Relational Instance Based Regression for Relational Reinforcement Learning

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

Abstract

Relational reinforcement learning (RRL) is a Q-learning technique which uses first order regression techniques to generalize the Q-function. Both the relational setting and the Q-learning context introduce a number of difficulties which must be dealt with. In this paper we investigate a few different methods that do incremental relational instance based regression and can be used for RRL. This leads us to different approaches which limit both memory consumption and processing times. We implemented a number of these approaches and experimentally evaluated and compared them to each other and an existing RRL algorithm. These experiments show relational instance based regression to work well and to add robustness to RRL.

1. Introduction

Q-learning (Watkins, 1989) is a model free approach to tackle reinforcement learning problems which calculates a Quality- or Q-function to represent the learned policy. The Q-function takes a state-action pair as input and outputs a real number which indicates the quality of that action in that state. The optimal action in a given state is the action with the highest Q-value.

The application possibilities of Q-learning is limited by the number of different state-action pairs that can occur. The number of these pairs grows exponentially in the number of attributes of the world and the possible actions and thus in the number of objects that exist in the world. This problem is usually solved by integrating some form of inductive regression technique into the Q-learning algorithm, which is able to generalize over state-action pairs. This generalized function is then able to make predictions about the Q-value of state-action pairs which it has never encountered.

One possible inductive algorithm that can be used for Q-learning is instance based regression. Instance based regression or nearest neighbor regression generalizes over seen examples by storing all or some of the seen examples and uses a similarity measure or distance between examples to make predictions about unseen examples. Instance based regression for Q-learning has been used by (Smart & Kaelbling, 2000) and (Forbes & Andre, 2002) with promising results.

Relational reinforcement learning (Džeroski et al., 1998; Driessens et al., 2001) is a Q-learning approach which incorporates a first order regression learner to generalize the Q-function. This makes Q-learning feasible in structured domains by enabling the use of objects, properties of objects and relations among objects in the description of the Q-function. Structural domains typically come with a very large state space, making it infeasible for regular Q-learning approaches to be used in them. Relational reinforcement learning (RRL) can handle relatively complex problems such as planning problems in a blocks world and learning to play computer games such as Digger and Tetris. However, we would like to include the robustness of instance based generalizations into RRL.

To apply instance based regression in the relational reinforcement learning context, a few problems have to be overcome. One of the most important problems deals with the number of examples that can be stored and used to make predictions. In the relational setting both the amount of memory to store examples and the computation time for the similarity measure between examples will be relatively large, so the amount of examples stored should be kept relatively small.

The rest of the paper is structured as follows. In Section 2 we give a small overview about related work on instance based regression and its use in Q-learning. Section 3 describes the relational Q-learning setting in which we will be using instance based regression. The considered approaches are then explained and tested in Section 4 and 5 respectively where we show that relational instance based regression works well as a generalization engine for RRL and that it leads to smoother learning curves as compared with the original decision tree approach to RRL. We conclude in Section 6.

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

2. Instance Based Regression

In this section we will discuss previous work on instance based regression and relate it to our setting.

Aha et al. introduced the concept of instance based learning for classification (Aha et al., 1991) through the use of stored examples and nearest neighbor techniques. They suggested two techniques to filter out unwanted examples to both limit the number of examples that are stored in memory and improve the behavior of instance based learning when confronted with noisy data. To limit the inflow of new examples into the database, the IB2 system only stores examples that are classified wrong by the examples in memory so far. To be able to deal with noise, the IB3 system removes examples from the database who's classification record (i.e. the ratio of correct and incorrect classification attempts) is significantly worse than that of other examples in the database. Although these filtering techniques are simple and effective for classification, they do not translate easily to regression.

The idea of instance based prediction of real-valued attributes was introduced by Kibler et.al. (Kibler et al., 1989). They describe an approach in which they use a form of local linear regression and although they refer to instance based classification methods for reducing the amount of storage space needed by the instance based techniques, they do not translate these techniques for real-value prediction tasks.

This idea of local linear regression is to greater detail explored in Atkeson et.al. (Atkeson et al., 1997), but again, no effort is made to limit the growth of the stored database. In follow-up work however (Schaal et al., 2000), they do describe a locally weighted learning algorithm that does not need to remember any data explicitly. Instead, the algorithm builds "*locally linear models*" which are updated with each new learning example. Each of these models is accompanied by a "*receptive field*" which represents the area in which this linear model can be used to make predictions. The algorithm also determines when to create a new receptive field and the associated linear model. Although we like this idea, building local linear models in our setting (where data can not be represented as a finite length vector) does not seem feasible.

An example where instance based regression is used in Q-learning is in the work of Smart and Kaelbling (Smart & Kaelbling, 2000) where they use locally weighted regression as a Q-function generalization technique for learning to control a real robot moving through a corridor. In this work, the authors do not look toward limiting the size of the example-set that is stored in memory. They focus on making safe predictions and accomplish this by constructing a convex hull around their data. Before making a prediction, they check whether the new example is inside this convex hull. The calculation of the convex hull again relies on the fact that the data can be represented as a vector, which is not the case in our setting.

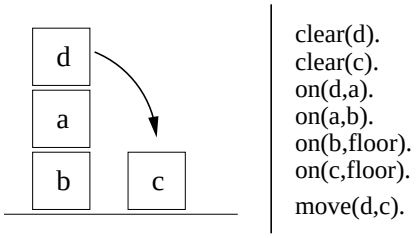

Figure 1. Notation example for state and action in the blocks world.

Another instance of Q-learning with the use of instance based learning is given by Forbes and Andre (Forbes & Andre, 2002) where Q-learning is used in the context of automated driving. In this work the authors do address the problem of large example-sets. They use two parameters that limit the inflow of examples into the database. First, a limit is placed on the density of stored examples. They overcome the necessity of forgetting old data in the Q-learning setting by updating the Q-value of stored examples according to the Q-value of similar new examples. Secondly, a limit is given on how accurately Q-values have to be predicted. If the Q-value of a new example is predicted within a given boundary, the new example is not stored. When the number of examples in the database reaches a specified number, the example contributing the least to the correct prediction of values is removed. We will adopt and expand on these ideas in this paper.

3. Relational Reinforcement Learning

Relational reinforcement learning or RRL (Džeroski et al., 1998) is a learning technique that combines Q-learning with relational representations for the states, actions and the resulting Q-function. The RRL-system learns through exploration of the state-space in a way that is very 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 datalog as a representation for encountered states and chosen actions. See Figure 1 for an example of this notation in the blocks world.

To build the generalized Q-function, RRL applies a first order logic incremental regression engine to the constructed example set. The resulting Q-function is then used to generate further episodes and updated by the new experiences that result from these episodes.

Regression algorithms for RRL need to cope with the Q-learning setting. Generalizing over examples to predict a real (and continuous) value is already much harder then doing regular classification, but the properties of Q-learning present the generalization engine with its own difficulties. For example, the regression algorithm needs to be incremental to deal with the almost continuous inflow of new (and probably more correct) examples that are presented to the generalization engine. Also, the algorithm needs to be able to do "*moving target regression*", i.e. deal with learning a function through examples which, at least in the beginning of learning, have a high probability of supplying the wrong function-value.

The relational setting we work in imposes its own 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 neglect-able. This, together with the larger memory requirements of datalog compared to less expressive data formats, force us to limit the number of examples that are stored in memory.

Also, a lot of existing instance based methods, especially for regression, rely on the fact that the examples are represented as a vector of numerical values, i.e. that the problem space can be represented as a vector space. Since we do not want to limit the applicability of our methods to that kind of problems, we will not be able to rely on techniques such as local linear models, instance averaging or convex hull building. Our use of datalog or herbrand interpretations to represent the state-space and actions allows us — in theory — to deal with worlds with infinite dimensions. In practice, it allows us to exploit relational properties of states and actions when describing both the Q-function and the related policies at the cost of having little more than a (relational) distance for calculation purposes.

4. Relational Instance Based Regression

In this section we will describe a number of different techniques which can be used with relational instance based regression to limit the number of examples stored in memory. As stated before, none of these techniques will require the use of vector representations. Some of these techniques are designed specifically to work well with Q-learning.

We will use c-nearest-neighbor prediction as a regression technique, i.e. the predicted Q-value $\hat{q}_i$ will be calculated as follows:

$$\hat{q}_i = \frac{\sum_j \frac{q_j}{dist_{ij}}}{\sum_j \frac{1}{dist_{ij}}} \quad (1)$$

where $dist_{ij}$ is the distance between example i and example j and the sum is calulated over all examples stored in memory. To prevent division by 0, a small amount δ can be added to this distance.

4.1. Limiting the inflow

In IB2 (Aha et al., 1991) the inflow of new examples into the database is limited by only storing examples that are classified wrong by the examples already stored in the database. However, when predicting a continuous value, one can not expect to predict a value correctly very often. A certain margin for error in the predicted value will have to be tolerated. Comparable techniques used in regression context (Forbes & Andre, 2002) allow an absolute error when making predictions as well as limit the density of the examples stored in the database.

We try to translate the idea of IB2 towards regression in a more adaptive manor. Instead of adopting an absolute error-margin we propose to use an error-margin which is proportional to the standard deviation of the values of the examples closest to the new example. This will make the regression engine more robust against large variations in the values that need to be predicted. So, examples will be stored if

$$|q - \hat{q}| > \sigma_{local} \cdot F_l \quad (2)$$

with q the real Q-value of the new example, $\hat{q}$ the prediction of the Q-value by the stored examples, σ_{local} the standard deviation of the Q-value of a representative set of the closest examples (we will use the 30 closest examples) and F_l a suitable parameter.

We also like the idea of limiting the number of examples which occupy the same region of the example space, but dislike the rigidity that a global maximum density imposes. Equation 2 will limit the number of examples stored in a certain area. However, when trying to approximate a function such as the one shown in Figure 2, it seems natural to store more examples of region A than region B in the database. Unfortunately, region A will yield a large σ_{local} in Equation 2 and will not cause the algorithm to store as many examples as we would like.

We will therefore adopt an extra strategy that stores examples in the database until the local standard-deviation (i.e. of the 30 closest examples) is only a

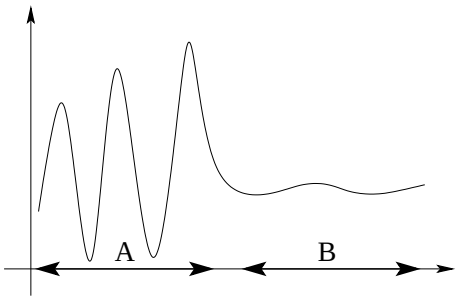

Figure 2. To predict the shown function correctly, an instance based learner should store more examples form area A than area B.

fraction of the standard deviation of the entire database, i.e. an example will be stored if

$$\sigma_{local} > \frac{\sigma_{global}}{F_g} \tag{3}$$

with σ_{local} the standard deviation of the Q-value of the 30 closest examples, σ_{global} the standard deviation of the Q-value of all stored examples and F_g a suitable parameter.

This will result in more stored examples in areas with large variance of the function value and less in areas with small variance. An example will be stored by the RRL-system if it meats one of the two criteria.

Both Equation 2 and Equation 3 can be tuned by varying the parameters F_l and F_g.

4.2. Throwing away stored examples

The techniques described in the previous section might not be enough to limit the growth of the database sufficiently. When memory limitations are reached, or when calculation times grow too large, one might have to place a hard limit on the number of examples that can be stored. The algorithm then has to decide which examples it can remove from the database.

IB3 uses a classification record for each stored example and removes the examples that perform worse than others. In IB3, this removal of examples is added to allow the instance based learner to deal with noise in the training data. Because Q-learning has to deal with moving target regression and therefore inevitably with noisy data, we will probably benefit from a similar strategy in our regression techniques. However, because we are dealing with continuous values, keeping a classification record which lists the number of correct and incorrect classifications is not feasible.

We suggest two separate scores that can be calculated for each example that will indicate which example we will remove from the database.

4.2.1. Error Contribution

Since we are in fact trying to minimize the prediction error, we can calculate for each example what the cumulative prediction error is with and without the example. The resulting score for example i looks as follows:

$$Score_i = (q_i - \hat{q}_i)^2 + \frac{1}{N}\sum_j [(q_j - \hat{q}_j^{-i})^2 - (q_j - \hat{q}_j)^2] \tag{4}$$

with N the number of examples in the database, $\hat{q}_j$ the prediction of the Q-value of example j by the database and $\hat{q}_j^{-i}$ the prediction of the Q-value of example j by the database without example i. The lowest scoring example is the example that should be removed.

4.2.2. Error Proximity

A more simple score to calculate is based on the proximity of examples in the database that are predicted with large errors. Since the influence of stored examples is inversely proportional to the distance, it makes sense to presume that examples which are close to the examples with large prediction errors are also causing these errors. The score for example i can be calculated as:

$$Score_i = \sum_j \frac{|q_j - \hat{q}_j|}{dist_{ij}} \tag{5}$$

where $\hat{q}_j$ is the prediction of the Q-value of example j by the database and $dist_{ij}$ the distance between example i and example j. In this case, the example with the highest score is the one that should be removed.

Another scoring function is used by (Forbes & Andre, 2002). In this work, the authors also suggest not just throwing out examples, but use instance-averaging instead. This is not possible using datalog representations and therefore is not used in our system.

4.3. Q-learning specific strategies: Maximum Variance

The major problem we encounter while using instance based learning for regression is that it is impossible to distinguish high function variation from actual noise. It seems impossible to do this without prior knowledge about the behavior of the function that we are trying to approximate. If one could pose a limit on the variation of the function to be learned, this limit might allow us to distinguish at least part of the noise from function variation. For example in Q-learning, one could know that

$$\frac{|q_i - q_j|}{dist_{ij}} < M \tag{6}$$

or one could have some other bound that limits the difference in Q-value in function of the distance between the examples.

Since we are using our instance based regression algorithm in a Q-learning setting, we can try to exploit some properties of this setting to our advantage. In a deterministic application and with the correct initialization of the Q-values (i.e. to values that are underestimations of the correct Q-value), the Q-values of tabled Q-learning follow a monotonically increasing path during calculation. This means that the values in the Q-table will always be an underestimation of the real Q-values.

When Q-learning is performed with the help of a generalization technique, this behavior will normally disappear. The Q-value of a new example is normally given by

$$Q(s,a) = R + max_{a'}\hat{Q}(s',a') \tag{7}$$

where s' is the state that is reached from performing action a in state s and $\hat{Q}(s',a')$ is the estimation of the Q-value of the state-action pair (s',a'). This estimation however, when done by Equation 1 might not be an underestimation.

By using the following formula for Q-value prediction

$$\hat{q}_i = \frac{\sum_j \frac{(q_j - (M \cdot dist_{ij}))}{dist_{ij}}}{\sum_j \frac{1}{dist_{ij}}} \tag{8}$$

where M is the same constant as the M in Equation 6. we ensure a generalization which is an underestimate.

With all the Q-value predictions being underestimations, we can use Equation 6 to eliminate examples from our database. Figure 3 shows the forbidden regions that result from the combination of the domain knowledge represented by the maximum derivative and the Q-learning property of having underestimations of the values to be predicted. We use these forbidden regions to eliminate examples from our database. In the example of Figure 3 this would mean that we can remove examples b and f. Example d will stay in the database.

The applicability of this approach is not limited to deterministic environments only. Since the algorithm will calculate the highest possible Q-value for each example it can also be used in stochastic environments where actions have a chance of failing. If accidental results of actions are of lesser quality than the normal results, the algorithm will still find the optimal strategy. If actions can have better than normal results, this approach can not be used.

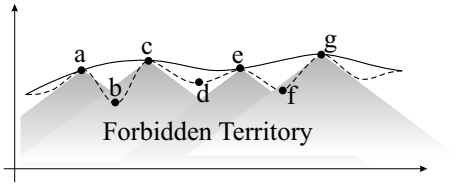

Figure 3. Using Maximum Variance to select Examples.

5. Experiments

In this section we describe the tests we ran to compare the database management approaches and also compare instance-based RRL to tree-induction-based RRL in a blocks world learning task.

5.1. A simple task

To test the different database management approaches we suggested, we devised a very simple (non-relational) Q-learning task. We let an agent walk through the corridor shown in Figure 4. The agents starts on one end of the corridor and receives a reward of 1.0 when he reaches the other end. The distance between two state-action-pairs that was used, is related to the number of steps it takes to get from one state to the other, slightly increased if the chosen actions differ.

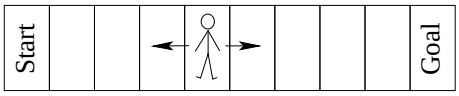

Figure 4. The corridor application.

The Q-function related to this problem is a very simple, monotonically increasing function, so that it only takes two (well chosen) examples for the Q-learner to learn the optimal policy. This being the case, we chose to compare the average prediction-error on all state-action-pairs for the different suggested approaches.

5.1.1. Inflow Behavior

To test the two inflow-filters of section 4.1 we ran several experiments varying the F_l and F_g values separately. Figure 5 shows the average prediction errors over 50 test trials. Figure 6 shows the corresponding database sizes.

The influence of F_g is exactly what one would expect. A larger value for F_g forces the algorithm to store more examples but lowers the average prediction error. It is worth noticing that in this application the influence on the size of the database and therefore on the calculation time is quite large with respect to the relatively

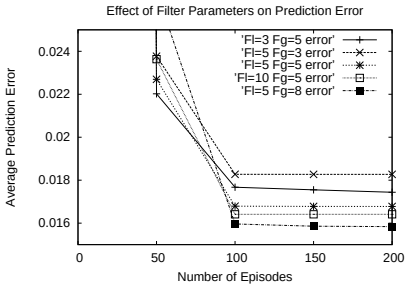

Figure 5. Prediction errors for varying inflow limitations.

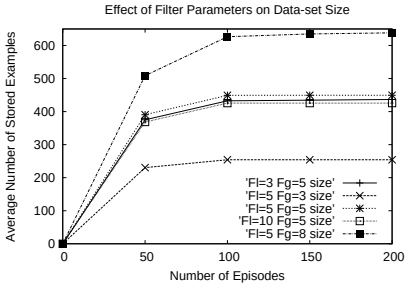

Figure 6. Database sizes for varying inflow limitations.

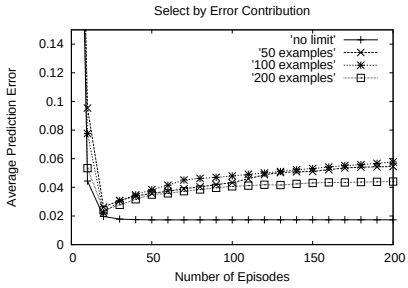

Figure 7. The effect of selection by Error Contribution.

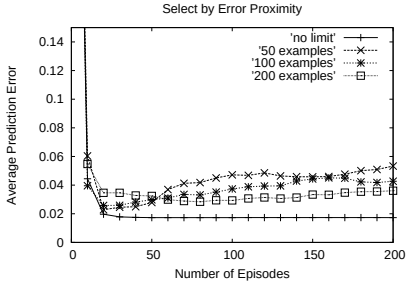

Figure 8. The effect of selection by Error Proximity.

small effect this has on the prediction errors.

The influence of F_l is not so predictable. First of all, the influence of this parameter on the size of the database seems limited to say the least. Second, one would expect that an increase of the value of F_l would cause an increase in the prediction error as well. Although the differences we measured were not significant enough to make any claims, this does not seem to be the case.

5.1.2. Adding an upper limit

We now test the two scoring functions from section 4.2 by adding an upper limit to the database size that RRL is allowed to use. We set the two parameters F_l and F_g to 5.0 — values that gave both average prediction errors and average database size — and varied the number of examples that RRL could store to make predictions.

Figure 7 shows the average prediction-error as a function of the number of learning episodes when using the error-contribution-score (ec-score) of Equation 4 for different maximum database sizes. The '*no limit*' curve in the graph shows the prediction error when no examples are removed.

In Figure 8 we show the average prediction-error when managing the database size with the error-proximity-score (ep-score) of Equation 5. Although differences with the ec-score are small, ep-score management performs at least as well and is easier to calculate.

5.1.3. The effects of Maximum Variance

Figure 9 shows the prediction-error when the maximum variance (or mv) strategy is used to manage the database. The prediction errors are a lot larger than with the other strategies, but RRL is still able to find the optimal strategy. The advantage of the mv-strategy lies in the number of examples stored in the database. With this particular application, only 20 examples are stored, one for each possible Q-value.

5.2. The Blocks World

To compare the new instance-based RRL with tree-induction-based RRL (RRL-Tg) we ran experiments in the blocks world with a variable number of blocks.

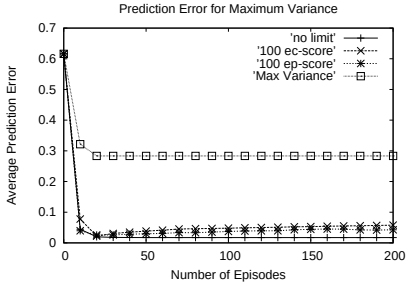

Figure 9. The effect of selection by Maximum Variance.

RRL-TG (Driessens et al., 2001) uses an incremental first-order regression tree algorithm as the Q-function approximation technique. We compared its performance to the algorithm that uses the error-proximity-score to remove examples and to the approach that uses the maximum variance to limit the examples stored in the database.

To train RRL we let it experiment in worlds which contain 3 to 5 blocks and allow it to ask for guidance as described in earlier work (Driessens & Džeroski, 2002a; Driessens & Džeroski, 2002b) in a world with 10 blocks. This guidance is provided in 10% of the training-episodes.

We test RRL on three different goals in the blocks world: stacking, unstacking and putting one specific block on top of another. In the *stack*-goal RRL receives a reward of 1.0 if it puts all the blocks in one stack in the minimum number of steps and 0.0 otherwise. In the *unstack*-goal, similar rewards are given when RRL succeeds in putting all the blocks on the floor in the minimum number of steps. The rewards for the *on(A,B)*-goal also behave similarly, but the specific blocks to be stacked can be changed in each learning episode.

To be able to use instance-based learning in the blocks world we need a distance defined on our representation of the blocks world. (See Figure 1). We define our distance as follows:

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 based on the distance between the stacks of blocks (Ramon & Bruynooghe, 2001).

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 & Fischer, 1974) between the resulting strings.

While this procedure defines a generic distance, it will adopt itself to deal with different goals as well as different numbers of blocks in the world. The renaming step (Step 1) even allows instance-based RRL to train on similar goals which refer to different specific blocks. This is comparable to RRL-TG which uses variables to represent blocks which appear in the action and goal description. Blocks which do not appear in the action or goal description are all regarded as generic blocks, i.e. without paying attention to the specific identity of these blocks.

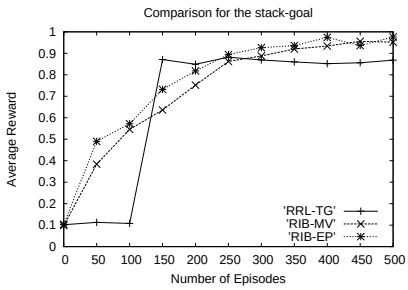

Figure 10. Comparison between RRL-TG and RRL-RIB for the stack-goal in the blocks world.

In the graphs we will refer to the algorithm that uses the error-proximity-score as RIB-EP and to the approach that uses the maximum variance as RIB-MV. Figure 10 shows the results for the *stack*-goal. We allowed the error-proximity approach to store 500 examples, a number it reaches after approximately 300 episodes. The graph shows that both instance-based policies outperform RRL-TG. It also shows that the learning progression is smoother than for RRL-TG. RRL-TG relies on finding the correct split for each node in the regression tree. When this node is found, this results in large improvements in the learned policy. Instance-based RRL does not rely on such key-decisions and therefore can be expected to be more robust than RRL-TG.

Figure 11 and 12 show the results for the *unstack*-goal and *on(A,B)*-goal respectively. It should be noted that for both tasks, the error-proximity based algorithm did not reach the 3000 examples we allowed it to store in its database and therefore did not remove any examples. Both graphs show that instance-based RRL clearly outperforms RRL-TG. RRL with instance based predictions is able to learn almost perfect behavior in worlds which are related to its training environment. RRL-TG never succeeded in this without the use of explicit policy learning (P-learning) (Džeroski et al., 1998; Driessens et al., 2001).

6. Conclusions

In this work, we introduced relational instance based regression, a new regression technique that can be used

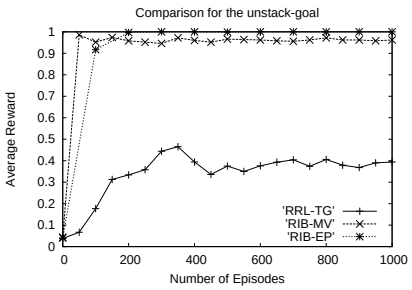

Figure 11. Comparison between RRL-TG and RRL-RIB for the unstack-goal in the blocks world.

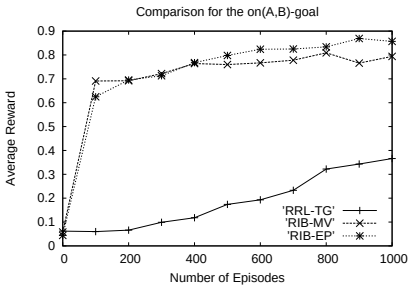

Figure 12. Comparison between RRL-TG and RRL-RIB for the on(A,B)-goal in the blocks world.

when instances can not be represented as vectors. We integrated this regression technique into relational reinforcement learning and thereby added the robustness of instance based generalizations to RRL.

Several database management approaches were developed to limit the memory requirements and computation times by limiting the number of examples that need to be stored in the database. We showed and compared the behavior of these different approaches in a simple example application and compared the behavior of instance-based RRL with another RRL-algorithm (RRL-TG) which uses a regression tree for Q-function generalization. Empirical results clearly show that instance-based RRL outperforms RRL-TG.

Acknowledgments

Jan Ramon is a post-doctoral fellow of the Katholieke Universiteit Leuven.

References

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

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, *11*, 11–73.

Driessens, K., & Džeroski, S. (2002a). Integrating experimentation and guidance in relational reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 115–122). Morgan Kaufmann Publishers, Inc.

Driessens, K., & Džeroski, S. (2002b). On using guidance in relational reinforcement learning. *Proceedings of Twelfth Belgian-Dutch Conference on Machine Learning* (pp. 31–38). Technical report UU-CS-2002-046.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *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. *Proceedings of the 15th International Conference on Machine Learning* (pp. 136–143). Morgan Kaufmann.

Forbes, J., & Andre, D. (2002). Representations for learning control policies. *Proceedings of the ICML-2002 Workshop on Development of Representations* (pp. 7–14). The University of New South Wales, Sydney.

Kibler, D., Aha, D. W., & Albert, M. (1989). Instance-based prediction of real-valued attributes. *Computational Intelligence*, *5*, 51–57.

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

Schaal, S., Atkeson, C. G., & Vijayakumar, S. (2000). Real-time robot learning with locally weighted statistical learning. *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 288–293). IEEE Press, Piscataway, N.J.

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

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

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