Combining Model-Based and Instance-Based Learning for First Order Regression

Kurt Driessens KURTD@WAIKATO.AC.NZ
Department of Computer Science, University of Waikato, Hamilton, New Zealand

Sašo Džeroski SASO.DZEROSKI@IJS.SI
Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

Abstract

The introduction of relational reinforcement learning and the RRL algorithm gave rise to the development of several first order regression algorithms. So far, these algorithms have employed either a model-based approach or an instance-based approach. As a consequence, they suffer from the typical drawbacks of model-based learning such as coarse function approximation or those of lazy learning such as high computational intensity.

In this paper we develop a new regression algorithm that combines the strong points of both approaches and tries to avoid the normally inherent draw-backs. By combining model-based and instance-based learning, we produce an incremental first order regression algorithm that is both computationally efficient and produces better predictions earlier in the learning experiment.

1. Introduction

With the development of relational reinforcement learning (Džeroski et al., 1998; Džeroski et al., 2001) came the need for incremental relational regression algorithms. A relational regression algorithm generalizes over learning examples with a continuous target value and makes predictions about the value of unseen examples, using a relational representation for both the learning examples and the resulting function. A number of these algorithms have already been developed. The TG algorithm builds regression trees (Driessens et al., 2001), RIB uses instance based regression with first order distances (Driessens & Ramon, 2003) and KBR relies on Gaussian processes and kernels for structured data as a similarity measure to make predictions (Gärtner et al., 2003). These algorithms, when compared by how they approach the learning task and how they make predictions, occupy opposite ends of a spectrum.

On the one hand, there is the model-based TG algorithm. Model-based algorithms represent the processed learning examples in an explicit generalizing theory. When used in an incremental setting, they usually discard learning examples when they are processed and as a consequence are often efficient both computationally and in the amount of memory that they use. The symbolic approach used in first order versions of these algorithms often allows a certain degree of human interpretation of the learning results. One of their drawbacks is that the constructed model is often limited by the used learning bias and, when used for regression, often makes a coarse approximation of the target function. This behavior surfaces more frequently when dealing with incremental data, as the learned model is often very basic at the start of learning so that small changes in the model can cause large differences in predictive accuracy.

On the other hand, instance based or lazy learning algorithms such as RIB or KBR represent their collected knowledge as a set of examples. In the simplest case, this could be just the set of learning examples, but more complex approaches that calculate "prototypes" by combining or selecting learning examples can also be used. They rely on a similarity measure between examples to make predictions about the target value of new examples. Instance based techniques are better suited for numerical predictions. They offer a finer degree of approximation with less amounts of learning data, so their predictive behavior is often less erratic when dealing with incremental data. They also suffer less from over-generalization than model-based techniques (Driessens & Džeroski, 2004). However, they do suffer computationally when a lot of learning exam-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

ples need to be processed. Although some techniques have been introduced that speed up the discovery of nearest neighbors — such as kd-trees and ball-trees (Moore, 1991; Freidman et al., 1977) — these methods were designed for the propositional setting and it is not immediately obvious how to translate them to the relational case.

In this paper, we develop a new, hybrid regression algorithm, using ideas from some of the algorithms mentioned above. The goal is to combine the advantages of the different approaches while trying to avoid the draw-backs that usually accompany them. The new algorithm should inherit the computational efficiency of model based techniques, and by dividing the example space into smaller parts, circumvent the computational complexity of lazy learning techniques at least partially. By incorporating instance based prediction, we also will try to avoid the jagged, step-like improvements and bias dependencies of model-based techniques and incorporate the robustness of instance based techniques in incremental settings.

The rest of this paper is structured as follows. Section 2 discusses related work that sets out to combine model-based and lazy learning, although not necessarily in a regression or first order setting. It also discusses the TG and RIB algorithms as parts of these will be used in the construction of the new algorithm. In section 3, the new algorithm TRENDI is presented. Section 4 describes the experimental setup of using the RRL algorithm in a blocks world setting to compare the performance of the new algorithm with previously developed first order regression algorithms and Section 5 shows and discusses the results of these experiments. Section 6 concludes and discusses some further work.

2. Related Work

2.1. Combining Model-Based and Instance-Based Learning

Many other techniques for combining model-based and instance-based learning have already been proposed. However, most of these have been based on propositional representations of the learning examples.

The RISE algorithm (Domingos, 1996) combines rule-based and instance-based induction by regarding rules as generalized instances and using an adapted distance measure between rules and instances. The algorithm starts with the full set of learning examples and tries to generalize examples (and rules) into more general rules by finding the closest not covered example and generating the most specific generalization that does cover the chosen example. These generalizations are accepted as long as they don't cause a decrease in predictive accuracy[1]. Duplicate rules are removed from the knowledge base.

Quinlan (1993) developed a technique of combining instance-based learning and model-based learning that is independent from the type of model learned. Instead of using the average of the stored class values of the nearest neighbors for predicting values of unseen examples, he proposes to compute the average of adjusted values based on the difference between the new example and its nearest neighbors as predicted by the learned model.

Although we have mentioned prototyping as part of instance-based learning, the prototype construction itself can be regarded as a model-based technique. Depending on the strategy used to compute the prototypes, prototypes are not restricted to being one of the initial learning examples (Aha et al., 1991).

Salzberg (1991) describes a technique based on *nested generalized exemplars* that uses hyper-rectangles defined by two examples (comparable to upper and lower bound) in a learning techniques that closely resembles instance-based learning. For new examples, the class of the smallest hyper-rectangle that contains the new example is predicted. If the new example is not within any hyper-rectangle, the algorithm predicts the class of the closest generalized exemplar.

Locally weighted learning is another learning techniques that combines building models with ideas from instance based approaches (Atkeson et al., 1997). In locally weighted learning, a distance measure is used to restrict the span of learning examples over which a model is built. To cover the entire example space, a set of several models is used.

2.2. Related First Order Regression Algorithms

Since parts of the TG and RIB algorithms will be used as parts of the new TRENDI algorithm, we will discuss them in more detail than the other related approaches. The TG algorithm (Driessens et al., 2001) incrementally builds first order regression trees. It uses a user-defined refinement operator that originated in the TILDE system (Blockeel & De Raedt, 1998). This refinement operator uses a "language bias" to specify which first order predicates can be used as possible tests in internal nodes of the regression tree. This language bias is application dependent and has to be defined by the user of the system. The selection of the

[1]The RISE system estimates predictive accuracy using the leave-one-out procedure.

tests is based on a number of statistics that are incrementally calculated for each leaf of the tree. These statistics indicate when and how to split a leaf and transform it into an internal node. Each leaf of the tree predicts the same value for each of the examples that is assigned to that leaf.

The RIB algorithm (Driessens & Ramon, 2003) does nearest neighbor prediction using a relational distance measure:

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

where $\hat{q}_i$ is the predicted value for example i, q_j is the stored target value of example j, $dist_{ij}$ is the distance between example i and example j and the sum is computed over all examples stored in memory. To prevent division by 0, a small amount δ is added to this distance. Several methods of selecting which examples to store and which to delete from the data-base are used. The one that will be adopted for use in our hybrid approach, employs a user defined "maximum variance" parameter M, for which the following holds:

$$\frac{|q_i - q_j|}{dist_{ij}} < M \quad (2)$$

This equation allows the algorithm (given certain conditions that cause all predictions made by the RIB algorithm to be guaranteed underestimates) to eliminate examples from the data-base. While this parameter is not always straightforward to derive for a given application, for planning problems such as those encountered in the blocks world, it is usually defined as the maximum difference between two different Q-values in the problem[2]. On the other hand, the RIB algorithm seems quite stable for slight variations of the value of this parameter.

3. Combining Trees and Instances

The algorithm we introduce uses a combination of the previously discussed TG and RIB algorithms. At a high level, it uses a TG like algorithm to divide the example space into regions and uses instance based predictions in each of the sub-spaces. The algorithm builds a first order decision tree incrementally and stores a copy of the RIB algorithm at each of its leafs. Each instance of RIB uses the "maximum variance" example selection method to control which learning examples are stored and which are discarded or removed. Since the new algorithm uses both TREes aND Instances to make predictions, we will refer to it as TRENDI.

The TG algorithm predicts the same value for each example that is classified to a leaf. For each leaf and for each test that can be used to further split the examples in that leaf, it keeps statistics about the target values of encountered examples and uses these to derive when that leaf should be split. A standard F-test is used to decide whether the target values of the examples that are classified positively and negatively by a certain test differ significantly. If such a test is found, the leaf is split and two new leafs are created. It should also be noted that TG uses a "minimal-sample-size" parameter that forces the algorithm to collect a minimal amount of data in a leaf before it can be split.

Since the predictions made by our algorithm in a single leaf are not constant but made by an instance based algorithm, we should not use the same splitting criterion as TG. Instead, we want to split a leaf when we discover that two instances of the RIB algorithm can make better predictions in their respective sub-spaces than a single RIB instance can for all the examples sorted to that leaf.

We approximate predictive accuracies based on the examples currently stored by the RIB algorithm in the leaf under consideration using leave-one-out predictions. By summing the prediction error made by the single RIB instance and the pair of RIB instances built according to each possible split over all examples stored in the leaf, we get an indication[3] of the improvement possible for each possible split. When the improvement is large enough, we use the selected split to create an internal node and add two new leafs to the tree. In each of the leafs, we build a new instance of the RIB algorithm using the corresponding subset of learning examples from the original RIB. Note that this also differs from the original TG algorithm, where new leafs are started with reset statistics, so that every split in TG causes a loss of learning experience. This was done in TG to ensure example turnover, but is unnecessary in the TRENDI algorithm because it uses the example selection strategy of the RIB algorithm. Algorithm 1 shows this approach in pseudo-code.

As the approach uses both the language bias needed for TG and the first order distance measure used by RIB, the TRENDI algorithm demands more user specification than any of the two approaches on their own. However, we feel that the improved performance of the technique (see Section 4) justifies these added require-

[2]In planning problems, a reward R is often only presented to the learning agent at the completion of the task. Using the discounted total reward as a value function causes the maximum difference between two different Q-values to be $(1 - \gamma)R$.

[3]In our current implementation of the algorithm, we use the sum of the errors as stated. Another possibility would be to compare the sum of the squared errors.

Algorithm 1 The TRENDI algorithm.

```
initialize by creating a tree with a single leaf with
  an empty instance of the RIB algorithm
for (each learning example that becomes available)
do
  sort the example down the tree using the tests of
    the internal nodes until it reaches a leaf
  update the instance of the RIB algorithm in the
    leaf according to the new example
  compare the predictive accuracy of the single RIB
    with that of the two RIB instances resulting
    from each of the possible test-extensions
  if (an extension results in better predictions)
  then
    generate a new internal node using the
      indicated test
    grow two new leafs and divide the examples
      from the original leaf over the new RIB
      instances corresponding to the chosen test
  end if
end for
```

ments.

The algorithm as described, has a number of parameters that can be tuned to deal with different environments. The first parameter is comparable to the "minimal-sample-size" parameter used by the TG algorithm. Similarly, we can introduce a "minimal-sample-size" parameter that specifies how many examples need to be stored in the local RIB instance, before we allow the algorithm to split a leaf.

Since we use the RIB algorithm without change, we can also use the "maximum variance" parameter M. Where the RIB algorithm tries to eliminate as many learning examples as possible to keep computing complexity at a minimum, we might be able to use the extra knowledge stored in these examples to decide on which splits to use. There will be a strong interaction between the values of the "minimal-sample-size" parameter and the "maximum variance" parameter, as the last one will try to limit the growth of the "sample-size" used by RIB in each leaf.

The last parameter emerges from the idea that we want predictions to improve when we split a leaf. Requiring an improvement when we split a leaf, will make the algorithm prefer smaller models, given equal performance. A parameter b $(0 < b \leq 1)$ can be used to specify how much of an improvement is needed before a leaf is split.

$$error_{\text{after}} < b \cdot error_{\text{before}} \tag{3}$$

4. Experimental Setup

To compare TRENDI with other available first-order regression algorithms, we decided to use the RRL algorithm as described by Driessens et al. (2001) and the Blocks World as a test-case. The blocks world is used with a varying number of blocks. During training, the number of blocks is varied between 3 and 5, and the resulting strategies are tested in worlds with 3-10 blocks. We also supply the algorithm with 5 guided traces in a world with 10 blocks every 50 learning episodes as specified by Driessens and Džeroski (2004).

We ran tests with the "Stack"-goal, i.e., trying to build one large stack of blocks, with no specific ordering of the blocks required, and the "On(A,B)"-goal, where two specific blocks need to be placed on top of another. This last goal is parameterized. The identity of the two blocks varies in each learning and testing episode. The last goal often considered with the RRL algorithm, "Unstacking", is easily solved by the RIB regression algorithm alone (Driessens & Ramon, 2003), so little or no improvement was possible for that task.

The performance graphs are generated by extracting the learned Q-function every 100 episodes, translating it into a deterministic policy (i.e., no exploration is included while testing) and testing it on 200 randomly generated starting positions with the number of blocks varying from 3 to 10. The Y-axis on the graphs indicates the percentage of tests in which the learned strategy reaches the desired goal state in the minimum number of steps possible, i.e., when the learned policy shows optimal behavior. The graphs show averaged results over 10 repeated experiments.

4.1. Blocks World Language Bias

The use of a TG like algorithm which builds first order decision trees requires the specification of a language bias for the blocks world. We used the same language bias as was used in previous experiments with RRL-TG in the blocks world (Driessens & Džeroski, 2004). Through the use of variables, the root of the tree referenced the 2 blocks included in the move action, and for the on(A,B) goal also the blocks included in the goal statement and included a count of the number of blocks in the current state. Possible tests available to the algorithm for use in lower nodes of the tree consisted of the following predicates[4]:

- *clear*/2 tests whether no block is on top of the referenced block in the given state

[4] All of these predicates include a reference to a blocks world state as the first argument.

- *on*/3 indicates whether a specified block is on top of another specified block or on the floor in the given state
- *equal*/2 tests whether two variables reference the same block (or the floor)
- *above*/3 tests whether a specified block is part of the stack on top of another given block
- *compheight*/3: compares the height of two different blocks, or the height of a block to a constant value (range 0-10)
- *compdiff*/4: calculates the difference between the height of a block and a previously initiated number-variable (such as the height of another block or the number of blocks) and compares this value to a constant value or the height of another block.

4.2. Blocks World Distance

The instance based part of our algorithm requires a distance to be defined between different learning examples, in this test case, between different $(state, action)$ pairs in the blocks world. We used the same distance measure as defined by Driessens and Ramon (2003), which is computed 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 compute the distance between the two "renamed" 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.

This application specific distance is not only easier (and thus faster) to compute than a more general first order distance, it also incorporates some background information about the blocks world, such as the importance of stacks and the identity of key blocks.

5. Experimental results

5.1. Parameter Influence

Since the existing regression algorithms used by RRL have the most difficulties with the 'On(A,B)' goal, we focused our tests on that task. We tested the influence of the RIB "maximum variance" parameter M and the "minimal-sample-size" mss parameter on a combination of different values. We let the M parameter vary between its standard value "0.1" (as used for RIB[5]) and 1.5, 2 and 4 times that value. The minimal sample size mms, we varied between 25, 50, 100 and 200. We show and discuss the most interesting results below.

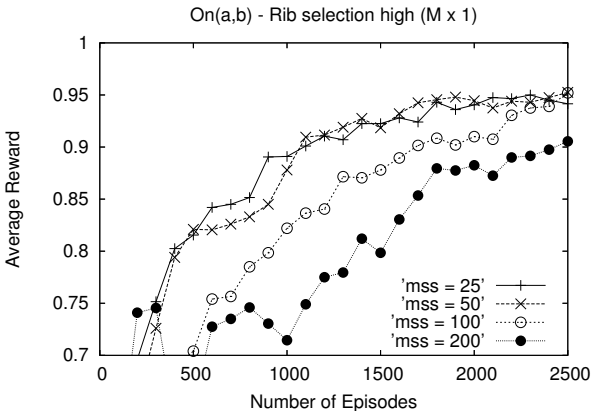

Figure 1. The influence of minimal sample size on the performance when example selection by RIB is high

As illustrated by Figure 1, larger minimal sample sizes lead to slower learning rates. This implies that the combination of trees and instance based prediction indeed leads to better predictions than instance based predictions on their own. Namely, the larger the minimal sample size is required to be, the longer the TRENDI algorithm will be equivalent to the RIB algorithm or consist of just a few instances thereof.

This behavior is observed for all values of the "maximum variance" M, but is most apparent when RIB uses high example selection, i.e., for low M values. The larger minimal sample sizes also cause a significant increase in computational complexity, as the TRENDI algorithm relies more on the computations within each RIB instance. This is illustrated by the algorithm's execution times shown in Table 1.

Lowering the amount of example selection used by the RIB algorithm can lead to better performance for the TRENDI approach. Figure 2 illustrates this with a

[5]The experiments use a reward of 1.0 for an accomplished task and a discount factor $\gamma = 0.9$.

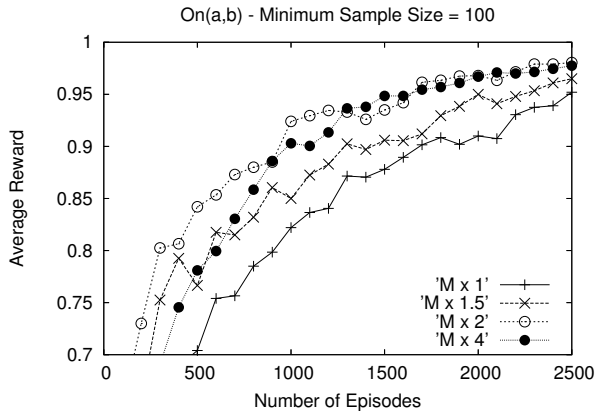

Figure 2. The influence of the RIB selection parameter on the performance with a minimal sample size of 100

medium minimal sample size of 100. Unfortunately, this better learning performance is accompanied by larger computation times and often, larger resulting trees. The good news is that some example selection ($M \times 2$) performs as well as almost no example selection ($M \times 4$), without a large increase in computation times.

Table 1 illustrates this with the average runtime of a single experiment for a number of different M and mss value combinations. The execution times include both learning and making predictions, as both are required during the experiment. The execution times for the set-up with almost no example selection by the RIB algorithm ($M \times 4$) vary largely between experiments. For the case with $mms = 100$ for example, we recorded execution times ranging from 624 CPU seconds (comparable to other values of M) to almost 20 000 CPU seconds. All experiments were performed on Pentium 4 machines running Linux, working at 2.6GHz with 512 MB RAM. Not shown on the graph, we did notice that when combined with small minimal sample sizes, low RIB example selection can slow down learning initially. This is probably caused by the higher number of learning examples with incorrect Q-values that influence the tree structure at the start of the experiment.

Table 1. Average run-times in CPU seconds of one experiment with varying minimal sample size and RIB example selection

$M \backslash mss$	> 25	> 50	> 100	> 200
$\times$ 1	182	344	606	1161
$\times$ 1.5	175	328	670	1264
$\times$ 2	183	333	677	1295
$\times$ 4	3227	3590	6229	15118

Table 2 shows the average size of the resulting tree for varying values of the mss and M parameters. As would be expected, larger mss values lead to smaller trees. One would expect less RIB example selection to have the same influence, and this seems to be the case when going from very high selection ($\times 1$) to medium selection ($\times 1.5$ and $\times 2$), but this trend does not continue for almost no example selection ($\times 4$). However, this last setting produces trees with largely varying sizes across different experiments. For the $mss = 100$ case, for example, the size of the learned tree ranged from 4 nodes to 23 nodes. This is tightly connected to the variance in execution times, with low tree sizes resulting in large computation times. For comparison, the TG algorithm, with a minimal sample size of 200 examples, produces trees with an average of 28 leafs.

Table 2. Average number of leafs in the resulting tree with varying minimal sample size and RIB example selection

$M \backslash mss$	> 25	> 50	> 100	> 200
$\times$ 1	76	36	19	10
$\times$ 1.5	81	41	21	11
$\times$ 2	83	40	21	9.9
$\times$ 4	68	34	15	4.6

All previous experiments were performed with a b-parameter (see Equation 3) set to 1.0. Increasing the error improvement that needs to be made before choosing to split a leaf node seems to have little effect for values very close to 1.0, but raising the boundary quickly leads the TRENDI algorithm to give up on building a tree, and behave even worse than a single RIB algorithm due to the larger M-value and its influence on example selection. Figure 3 compares the performance of the TRENDI algorithm for different values of b with $mss = 100$ and $M \times 2$. In the experiment with $b = 0.95$, TRENDI built trees with only 4 to 5 leafs. For comparison, we included the learning curve of the standard RIB algorithm in the graph. Thus, it seems that just requiring an improvement as a precondition to splitting a leaf is the best strategy.

5.2. Comparing to other algorithms

Next we compare the performance of the TRENDI algorithm with that of the three existing relational regression algorithms that have so far been used within the RRL setting. For TRENDI , we use the results obtained with parameter values $b = 1.0$, $mss = 100$ and $M \times 2$. Figure 4 shows the resulting graphs.

It is clear that TRENDI outperforms both of its "par-

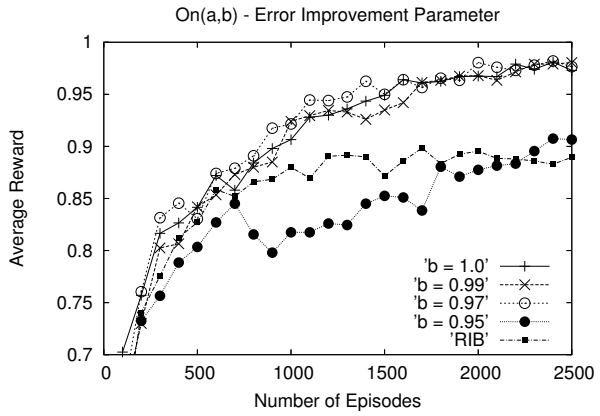

Figure 3. The influence of the error improvement parameter on TRENDI performance

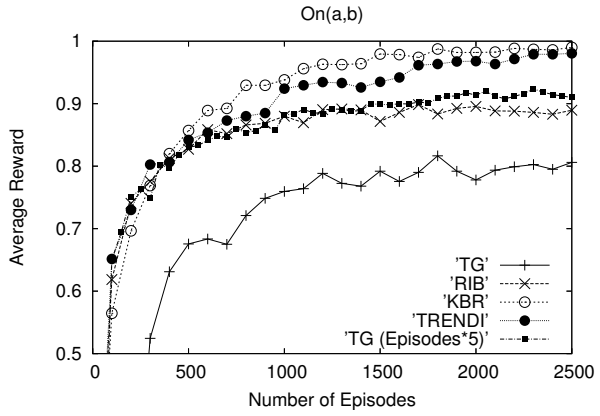

Figure 4. Comparing the TRENDI algorithm with other first order regression algorithms used in RRL on the *On(a,b)* goal

ents". Not only does it perform significantly[6] better then both TG and RIB , but when comparing the execution times of the experiments (Tables 1 and 3), the new algorithm also succeeds in retaining some of the computational efficiency of model-based algorithms. Although TRENDI retains an average of approximately 1200 examples in all the RIB instances combined, compared to approximately 730 in the standard RIB algorithm, the fact that TRENDI does not deal with all of these examples at the same time, makes it computationally much more efficient. Since the TG algorithm is still quite a bit faster than TRENDI we included an experiment that gave TG more episodes to learn from.

[6]The student's t-test indicates t-values of 5.3 and 7.1 for comparison to TG and RIB respectively, where only a value of 3.92 is required for a 0.001 probability that the two techniques perform equally well.

As shown on the graph, this still does not seem to help TG in reaching the same level of performance as reached with the TRENDI algorithm.

Table 3. Average run-times in CPU seconds of the previous RRL systems

System	CPU Time
TG	31
TG (12500 Episodes)	294
RIB	5000
KBR	150 000
TRENDI ($mss = 100, M \times 2$)	677

Although the difference is small[7], KBR still has the upper hand when it comes to performance gain per learning episode. However, the computational complexity of KBR, and the very long learning and prediction times could limit the practical use of this extra performance.

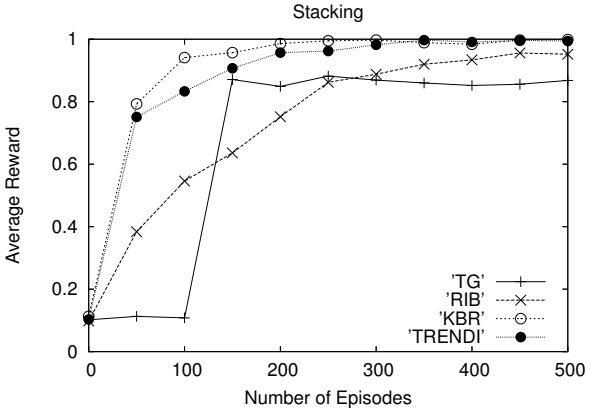

Figure 5. Comparing the TRENDI algorithm with previous first order regression algorithms on the *Stack* goal

On the "Stacking" goal, the TRENDI and KBR algorithm again perform similarly, and both again better than the TG and RIB algorithms. Figure 5 also shows that the TRENDI algorithm causes a gentler development of performance, comparable to that of instance-based approaches. The step-wise improvement of a model building algorithm like TG is avoided and a more robust development of the learned policy is obtained.

6. Conclusions

In this paper, we introduced a new incremental relational regression algorithm TRENDI that combines relational decision trees with (relational) instance based

[7]A t-value of 1.92 on the student's t-test, indicating a probability of 0.08 that the two perform equally well.

regression to make predictions on continuous target values using relational learning data. This combination of model-based and instance-based learning techniques retains the predictive accuracy and robustness of instance-based techniques as well as having a computational efficiency which is comparable to model-based learning. However, it does eliminate the coarse approximations that usually accompany model-based techniques.

In an experimental evaluation using relational reinforcement learning and the blocks world as a test-case, the TRENDI algorithm performed significantly better than its two "parent"-techniques TG and RIB and competitive with the much slower KBR algorithm.

A downside of the relational tree based algorithm, is that decisions made early in the learning experiment (tests chosen at the top of the tree), can not be undone. This might complicate the learning task at later stages in applications where data is only available incrementally, such as in e.g. reinforcement learning. In the future, we intend to investigate the combination of rule-based and instance-based learning as another approach to relational regression. The more loosely structured model that results from rule-learning (i.e., a rule-set or decision list) might make it easier to adapt the learned model in later stages of learning.

Although we've only tested the new algorithm on relational reinforcement learning, and although the algorithm was made incremental with this application in mind, the technique is readily applicable to other (relational) regression problems. In future work, we intend to experiment with TRENDI on other (relational) regression problems, possibly also in non-incremental settings.

Acknowledgements

Part of this research was performed during a sabbatical exchange funded by the PASCAL network of excellence.

References

Aha, D., Kibler, D., & Albert, M. (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.

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

Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning*, *24*, 141–168.

Driessens, K., & Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, *57*, 271–304.

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 123–130). AAAI Press.

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., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

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.

Freidman, J., Bentley, J., & Finkel, R. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, *3*, 209–226.

Gärtner, T., Driessens, K., & Ramon, J. (2003). Graph kernels and Gaussian processes for relational reinforcement learning. *Inductive Logic Programming, 13th International Conference, ILP 2003, Proceedings* (pp. 146–163). Springer.

Moore, A. (1991). *An introductory tutorial on kd-trees* (Technical Report). Robotics Institute, Carnegie Mellon University.

Quinlan, J. (1993). Combining instance-based and model-based learning. *Proceedings of the 10th International Conference on Machine Learning*. Morgan Kaufmann.

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

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, *6*, 252–276.

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