Incremental Reduced Error Pruning

Johannes Fürnkranz[1] and Gerhard Widmer[1,2]

[1]Austrian Research Institute for Artificial Intelligence
Schottengasse 3
A-1010 Vienna
Austria

[2]Department of Medical Cybernetics and Artificial Intelligence
University of Vienna

E-mail: [juffi,gerhard]@ai.univie.ac.at

Keywords: Inductive Logic Programming, Pruning, Noise

Abstract

This paper outlines some problems that may occur with *Reduced Error Pruning* in *Inductive Logic Programming*, most notably efficiency. Thereafter a new method, *Incremental Reduced Error Pruning*, is proposed that attempts to address all of these problems. Experiments show that in many noisy domains this method is much more efficient than alternative algorithms, along with a slight gain in accuracy. However, the experiments show as well that the use of this algorithm cannot be recommended for domains with a very specific concept description.

OEFAI-TR-94-09

1 Introduction

Being able to deal with noisy data is a must for algorithms that are meant to learn concepts in real-world domains. Significant effort has gone into investigating the effect of noisy data on decision tree learning algorithms (see e.g. [Quinlan, 1993, Breiman *et al.*, 1984]). Not surprisingly, noise handling methods have also entered the emerging field of *Inductive Logic Programming* (ILP) [Muggleton, 1992]. LINUS [Lavrač and Džeroski, 1992] relies directly on the noise handling abilities of decision tree learning algorithms like CN2 [Clark and Niblett, 1989, Clark and Boswell, 1991] or ASSISTANT [Bratko and Kononenko, 1986]. Others, like *m*FOIL [Džeroski and Bratko, 1992], have adapted some of these well-known methods from attribute-value learning for the ILP framework.

Pruning is a standard way of dealing with noise in decision tree learning (see e.g. [Mingers, 1989] or [Esposito *et al.*, 1993]). There are two fundamentally different approaches [Cestnik *et al.*, 1987]:

Pre-Pruning means that during concept generation some training examples are deliberately ignored, so that the final concept description does not classify all training instances correctly.

Post-Pruning means that first a concept description is generated that perfectly explains all training instances. This theory will subsequently be generalized by cutting off branches of the decision tree (as in [Quinlan, 1987] or [Breiman *et al.*, 1984]).

In ILP, pre-pruning has been common in the form of *stopping criteria* as used in FOIL [Quinlan, 1990], *m*FOIL [Džeroski and Bratko, 1992], or FOSSIL [Fürnkranz, 1994a]. Post-pruning was introduced to ILP with *Reduced Error Pruning* (REP) [Brunk and Pazzani, 1991] based on ideas by [Quinlan, 1987] and [Pagallo and Haussler, 1990]. First the training set is split into two subsets: a *growing set* and a *pruning set*. A concept description explaining all of the examples in the growing set is generated with a relational learning algorithm. The resulting concept is then generalized by deleting literals and clauses from the theory until any further deletion would result in a decrease of predictive accuracy measured on the pruning set.

However, this approach has several disadvantages, which we will highlight in section 2. Section 3 briefly presents the approach of [Cohen, 1993] designed to solve some of these problems. In section 4 we propose *Incremental Reduced Error Pruning* — a method that integrates pre- and post-pruning — as an alternative solution. Section 5 then reports some experiments with two versions of this algorithm.

2 Some Problems with Reduced Error Pruning

Reduced Error Pruning (REP) [Brunk and Pazzani, 1991] has proven to be quite effective in raising predictive accuracy in noisy domains. However, this method has several shortcomings, which we will discuss in this section.

2.1 Efficiency

In [Cohen, 1993] it was shown that the worst-case time complexity of REP is as bad as $\Omega(n^4)$ on random data (n is the number of examples). The growing of the initial concept, on the other hand, is only $\Omega(n^2 \log n)$. The derivation of these numbers as given in [Cohen, 1993] rests on the assumption that for random and thus incompressible data the concept description after the growing phase will contain about 1 rule for each example (n rules altogether), each of them having about $\log n$ conditions, because each literal will cover about half of the random instances. It is further assumed that the final (pruned) theory will only have constant size, i.e. its size is independent of n. The costs of adding 1 literal to the rule are proportional to n, because each of the constant number of literals[1] has to be tested once against each of the n instances in the growing set. As there will be about $n \log n$ literals in the concept description, we have a total cost of the order of $n^2 \log n$ for the growing phase.

In each step of the pruning phase each of the n clauses can be simplified by deleting the last literal or deleting the whole clause. Each of the $2n$ simplifications has to test its $\leq n$ rules against the n examples. Assuming that each rule will be modified or deleted at least once until the final concept description of constant size will be found, this simplification loop has to be performed at least n times. Therefore we get a total cost of $\Omega(n^4)$. A detailed proof can be found in [Cohen, 1993]. It has also been pointed out there that this result for random data generalizes to data containing noise, i.e. a constant fraction of random and incompressible data. Therefore it can be concluded that in the long run the costs of pruning will by far outweigh the costs of generating the initial concept description, which has been shown experimentally on a variety of data sets.

2.2 Split of Training Data

A disadvantage of REP is that the training data has to be split into two sets, a *growing set* (usually 2/3) and a *pruning set* (usually 1/3). A two-fold problem results from this: Learning occurs from examples in the growing set only. Therefore the algorithm might miss to learn important rules, if some or most of the examples for this rule are in the pruning set and not in the growing set. On the

[1] While the number of literals is constant, the number of variabilizations for each literal might be increasing with the introduction of new variables. We assume that the number of variables is bounded by a constant.

other hand, each learned rule has to have support in the pruning set, because if some or all examples for this rule are in the growing set, the rule — although learned correctly — might be pruned or deleted altogether. Thus a bad split of the given examples can have a negative influence on the behavior of both the learning and the pruning algorithm.

2.3 Separate-and-Conquer Strategy

In decision tree learning usually a *divide-and-conquer* strategy is used. This means that the training set is split into disjoint sets according to the outcome of the test chosen for the top level decision. After this, the algorithm is recursively applied to each of these sets independently. Greedy covering algorithms like FOIL follow a *separate-and-conquer* strategy [Pagallo and Haussler, 1990]. This method first learns a rule from the whole training set and subsequently removes all examples that are covered by this rule. Then the algorithm recursively tries to find rules that explain the remaining examples.

Although the separate-and-conquer approach shares many similarities with the divide-and-conquer strategy, there is one important difference: Pruning of branches in a decision tree will never affect the neighboring branches, whereas pruning of literals of a rule will affect all subsequent rules. Pruning a literal from a clause means that the clause is generalized, i.e. it will cover more positive instances along with some negative instances. These negative instances are deliberately ignored, i.e. they are practically identified to be wrong or noisy. Consequently those additional positive and negative instances should be removed as well so that they cannot influence the learning of subsequent clauses. However, the initial growing phase of REP does not know which of the instances are noisy and will henceforth carry along instances that should already be covered by one of the previous clauses. In the best case those superfluous examples in the growing phase only lead to the generation of some additional clauses that will be pruned in the pruning phase. In the worst case, however, those instances may lead the learner down a garden path, because they may change the evaluation of the literals in subsequent learning and thus the "correct" literals might not be selected. A wrong choice of a literal cannot be undone by pruning.

2.4 Bottom Up Hill-Climbing

REP employs a greedy hill-climbing strategy: Literals and clauses will be deleted from the concept definition so that predictive accuracy on the pruning set is maximized. When each possible operator leads to a decrease in predictive accuracy, the search process stops.

However, in noisy domains it can be expected that the theory that has been generated in the growing phase is much too specific, i.e. REP will have to do a lot of pruning and therefore has ample opportunity to get caught in a local

maximum. Therefore REP's specific-to-general search can be expected to be slow and imprecise for noisy data, because it has to prune a significant portion of the theory previously generated in the growing phase and is likely to stop at a local maximum during this process. [Fürnkranz, 1994b] reports experiments with an algorithm that is able to find a starting theory much closer to the final theory than the most specific theory. This method succeeded in improving both run-time and accuracy of REP.

3 Cohen's GROW algorithm

In [Cohen, 1993] several of the problems of section 2 — in particular efficiency — have been recognized. Cohen has then proposed a pruning algorithm based on the technique used in the GROVE learning system [Pagallo and Haussler, 1990]. Like REP, GROW first finds a theory that overfits the data. But instead of pruning the intermediate theory until any further deletion results in a decrease in accuracy on the pruning set, in a first step the intermediate theory is augmented with generalizations of all its clauses. In a second step, clauses from this expanded theory are subsequently selected to form the final concept description until no further clause that improves predictive accuracy on the pruning set can be found. The generalizations of the clauses are formed by repeatedly deleting a final sequence of conditions from the clause so that the error on the *growing* set goes up the least. For a detailed description of the GROW algorithm see [Cohen, 1993].

This algorithm solves several of the problems from section 2:

- Under the assumptions of section 2.1 the costs of pruning on random data are reduced to $O(n^2 \log n)$: Each clause contains about $\log n$ literals, each of which can be the last literal in a generalized clause. So in the worst case we get a total of $n \log n$ clauses in the augmented intermediate theory. Each of them is tentatively added to the initially empty final theory and the resulting set of clauses is tested on the n training examples. This is repeated until all clauses in the final concept description of constant size have been found. Therefore the costs of this algorithm are $O(n^2 \log n)$. Again, consult [Cohen, 1993] for a detailed proof.

- GROW replaces the bottom-up hill-climbing search of REP by a top-down approach (see section 2.4). It does not remove the most useless clause or literal from the specific theory, but instead adds the most promising generalization of a rule to an initially empty theory. This results in a significant gain in efficiency, along with a slight gain in accuracy. An explanation for the latter could be that top-down hill-climbing starts from the empty theory, which in many domains is much closer to the correct theory than the most specific one.

However, Cohen's GROW algorithm does not attempt to solve the remaining problems with bad splits and the separate-and-conquer strategy. Besides, the overall costs of the GROW algorithm still include the cost of overfitting the data and thus are unnecessarily high. Consequently Cohen tried to improve GROW by adding two stopping heuristics to the initial stage of overfitting, and thus achieved a further speed-up of the algorithm. Another way of *combining* pre-pruning and post-pruning methods to get better results can be found in [Fürnkranz, 1994b].

In section 4 we present an alternative approach that *integrates* pre-pruning and post-pruning in a way that avoids the expensive initial phase of overfitting altogether.

4 Incremental REP

As an attempt to solve the problems mentioned in section 2 we have developed a new algorithm that integrates pre-pruning and post-pruning into learning. The basic idea is that instead of first growing a complete concept description and pruning it thereafter, each clause will be pruned right after it has been generated (see figure 1): After learning a clause from the growing set, literals will be deleted from this clause in a greedy fashion until any further deletion would decrease the accuracy of this clause on the pruning set. Each literal is considered for pruning (not only a final sequence of literals). The resulting rule will then be added to the concept description and all covered positive and negative examples will be removed from the training — growing *and* pruning — set. The remaining instances in the training set are then redistributed into a growing and a pruning set and a new clause is learned. When the predictive accuracy of the pruned clause is below the predictive accuracy of the empty clause (i.e. the clause with the body `fail`), the clause will not be added to the concept description and I-REP returns the learned clauses. Thus the accuracy of the pruned clauses on the pruning set also serves as a stopping criterion.

As this algorithm does not prune on the entire set of clauses, but prunes each one of them successively, we have named it *Incremental Reduced Error Pruning* (I-REP). It addresses the problems of section 2 in the following ways:

Efficiency: I-REP does not generate an intermediate concept description. Thus the costs of I-REP are roughly the costs for generating the final theory, while REP and GROW have to generate a more specific theory first. As in REP, growing one clause from purely random data costs $n \log n$ (see section 2.1). I-REP considers every literal in the clause for pruning, i.e. each of the $\log n$ literals has to be tested against n examples until the final clause has been found, i.e. at most $\log n$ times. Thus the costs of pruning one clause are $n \log^2 n$. As the final theory only has a constant number of clauses, the overall costs are also of the order $n \log^2 n$.

```
procedure I-REP (Pos, Neg, SplitRatio)

Clauses = ∅
while Pos ≠ ∅
      SplitExamples(SplitRatio, Pos, PosGrow, PosPrune)
      SplitExamples(SplitRatio, Neg, NegGrow, NegPrune)
      Clause = ∅
      while NegGrow ≠ ∅
            Clause = Clause ∪ FindLiteral(Clause, PosGrow, NegGrow)
            PosGrow = Cover(Clause, PosGrow)
            NegGrow = Cover(Clause, NegGrow)
      Clause = PruneClause(Clause, PosPrune, NegPrune)
      if Accuracy(Clause) ≤ Accuracy(fail)
         return(Clauses)
      else
         Pos = Pos − Cover(Clause, Pos)
         Neg = Neg − Cover(Clause, Neg)
         Clauses = Clauses ∪ Clause
return(Clauses)
```

Figure 1: Incremental Reduced Error Pruning

Split of Training Data: I-REP redistributes its pruning and growing sets after a clause has been found. Thus the scope of the problems discussed in section 2.2 is reduced to the learning of a single clause. However, it may happen that a clause learned from a bad growing set or evaluated on a bad pruning set appears worse than the empty clause, which causes I-REP to stop learning, while REP would continue learning and prune the bad clause later on.

Separate-and-Conquer Strategy: I-REP learns the clauses in the order in which they will be used by a PROLOG interpreter. Before subsequent rules will be learned, each clause is completed (learned *and* pruned) and all covered examples are removed. For this reason the problems discussed in section 2.3 cannot appear in I-REP.

Bottom-Up Hill-Climbing: Similarly to GROW, I-REP uses a top-down approach, instead of REP's bottom-up search: Final programs are not found by removing unnecessary clauses and literals from an overly specific theory, but by repeatedly adding clauses to an initially empty theory. However, GROW still has to generate an intermediate, specific concept description, while I-REP directly constructs the final theory. In cases where the correct concept definition is fairly simple, the top-down approach can be expected to be less sensitive to local optima as discussed in section 2.4.

Most of the efficiency of the I-REP algorithm comes from the integration of pre-pruning and post-pruning by defining a stopping criterion based on the accuracy of the pruned clause on the pruning set. However, this may also cause problems: Whenever the pruned clause does not have an accuracy above the accuracy of the empty clause, no more clauses will be learned. This is more or less the inversion of REP's `delete-clause` operator. However, if this accuracy is not measured appropriately, either because there are not enough examples left, or because of a bad split of the examples, I-REP will be prone to over-generalization. Using the terminology of [Schaffer, 1993], I-REP has a strong *Over-fitting Avoidance Bias*, which can be detrimental in some domains.

5 Experiments

5.1 Implementations of the Algorithms

We have tested two different implementations of I-REP, which differ in the way they prune the clauses (let p (n) be the number of positive (negative) examples covered by the current clause from a total of P (N) positive (negative) examples in the current pruning set):

I-REP prunes clauses so that the number of covered positive examples plus the number of not covered negative examples is maximized ($\frac{p+(N-n)}{P+N}$). The accuracy of the empty clause (i.e. the clause with the body `fail`) is $\frac{N}{P+N}$. Whenever the accuracy of the best pruned clause is below this value, learning stops.

I-REP-2 prunes clauses so that the "purity" of each clause ($\frac{p}{n+p}$) is maximized. As the purity of the empty clause is meaningless ($p = n = 0$), we have adopted the following stopping criterion: Only clauses that cover more positive than negative examples ($\frac{p}{n+p} > 0.5$) are permitted, as only those may increase the overall accuracy of the concept.

Both algorithms, REP and GROW, were implemented as described in [Cohen, 1993] with the exception that `delete-last-literal` was used as a clause pruning operator (as in [Brunk and Pazzani, 1991]) instead of Cohen's operator that deletes a final sequence of literals from a clause.[2] All systems were implemented in Sicstus PROLOG, run-times were measured on a SUN SPARCstation IPX.

[2]Cohen used his `delete-final-sequence` operator in both REP and GROW, while we have used `delete-last-literal` in both algorithms. Besides, [Cohen, personal communication] has pointed out that his implementation of GROW in most cases produces all generalizations that would be produced when using `delete-last-literal` instead.

5.2 Domains

We have run experiments in a variety of domains. As our current implementation of the algorithms is not capable of handling continuous attributes or multi-valued classes, our choice of test domains was somewhat limited. Tests were performed for most of the datasets in the UCI Machine Learning repository that had only two classes and only symbolic attributes, and also for the KRK [Muggleton *et al.*, 1989] and the *Mesh* [Dolšak and Muggleton, 1992] domains. Table 1 gives an overview of the used databases along with a comparison of the run-times of the different algorithms. In some domains artificial noise was generated by inverting the classification of 10% of the examples. The column *Overfitting* refers to the initial growing phase that REP and GROW have in common, while REP and GROW give the results for the pruning phases only. Thus the total run-time of REP (GROW) is the run-time of *Overfitting* plus the run-time of REP (GROW). The best result(s) in each line are emphasized.

Domain	*Overfitting*	REP	GROW	I-REP	I-REP-2
KRK-100 (10%)	8.36	2.44	1.66	**4.20**	4.37
KRK-250 (10%)	91.31	104.98	19.81	**17.30**	18.09
KRK-500 (10%)	456.56	1578.16	100.81	**46.32**	57.05
KRK-750 (10%)	1142.78	7308.84	361.41	**83.64**	118.99
KRK-1000 (10%)	2129.89	23125.34	806.89	**115.35**	178.26
Monks 1	**0.70**	**0.03**	0.24	1.03	1.02
Monks 2	43.43	52.56	18.45	**9.69**	12.03
Monks 3	5.82	2.09	1.14	**3.30**	3.93
Mesh	3928.89	21168.56	2386.09	**379.57**	466.47
Promoters	241.03	1.00	0.63	**179.64**	195.79
Votes	59.05	13.26	4.87	**23.87**	37.63
Votes (VI)	185.94	101.87	30.83	**41.59**	69.97
Mushroom	108.03	2.30	5.26	**106.95**	110.49
KRKN	**24.49**	**0.57**	2.45	28.19	28.59
KRKPa7	937.93	26.29	17.41	**608.78**	728.59
Tic-Tac-Toe	191.47	45.46	57.74	**112.98**	131.06
Mushroom (10%)	1454.44	612.20	75.51	**254.40**	372.52
KRKN (10%)	862.84	504.77	48.19	**86.29**	116.30
KRKPa7 (10%)	3951.48	296.01	69.81	**812.66**	1068.26
Tic-Tac-Toe (10%)	443.76	390.02	114.33	**92.71**	134.89

Table 1: Average run-time

On the *Votes* and *Promoters* data a 10-fold cross-validation was performed. The *Mesh* data were tested in 5 runs as described in [Džeroski and Bratko, 1992], but classification accuracy on negative examples was measured as well. The KRK data were tested on 5 different example set sizes evaluated on 5000 noise-free examples. For each training set size we used 10 different example sets except

for the sets with 1000 examples, which were only tested on 6 sets because of the high run-times of this task. The same was done with the KRKN data on a training set size of 500. The *Monks* data have a dedicated testing set, so 10 runs were performed using different pruning and growing set distributions, all evaluated on the same testing set. In all other domains 10 sample sets with a size of about 500 examples were generated and the rules were tested on the remaining examples. The *Votes (VI)* set is the *Votes* data set with the most significant attribute removed. In all of the propositional domains the equality relation was added as background knowledge. The *Promoters* data also included two background relations specifying that the 4 DNA bases can be split into 2 groups [Ali and Pazzani, 1993], and in the KRKN data the $<$ relation was added for the 6 integer valued attributes. In all domains the system was given the task of learning definitions for the minority class.

5.3 Results

Looking at table 1 it can easily be seen that GROW is much faster than REP, but I-REP does not have to grow an intermediate theory and thus is faster than both. In fact, I-REP is usually significantly faster than the initial growing phase that both REP and GROW have in common. Figure 2 illustrates the effect of training set size on the performance of the algorithms in the KRK domain.

Looking at the classification accuracies (table 2) the picture is more diverse. However, several conclusions can be drawn: The experiments in the *KRK* domain illustrate that I-REP is more sensitive to small training set sizes, but improves much at larger sizes (see also figure 2). The reason for this is that a bad distribution of growing and pruning examples may cause I-REP's stopping criterion to prematurely stop learning. Redistributing the examples before the learning of each clause cannot help here, as there is little redundancy in the data because of the small sample size. Thus each example has a significant influence on the outcome of learning. At larger example set sizes I-REP does better than the other algorithms, because REP often gets caught in local maxima and is not able to generalize to the right level. Interestingly, despite its top-down search strategy, GROW also occasionally overfits the noise in the data: Some of the highly specialized clauses in the intermediate theory sometimes also fit a few noisy examples in the pruning set and thus will be added to the concept description. In I-REP this is less likely to happen, because it generates fewer clauses and stops after the first clause has been found that fits noisy examples in the growing, but not in the pruning set.

In domains with a very specific theory, where not much pruning has to be done or pruning is even detrimental, the bottom-up approach to pruning is more appropriate. This can be see most clearly in the noise-free *Tic-Tac-Toe* and *Monks 2* domains, where REP was significantly more accurate and even a little faster than GROW, although clearly some amount of pruning was performed.

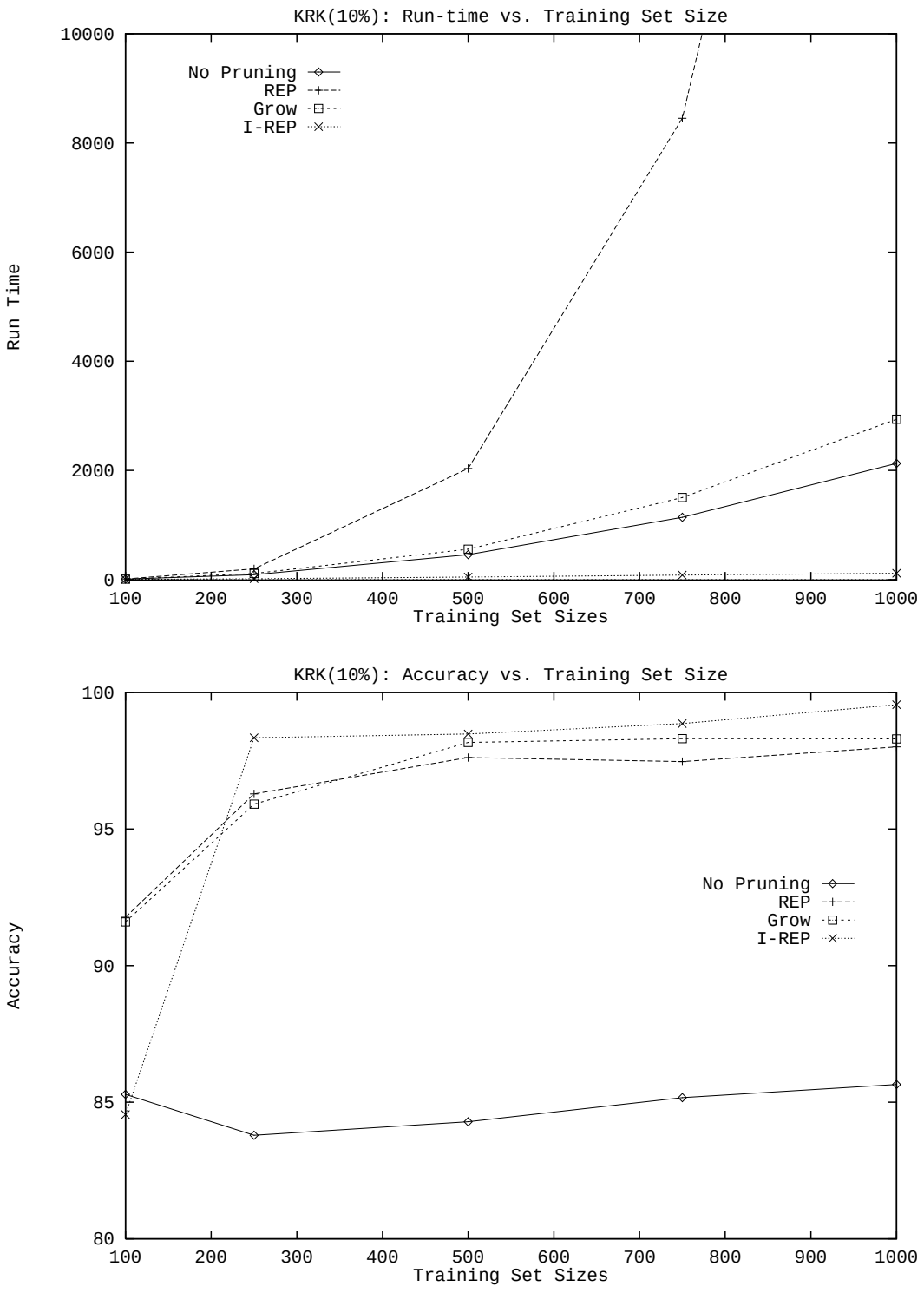

Figure 2: Different training set sizes in the KRK domain.

Domain	*Overfitting*	REP	GROW	I-REP	I-REP-2
KRK-100 (10%)	85.29	**91.77**	91.60	84.55	85.37
KRK-250 (10%)	83.79	96.29	95.91	**98.34**	97.93
KRK-500 (10%)	84.29	97.62	98.17	**98.48**	95.67
KRK-750 (10%)	85.17	97.47	98.31	**98.86**	98.49
KRK-1000 (10%)	85.65	98.01	98.30	**99.55**	98.30
Monks 1	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Monks 2	57.80	**63.65**	60.58	61.63	63.31
Monks 3	92.48	96.90	**98.06**	**98.06**	96.16
Mesh	84.63	88.87	88.81	**91.00**	90.97
Promoters	65.00	68.00	71.64	**71.91**	71.64
Votes	94.71	**95.85**	95.39	95.62	94.71
Votes (VI)	86.66	87.53	87.13	**88.72**	87.80
Mushroom	**99.08**	98.78	98.40	97.14	98.10
KRKN	99.54	**99.76**	**99.76**	99.28	99.68
KRKPa7	92.92	94.72	**94.98**	94.76	92.80
Tic-Tac-Toe	92.78	**93.40**	86.95	82.71	83.83
Mushroom (10%)	79.57	**87.96**	87.46	87.21	86.93
KRKN (10%)	85.12	97.86	97.72	**99.00**	98.37
KRKPa7 (10%)	74.89	83.41	**83.53**	81.93	82.73
Tic-Tac-Toe (10%)	72.41	**74.43**	73.29	71.83	73.14

Table 2: Average accuracy

I-REP had the worst classification accuracy on these domains. The reason for this is that I-REP's top-down approach to pruning has an even stronger *Overfitting Avoidance Bias* than GROW, which can be inappropriate in some domains [Schaffer, 1993].

I-REP-2 in general seems to be worse than I-REP. Its purity criterion for evaluating a clause that tries to maximize the percentage of the covered positive examples seems to have a preference for more specific clauses than I-REP (which can also be seen from the higher run-times). This might be the reason why I-REP-2 is worse than I-REP in domains with a fairly general theory, but seems to do a little better in domains where specific theories have to be found.

In the natural domains *Mesh, Promoters* and *Votes* I-REP is at least equal to the other algorithms in terms of predictive accuracy, but significantly faster than both of them. This seems to confirm the observation of [Holte, 1993] that in most commonly used data sets, simple rules perform reasonably well.

As a general conclusion we may say that GROW outperforms REP and that I-REP is better than REP and GROW whenever a fairly general theory has to be found, whereas REP is appropriate when the underlying theory is rather specific.

6 Conclusion

In this paper we have introduced a new method of integrating pruning and learning — *Incremental Reduced Error Pruning* (I-REP). The system builds upon ideas introduced by [Brunk and Pazzani, 1991] and [Cohen, 1993], but improves upon them in the following ways:

Efficiency: [Cohen, 1993] has shown that the complexity of REP is $\Omega(n^4)$ on random data and has proposed an alternative algorithm — GROW — with time complexity $\Omega(n^2 \log n)$. The complexity of our algorithm, due to its new method of integrating pruning into learning, is of the order $n \log^2 n$. Experiments confirm the significant run-time improvement over REP and GROW, although I-REP uses a more expensive pruning operator.[3]

Separate-and-Conquer Strategy: Section 2.3 argued that the separate-and-conquer strategy of many relational learning algorithms may lead to problems when used for overfitting noisy data. Our algorithm avoids this problem, because the rules are pruned right after they are generated. Thus they are immediately adjusted to the right level of generality and the learning of subsequent clauses cannot be disturbed by the influence of an overly specific first clause.

Split of Training Data: The performance of the above algorithms depends on a reasonable split of the training set into a growing and a pruning set. I-REP is also prone to this problem, but its method of redistributing the examples after a clause has been learned may help to stabilize the behavior of the algorithm.

I-REP's efficiency stems from the tight integration of post-pruning and pre-pruning. Whenever the algorithm learns a clause that is worse than the empty clause, learning stops. However, this may cause the algorithm to over-generalize in domains with a rather specific concept description [Schaffer, 1993]. Similar problems may occur with small training sets.

In the near future I-REP should be adapted to be capable of dealing with numeric data and multi-valued classes to allow a test in a broader variety of real-world domains as in [Cohen, 1993]. A direct comparison of our results and the work reported there is not possible, because Cohen's experiments were performed with a propositional learning algorithm, while in our experiments relations like equality were made available as background knowledge to all learners.

[3] Experiments with Cohen's GROW and the `delete-any-literal` operator used in I-REP indicate that its usage may result not only in an increase in runtime, but surprisingly also in a decrease in accuracy. Whether using the `delete-last-literal` operator in I-REP would also improve predictive accuracy remains to be investigated, but it would further speed up the algorithm in any case.

Acknowledgements

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P8756-TEC. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Research. We would like to thank our colleagues B. Pfahringer and Ch. Holzbaur for help on many improvements of the PROLOG implementation of FOIL.

References

[Ali and Pazzani, 1993] Kamal M. Ali and Michael J. Pazzani. HYDRA: A noise-tolerant relational concept learning algorithm. In *Proceedings of the Thirteenth Joint International Conference on Artificial Intelligence*, pages 1064–1071, Chambèry, France, 1993.

[Bratko and Kononenko, 1986] Ivan Bratko and Igor Kononenko. Learning diagnostic rules from incomplete and noisy data. In B. Phelps, editor, *Interactions in AI and Statistical Methods*, pages 142–153, London, 1986.

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA, 1984.

[Brunk and Pazzani, 1991] Clifford A. Brunk and Michael J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 389–393, Evanston, Illinois, 1991.

[Cestnik *et al.*, 1987] Bojan Cestnik, Igor Kononenko, and Ivan Bratko. ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In Ivan Bratko and Nada Lavrač, editors, *Progress in Machine Learning*, pages 31–45, Wilmslow, England, 1987. Sigma Press.

[Clark and Boswell, 1991] Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of the 5th European Working Session of Learning*, pages 151–163, Porto, Portugal, 1991.

[Clark and Niblett, 1989] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[Cohen, 1993] William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 988–994, Chambery, France, 1993.

[Dolšak and Muggleton, 1992] Bojan Dolšak and Stephen Muggleton. The application of Inductive Logic Programming to finite-element mesh design. In Stephen Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press Ltd., London, 1992.

[Džeroski and Bratko, 1992] Sašo Džeroski and Ivan Bratko. Handling noise in Inductive Logic Programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan, 1992.

[Esposito *et al.*, 1993] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. Decision tree pruning as a search in the state space. In *Proceedings of the European Conference on Machine Learning*, pages 165–184, Vienna, Austria, 1993. Springer-Verlag.

[Fürnkranz, 1994a] Johannes Fürnkranz. FOSSIL: A robust relational learner. In *Proceedings of the European Conference on Machine Learning*, Catania, Italy, 1994. Springer-Verlag.

[Fürnkranz, 1994b] Johannes Fürnkranz. Top-down pruning in relational learning. Technical Report OEFAI-TR-94-03, Austrian Research Institute for Artificial Intelligence, 1994.

[Holte, 1993] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[Lavrač and Džeroski, 1992] Nada Lavrač and Sašo Džeroski. Inductive learning of relations from noisy examples. In Stephen Muggleton, editor, *Inductive Logic Programming*, pages 495–516. Academic Press Ltd., London, 1992.

[Mingers, 1989] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Muggleton *et al.*, 1989] Stephen Muggleton, Michael Bain, Jean Hayes-Michie, and Donald Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113–118, 1989.

[Muggleton, 1992] Stephen Muggleton, editor. *Inductive Logic Programming.* Academic Press Ltd., London, 1992.

[Pagallo and Haussler, 1990] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[Quinlan, 1987] John Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Quinlan, 1990] John Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Quinlan, 1993] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Schaffer, 1993] Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.