On and Off-Policy Relational Reinforcement Learning

Christophe Rodrigues, Pierre Gérard, and Céline Rouveirol

LIPN, UMR CNRS 7030, Institut Galilée - Université Paris-Nord
first.last@lipn.univ-paris13.fr

Abstract. 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 generalization 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.

In this paper, we study how even slight modifications in the RL algorithm employed may impact significantly on the performance of the relational regression system. In section 2, we briefly present the RL problem in the relational framework and we present three very similar RRL algorithms: the former Q-RRL [2], and two regular algorithms upgraded to relational representations: Q-learning (off-policy) and Sarsa (on-policy). We combine all these RL techniques with the same relational function approximator: RIB [3]. In section 3, we compare those algorithms experimentally and show a significant impact on RIB performance. Indeed, the size of the models learned by RIB decrease, resulting in an overall reduction of computation time.

2 Relational Temporal Difference

Relational Reinforcement Learning (RRL) addresses the development of RL algorithms operating on relational representations of states and actions.

The relational Reinforcement Learning task can be defined as follows. 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 nondeterministic)
– an unknown real-valued reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$,

the goal is to learn a policy for selecting actions $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes 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 $V : \mathcal{S} \to \mathbb{R}$ or a Q-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that $Q(s, a) \approx E\{R_t \mid s_t = s, a_t = a\}$.

Algorithm 1 Off-policy TD RRL algorithm: Qlearning-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}
  initialize 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

States are relational interpretations, as used in the "learning from interpretations" setting [4]. 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.

Among other incremental relational function approximators used in RRL [2, 5–7], the RIB system [3] is a quite good performance/efficiency compromise. It 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-neighbor algorithm. It takes advantage of a relational distance adapted to the problem to solve (see [8] for a distance for the blocks world problem). RIB handles incrementality since it forgets prototypes that are not necessary to reach a good prediction performance or have a bad prediction performance.

As opposed to regular Q-learning, in the RRL algorithm introduced in [2], learning occurs only at the end of episodes, and not at each time step. It 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:

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

In order to learn at each time step, we propose (algorithm 1) a regular adaptation of Q-learning to a relational framework and use RIB for the relational regression part.

Algorithm 2 On-policy 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}
  initialize 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

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 [1].

Q-learning is said off-policy because it learns an optimal Q-value function, even if it does not always choose optimal actions. With minor modifications, we propose (algorithm 2) an upgraded version of Sarsa [1], an on-policy algorithm which learns the Q-value function corresponding to the policy it actually follows.

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

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

the regression algorithm, resulting in a different generalization of the Q-value function.

3 Experimental study

The experiments are performed on the blocks world problem as described in [9]. 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.

In every experiment, the discount factor γ is set to 0.9. Each episode is interrupted after N time steps, depending of the number of blocks in the environment: $N = (n_{blocks} - 1) \times 3$. All other parameters are set according to [9].

Figure 1 compares the different algorithms facing the *on(a,b)* problem with 5 and 7 blocks. Table 1 shows the average number of instances used by RIB after 1000 episodes, indicating the complexity of the model obtained by each algorithm by each algorithm on the different problems.

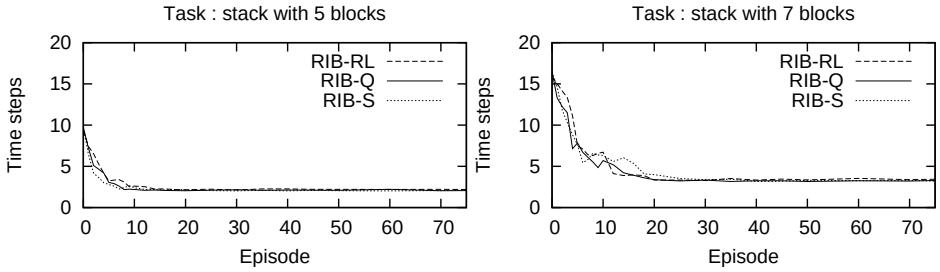

Fig. 1. Evolution of the number of time steps to complete an episode

stacking	5 blocks		6 blocks		7 blocks	
goal	Average	σ	Average	σ	Average	σ
RIB-RL	21	0	38	0	63	1.0
RIB-Q	19	0.2	32	0.6	51	1.9
RIB-S	19	0	32	0.5	50	1.3

Table 1. Number of prototypes used by RIB after 1000 episodes

Figure 2 compares the different algorithms facing the *stacking* problem with 5 and 7 blocks. Table 2 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.

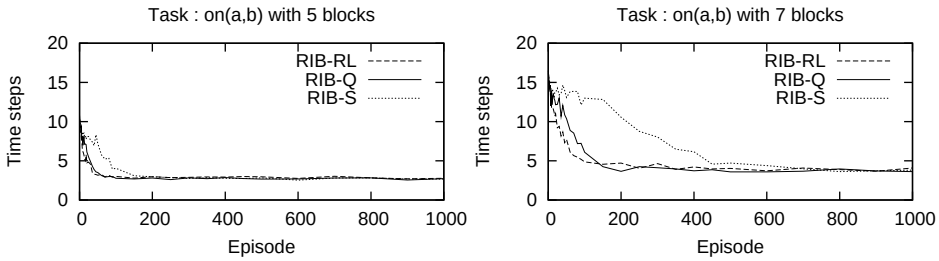

Fig. 2. Evolution of the number of time steps to complete an episode

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

Table 2. Number of prototypes used by RIB after 1000 episodes

The experimental results show that, as one might expect on such a task where exploration actions do not lead to catastrophic actions, the off-policy TD algorithms (RIB-RL and RIB-Q) outperform slightly the on-policy one (RIB-S). Moreover, RIB-Q does not differ that much from the original Q-RRL (here, RIB-RL), considering the convergence speed wrt the number of episodes. Most important is the level of performance (computation time) reached by all presented RRL algorithms. Our adaptations of Q-learning and Sarsa, namely RIB-Q and RIB-S, don't provide more examples to the regression system than the former Q-RRL. Thus, since RIB's learning is linear in the number of prototypes, and having less prototypes saves computation time. RIB-S (on-policy) learns less prototypes for these relatively simple tasks, it explores a smaller portion of the state space than off-policy algorithms due to its policy, and therefore needs less prototypes to reach a good predictive accuracy on those states. As a consequence, RIB-S outperforms RIB-Q and RIB-RL as far as computation time is concerned, demonstrating than despite its slower convergence speed wrt the number of episodes, Sarsa remains a good candidate for scaling-up.

RIB is instance-based and thus strongly relies on its distance: generalization only takes place through the distance computation during the k-nearest-neighbor prediction. The distance used in RIB [9] 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 equal to 0 for pairs differing only by a permutation of constants that do not occur in the action literal. This distance also takes into account bindings of variables occurring in the goal. Without this prior knowledge, the system cannot solve

problems like $on(a, b)$, where two specific blocks have to be stacked on each other.

4 Conclusion

We have observed that even small differences in the RL techniques significantly influence the behavior of a fixed relational regression algorithm, namely RIB. We have proposed two RRL algorithms, RIB-Q and RIB-S, and have tested them on usual RRL benchmarks, showing performance improvements.

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 already made experiments with relational TD(λ) with eligility traces, without noticing a significant improvement, neither on the number of prototypes nor on the computation time. A possible explanation is that the distance is to well fitted to the problem that eligibility traces are useless in that case. 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. De Raedt, L., Džeroski, S.: First-order jk-clausal theories are PAC-learnable. Artificial Intelligence **70**(1–2) (1994) 375–392
5. 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
6. 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
7. 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
8. Ramon, J., Bruynooghe, M.: A polynomial time computable metric between point sets. Acta Informatica **37**(10) (2001) 765–780
9. Driessens, K.: Relational reinforcement learning. PhD thesis, K. U. Leuven (2004)