Machine Learning, 27, 139–172 (1997)

Pruning Algorithms for Rule Learning

JOHANNES FÜRNKRANZ juffi@ai.univie.ac.at
Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Vienna, Austria

Editor: Raymond J. Mooney

Abstract. Pre-pruning and Post-pruning are two standard techniques for handling noise in decision tree learning. Pre-pruning deals with noise during learning, while post-pruning addresses this problem after an overfitting theory has been learned. We first review several adaptations of pre- and post-pruning techniques for separate-and-conquer rule learning algorithms and discuss some fundamental problems. The primary goal of this paper is to show how to solve these problems with two new algorithms that combine and integrate pre- and post-pruning.

Keywords: Pruning, Noise Handling, Inductive Rule Learning, Inductive Logic Programming

1. Introduction

Separate-and-conquer rule-learning systems have recently gained popularity through the success of the Inductive Logic Programming algorithm FOIL (Quinlan, 1990, Quinlan & Cameron-Jones, 1995). In this paper, we will analyze different pruning techniques for this type of inductive rule learning algorithm and discuss some of their problems. Its main contributions are two new algorithms: *Top-Down Pruning* (TDP), an approach that combines pre- and post-pruning, and *Incremental Reduced Error Pruning* (I-REP), a very efficient integration of pre-and post-pruning.

Pruning is the common framework for avoiding the problem of *overfitting* noisy data. The basic idea is to incorporate a bias towards simpler theories in order to avoid complex rules with low coverage that contain irrelevant literals that have only been added to exclude noisy examples.

Pre-pruning methods deal with noise during learning. Instead of trying to find a theory that is complete and consistent with the given training data, heuristics (i.e., *stopping criteria*) are used to relax this constraint by stopping the learning process although some positive examples may not yet be explained and some of the negative examples may still be covered by the current theory. The final theory is learned in one pass (see figure 1). Most separate-and-conquer rule learners, like CN2 (Clark & Niblett, 1989), FOIL (Quinlan, 1990), and FOSSIL (Fürnkranz, 1994a), use this form of noise handling.

Another family of algorithms deals with noise by simplifying a previously learned overfitting theory. These *post-pruning* algorithms typically first induce a theory that is complete and consistent with the training data. Then, this theory is examined in order to discard rules and conditions that only seem to explain characteristics of the particular training set and do not reflect true regularities of the domain (see figure 1). The quality of the learned rules and conditions is commonly evaluated on a separate set of training examples that have not been seen during learning. Post-pruning algorithms include *Reduced Error Pruning* (REP) (Brunk & Pazzani, 1991) and GROW (Cohen, 1993). Both have been shown to be very

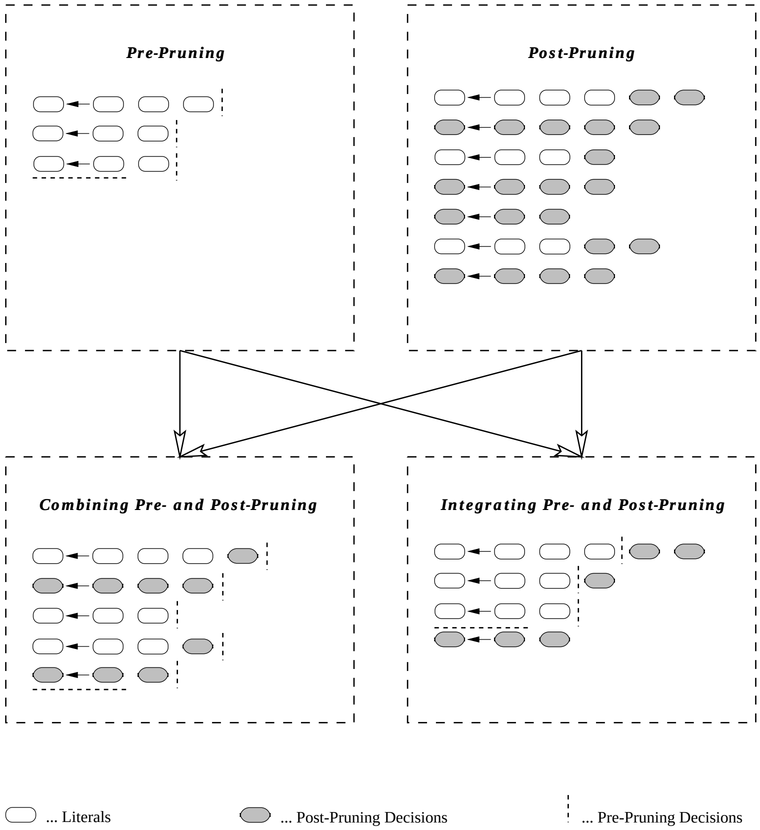

Pre-Pruning
Post-Pruning
Combining Pre- and Post-Pruning
Integrating Pre- and Post-Pruning
... Literals
... Post-Pruning Decisions
... Pre-Pruning Decisions

Figure 1. Pruning methods for separate-and-conquer rule learning algorithms.

effective in noise-handling. However, they are also inefficient, because they waste time by learning an overfitting concept description and subsequently pruning a significant portion of its rules and conditions.

One remedy for this problem is to *combine* pre- and post-pruning (figure 1, part 3). Pre-pruning heuristics are used to reduce (not entirely eliminate) the amount of overfitting, so that learning and pruning will be more efficient. Our particular implementation of this approach, *Top-Down Pruning* (TDP) (Fürnkranz, 1994b), uses a simple algorithm to generate a set of theories pruned to different degrees in a top-down, simple-to-complex order. The accuracies of the theories are evaluated on a separate set of data and the most complex theory with an accuracy comparable to the accuracy of the best theory so far will be submitted to a subsequent post-pruning phase. Experiments show that this initial top-down search for a better starting theory can be more efficient than the overfitting phase of classical post-pruning algorithms. As this search will typically return a theory that is closer to the

```
procedure SeparateAndConquer(Examples)

Theory $= \emptyset$
while Positive(Examples) $\neq \emptyset$
    Clause $= \emptyset$
    Cover $=$ Examples
    while Negative(Cover) $\neq \emptyset$
        Clause $=$ Clause $\cup$ FindLiteral($Clause, Cover$)
        Cover $=$ Cover(Clause,Cover)
    Theory $=$ Theory $\cup$ Clause
    Examples $=$ Examples $-$ Cover
return(Theory)
```

Figure 2. A separate-and-conquer rule learning algorithm

final theory, the post-pruning phase will also be sped up, because fewer pruning operations are needed to get to the final theory.

Motivated by the success of this method, we have developed a more rigorous approach that tightly *integrates* pre- and post-pruning. Instead of learning an entire theory and pruning it thereafter, *Incremental Reduced Error Pruning* (I-REP) (Fürnkranz & Widmer, 1994) prunes single clauses right after they have been learned. Thus, by using post-pruning methods as a pre-pruning stopping criterion (as sketched in figure 1), this new algorithm avoids learning an overfitting theory and achieves a significant speedup in noisy domains. Because it avoids some problems with other approaches that incorporate post-pruning, I-REP also learns more accurate theories.

2. Separate-and-conquer rule learning algorithms

Many rule learning algorithms including the propositional learner CN2 (Clark & Niblett, 1989), and the relational learner Foil and its successors (Quinlan & Cameron-Jones, 1995) construct rules using the *separate-and-conquer* strategy. The term separate-and-conquer was coined by Pagallo and Haussler (1990) in the context of learning decision lists, but the strategy has its origins in the AQ family of covering algorithms (Michalski, 1980, Michalski, Mozetič, Hong, & Lavrač, 1986). Fürnkranz (1996) presents an extensive survey of this family of learning algorithms.

Figure 2 sketches the basic SeparateAndConquer rule learning algorithm. The input to the algorithm is a set of positive and negative examples of the target concept. The output is a set of rules that are able to prove the given positive examples, but none of the negative examples. We will represent rules in the form of PROLOG clauses as in Foil:

```
Concept :- Literal1, Literal2, ..., LiteralN.
```

In propositional learning, literals can only be tests for the values of certain attributes of the concept, while in relational learning (as in Foil) one can also specify relations between these attributes, so that the head and the conditions of a rule can be general PROLOG literals. We will consider a set of rules as a PROLOG program, i.e., the rules will be checked in order until one of them "fires". The example that has satisfied the conditions of the rule will consequently be classified as an instance of the target concept. If no rule "fires", the instance will not be considered as a member of the concept. Although we will use PROLOG terminology in the rest of the paper — rules will be referred to as *clauses*, while their conditions will be denoted as *literals* — our results also apply to propositional separate-and-conquer rule learning algorithms like CN2 (Clark & Niblett, 1989) or SWAP-1 (Weiss & Indurkhya, 1991).

SEPARATEANDCONQUER learns clauses by successively adding literals to their right-hand side until the clause covers no more negative examples. All covered positive examples are then separated from the training set and the next rule is learned from the remaining examples (hence the name *separate-and-conquer*). Rules are learned in this way until no positive examples are left. This method guarantees that each positive example is covered by at least one rule (*completeness*), and that no rule covers a negative example (*consistency*).

However, the basic SEPARATEANDCONQUER algorithm has a severe drawback: real-world data may be noisy. Noisy data are a problem for many learning algorithms, because it is hard to distinguish between rare exceptions and erroneous examples. As we have seen, the algorithm of figure 2 forms a complete and consistent theory, i.e., it tries to cover all of the positive and none of the negative examples. In the presence of noise, it will therefore attempt to add literals to rules in order to exclude positive examples that have a negative classification in the training set and add rules in order to cover negative examples that have erroneously been classified as positive. Thus, complete and consistent theories generated from noisy examples are typically very complicated and exhibit low predictive accuracy on classifying unseen examples. This problem is known as *overfitting the noise*.

One remedy for this problem is to try to increase the predictive accuracy by considering not only complete and consistent theories, but also simple approximate theories. A simple theory that covers most positive examples and excludes most negative examples of the training set will often be more predictive than a complex, complete and consistent theory. Such simple theories are usually discovered using *pruning* heuristics.

3. Pre-pruning

Figure 3 shows an adaptation of the simple SEPARATEANDCONQUER algorithm that handles noisy data with a *pre-pruning* heuristic. The algorithm is identical to the one of figure 2 except that both loops can terminate not only when no more negative examples are covered (inner loop) or when all positive examples are covered (outer loop), but also when stopping criteria are satisfied. The LITERALSTOPPINGCRITERION decides heuristically when to stop adding literals to a clause, while the CLAUSESTOPPINGCRITERION decides when to stop adding clauses to the theory. If the current rule with the new literal added satisfies the LITERALSTOPPINGCRITERION, the inner `while` loop will terminate and the inconsistent clause will be added to the concept description, unless the CLAUSESTOPPINGCRITERION is satisfied. In that case, it is assumed that no further clause can be found that covers the remaining positive examples and the incomplete theory without the clause that triggered the

```
procedure PrePruning(Examples)

Theory $= \emptyset$
while Positive(Examples) $\neq \emptyset$
        Clause $= \emptyset$
        Cover $=$ Examples
        while Negative(Cover) $\neq \emptyset$
                NewClause $=$ Clause $\cup$ FindLiteral(Clause, Cover)
                if LiteralStoppingCriterion(Theory,NewClause,Cover)
                   exit while
                Clause $=$ NewClause
                Cover $=$ Cover(Clause,Cover)
        if ClauseStoppingCriterion(Theory,Clause,Cover)
           exit while
        Theory $=$ Theory $\cup$ Clause
        Examples $=$ Examples $-$ Cover
return(Theory)
```

Figure 3. A rule learning algorithm using pre-pruning

criterion is returned as the final theory. The remaining positive examples are thus considered to be noisy and will be classified as negative by the returned theory.

Most separate-and-conquer algorithms employ stopping criteria for noise handling. Among them, the most commonly used are:

- *Encoding Length Restriction*: This heuristic used in the Inductive Logic Programming algorithm FOIL (Quinlan, 1990) is based on the Minimum Description Length principle (Rissanen, 1978). It tries to avoid learning complicated rules that cover only a few examples by making sure that the number of bits that are needed to encode a clause is less than the number of bits needed to encode the instances covered by it.[1] When no literal can be added without exceeding this limit, the incomplete clause is added provided that a certain percentage (usually 80%) of the examples it covers is positive.

- *Significance Testing* was first used in the propositional CN2 induction algorithm (Clark & Niblett, 1989) and later on in the relational learner mFOIL (Džeroski & Bratko, 1992). It tests for significant differences between the distribution of positive and negative examples covered by a rule and the overall distribution of positive and negative examples by comparing the likelihood ratio statistic to a χ^2 distribution with 1 degree of freedom at the desired significance level.[2] Insignificant rules are rejected.

- *The Cutoff Stopping Criterion* compares the heuristic evaluation of a literal to a user-set threshold and only admits literals that have an evaluation above this *cutoff*. This simple stopping criterion was first employed in the relational separate-and-conquer learning system FOSSIL (Fürnkranz, 1994a). As it forms the basis of the top-down pruning approach, which we will discuss in section 5, we describe FOSSIL in more detail in the following section.

3.1. FOSSIL

FOSSIL (Fürnkranz, 1994a) is a relational separate-and-conquer learning algorithm that uses a search heuristic based on statistical correlation.

Suppose FOSSIL has learned an incomplete clause, which currently covers m instances, p positive and n negative. Of these m instances, a candidate literal will cover c instances and leave u instances uncovered. Of the c covered instances, there will be p_c positive and n_c negative instances. Likewise, n_u negative and p_u positive examples will remain uncovered by the literal. An optimal literal will perfectly discriminate between all positive and negative instances, i.e., there will be no *false positives* ($n_c = 0$) and no *false negatives* ($p_u = 0$).

We now arbitrarily assign the numeric values $+1$ to positive examples and -1 to negative instances. Similarly we assign $+1$ to all covered instances and -1 to all uncovered instances. Each of the m instances is now associated with two numbers (PN, CU) and we will measure their correspondence by computing their correlation coefficient. The *correlation coefficient* of PN and CU is defined as

$$\begin{aligned} corr(PN, CU) = & \tfrac{\mathrm{E}((PN - \mathrm{E}(PN)) \times (CU - \mathrm{E}(CU)))}{\mathrm{Var}(PN) \times \mathrm{Var}(CU)} \\ = & \tfrac{\mathrm{E}(PN \times CU) - \mathrm{E}(PN) \times \mathrm{E}(CU)}{\mathrm{Var}(PN) \times \mathrm{Var}(CU)} \end{aligned} \tag{1}$$

The expected values in (1) will be estimated by the means

$$\mathrm{E}(PN) = \frac{p - n}{m}, \ \ \mathrm{E}(CU) = \frac{c - u}{m}, \tag{2}$$

As all values of PN and CU can only be $+1$ or -1, the variance can be simplified to

$$\mathrm{Var}(X) = \mathrm{E}(X^2) - \mathrm{E}(X)^2 = 1 - \mathrm{E}(X)^2 \tag{3}$$

Finally, as $PN \times CU$ is $+1$ for all covered positive and uncovered negative examples and -1 for all uncovered positive and covered negative examples, we get

$$\mathrm{E}(PN \times CU) = \frac{p_c + n_u - p_u - n_c}{m} \tag{4}$$

The partial results (2) – (4) will be substituted into the formula for the correlation coefficient (1) resulting in a value between -1 and $+1$. $corr(PN, CU) = +1$ indicates a perfect match between the new literal and the examples covered so far, whereas $corr(PN, CU) = -1$ shows that the literal classifies all positive examples as negative and vice versa, i.e., that its negation will be a perfect choice. A correlation value of 0 signals that the literal and the classification of the examples are independent. The literal L with the highest absolute value of the correlation coefficient (or its negation if the sign of the coefficient is negative) is finally chosen to extend the partial clause.

Utilizing the fact that each literal has an evaluation between 0 and 1, FOSSIL employs the *cutoff stopping criterion*. The user can require that all literals considered for clause extension must have a certain minimum correlation value, the *Cutoff* parameter. Different settings of the value will cause different amounts of pre-pruning. A setting of *Cutoff* $= 0.0$ results in learning a complete and consistent theory for the training set, because all correlation values are ≥ 0.0 and thus no literals will be excluded. On the other hand, an empty theory will be learned at *Cutoff* $= 1.0$, because only trivial learning problems have background literals with a correlation $= 1.0$ (e.g. `parent(A,B) :- child(B,A)`).

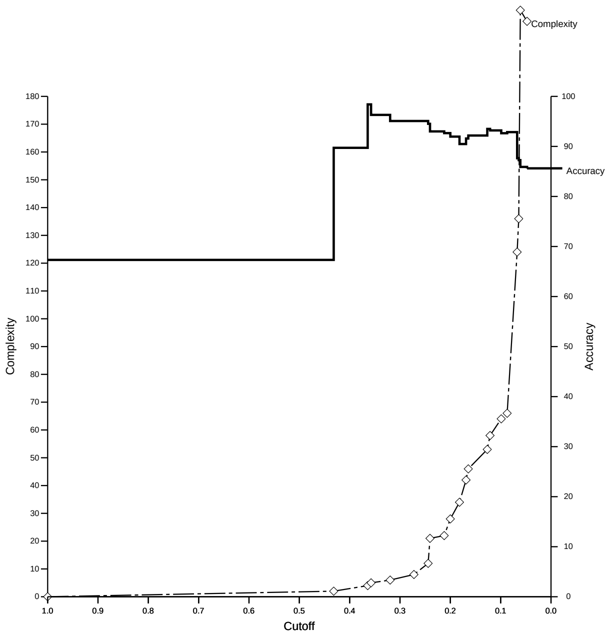


Figure 4. Accuracy and complexity vs. cutoff in a noisy domain

Figure 4 shows a typical plot of accuracy and theory complexity (number of literals in the learned theory) vs. different values of the cutoff parameter for the commonly used KRK endgame classification task using 500 training examples with 10% noise.[3] The most accurate theories are found for cutoff values between approximately 0.25 and 0.35. Higher cutoff values result in too simple theories, while lower settings of the cutoff obviously result in overfitting of the data. Also note that the differences in cutoff values for neighboring theories decrease with increasing theory complexity. Contrary to figure 8, where the most complex theory has been learned with a cutoff of 0.36 from a noise-free data set, this indicates that for lower cutoffs more and more literals have a correlation above this threshold, several of them fitting noisy examples by chance.

FOSSIL's cutoff parameter may therefore be viewed as a means for directly controlling the *Overfitting Avoidance Bias* (Schaffer, 1993, Wolpert, 1993). A setting of *Cutoff* $= 0.3$ is a good general heuristic which seems to be independent of the noise level in the data. Fürnkranz (1994a) discusses this in more detail and compares FOSSIL to FOIL and mFOIL.

4. Post-pruning

While pre-pruning approaches try to avoid overfitting during rule generation, *post-pruning* approaches at first ignore the problem of overfitting and learn a complete and consistent theory. The quality of this theory is then estimated with some quality measure (usually predictive accuracy). If this quality measure can be improved by simplifying the theory, this will be repeatedly done until all further simplifications would harm the quality of the theory.

Post-pruning approaches have been commonly used in decision tree learning algorithms (Breiman, Friedman, Olshen, & Stone, 1984, Quinlan, 1987, Niblett & Bratko, 1987) Mingers (1989) and Esposito, Malerba, and Semeraro (1993) present an overview and comparison of various approaches.

4.1. Reduced Error Pruning

The most common among these methods is *Reduced Error Pruning (REP)*. Pagallo and Haussler (1990), Weiss and Indurkhya (1991), and Brunk and Pazzani (1991) employ a straightforward adaptation of REP to the separate-and-conquer rule learning framework. First, the training data are split into two subsets: a *growing set* (usually 2/3) and a *pruning set* (1/3). In the first phase, no attention is paid to the noise in the data and a concept description that covers all of the positive and none of the negative examples is learned from the growing set. The resulting theory is then repeatedly simplified by greedily deleting literals and rules from the theory until any further deletion would result in a decrease of predictive accuracy as measured on the pruning set. Pseudo-code for this algorithm is given in figure 5.

The subroutine BESTSIMPLIFICATION selects the theory with the highest accuracy on the pruning set from the set of simplifications of the current theory. Simplifications that are usually tried are deleting an entire clause, or deleting the last literal of a clause. Variants of REP can also employ additional simplification operators like deleting each literal of a clause, deleting a final sequence of literals (Cohen, 1993), or finding the best replacement of a literal (Weiss & Indurkhya, 1991). If the accuracy of the best simplification is not below the accuracy of the unpruned theory, REP will continue to prune the new theory. This is repeated until the accuracy of the best pruned theory is below that of its predecessor.

REP has been shown to learn more accurate theories than the pre-pruning algorithm FOIL in the KRK domain at several levels of noise (Brunk & Pazzani, 1991). However,

```
procedure PostPruning(Examples, SplitRatio)

SplitExamples(SplitRatio, Examples, GrowingSet, PruningSet)
Theory = SeparateAndConquer(GrowingSet)
loop
    NewTheory = BestSimplification(Theory,PruningSet)
    if Accuracy(NewTheory,PruningSet) <
      Accuracy(Theory,PruningSet)
      exit loop
    Theory = NewTheory
return(Theory)
```

Figure 5. A post-pruning algorithm

this straightforward adaptation of REP brings several problems for separate-and-conquer rule-learning algorithms, as we will see in the next section.

4.2. Problems with Reduced Error Pruning

Although REP is quite effective in raising predictive accuracy in noisy domains (Brunk & Pazzani, 1991), it has several shortcomings.

Complexity

REP's time complexity has been shown to be as bad as $\Omega(n^4)$ for noisy data (n being the number of examples).[4] We will follow Cohen (1993) and present an intuitive sketch of the arguments used in deriving these results. Formal proofs have been derived by Cameron-Jones (1996). We assume a propositional dataset consisting of n examples, each described with a number of binary attributes that is fixed and independent of n.[5] A constant fraction of the examples has a random classification, i.e., $\Theta(n)$ examples form incompressible noise. Following Cohen (1993) we further assume that each noisy example has to be covered by a separate rule, i.e., noisy data will produce $\Theta(n)$ rules.[6] Each of these rules will need $\Theta(\log n)$ tests for discriminating the noisy example from other examples, because each test for a binary attribute will exclude about half of the random instances. Thus, the size of the overfitting theory is $\Theta(n \log n)$ literals.

In each step of the pruning phase, each of the $\Theta(n)$ clauses can be simplified by deleting the last literal or deleting the whole clause, i.e., the theory can be simplified in $\Theta(n)$ different ways. Each simplification has to be tested on the pruning set in order to select the simplification with the highest accuracy. For the $\Theta(n)$ examples of the pruning set that will be classified as negative by a theory, at least the first literals of all $\Theta(n)$ clauses have to be evaluated, so that the complexity of testing a theory is $\Omega(n^2)$. If we further assume that REP works as intended, it should prune the overfitting theory to the correct theory, whose size should be independent of the size of training set (provided it has a certain minimum size). Therefore, REP must at least remove all but a constant number of the $\Theta(n)$ clauses of the overfitting theory, i.e., it has to loop $\Omega(n)$ times (when it frequently prunes single literals there may be considerably more iterations). Thus, we get a total cost of $\Omega(n^4)$.[7]

With similar arguments, Cohen (1993) has derived a complexity bound of $\Omega(n^2 \log n)$ for the initial growing phase. This lower bound is tight, because each of the $\Theta(n \log n)$ literals in the overfitting theory has been tested on at most $O(n)$ examples of the growing set. This result shows that the costs of pruning will outweigh the costs of generating the initial concept description, which already are higher than the costs of using a pre-pruning algorithm that entirely avoids overfitting.

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 greedily maximized. When each possible operator leads to a decrease in predictive accuracy, the search process stops at this local maximum.

However, in noisy domains the theory that has been generated in the growing phase will be much too complex (see figure 4). REP has to prune a significant portion of this theory and has ample opportunity to err on its way. Therefore, we can expect REP's complex-to-simple search to be not only slow, but also inaccurate on noisy data.

Separate-and-conquer strategy

Post-pruning algorithms originate from research in decision tree learning where usually the well-known *divide-and-conquer* learning strategy is used. At each node, the current training set is divided into disjoint sets according to the outcome of the chosen test. After this, the algorithm is recursively applied to each of these sets independently.

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. Figure 6a illustrates how post-pruning works in decision tree learning. The right half of the overfitting tree covers the sets C and D of the training instances. When the pruning algorithm decides to prune these two leaves, their ancestor node becomes a leaf that now covers the examples $C \cup D$. The left branch of the decision tree is not influenced by this operation.

On the other hand, pruning a literal from a clause means that the clause is generalized, i.e., it will cover more positive and negative instances. Consequently, those additional positive and negative instances should be removed from the training set so that they cannot influence the learning of subsequent clauses. In the example of figure 6b, the first of three rules is simplified and now covers not only the examples its original version has covered, but also all of the examples that the third rule has covered and several of the examples that the second rule has covered. While the third rule could easily be removed by a post-pruning algorithm, the situation is not as simple with the remaining set of examples B2. The second rule will

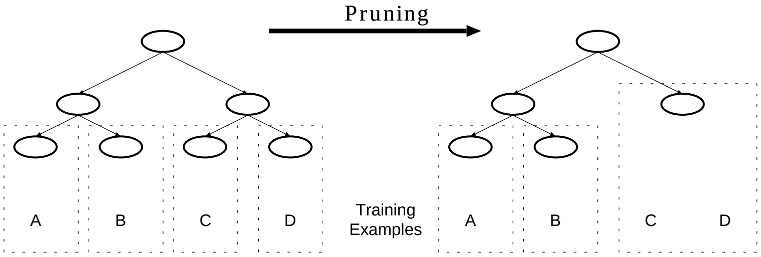


Figure 6a. Post-pruning in divide-and-conquer learning algorithms

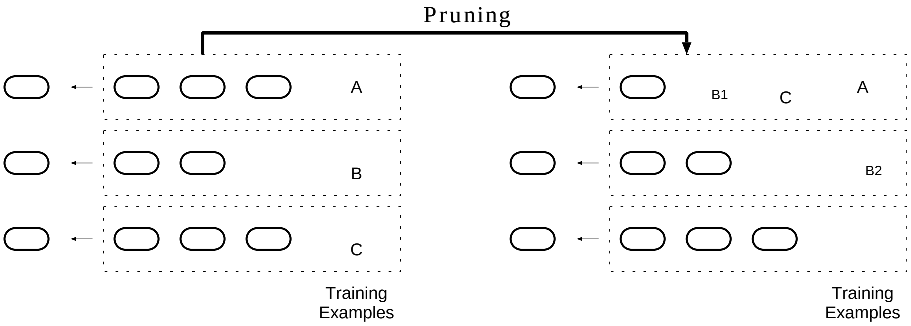


Figure 6b. Post-pruning in separate-and-conquer learning algorithms.

naturally cover all examples of the set B2, because it has been learned in order to cover the examples of its superset B. However, it might well be the case that a different rule could be more appropriate for discriminating the positive examples in B2 from the remaining negative examples. As pruning literals from a clause can only generalize the concept, i.e., increase the set of covered examples, a post-pruning algorithm has no means for adjusting the second rule to this new situation. Thus, the learner may be lead down a garden path, because the set of examples that remain uncovered by the unpruned clauses at the beginning of a theory may yield a different evaluation of candidate literals for subsequent clauses than the set of examples that remain uncovered by the pruned versions of these clauses. A wrong choice of a literal cannot be undone by pruning.

4.3. The Grow algorithm

To solve some of the problems of section 4.2, in particular efficiency, Cohen (1993) has proposed a top-down post-pruning algorithm based on a technique used by Pagallo and Haussler (1990). Like REP, the Grow algorithm first finds a theory that overfits the data. But instead of pruning this intermediate theory until any further deletion results in a decrease of accuracy, it uses it to grow a pruned theory. For this purpose, GROW augments the intermediate theory with generalizations of all its clauses by repeatedly deleting a final sequence of literals from each clause so that its error on the *growing* set increases the least. Then, it iteratively selects clauses from this expanded theory to form the final concept description. When no clause further improves this theory's predictive accuracy on the pruning set, GROW stops.

Thus, GROW improves upon REP by replacing the bottom-up hill-climbing search of REP with a top-down approach. Instead of removing the most useless clause or literal from the overfitting theory, it adds the most promising generalization of a rule to an initially empty theory. It has been experimentally confirmed that this results in a significant gain in efficiency, along with a slight gain in accuracy (Cohen, 1993). However, Cameron-Jones (1996) has demonstrated that the asymptotic time complexity of the GROW post-pruning method is still above the complexity of the initial rule growing phase.

The explanation for the speedup that can be gained with the top-down strategy is that it starts from the empty theory, which in many noisy domains is much closer to the final theory than the overfitting theory. For example, compare the complexities of the most complex theory (*Cutoff* $= 0.0$) and the best theories ($0.25 \leq Cutoff \leq 0.35$) in figure 4.

Thus, it is not surprising that GROW has been shown to outperform REP on a variety of data sets (Cohen, 1993).[8] Nevertheless, it still suffers from the inefficiency caused by the need of generating an overly specific theory in a first pass.

5. Combining pre- and post-pruning

In section 4, we have seen that the intermediate theory resulting from the initial overfitting phase can be much more complex than the final theory. Post-pruning is very inefficient in this case, because most of the work performed in the learning phase has to be undone in the pruning phase.

A natural solution to this problem would be to start the pruning phase with a simpler theory. Cohen (1993) has first investigated this idea by combining the efficient post-pruning algorithm GROW (see section 4.3) with some weak pre-pruning heuristics that speed up the learning phase. The goal of pre-pruning in this context is not to entirely prevent overfitting, but to reduce its amount so that a subsequent post-pruning phase has to do less work and is less likely to go wrong.

However, there is always the danger that a predefined stopping criterion will return an overly simple theory. In this section, we will therefore discuss an alternative approach that searches for an appropriate starting point for the post-pruning phase.

5.1. Top-Down Pruning

One advantage of FOSSIL's simple and efficient cutoff stopping criterion is its closeness to the search heuristic (see section 3.1) . FOSSIL needs to do a mere comparison between the heuristic value of the best candidate literal and the cutoff value in order to decide whether to

```
procedure AllTheories(Examples)

$Cutoff = 1.0$
$Theories = \emptyset$
while ($Cutoff > 0.0$) do
        $Theory = $ Fossil(Examples,Cutoff)
        $Cutoff = $ MaximumPrunedCorrelation(Theory)
        $Theories = Theories \cup Theory$
return(Theories)
```

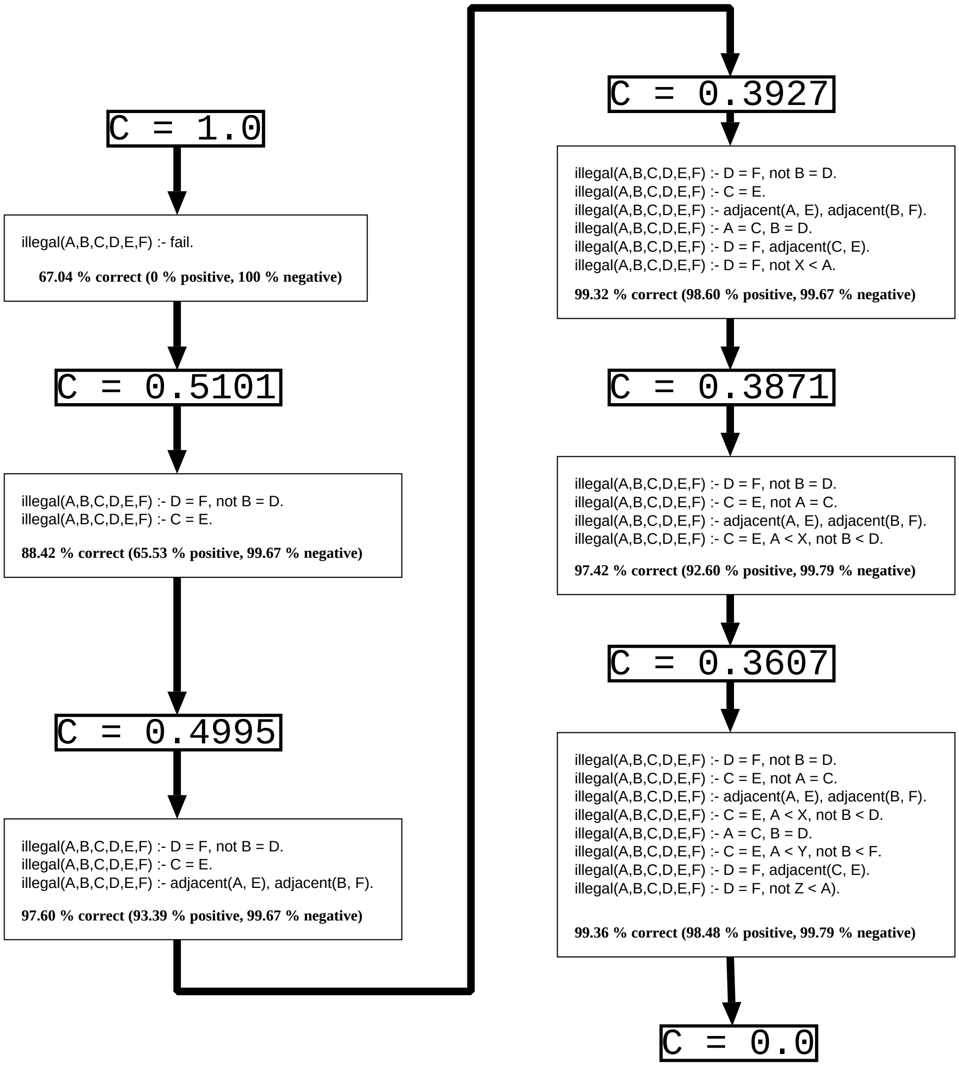
Figure 7. Algorithm to generate all theories learnable by Fossil

add the candidate literal to the clause at hand or not. This property can be used to generate *all* theories that could be learned by Fossil with any setting of the cutoff parameter (see figure 7).

The basic idea behind this algorithm is the following: Assume that Fossil is trying to learn a theory with a cutoff of 1.0. Unless there is one literal in the background knowledge that perfectly discriminates between positive and negative examples, we will not find a literal with a correlation of 1.0 and thus learn an empty theory.

However, we can remember the literal that had the maximum correlation and use this information in the following way: If we make another call to Fossil with the cutoff set to exactly this maximum correlation value, at least one literal (the one that produced this maximum correlation) will be added to the theory, typically followed by several other literals that have a correlation value higher than the new cutoff. The result is a new theory, which usually is a little more complex than its predecessor. Again, the maximum correlation of the literals that have been cut off will be remembered. Obviously, for all values between the old cutoff and the new maximum, the same theory would have been learned. Thus, we can choose this value as the cutoff for the next run. It can also be expected that the new theory will be more complex than the previous one. This process is repeated until at a certain setting of the *Cutoff* no further literal is pruned (MaximumPrunedCorrelation $= 0.0$) and thus the most complex theory has been reached.

Figure 8 shows a complete series of theories generated by Fossil from 1000 noise-free examples in the domain of distinguishing legal from illegal positions in the KRK endgame domain.[3] Any setting of the cutoff parameter would yield one of these six theories (on the same training set). It can be seen that the theories are generated roughly in a simple-to-complex order (*top-down*).[9] As simpler theories can be expected to be more accurate in noisy domains, the best theories will be learned after a few iterations. Therefore, it may be possible to stop the generation of theories as soon as a reasonably good theory has been found in order to avoid expensive learning of many overly-specific theories. This may save a lot of work, as figure 4 indicates. Besides, it is also possible to reuse parts of the previous theory (up to the point where the highest cutoff has occurred) so that the total cost of generating a complete series of concept descriptions may not be much higher than the



```
illegal(A,B,C,D,E,F) :- fail.

67.04 % correct (0 % positive, 100 % negative)
```

```
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E.

88.42 % correct (65.53 % positive, 99.67 % negative)
```

```
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E.
illegal(A,B,C,D,E,F) :- adjacent(A, E), adjacent(B, F).

97.60 % correct (93.39 % positive, 99.67 % negative)
```

```
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E.
illegal(A,B,C,D,E,F) :- adjacent(A, E), adjacent(B, F).
illegal(A,B,C,D,E,F) :- A = C, B = D.
illegal(A,B,C,D,E,F) :- D = F, adjacent(C, E).
illegal(A,B,C,D,E,F) :- D = F, not X < A.

99.32 % correct (98.60 % positive, 99.67 % negative)
```

```
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E, not A = C.
illegal(A,B,C,D,E,F) :- adjacent(A, E), adjacent(B, F).
illegal(A,B,C,D,E,F) :- C = E, A < X, not B < D.

97.42 % correct (92.60 % positive, 99.79 % negative)
```

```
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E, not A = C.
illegal(A,B,C,D,E,F) :- adjacent(A, E), adjacent(B, F).
illegal(A,B,C,D,E,F) :- C = E, A < X, not B < D.
illegal(A,B,C,D,E,F) :- A = C, B = D.
illegal(A,B,C,D,E,F) :- C = E, A < Y, not B < F.
illegal(A,B,C,D,E,F) :- D = F, adjacent(C, E).
illegal(A,B,C,D,E,F) :- D = F, not Z < A).

99.36 % correct (98.48 % positive, 99.79 % negative)
```

Figure 8. Generating a series of theories from 1000 noise-free examples in the KRK domain

```
procedure TDP(Examples, SplitRatio)

Cutoff $= 1.0$
BestTheory $= \emptyset$
BestAccuracy $= 0.0$
SPLITEXAMPLES(SplitRatio, Examples, GrowingSet, PruningSet)
repeat
    NewTheory = FOSSIL(GrowingSet,Cutoff)
    NewAccuracy = ACCURACY(NewTheory,PruningSet)
    if NewAccuracy $>$ BestAccuracy
        BestTheory = NewTheory
        BestAccuracy = NewAccuracy
        LowerBound = BestAccuracy $-$ STANDARDERROR(BestAccuracy,PruningSet)
    Cutoff = MAXIMUMPRUNEDCORRELATION(NewTheory)
until (NewAccuracy $<$ LowerBound) or (Cutoff $= 0.0$)
loop
    NewTheory = BESTSIMPLIFICATION(Theory,PruningSet)
    if ACCURACY(NewTheory,PruningSet) $<$ ACCURACY(Theory,PruningSet)
        exit loop
    Theory = NewTheory
return(Theory)
```

Figure 9. Combining pre- and post-pruning with *Top-Down Pruning*.

cost of generating merely the most complex theory (at least in cases where the cutoff occurs near the end of the learned theory, which is frequently the case).

After some experimentation where we tried to automatically select the best theory (Fürnkranz, 1994a) or the best cutoff parameter (in a way very similar to CART's cost-complexity pruning (Breiman, Friedman, Olshen, & Stone, 1984)), we found that pre-pruning is too rigid and decided to combine it with the flexibility of post-pruning. The algorithm of figure 9 uses the basic algorithm of figure 7 not to find the best theory, but — in order to avoid too simple theories — to find the most complex among all reasonably good theories that can be learned by FOSSIL. This theory is then used as a starting point for a post-pruning phase. More precisely, theories are generated in a simple-to-complex order and evaluated on a designated test set of the data (usually $1/3$). When the measured classification accuracy of one of the theories falls below the measured accuracy of the best theory so far minus one *standard error*, no more theories will be generated and the last theory within the 1-SE margin will be submitted to the REP algorithm as described in section 4.1.[10] Usually this theory will be a little too complex, but simple enough so that the final theory can be found with a small amount of post-pruning. Because of this initial top-down simple-to-complex search for a good theory, we have named the method *Top-Down Pruning* (TDP).

If this algorithm succeeds in finding a starting theory that is close to the final theory, we can expect our algorithm to be faster than basic REP, because the initial search for a good starting theory will

- *speed up the growing phase*, because the most expensive theories will not be generated,[11]

- *speed up the pruning phase*, because pruning starts from a simpler theory and thus the number of possible pruning operations is smaller.

In preliminary experiments, it turned out that lowering the cutoff may in some cases result in specializations of the rules learned in the previous iteration without learning new rules that would cover the positive examples that are no longer covered by the specialized rules. This may result in theories that cover only a small fraction of the available positive examples, which can cause a sudden drop in accuracy. The problem can be easily avoided by forcing TDP to learn additional rules that cover the remaining examples. Thus, we have added the constraint that only theories that cover at least 50% of the positive examples in the growing set will be evaluated on the pruning set. If a theory does not satisfy this criterion, it will be improved by adding more clauses. This is achieved by lowering the cutoff to the value that would be needed to start a new clause.[12] If the clauses that are added during this phase do not improve the predictive accuracy on the pruning set, they will be removed in the post-pruning phase. A possible danger of this technique is that it can force TDP to learn overly complex theories in domains where only a small fraction of the positive examples can be covered by predictive rules.

5.2. Experimental results

We have compared *Top-Down Pruning* (TDP) to two algorithms, Fossil* and *Reduced Error Pruning* (REP) in the KRK endgame domain with 10% artificial noise added.[3] Fossil* is a version of the pre-pruning algorithm Fossil that cheats by consulting the test data during learning. It uses the algorithm described in figure 7 to generate all theories learnable by Fossil from the entire training set, evaluates each of them on the test set of 5000 noise-free examples, and selects the one with the highest accuracy. We have included this algorithm as an upper bound for the accuracy that could have been obtained by learning with pre-pruning alone.

All algorithms have been implemented in PROLOG. REP and TDP split the training data into the same growing (ca. $2/3$) and pruning sets (ca. $1/3$). In order to exclude possible influences from the underlying learning algorithm, we ran REP using Fossil with *Cutoff* $= 0.0$ as its basic learning module.[13]

Table 1. Accuracy in the KRK domain with 10% noise.

Average Accuracy (10 runs)		100	250	500	750
Fossil*		**92.64**	**95.68**	**98.00**	**98.28**
REP	Before Pruning	84.84	86.88	87.11	89.21
	After Pruning	**94.67**	**96.72**	**97.80**	**98.51**
TDP	Before Pruning	89.15	91.02	95.89	95.85
	After Pruning	**95.14**	**95.93**	**98.29**	**98.70**

Table 1 shows the results of some experiments with varying training set sizes. TDP is constantly better than FOSSIL* which shows that it would outperform FOSSIL in this domain with any (fixed or dynamic) setting of the cutoff parameter. REP is also slightly better than FOSSIL* which confirms earlier results where REP was shown to outperform FOIL (Brunk & Pazzani, 1991). Thus, using post-pruning in some form seems to be a reasonable approach for improving accuracy.

REP was only better than TDP at a training set size of 250, where TDP heavily overpruned in one of the 10 cases: TDP started off with a theory that was 98.42% correct, but unfortunately one of the literals had no support in the pruning set and consequently was pruned, thus yielding a theory with a mere 81.34%. This did not happen to REP because it got caught in a 91.36% correct theory, and did not even get to the 98.42% theory. With increasing training set sizes, TDP seems to be slightly superior to REP, although the differences are not statistically significant.

A comparison of the accuracies of the intermediate theories reveals that TDP starts with significantly better theories than REP. Obviously, the top-down search for better starting theories is successful. In particular at higher training set sizes, REP sometimes gets stuck in a local optimum and returns bad theories. However, we have seen above that REP may profit from this in some rare cases. TDP is less likely to get stuck in a local optimum during pruning because it starts with an initial theory that is already quite close to the final theory. The problem of local optima with greedy hill-climbing is also not likely to appear in TDP's top-down search for a starting theory, because (at least in this domain) the intermediate theories usually appear after only a few iterations of TDP's top-level loop.

Table 2. Run-time in the KRK domain with 10% noise.

Average	*Run-time (10 runs)*	100	250	500	750
REP	Growing	6.66	75.22	397.17	845.76
	Pruning	2.93	91.46	1248.48	2922.66
	Total	**9.59**	**166.68**	**1645.65**	**3768.42**
TDP	Growing	7.23	51.37	80.17	190.66
	Pruning	1.24	22.49	16.39	151.52
	Total	**8.47**	**73.86**	**96.56**	**342.18**

Comparing the run-times of REP and TDP (table 2) confirms that TDP is significantly faster than REP. In fact, it is even faster than REP's initial phase of overfitting alone. TDP only has to find a few fairly simple theories, while REP generates huge theories that fit all the noisy examples. Expectedly, with increasing training set sizes, the costs of REP are dominated by the pruning process. TDP on the other hand, even manages to decrease pruning time with training set sizes 250 to 500. The significant run-time increase from 500 to 750 examples is mainly due to one of the 10 sets, where a much too complex theory was learned in 855.94 CPU secs. growing and 1399.35 CPU secs. pruning time. For the remaining 9 sets, the average run-time was 116.74 CPU secs. for growing and 12.88 CPU secs. for pruning.

These results confirm that combining pre- and post-pruning is a good idea. TDP is more accurate than pre-pruning and faster than post-pruning in both, learning *and* pruning. The starting theories learned by FOSSIL become increasingly more accurate as the training set grows, which means that not only learning will be faster, but also less and less pruning has to be performed. The problem of the incompatibility of separate-and-conquer learning with post-pruning as discussed in section 4.2 is also alleviated when the starting theory is already close to the final theory. Nevertheless, we have seen that in one of the experiments with a training set size of 750 the search for a good initial theory failed resulting in an increase in pruning time by two orders of magnitude. Motivated by the observation that TDP might fail occasionally, because we have only alleviated, but not solved the problems discussed in section 4.2, we have developed an algorithm that *integrates* pre- and post-pruning.

6. Integrating pre- and post-pruning

The algorithm that we will present in this section was mainly motivated by the observation that post-pruning is incompatible with the separate-and-conquer learning strategy as we have discussed in section 4.2. The problem is that post-pruning approaches do not take into account that pruning a clause will generalize it so that it will eventually cover more examples of the training set. This may have a considerable influence on the evaluation of candidate literals for subsequent clauses.

6.1. Incremental Reduced Error Pruning

The basic idea of *Incremental Reduced Error Pruning* (I-REP) is that instead of first growing a complete concept description and pruning it thereafter, each individual clause will be pruned right after it has been generated. This ensures that the algorithm can remove the training examples that are covered by the pruned clause before subsequent clauses are learned thereby preventing these examples from influencing the learning of subsequent clauses.

Figure 10 shows pseudo-code for this algorithm. As usual, the current set of training examples is split into a growing (usually 2/3) and a pruning set (usually 1/3). However, not an entire theory, but only one clause is learned from the growing set. Then, literals are deleted from this clause in a greedy fashion until any further deletion would decrease the accuracy of this clause on the pruning set.[14] Single pruning steps can be performed by submitting a one-clause theory to the same BESTSIMPLIFICATION subroutine used in REP or, as in our implementation, one can use a more complex pruning operator that considers every literal in a clause for pruning. The best rule found by repeatedly pruning the original clause is added to the concept description and all covered positive and negative examples are removed from the training — growing *and* pruning — set. The remaining training instances are then redistributed into a new growing and a new pruning set to ensure that each of the two sets contains the predefined percentage of the remaining examples. From these sets the next 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

```
procedure I-REP (Examples, SplitRatio)

$Theory = \emptyset$
while Positive(Examples) $\neq \emptyset$
      $Clause = \emptyset$
      SplitExamples(SplitRatio, Examples, GrowingSet, PruningSet)
      $Cover = GrowingSet$
      while Negative(Cover) $\neq \emptyset$
            $Clause = Clause \cup$ FindLiteral($Clause, Cover$)
            $Cover =$ Cover(Clause,Cover)
      loop
           NewClause = BestSimplification(Clause,PruningSet)
           if Accuracy(NewClause,PruningSet) $<$ Accuracy(Clause,PruningSet)
              exit loop
           Clause = NewClause
      if Accuracy(Clause,PruningSet) $\leq$ Accuracy(fail,PruningSet)
         exit while
      $Theory = Theory \cup Clause$
      $Examples = Examples - Cover$
return(Theory)
```

Figure 10. Integrating pre- and post-pruning with *Incremental Reduced Error Pruning*

clause is not 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. Post-pruning methods are used as pre-pruning heuristics.

Because 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). We can expect I-REP to improve upon post-pruning algorithms, because it is aimed at solving the problems we discussed in section 4.2:

Complexity: Using the same assumptions as in the analysis of the complexity of REP (section 4.2), we will show that I-REP's asymptotic complexity is $O(n \log^2 n)$, n being the size of the training set. The cost of growing one clause in REP is $O(n \log n)$, because for selecting each of the $\Theta(\log n)$ literals. Thus, a constant number of conditions is tested against $O(n)$ examples. I-REP considers *every* literal in the clause for pruning. Therefore, each of the $\Theta(\log n)$ literals has to be evaluated on the $\Theta(n)$ examples in the pruning set until the final clause has been found, i.e., at most $O(\log n)$ times. Thus, the cost of pruning one clause is $O(n \log^2 n)$. Assuming that I-REP stops when the correct theory of constant size has been found, the overall cost is also $O(n \log^2 n)$. This is significantly lower than the cost of growing an overfitting theory which has been shown to be $\Omega(n^2 \log n)$ under the same assumptions (Cohen, 1993).

Bottom-up hill-climbing: Like Grow, I-REP uses a top-down approach instead of REP's bottom-up search: Final theories are not found by removing unnecessary clauses and literals from an overly complex theory, but by repeatedly adding clauses to an initially empty theory. However, while Grow first generates an overfitting theory and thereafter selects the best generalized clauses from this theory, I-REP selects the best generalization of a clause right after the clause has been learned.

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 are learned, each clause is completed (learned *and* pruned) and all covered examples are removed. Therefore, the I-REP approach eliminates the problem of incompatibility between the separate-and-conquer learning strategy and the reduced-error pruning strategy.

6.2. Experimental results

Table 3 shows a comparison of the run-times of post-pruning algorithms and I-REP in the KRK domain with 10% artificial noise added.[3] All algorithms used FOIL's information gain criterion as a search heuristic. The column *Initial Rule Growth* refers to the initial growing phase that REP and GROW have in common, while the columns REP and GROW give the results for the pruning phases only. The total run-time of REP (GROW) is the run-time of *Initial Rule Growth* plus the run-time of REP (GROW). In I-REP, both phases are tightly integrated so that only the total value of the run-time can be given.

Table 3. Average run-time

Domain	Initial Rule Growth	REP	GROW	I-REP
KRK-100 (10%)	8.36	2.44	1.66	4.20
KRK-250 (10%)	91.31	104.98	19.81	17.30
KRK-500 (10%)	456.56	1578.16	100.81	46.32
KRK-750 (10%)	1142.78	7308.84	361.41	83.64
KRK-1000 (10%)	2129.89	23125.34	806.89	115.35

It is obvious that I-REP is significantly faster than the post-pruning algorithms. In fact, it is always faster than REP's and GROW's initial growing phase alone, because I-REP avoids learning an intermediate overfitting theory. It can also be seen that GROW's pruning algorithm is much faster than REP's, which confirms earlier results (Cohen, 1993).

In order to get an idea on the asymptotic complexity of the various algorithms, we have performed a log-log analysis (Cameron-Jones, 1996). Note that the KRK domain can be reformulated as a binary problem using examples that are represented with one attribute for each possible condition that can appear in the body of a rule (Lavrač, Džeroski, & Grobelnik, 1991). Noise was simulated by flipping the classification of a fixed percentage of the training examples. Thus, this domain conforms to the assumptions made in the complexity analysis of I-REP and REP. The asymptotic time complexity of an algorithm can be empirically estimated by dividing the differences between the logarithms of two run-times by the differences of the logarithms of the corresponding training set sizes. Table 4 suggests

Table 4. Log-log analysis of the run-times on noisy KRK data.

Domain	Initial Rule Growth	REP	GROW	I-REP
100-250	2.61	4.11	2.71	1.54
250-500	2.32	3.91	2.35	1.42
500-750	2.26	3.78	3.15	1.46
750-1000	2.16	4.00	2.79	1.12

that I-REP has a sub-quadratic time complexity. It is consistent with our conjecture that I-REP's time complexity is $O(n \log^2 n)$. In general, the results we get are consistent with previous analyses (Cohen, 1993, Cameron-Jones, 1996). In particular, the evidence supports the result that REP has a complexity of $\Omega(n^4)$ and that the initial rule growing phase is $O(n^2 \log n)$. It also confirms the main result of Cameron-Jones (1996), namely that the asymptotic complexity of GROW is also above the asymptotic complexity of the initial rule growing phase. However, in all our experiments the absolute values for the run-time of GROW's pruning phase were negligible compared to the initial overfitting phase.

REP often gets caught in local maxima and is not able to simplify the theories to the right level. Interestingly we have observed that, despite its top-down search strategy, GROW also occasionally overfits the noise in the data, a phenomenon that has also been predicted by Cameron-Jones (1996). I-REP, on the other hand, will stop generating clauses whenever it has found a clause that has no support in the pruning set. Therefore, I-REP can be expected to have very fast run-times on purely random data (where REP and GROW are most expensive), because there is a high chance that the first clause will not fit any of the examples in the pruning set. This will stop the algorithm immediately without accepting a single clause and thus effectively avoid overfitting.

Table 5. Average accuracy

Domain	Initial Rule Growth	REP	GROW	I-REP
KRK-100 (10%)	85.29	91.77	91.60	84.55
KRK-250 (10%)	83.79	96.29	95.91	98.34
KRK-500 (10%)	84.29	97.62	98.17	98.48
KRK-750 (10%)	85.17	97.47	98.31	98.86
KRK-1000 (10%)	85.65	98.01	98.30	99.55

In terms of accuracy (table 5), I-REP also is superior to the post-pruning algorithms, although it seems to be more sensitive to small training set sizes. 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 into new growing and pruning sets before learning a new clause cannot help when there is little redundancy in the data because of small sample sizes. However, at larger example set sizes I-REP outperforms the other algorithms.

7. Experimental evaluation

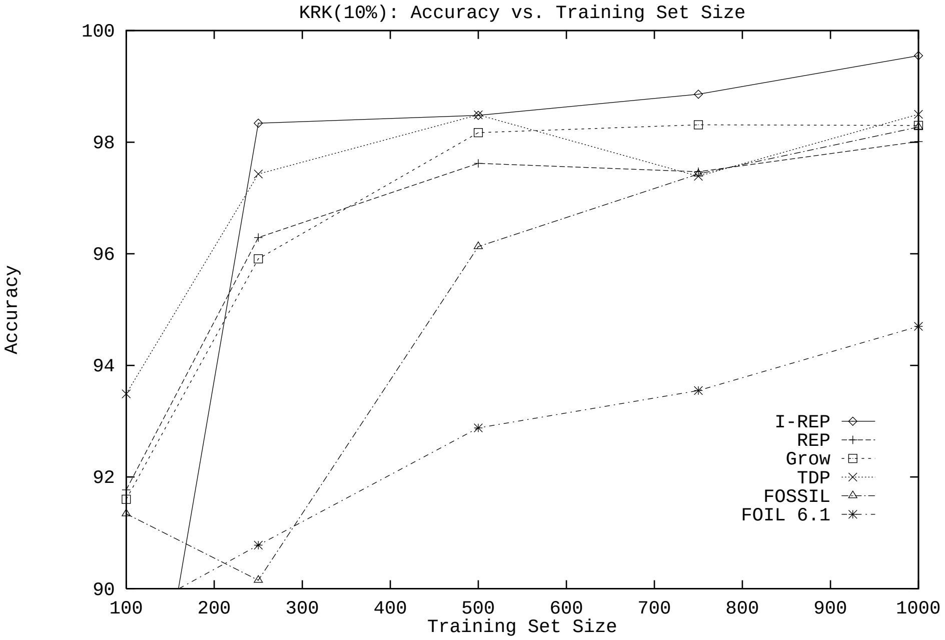
We have tested the algorithms presented in this paper on a variety of domains. All algorithms except Quinlan's FOIL (Quinlan & Cameron-Jones, 1995), which is written in C,[15] were implemented in SICStus PROLOG and had major parts of their implementations in common. In particular, they shared the same interface to the data and used the same procedures for splitting the training sets. Mode, type and symmetry information about the background relations was used to restrict the search space wherever applicable. Information gain was used as a search heuristic for REP, GROW and I-REP, and FOSSIL's correlation heuristic was used in FOSSIL and TDP. Run-times were measured in CPU seconds for SUN SPARCstations ELC. All algorithms were tested on identical training and test sets.

7.1. Summary of the experiments in the KRK domain

First, we will summarize the experiments in the domain of recognizing illegal chess positions in the KRK endgame (Muggleton, Bain, Hayes-Michie, & Michie, 1989). This domain has become a standard benchmark problem for relational learning systems, as it cannot be solved in a trivial way by propositional learning algorithms, because the background knowledge has to contain relations like `X = Y`, `X < Y`, and `adjacent(X,Y)`.

Class noise in the training instances has been generated according to the *classification noise process* (Angluin & Laird, 1988). In this model, a noise level of η means that the sign of each example is reversed with a probability of η.[16] The learned concepts were evaluated on test sets with 5000 noise-free examples. FOIL 6.1 was used with its default settings except that the `-V` option was set to 0 to avoid the introduction of new variables, which is not necessary for this task. All the other algorithms had their argument modes declared as input, which has the same effect. All algorithms were trained on identical sets of sizes from 100 to 1000 examples. All reported results are averages over 10 runs, except for the training set size 1000, where only 6 runs have been performed, because of the complexity of this task for some algorithms.

Figure 11 shows curves for accuracy and run-times over 5 different training set sizes. I-REP — after a bad start with only 84.55% accuracy on 100 examples — achieves the highest accuracy. In predictive accuracy, FOIL did poorly. Its stopping criterion (encoding length) is dependent on the training set size and thus too weak to effectively prevent overfitting the noise. From 1000 examples FOIL learns concepts that have more than 20 rules and are incomprehensible (Fürnkranz, 1994a). I-REP, on the other hand, consistently produces a 99.57% correct, understandable 4-rule approximation of the correct concept description. This theory correctly identifies all illegal positions, but is over-general for legal positions where the white king is between the black king and the white rook thus blocking a check



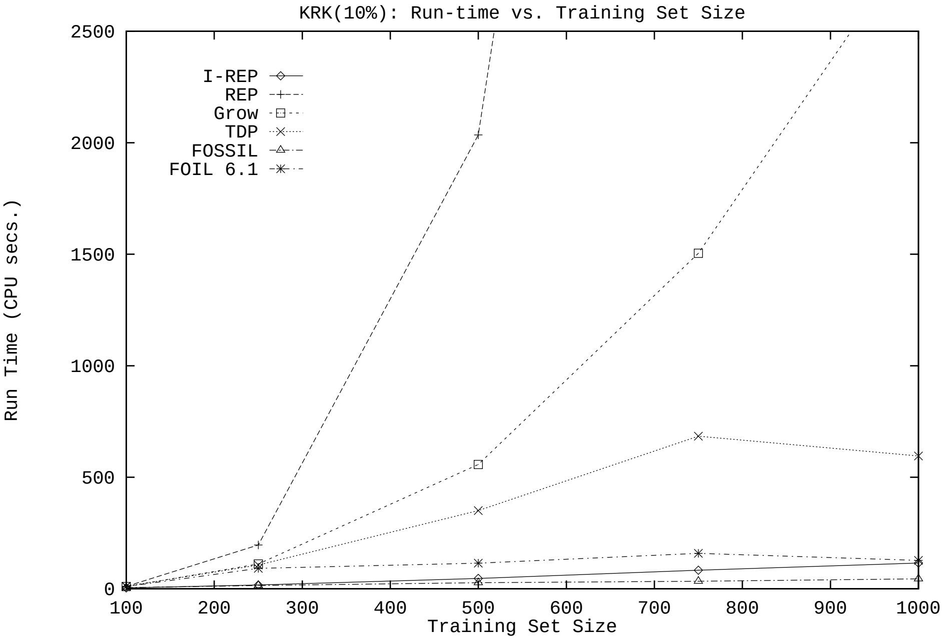


Figure 11. KRK domain (10% noise), different training set sizes

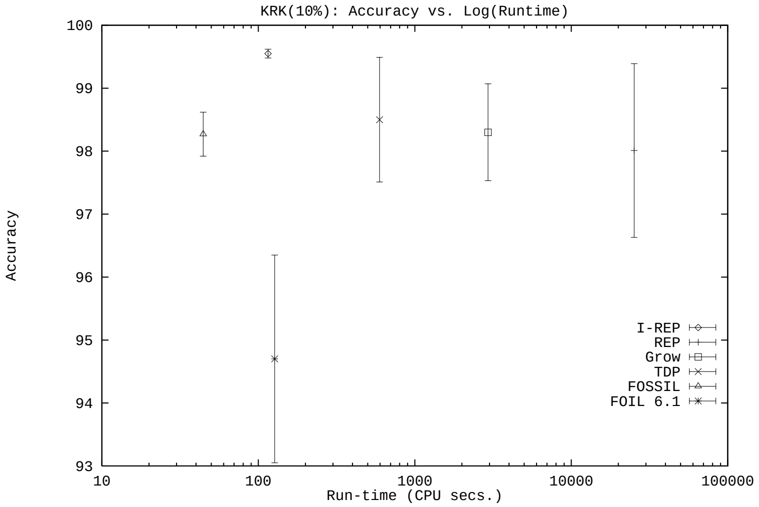


Figure 12. KRK domain (10% noise), 1000 examples

that would make the position illegal when white is to move. The post-pruning approaches REP and GROW are about equal, and TDP does not lose accuracy compared to them. All three, however, only rarely find the 4th rule that specifies that the white king and the white rook must not be on the same square. It can also be seen that the pre-pruning approach taken by FOSSIL needs many examples in order to make its heuristic pruning decisions more reliable.

On the other hand, FOSSIL is the fastest algorithm. FOIL, although implemented in C, is slower, because with increasing training set sizes it learns more complex theories than FOSSIL (Fürnkranz, 1994a). REP proves that its pruning method is very inefficient. GROW has an efficient pruning algorithm, but still suffers from the expensive overfitting phase. TDP is faster than REP and GROW, because it is able to start post-pruning with a much better theory than REP or GROW.

I-REP, however, learns a much better theory and is faster than both the growing and the pruning phase of TDP. In fact, I-REP is only a little slower than FOSSIL, but much more accurate. Thus, it can be said that it truly combines the merits of post-pruning (accuracy) and pre-pruning (efficiency). This also becomes apparent in figure 12, where accuracy (with the standard deviations observed in the different runs) is plotted against the logarithm of the run-time.

Table 6. Experiments in the mesh domain

Algorithm	Accuracy	Only +	Run-time
FOIL 6.3	90.27	23.32	20.74
FOSSIL	90.97	0.00	15.99
Initial Theory (REP & GROW)	87.42	31.47	6355.69
REP	88.74	26.86	28263.80
GROW	89.27	23.75	9880.32
Initial Theory (TDP)	88.99	28.89	3762.94
TDP	89.12	23.89	10111.27
I-REP	90.14	12.81	471.25

7.2. The mesh domain

We have also tested our algorithms on the finite element mesh design problem first studied and described in detail by Dolšak and Muggleton (1992). The problem of mesh design is to break complex objects into a number of finite elements in order to be able to compute pressure and deformations when a force is applied to the object. The basic problem during manual mesh design is the selection of an optimal number of finite elements on the edges of the structure. Several authors have tried ILP methods on this problem (Dolšak & Muggleton, 1992, Džeroski & Bratko, 1992, Quinlan, 1994). The available background knowledge consists of an attribute-based description of the edges and of topological relations between the edges.

The setup of our experiments was the same as in a previous study (Quinlan, 1994), i.e., we learned rules from four of the five objects in the data set and tested the learned concept on the fifth object. For estimating predictive accuracy, we have tested whether the learned theory can derive a given number of finite elements on an edge (Quinlan, 1994), while Džeroski and Bratko (1992) have used the percentage of all positive instances for which the learned theory predicts the right number of finite elements on an edge. To make sure that the learned theories are not over-general, we also had to test them on negative examples. In order to allow a rough comparison to earlier results (Džeroski & Bratko, 1992), we have added the accuracy on the positive examples to table 6, which can serve as an upper bound of the accuracy that would have been obtained by our algorithms using the other evaluation procedure. The given run-times are the total run-times (learning *and* pruning).

I-REP again is clearly faster than all post-pruning algorithms without losing predictive accuracy. TDP finds a more accurate starting theory than REP in a shorter time span. Consequently, its pruning time is much shorter than REP's and the learned theory is a little more accurate. TDP's pruning phase is slower than GROW's pruning phase, although it starts with a simpler theory. The reason for this is that our implementation of TDP uses REP for pruning. It might be worthwhile to further improve TDP by using the faster GROW algorithm for its post-pruning phase. However, this also indicates that in this domain TDP's initial top-down search for a good starting theory is not as effective as in the KRK domain, because more work was left for the post-pruning phase.

The only PROLOG algorithm faster and more accurate than I-REP is FOSSIL with a cutoff of 0.3. It is even faster than the C implementation of FOIL 6.3. However, FOSSIL couldn't discover any significant regularities in the data and thus consistently learned empty theories (all literals in the background knowledge had a correlation below 0.3). Nevertheless, it is still the best algorithm in terms of accuracy (followed closely by FOIL 6.3 and I-REP) which shows how poorly all algorithms perform in this domain (all are below FOSSIL, i.e., below default accuracy). We hope to be able to improve our results in this domain by trying the faster algorithms on the new data set (Dolšak, Bratko, & Jezernik, 1994) which contains a total of 10 objects (and thus hopefully provides more redundancy). However, the new data set was too big for the post-pruning algorithms of this comparative study.

An interesting phenomenon is that although pruning literals generalizes the clauses so that more positive examples will be covered, the pruned theories as a whole cover fewer positive examples. Obviously, for many learned rules generalization by deleting irrelevant literals did not improve accuracy as much as did specializing the theory by removing the entire rule. This can also be taken as evidence that most regularities detected by the basic separate-and-conquer induction module were not very reliable.

7.3. Propositional data sets

We have also experimented with data sets from the UCI repository of Machine Learning databases that have previously been used to compare propositional learning algorithms. Holte (1993) gives a summary of the results achieved by various algorithms on some of the most commonly used data sets of the UCI repository and a short description of these sets. We selected nine of them for our experiments. The remaining sets were not used because either the description of the data sets was unclear or they had more than two classes, which could not be handled by our implementation of the learning algorithms. In the *Lymphography2* data set, we removed the 6 examples for the classes "normal find" and "fibrosis" in order to get a 2-class problem. All other data were used in the same way as Holte (1993) has used them. For all data sets, the task was to learn a definition for the minority class.

In all data sets, the background knowledge consisted of < and = relations with one variable and one constant argument. Wherever appropriate, comparisons between two different variables of the same data type were allowed as well. In all experiments, the value of FOSSIL's cutoff parameter was set to 0.3. FOIL 6.3 was used with its default parameter settings. It has been previously shown that FOIL is competitive with propositional decision tree learning algorithms (Cameron-Jones & Quinlan, 1993). Run-times for all data sets were measured in CPU seconds for SUN SPARCstations ELC except for the *Mushroom* and *KRKPa7* data sets which are quite big and thus had to be run on a considerably faster SPARCstation S10. All experiments followed Holte's setup, i.e., the algorithms were trained on $2/3$ of the data and tested on the remaining $1/3$. However, only 10 runs were performed for each algorithm on each data set.

The results on predictive accuracy can be found in table 7. Each line shows the average accuracy on the 10 sets and its standard deviation. To allow a more structured analysis, the

Table 7. Accuracy results for some propositional data sets.

	Sick Euthyroid	*Breast Cancer*	*Hepatitis*
Foil 6.3	**96.59** ± 0.67	**67.59** ± 3.48	**77.04** ± 5.62
Fossil	**97.58** ± 0.42	**73.33** ± 4.81	**76.07** ± 6.08
No Pruning	96.25 ± 0.54	65.39 ± 4.44	73.66 ± 5.25
REP	**97.55** ± 0.34	**69.97** ± 4.01	**76.96** ± 4.14
Grow	**97.52** ± 0.50	**68.46** ± 4.98	**76.45** ± 4.47
No Pruning (TDP)	97.37 ± 0.54	67.98 ± 5.86	76.33 ± 3.58
TDP	**97.49** ± 0.45	**71.74** ± 4.00	**79.42** ± 4.09
I-REP	**97.48** ± 0.53	**70.89** ± 5.51	**78.66** ± 2.95
	Glass (G2)	*Votes*	*Votes (VI)*
Foil 6.3	**75.78** ± 7.11	**92.50** ± 1.94	**84.30** ± 2.99
Fossil	**77.32** ± 5.05	**95.35** ± 1.23	**89.07** ± 2.78
No Pruning	75.24 ± 5.54	94.69 ± 1.99	86.46 ± 2.12
REP	**77.76** ± 4.54	**95.84** ± 1.47	**86.72** ± 3.65
Grow	**75.63** ± 4.94	**95.63** ± 1.43	**87.49** ± 3.53
No Pruning (TDP)	77.23 ± 4.23	95.33 ± 1.29	87.57 ± 1.43
TDP	**75.90** ± 6.51	**95.22** ± 1.92	**85.85** ± 2.76
I-REP	**76.31** ± 5.15	**94.75** ± 1.84	**87.25** ± 3.45
	KRKPa7	*Lymphography2*	*Mushroom*
Foil 6.3	**98.48** ± 0.57	**81.04** ± 5.08	**100.00** ± 0.00
Fossil	**95.17** ± 2.80	**87.22** ± 4.63	**99.96** ± 0.03
No Pruning	97.92 ± 0.61	83.25 ± 5.05	100.00 ± 0.01
REP	**97.84** ± 0.57	**81.85** ± 5.12	**99.97** ± 0.05
Grow	**97.48** ± 0.43	**82.10** ± 5.57	**99.57** ± 0.69
No Pruning (TDP)	96.26 ± 1.95	83.73 ± 5.80	100.00 ± 0.01
TDP	**96.41** ± 1.97	**81.86** ± 4.63	**99.97** ± 0.05
I-REP	**97.74** ± 0.38	**79.17** ± 4.66	**99.97** ± 0.04

9 domains can be grouped into 3 subclasses: In the 3 domains of the upper part of table 7, overfitting is harmful, i.e., REP's post-pruning phase substantially improves the concepts learned by the initial overfitting phase.[17] The middle part contains domains where pruning does not make a significant difference, and in the domains of the lower part pruning cannot be recommended as exemplified by the *Mushroom* data, where the overfitting phases learned 100% correct concept descriptions that were substantially better than those learned by all pruning algorithms (with the exception of Foil). The *Mushroom* and *KRKPa7* domains are known to be free of noise, while the three medical domains are noisy. We thus assume that our grouping of the domains corresponds to the amount of noise contained in the data.

I-REP and TDP outperform the post-pruning algorithms in the three noisy domains, perform inconsistently in domains where pruning does not make much difference and seem to be harmful in noise-free domains. Compared to the other algorithms, post-pruning counter-intuitively performs better in domains with low noise levels. This indicates that

Table 8. Run-time results for some propositional data sets.

	Sick Euthyroid	*Breast Cancer*	*Hepatitis*
Foil 6.3	**132.08**	**1.83**	**0.76**
Fossil	**891.40**	**19.68**	**217.40**
No Pruning	4554.65	169.70	101.66
REP	**5040.23**	**257.29**	**102.28**
Grow	**4635.26**	**183.67**	**102.39**
No Pruning (TDP)	2965.51	154.05	115.41
TDP	**3010.97**	**173.31**	**116.24**
I-REP	**970.70**	**28.97**	**60.40**
	Glass (G2)	*Votes*	*Votes (VI)*
Foil 6.3	**0.31**	**1.68**	**2.83**
Fossil	**216.42**	**105.22**	**88.94**
No Pruning	91.89	50.45	124.47
REP	**93.31**	**57.41**	**163.26**
Grow	**93.11**	**53.84**	**137.49**
No Pruning (TDP)	85.56	60.88	105.67
TDP	**87.39**	**62.17**	**113.05**
I-REP	**63.01**	**22.43**	**38.78**
	KRKPa7	*Lymphography2*	*Mushroom*
Foil 6.3	**78.35**	**0.67**	**40.34**
Fossil	**2383.61**	**20.79**	**3538.19**
No Pruning	4063.80	17.05	1878.51
REP	**4243.08**	**18.81**	**1931.75**
Grow	**4219.00**	**18.42**	**2088.81**
No Pruning (TDP)	2368.28	18.66	4595.23
TDP	**2376.28**	**20.27**	**4656.31**
I-REP	**1785.50**	**10.14**	**2493.77**

overfitting can not entirely be prevented with post-pruning alone. Similarly, Foil 6.3 outperforms TDP and I-REP in noise-free domains while they in turn seem to be preferable in noisy domains. This confirms previous findings that Foil's encoding length criterion does not effectively prevent overfitting (Fürnkranz, 1994a), but also shows that TDP's and I-REP's bias towards simple theories may be too strong. The latter has also been noted by Cohen (1995), where it has been shown that RIPPER, an algorithm that improves I-REP by employing different pruning and stopping criteria and an additional post-pruning phase, outperforms C4.5rules on a variety of domains.[18] Pre-pruning with Fossil at a cutoff of 0.3 gives a good overall performance which confirms the robustness of this algorithm towards different noise-levels (Fürnkranz, 1994a).

Some differences in accuracies are statistically significant (measured with a standard two-tailed t-test): In the *KRKPa7* chess endgame domain, Foil 6.3 was significantly better than the other five algorithms at the 1% error level (for REP and TDP only at 5%), while

FOSSIL performed significantly worse (5%) than all other algorithms except TDP. On the other hand, FOSSIL outperformed all other algorithms except GROW in the *Lymphography2* domain (5%, I-REP even with 1%). In the *Votes (VI)* domain, FOSSIL was better than FOIL (1%) and TDP (5%). FOIL performed significantly worse (1%) than all other algorithms in the *Votes* and *Sick Euthyroid* domains. It was also worse (5%) than TDP and FOSSIL in the *Breast Cancer* domain, but better than FOSSIL(1%) in the *Mushroom* domain. No significant differences could be found in the *Glass (G2)* and *Hepatitis* domains.

While the results vary in terms of accuracy the results for run-times are quite consistent (see table 8): FOIL 6.3 is clearly the fastest algorithm, but as it is implemented in C this comparison is not entirely fair. Among the PROLOG algorithms I-REP is the fastest in 6 of the 9 test problems, while it is second-best in 2 of the remaining 3. The table also confirms that GROW is usually faster than REP. TDP's results are not consistent, but it is faster than REP and GROW in some cases, which indicates that its initial top-down search for a good starting theory does not overfit the data as much as the initial rule growing phase of REP and GROW does. FOSSIL's run-times are very unstable. It is the fastest algorithm on some data sets, but by far the slowest on data sets, where not much pruning has to be performed and thus the algorithms that only learn from $2/3$ of the data can be faster.

8. Related work

I-REP and TDP have been deliberately designed to closely resemble the basic post-pruning algorithm, REP. For instance, we have already pointed out that TDP can be further improved by using GROW instead of REP in TDP's post-pruning phase. In the case of I-REP, we have chosen accuracy of a one-clause theory on the pruning set as the basic pruning and stopping criterion in order to get a fair comparison to REP and to concentrate on the methodological differences between post-pruning and I-REP's integration of pre- and post-pruning. An important advantage of post-pruning methods is that the way of evaluating theories (or rules in I-REP's case) is entirely independent from the basic learning algorithm. Other pruning and stopping criteria can further improve the performance and eliminate weaknesses. Currently, we are investigating pruning criteria based on the *Minimal Description Length* principle (Rissanen, 1978), which have the merit of avoiding the loss of information that is caused by the need of splitting the training set into separate growing and pruning sets (Quinlan, 1994).

The accuracy-based pruning criterion used in I-REP basically optimizes the difference between the positive and negative examples covered by a clause.[14] Cohen (1995) points out that this measure can lead to undesirable choices. For example, it would prefer a rule that covers 2000 positive and 1000 negative instances over a rule that covers 1000 positive and only 1 negative instance. As an alternative, Cohen (1995) suggests to divide this difference by the total number of covered examples and shows that this choice leads to significantly better results. In addition he shows that an alternative clause stopping criterion based on theory description length and an additional post-pruning phase can further improve I-REP. The resulting algorithm, RIPPER, is competitive with C4.5rules[18] without losing I-REP's efficiency. Cohen (1995) has also noted that accuracy estimates for low-coverage rules will have a high variance and therefore I-REP is likely to over-generalize in domains that are susceptible to the *Small Disjuncts Problem* (Holte, Acker, & Porter, 1989). This is a generalization of our observation that I-REP performs badly in domains with small training sets.

Fürnkranz (1995b) has made another attempt to improve I-REP. Just as I-REP avoids learning an overfitting theory by pruning at the clause level instead of the theory level, we have investigated a way of taking this further by pruning at the literal level to avoid learning overfitting clauses. The resulting algorithm, I^2-REP, splits the training set into two sets of equal size, selects the best literal for each of them, and chooses the one that has the higher accuracy on the entire training set. This procedure is quite similar to a two-fold cross-validation, which has been shown to give reliable results for ranking classifiers at low training set sizes (Weiss & Indurkhya, 1994). Literals are added to the clause until the best choice does not improve the accuracy of the clause, and clauses are added as long as this betters the accuracy of the theory. I^2-REP seems to be a little more stable than I-REP with small training sets, but no significant run-time improvements can be observed (Fürnkranz, 1995a). It even appears to be a little slower than I-REP, although asymptotically both algorithms exhibit approximately the same subquadratic behavior.

9. Conclusion

In this paper, we have discussed different pruning techniques for separate-and-conquer rule learning algorithms. Conventional pre-pruning methods are very efficient, but not always as accurate as post-pruning methods. The latter, however, tend to be very expensive, because they have to learn an over-specialized theory first. In addition to their inefficiency we have pointed out a fundamental incompatibility of post-pruning methods with separate-and-conquer rule learning systems. As a solution, we have investigated two methods for combining and integrating pre- and post-pruning algorithms.

TDP performs an initial top-down search through the hypothesis space to find a theory that overfits the training data, but is still fairly simple. This theory is then used as a starting theory for a subsequent post-pruning phase that tries to simplify this theory to an appropriate level. A systematic algorithm for varying the cutoff parameter of the pre-pruning algorithm FOSSIL provides an efficient way of generating theories in an approximate simple-to-complex order so that a good starting theory can often be found in considerably shorter time than would be needed to generate a complex theory that fits all of the training examples. Of course, the pruning phase for the simpler theory is also shorter than the pruning phase for the more complex theory.

I-REP integrates pre- and post-pruning into one algorithm. Instead of post-pruning entire theories, each rule is pruned right after it has been learned. Our experiments show that this approach effectively combines the efficiency of pre-pruning with the accuracy of post-pruning in noisy domains. As real-world databases are typically large and noisy, and thus require learning algorithms that are both efficient and noise-tolerant, I-REP seems to be an appropriate choice for this purpose.

Acknowledgments

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P10489-MAT. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science, Transport, and the Arts. I would like to thank Gerhard Widmer for patiently reading and improving numerous versions of this paper. Thanks are also due to Ross Quinlan, Mike Cameron-Jones, William Cohen, Bernhard Pfahringer, Johann Petrak, Arthur Flexer and Stefan Kramer for valuable suggestions and discussions, to the maintainers of the UCI repository of machine learning databases, and to the editor and three anonymous reviewers for their thoughtful comments and constructive criticism.

Notes

1. For encoding p positive and n negative training instances, one needs at least $\log_2(p+n) + \log_2(\begin{pmatrix} p+n \\ p \end{pmatrix})$ bits. Literals can be encoded by specifying one out of r relations ($\log_2(r)$ bits), one out of v variabilizations ($\log_2(v)$ bits) and whether it is negated or not (1 bit). The sum of these terms for all l literals of the clause has to be reduced by $\log_2(l!)$ since the ordering of literals within a clause is in general irrelevant.
2. $LRS = 2 \times (p \log \left(\frac{\frac{p}{p+n}}{\frac{P}{P+N}} \right) + n \log \left(\frac{\frac{n}{p+n}}{\frac{N}{P+N}} \right))$ where p and n are the number of positive and negative examples covered by the clause and P and N are the number of positive and negative examples in the entire training set.
3. A short description of the KRK domain along with the experimental setup can be found at the beginning of section 7.1.
4. Recall that $g(x) = \Omega(f(x))$ denotes that g grows at least as fast as f, i.e., f forms a lower bound for g. Similarly, $g(x) = O(f(x))$ means that f forms an upper bound for g. We will also use $g(x) = \Theta(f(x))$ to denote that f and g have the same asymptotic growth, i.e., there exist two constants c_1 and c_2 such that $c_1 f(x) \leq g(x) \leq c_2 f(x)$.
5. The number of attributes will only be independent of n as long as the training set size does not approach the complete enumeration of the domain, in which case there will be $\log n$ attributes. As this is in general not the case for practical purposes, we feel this assumption is justified. Also note that our assumptions include many first-order learning problems, such as the KRK domain, that can be transformed into a propositional representation by using one binary attribute for each literal that can appear in the body of a rule (Lavrač, Džeroski, & Grobelnik, 1991). Their applicability to domains that require numerical threshold conditions and/or full first-order horn-clause logic is not so straight-forward.
6. Cameron-Jones (1996) points out that for training sets that approach towards complete enumeration, chances increase that rules may cover more than one random example, but derives compatible results for this case.
7. One may argue that the number of simplifications decreases significantly when the theories get simpler, so that the total cost will be lower. However, until half of the rules have been deleted, the size of the concepts is greater than $n/2$, so that we still have $\Theta(n)$ simplifications for $\Omega(n)$ iterations.
8. Cohen used a slightly different version of REP that prunes to an empty theory and selects the best pruned theory thereafter, while the original version stops pruning when the accuracy on the pruning set goes up. However, it is unlikely that these changes have been disadvantageous for REP.
9. The theories are not learned in a strict simple-to-complex order, but simpler theories can also follow more complex theories (see e.g. theories 4 and 5 of figure 8). However, the general trend is that the complexity of the learned theories increases with lower settings of the cutoff parameter. Note that more complex theories often contain several specializations of rules of a simpler theory, so that simpler theories are often also more general than complex theories.

[10] This is based on an idea in CART (Breiman, Friedman, Olshen, & Stone, 1984), where the most *general* pruned decision tree within one SE of the best will be selected. The standard classification error can be computed with $SE = \sqrt{\frac{p \times (1-p)}{N}}$ where p is the probability of misclassification (estimated on the pruning set) and N is the number of examples in the pruning set.

[11] This argument, of course, only applies to noisy domains. In non-noisy domains, the most complex theory will in general be the most precise and thus our algorithm will be slower, because it has to generate all theories down to a cutoff of 0.0.

[12] Note that this method may yield a theory that is not learnable by the original FOSSIL as the value of the cutoff parameter is changed during the generation of the theory.

[13] The version of REP using FOSSIL did better than the version using FOIL. In section 7.1 we show the results obtained by using an implementation of FOIL to generate the initial theory for REP.

[14] We measure the accuracy of a clause with $\frac{p+(N-n)}{P+N}$, the percentage of the covered positive examples plus the uncovered negative examples in the set of examples that have not yet been covered by the previous clauses. We chose this function because it is directly related to the accuracy-based optimization used in REP. As P and N are constant for all versions of a clause, pruning will basically maximize the difference $p - n$. A problem with this measure is discussed in section 8.

[15] The current version of FOIL is available by anonymous ftp from `ftp.cs.su.oz.au` or `129.78.8.1` file name, `pub/foilN.sh` for some integer `N`.

[16] Note that often a different model is used in the literature, where a noise level of η means that the sign of each example is randomly chosen with a probability of η. A noise level of η in our experiments is roughly equivalent to a noise level of 2η in the other model. In our model, 50% noise means totally random data, while 100% noise would result in a training set with reversed class labels.

[17] It might be (justifiably) argued here that we should have used a separate run with no pruning on all of the data for a comparison. Our main purpose, however, was to compare different pruning approaches and not evaluate the merits of pruning by itself. The results for the initial overfitting phases of REP, GROW and TDP may nevertheless be an indicator for the latter (and they come at no additional cost).

[18] C4.5rules is an algorithm that generates propositional rules from decision trees that have been learned with C4.5 (Quinlan, 1993).

References

Angluin, D., & Laird, P. (1988). Learning from noisy examples. *Machine Learning*, *2*(4), 343–370.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Brooks.

Brunk, C. A., & Pazzani, M. J. (1991). An investigation of noise-tolerant relational concept learning algorithms. *Proceedings of the 8th International Workshop on Machine Learning (ML-91)* (pp. 389–393). Evanston, IL: Morgan Kaufmann.

Cameron-Jones, R. M. (1996). The complexity of batch approaches to reduced error rule set induction. *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI-96)* (pp. 348–359). Cairns, Australia: Springer-Verlag.

Cameron-Jones, R.M., & Quinlan, J. R. (1993). First order learning, zeroth order data. *Proceedings of the 6th Australian Joint Conference on AI*. World Scientific.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proceedings of the 5th European Working Session on Learning (EWSL-91)* (pp. 151–163). Porto, Portugal: Springer-Verlag.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*(4), 261–283.

Cohen, W. W. (1993). Efficient pruning methods for separate-and-conquer rule learning systems. *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)* (pp. 988–994). Chambéry, France: Morgan Kaufmann.

Cohen, W. W. (1995). Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning (ML-95)* (pp. 115–123). Lake Tahoe, CA: Morgan Kaufmann.

Dolšak, B., Bratko, I., & Jezernik, A. (1994). Finite element mesh design: An engineering domain for ILP application. *Proceedings of the 4th International Workshop on Inductive Logic Programming (ILP-94)* (pp. 305–320). Bad Honnef, Germany: GMD-Studien (Vol. 237).

Dolšak, B., & Muggleton, S. (1992). The application of inductive logic programming to finite-element mesh design. In S. H. Muggleton (Ed.), *Inductive Logic Programming*. London, UK: Academic Press.

Džeroski, S., & Bratko, I. (1992). Handling noise in inductive logic programming. *Proceedings of the International Workshop on Inductive Logic Programming (ILP-92)*. Tokyo, Japan.

Esposito, F., Malerba, D., & Semeraro, G. (1993). Decision tree pruning as a search in the state space. *Proceedings of the 6th European Conference on Machine Learning (ECML-93)* (pp. 165–184). Vienna, Austria: Springer-Verlag.

Fürnkranz, J. (1994a). FOSSIL: A robust relational learner. *Proceedings of the 7th European Conference on Machine Learning (ECML-94)* (pp. 122–137). Catania, Italy: Springer-Verlag.

Fürnkranz, J. (1994b). Top-down pruning in relational learning. *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)* (pp. 453–457). Amsterdam, The Netherlands: John Wiley & Sons.

Fürnkranz, J. (1995a). *A tight integration of pruning and learning* (Technical Report OEFAI-TR-95-03). Vienna, Austria: Austrian Research Institute for Artificial Intelligence.

Fürnkranz, J. (1995b). A tight integration of pruning and learning (extended abstract). *Proceedings of the 8th European Conference on Machine Learning (ECML-95)* (pp. 291–294). Heraclion, Greece: Springer-Verlag.

Fürnkranz, J. (1996). *Separate-and-conquer rule learning* (Technical Report OEFAI-TR-96-25). Vienna, Austria: Austrian Research Institute for Artificial Intelligence. Submitted for publication.

Fürnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. *Proceedings of the 11th International Conference on Machine Learning (ML-94)* (pp. 70–77). New Brunswick, NJ: Morgan Kaufmann.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–91.

Holte, R. C., Acker, L., & Porter, B. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)* (pp. 813–818). Detroit, MI: Morgan Kaufmann.

Lavrač, N., Džeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. *Proceedings of the 5th European Working Session on Learning (EWSL-91)* (pp. 265–281). Porto, Portugal: Springer-Verlag.

Michalski, R. S. (1980). Pattern recognition and rule-guided inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*, 349–361.

Michalski, R. S., Mozetič, I., Hong, J., & Lavrač, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)* (pp. 1041–1045). Philadelphia, PA.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, *4*, 227–243.

Muggleton, S., Bain, M., Hayes-Michie, J., & Michie, D. (1989). An experimental comparison of human and machine learning formalisms. *Proceedings of the 6th International Workshop on Machine Learning (ML-89)* (pp. 113–118). Morgan Kaufmann.

Niblett, T., & Bratko, I. (1987). Learning decision rules in noisy domains. In M. Bramer (Ed.), *Research and Development in Expert Systems*. Cambridge, UK: Cambridge University Press.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*, 71–99.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, *27*, 221–234.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1994). The minimum description length principle and categorical theories. *Proceedings of the 11th International Conference on Machine Learning (ML-94)* (pp. 233–241). New Brunswick, NJ.

Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, *13*(3,4), 287–312.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465–471.

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, *10*, 153–178.

Weiss, S. M., & Indurkhya, N. (1991). Reduced complexity rule induction. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)* (pp. 678–684). Morgan Kaufmann.

Weiss, S. M., & Indurkhya, N. (1994). Small sample decision tree pruning. *Proceedings of the 11th Conference on Machine Learning* (pp. 335–342). New Brunswick, NJ: Morgan Kaufmann.

Wolpert, D. H. (1993). *On overfitting avoidance as bias* (Technical Report SFI TR 92-03-5001). Santa Fe, NM: The Santa Fe Institute.

Received January 2, 1996
Accepted September 26, 1996
Final Manuscript October 21, 1996