 and 5.3 show that, as one might expect on such a tasks where exploration actions do not lead to catastrophic actions, the off-line TD algorithms (RIB-RL and RIB-Q) outperform slightly the online one (RIB-S). Moreover, RIB-Q does not differ that much from the original RIB-RL *wrt* to convergence speed. This means that RIB is robust enough to support samples provided one by one rather than in a batch way.

Thus, keeping a memory of only one single $(s_t, a_t, r_{t+1}, s_{t+1})$ tuple is enough, and there is no need for storing complete episode trajectories, as soon as one runs a fully incremental regression algorithm like RIB. Trajectories could be useful within other algorithms like Monte Carlo (MC) for instance, but in that case we would rather use the TD(λ) generalisation of TD and MC [1]. Most important is the level of performance reached by all presented RRL algorithms. Blocks world problems do not need very long sequences of actions to be solved, but they involve many states, especially as the number of blocks grows. This number of states has to be compared to the number of episodes required to solve the problem (for 9 blocks stacking: less than 50 episodes to learn about 4 596 553 states). These good results show at least two points:

- When a relational representation is available, it may be a good idea to use it in order to draw benefits from additional generalisation capacities.
- The distance used in RIB to discriminate between (state, action) pairs fits the problem well.

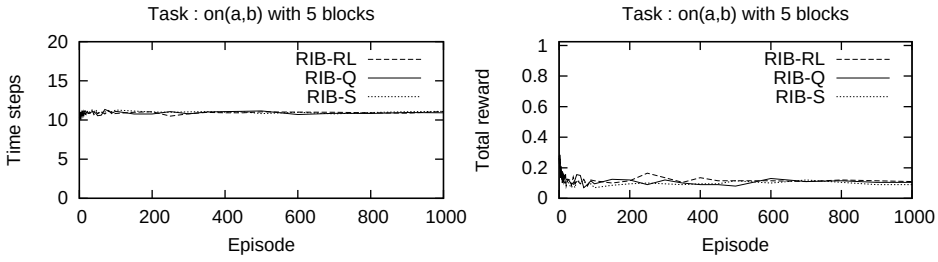

Fig. 5. Evolution of the number of time steps to complete an episode (task *on(a,b)* with 5 blocks, goal information not taken into account)

RIB stores prototypes in the form of specific (state, action) pairs ; generalisation only takes place through the distance computation. The distance used in RIB [11] is well suited for blocks world problems: it relies on a distance similar to the Levenshtein edit distance between sets of block stacks, seen as strings. The distance between (state, action) pairs is such that it is 0 between pairs differing only by a permutation of constants that do not occur in the action literal (figure 2).

This distance takes into account relationships between objects but not their specific identity, unless prior knowledge is given. It behaves very well for the *stacking* task where all blocks should be stacked regardless on their order, but it is less adapted when it is not the case – like the $on(a, b)$ problem. To make it work nevertheless, [11] proposes to provide the system with prior information about the constants involved in goal states. When this information is not taken into account, the system is not able to learn a correct behaviour, as shown on the experiments on figure 5.

Beyond similar convergence speed of RIB-Q and RIB-RL, tables 3 and 4 show the main difference between former and our new RRL algorithms : the complexity of the Q-function model. Indeed, for every task, RIB-Q achieves similar performance with a smaller number of prototypes than RIB-RL. In addition, RIB-S achieves even better generalisation despite a lower convergence speed. This is particularly obvious for our most complex $on(a, b)$ tasks. Even if the convergence speed of the online RIB-S is lower, it stores less prototypes and therefore it remains a good candidate for scaling-up. This reduced number of prototypes is due to changes in the order of presentation of the examples, our algorithms providing RIB with better opportunities for generalisation.

6 Conclusion

Our goal in this work was to investigate the gain of upgrading the RL algorithm for a fixed relational regression algorithm, namely RIB. We have proposed two simple RRL algorithms, RIB-Q and RIB-S, and have tested them on usual RRL benchmarks.

From a RL viewpoint, these are the first systems to date that implement regular and simple TD in a RRL context. Without storing full trajectories, we achieved similar convergence speed. From an ILP viewpoint, the experiments show the robustness of RIB in a fully incremental context. Less expected was that this new interaction between RL and relational regression yields a more compact model for the Q-function. We may expect that this will improve the scaling-up ability of RRL systems.

This work opens up several new research directions. We plan to adapt more sophisticated RL algorithms that will provide more useful information to the relational regression algorithms. We have observed that the RIB performance is highly dependant on the distance adequation with the problem. It might be interesting to study how RL may balance the effects of a misleading distance or even further, how RL may help in adapting the distance to the problem at hand.

Acknowledgements The authors would like to thank Kurt Driessens and Jan Ramon for very nicely and helpfully providing the authors with RIB-RL.

References

1. Sutton, R.S., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
2. Dzeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning **43** (2001) 7–52
3. Driessens, K., Ramon, J.: Relational instance based regression for relational reinforcement learning. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003). (2003) 123–130
4. Tesauro, G.J.: Temporal difference learning and td-gammon. Communications of the ACM **38**(3) (1995) 58–68
5. De Raedt, L., Džeroski, S.: First-order jk-clausal theories are PAC-learnable. Artificial Intelligence **70**(1–2) (1994) 375–392
6. Blockeel, H., de Raedt, L., Jacobs, N., Demoen, B.: Scaling up inductive logic programming by learning from interpretations. Data Mining and Knowledge Discovery **3**(1) (1999) 59–93
7. Driessens, K., Ramon, J., , Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In: Proceedings of the European Conference on Machine Learning (ECML 2001), LNAI vol 2167. (2001) 97–108
8. Driessens, K., Dzeroski, S.: Combining model-based and instance-based learning for first order regression. In: Proceedings of the 22nd International Conference on Machine Learning (ICML 2005). (2005) 193–200
9. Gartner, T., Driessens, K., Ramon, J.: Graph kernels and gaussian processes for relational reinforcement learning. In: Proceedings of the 13th Inductive Logic Programming International Conference (ILP 2003), LNCS vol 2835. (2003) 146–163
10. Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: Proceedings of the 15th International Conference on Machine Learning (ICML 1998). (1998) 55–63
11. Driessens, K.: Relational reinforcement learning. PhD thesis, K. U. Leuven (2004)
12. Ramon, J., Bruynooghe, M.: A polynomial time computable metric between point sets. Acta Informatica **37**(10) (2001) 765–780
13. Tadepalli, P., Givan, R., Driessens, K.: Relational reinforcement learning: An overview. In: Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning. (2004) 1–9
14. Van Otterlo, M.: A survey of reinforcement learning in relational domains. Technical report, Centre for Telematics and Information Technology, University of Twente, Enschede (2005)
15. Asgharbeygi, N., Stracuzzi, D., Langley, P.: Relational temporal difference learning. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML 2006). (2006) 49–56
16. Watkins, C.J.: Learning with delayed rewards. PhD thesis, Psychology Department, University of Cambridge, England (1989)