Pruning Algorithms for Rule Learning

Johannes Fürnkranz
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria
E-mail: juffi@ai.univie.ac.at

OEFAI-TR-96-07

Abstract

Pre-Pruning and Post-Pruning are two standard methods of dealing with noise in decision tree learning. Pre-Pruning methods deal with noise during learning, while post-pruning methods try to address this problem after an overfitting theory has been learned. This paper shows how pre- and post-pruning algorithms can be used for separate-and-conquer rule learning algorithms. We discuss some fundamental problems and show how to solve them 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 gained in popularity through the recent success of the Inductive Logic Programming algorithm FOIL (Quinlan 1990). We will analyze different pruning methods for this type of inductive rule learning algorithm and discuss some of their problems. The main contribution of this paper 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.

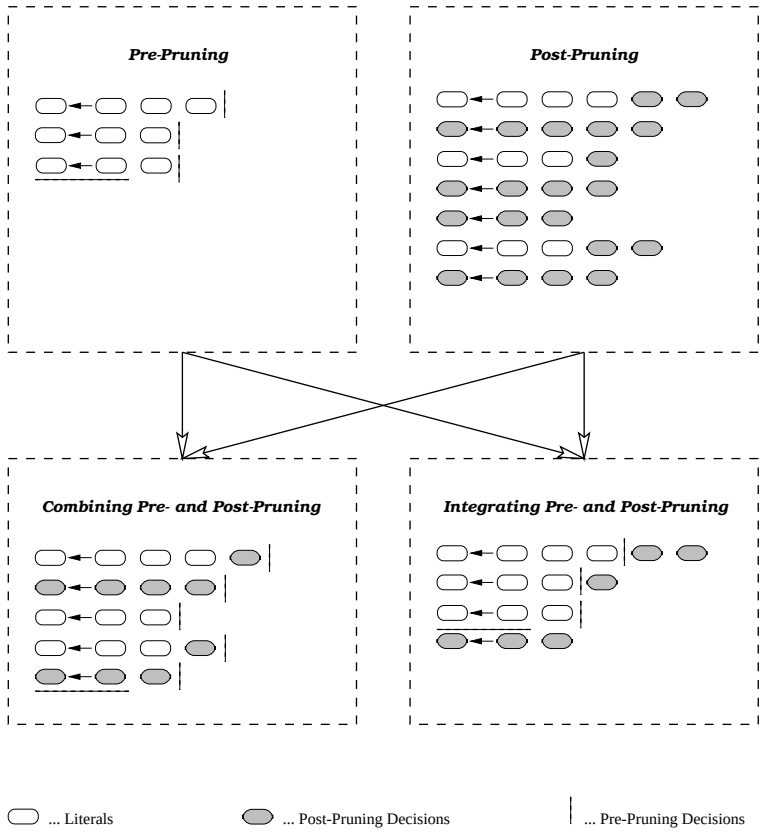


Figure 1: Pruning methods for separate-and-conquer rule learning algorithms.

Pruning is the common framework for avoiding the problem of *overfitting* noisy data. The basic idea is to incorporate a bias towards more general and simpler theories in order to avoid overly specific theories that try to find explanations for 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 — so-called *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 and Niblett 1989), FOIL (Quinlan 1990), and FOSSIL (Fürnkranz 1994), use this form of noise handling.

Another family of algorithms deals with noise after learning. These *post-pruning* algorithms typically first induce a theory that is complete and consistent with the training data. Then this theory is examined and those rules and conditions are discarded that seem to

explain only characteristics of the particular training set and thus do not reflect true regularities of the domain. Figure 1 shows a schematic depiction of this process. The quality of the found 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 and Pazzani 1991) and GROW (Cohen 1993). Both have been shown to be very 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. For this purpose pre-pruning heuristics are used to reduce (not entirely prevent) the amount of overfitting, so that learning and pruning will be more efficient as sketched in the third part of figure 1. Our particular implementation of this approach, *Top-Down Pruning* (TDP) (Fürnkranz 1994), uses a simple algorithm to generate a set of theories pruned to different degrees in a top-down, general-to-specific order. The accuracies of the theories are evaluated on a separate set of data and the most specific 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 final theory, the post-pruning phase will also be sped up, because less 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 and Widmer 1994) prunes single clauses right after they have been learned. This new algorithm entirely avoids the learning of an overfitting theory by using post-pruning methods as a pre-pruning stopping criterion as shown in figure 1. With this method a significant speedup can be achieved in noisy domains. As 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 try to construct rules with the so-called *separate-and-conquer* strategy. This method has its roots in the early days of Machine Learning in the covering algorithm of the famous AQ family (Michalski 1980; Michalski, Mozetič, Hong, and Lavrač 1986). CN2 (Clark and Niblett 1989; Clark and Boswell 1991) combined AQ's covering strategy with the greedy information-based test selection of ID3 (Quinlan 1983), which yielded a powerful rule learning algorithm. The term separate-and-conquer has been coined in (Pagallo and Haussler 1990) in the context of learning decision lists. Finally, separate-and-conquer learning is the basic control structure in the FOIL algorithm for efficiently inducing logic programs (Quinlan 1990), which pioneered significant research in the field of relational learning and Inductive Logic Programming.

Figure 2 shows 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 the most general separate-and-conquer learning algorithm, FOIL.

```
procedure SeparateAndConquer(Examples)

Theory = ∅
while Positive(Examples) ≠ ∅
        Clause = ∅
        Cover = Examples
        while Negative(Cover) ≠ ∅
                Clause = Clause ∪ FindLiteral(Clause, Cover)
                Cover = Cover(Clause, Cover)
        Examples = Examples − Cover
        Theory = Theory ∪ Clause
return(Theory)
```

Figure 2: A Separate-and-Conquer Rule Learning Algorithm

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

In propositional learning (as in CN2) the conditions 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 fulfilled the conditions of the rule will consequently be classified as an instance of the learned concept. If no rules "fires", the instance will not be considered as a member of the concept.

SeparateAndConquer constructs rules by successively adding conditions to the right-hand side of the current rule. This process is repeated until enough conditions have been found to rule out all of the negative examples. All positive examples covered by this rule 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*).

The simple SeparateAndConquer algorithm of figure 2 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. The fundamental algorithm of figure 2 forms a complete and consistent theory, i. e., it tries to explain all of the positive examples and none of the negative examples. In the presence of noise it will therefore attempt to find explanations for negative examples that have erroneously been classified as positive and try to exclude positive examples that have a negative classification in the training set. Explanations for noisy examples typically are 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 theories that may be over-general on the training examples. The final theory will be allowed to deliberately cover some negative training examples and leave some positive training examples uncovered in order to learn simpler and more predictive theories. This is usually achieved via *pruning* heuristics.

```
procedure PrePruning(Examples)

Theory = ∅
while Positive(Examples) ≠ ∅
      Clause = ∅
      Cover = Examples
      while Negative(Cover) ≠ ∅
            NewClause = Clause ∪ FindLiteral(Clause, Cover)
            if StoppingCriterion(Theory, NewClause, Cover)
               exit while
            Clause = NewClause
            Cover = Cover(Clause, Cover)
      if Clause = ∅
         exit while
      Examples = Examples − Cover
      Theory = Theory ∪ Clause
return(Theory)
```

Figure 3: A Rule Learning Algorithm Using Pre-Pruning

3 Pre-Pruning

Figure 3 shows an adaptation of the simple SeparateAndConquer algorithm in order to address noisy data with a *pre-pruning* heuristic. The algorithm is identical to the one of figure 2 except that the inner `while` loop contains a call to the subroutine StoppingCriterion. The *stopping criterion* is a heuristic which determines when to stop adding conditions to a rule, and when to stop adding rules to the concept description. If the current rule with the new condition added fulfills the stopping criterion the inner `while` loop will terminate and the incomplete clause will be added to the concept description. If this clause contains no literal it is assumed that no further clause can be found that explains the remaining positive examples and the theory without this clause is returned. 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. The most commonly used among them 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 the current clause is less than the number of bits needed to encode the instances covered by it.[1]
- *Significance Testing* was first used in the propositional CN2 induction algorithm (Clark and Niblett 1989) and later on in the relational learner *m*Foil (Džeroski and Bratko 1992). It tests for significant differences between the distribution of positive and negative

[1] The number of bits needed to encode the training instances is $log_2(n)+log_2(\binom{n}{p})$ where n is the number of training instances and p positive instances are covered by the current clause. Literals can be encoded by specifying the relation ($\log_2$(# relations) bits), the variables ($\log_2$(# variabilizations) bits) and whether it is negated or not (1 bit). The sum of these terms for all literals has to be reduced by $log_2(n!)$ since the ordering of these literals within the clause is in general irrelevant.

examples covered by a rule and the overall distribution of positive and negative examples by comparing the likelihood ratio statistic[2] to a χ^2 distribution with 1 degree of freedom at the desired significance level. Insignificant rules are rejected.

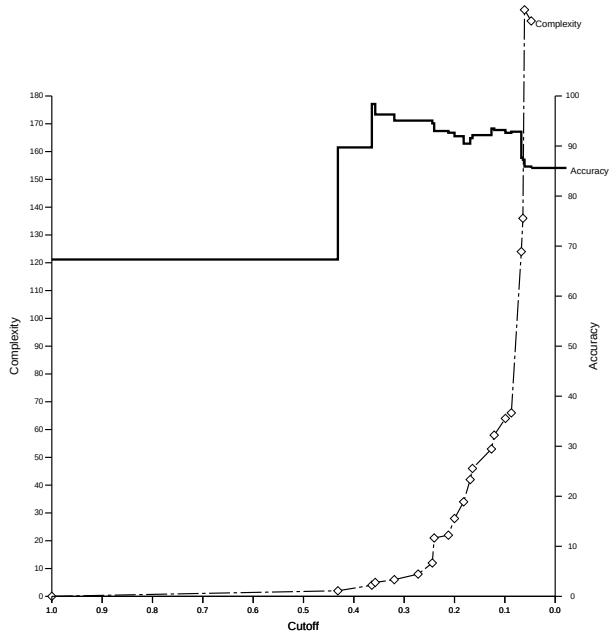


Figure 4: Accuracy and Complexity vs. Cutoff

- *The Cutoff Stopping Criterion* has been used in the separate-and-conquer learning system FOSSIL (Fürnkranz 1994). FOSSIL uses a search heuristic based on statistical correlation, which enables it to judge the relevance of all literals on the same uniform scale from 0 to 1. Thus the user can require that conditions considered for clause construction have a certain minimum correlation value — the *cutoff parameter*. This property can be used as a simple, but robust criterion for filtering out noise, as it can be expected that tuples originating from noise in the data will only have a low correlation with predicates in the background knowledge. Different settings of the values will cause different amounts of pre-pruning. A setting of *Cutoff* = 0.0 results in learning a theory that is complete and consistent for the current training set, because every literal has a correlation > 0.0. On the other hand, at *Cutoff* = 1.0 in general an empty theory will be learned, because only trivial learning problems have background literals with a correlation ≥ 1.0.

  Figure 4 shows a typical plot of accuracy and rule complexity vs. different values of the cutoff parameter for the commonly used KRK endgame classification task with 10% noise added[3]. The most accurate rules are found for cutoff values between approximately 0.25 and 0.35. Higher cutoff values result in over-general theories, while lower settings of the cutoff obviously result in overfitting of the data. Thus FOSSIL's cutoff parameter may be viewed as a means for directly controlling the *Overfitting Avoidance Bias* (Schaffer

[2] $LRS = 2 \times (p \log \left(\frac{\frac{p}{p+n}}{\frac{P}{P+N}} \right) + n \log \left(\frac{\frac{n}{p+n}}{\frac{N}{P+N}} \right))$

[3] A short description of the KRK domain along with the experimental setup can be found at the beginning of section 7.1.

```
procedure PostPruning(Examples, SplitRatio)

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

Figure 5: A Post-Pruning algorithm

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 1994).

4 Post-Pruning

While pre-pruning approaches try to avoid overfitting during rule generation, *post-pruning* approaches at first ignore the problem of overfitting the noise and learn a complete and consistent concept description. The resulting theory is subsequently analyzed and (if necessary) simplified and generalized in order to increase its predictive accuracy on unseen data.

Post-pruning approaches have been commonly used in the decision tree learning algorithms CART (Breiman, Friedman, Olshen, and Stone 1984), ID3 (Quinlan 1987) and ASSISTANT (Niblett and Bratko 1986). An overview and comparison of various approaches can be found in (Mingers 1989) and (Esposito, Malerba, and Semeraro 1993).

4.1 Reduced Error Pruning

The most common among these methods is *Reduced Error Pruning (REP)*. This simple algorithm has been adapted from decision tree learning (Quinlan 1987) to the separate-and-conquer rule learning framework by (Pagallo and Haussler 1990) and (Brunk and Pazzani 1991). At the beginning 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 explains all of the positive and none of the negative examples is learned from the growing set. The resulting theory is then simplified by greedily deleting conditions and rules from the theory until any further deletion would result in a decrease of predictive accuracy as measured on the pruning set. A pseudo-code version of this algorithm is shown in figure 5.

The subroutine PruneTheory simplifies the current theory by deleting conditions and rules, usually one at a time. From the resulting set of theories it then selects the one with the highest accuracy on the pruning set and continues to prune this 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 and Pazzani 1991).

4.2 Problems with Reduced Error Pruning

Although REP is quite effective in raising predictive accuracy in noisy domains (Brunk and Pazzani 1991), it has several shortcomings, which we will discuss in this section. In particular we will suggest that post-pruning is incompatible with the separate-and-conquer learning strategy.

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 $\Omega(n^2 \log n)$. Therefore in the long run the costs of pruning will by far 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.

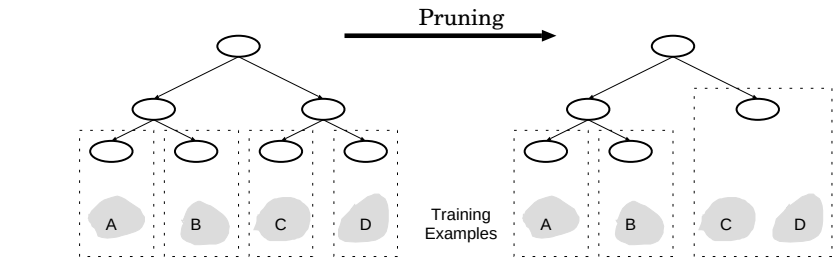
However, in noisy domains the theory that has been generated in the growing phase will be much too specific (see figure 4). REP will have to prune a significant portion of this theory and has ample opportunity to err on its way. Therefore we can also expect REP's specific-to-general search not only to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

Separate-and-Conquer Strategy

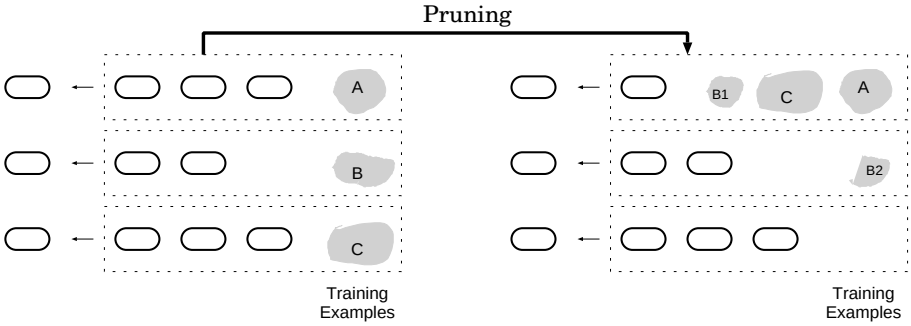
Post-pruning algorithms originate from research in decision tree learning where usually the well-known *divide-and-conquer* learning strategy is used. At each node the current training set is divided into disjoint sets according to the outcome of the chosen test. After this, the algorithm is recursively 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 neighbouring branches, whereas pruning of literals of a rule will affect all subsequent rules. Figure 6 (a) illustrates how post-pruning in decision tree learning works. The right half of the initially grown 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 instances along with some 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 6 (b) 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, this is not necessarily the case. For example there is no guarantee



(a)



(b)

Figure 6: Post-Pruning in (a) Divide-and-Conquer and (b) Separate-and-Conquer learning algorithms.

that the second rule or one of its pruned versions are good explanations for the remaining set of examples B2, because B2 is a subset of the original set B and pruning operators can only generalize the concept, i.e. increase the set of covered examples. It might well be that a good explanation for B2 needs a totally different set of literals than an explanation for its superset B. Thus the learner may be lead down a garden path, because the unpruned clauses at the beginning of the theory may change the evaluation of candidate literals for subsequent 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, a top-down post-pruning algorithm based on a technique used in (Pagallo and Haussler 1990) has been proposed in (Cohen 1993). Like REP, the GROW algorithm 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 iteratively 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 of

the intermediate theory 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.

Thus GROW improves upon REP by replacing the bottom-up hill-climbing search of REP with a a top-down approach. Instead of removing the most useless clause or literal from the specific theory it 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 as the experiments in (Cohen 1993) show. However, the asymptotic time complexity of the GROW post-pruning method is still above the complexity of the initial rule growing phase as has recently been shown in (Cameron-Jones 1994).

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. This can also been seen from figure 4 when we look at the complexities of the most specific theory (*Cutoff* = 0.0) and the complexities of the optimal theories (*Cutoff* between 0.25 and 0.35).

Thus it is not surprising that GROW has been shown to outperform REP on a variety of datasets (Cohen 1993). However, 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. This idea has first been investigated in (Cohen 1993), where the efficient post-pruning algorithm GROW (see section 4.3) has been combined 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. Thus 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 over-generalize the 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 (Fürnkranz 1994) is its closeness to the search heuristic. 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 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 (which will only be the case in trivial examples such as `parent(A,B) :- child(B,A).`), we will not find a literal with a correlation of 1.0 and thus learn an empty theory.

```
procedure AllTheories(Examples)

Cutoff = 1.0
Theories = ∅
while (Cutoff > 0.0) do
        Theory = Fossil(Examples, Cutoff)
        Cutoff = MaximumPrunedCorrelation(Theory)
        Theories = Theories ∪ Theory
return(Theories)
```

Figure 7: Algorithm to generate all theories learnable by FOSSIL

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 specific 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 specific 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 specific 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 a king-rook-king (KRK) chess endgame. 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 in a more or less general to specific order (*top-down*). 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 cost of generating merely the most specific theory (at least in cases where the cutoff occurs near the end of the learned theory, which is frequently the case).

Based on the above ideas, we have conceived the algorithm shown in figure 9. It uses the basic algorithm of figure 7 not to find the best theory, but — in order to avoid over-generalization — tries to find the most specific among all reasonably good theories that can be learned by FOSSIL and uses this theory as a starting point for *Reduced Error Pruning*. More precisely it generates theories in a general-to-specific order, evaluates them on a designated test set of the data (usually 1/3) and stops when the measured accuracy of one of the theories falls below the measured accuracy of the best theory so far minus one *Standard Error* for classification[4]. The last theory within the 1-SE margin, which hopefully is a little too specific,

[4]This is based on an idea in CART (Breiman, Friedman, Olshen, and Stone 1984), where the most *general*

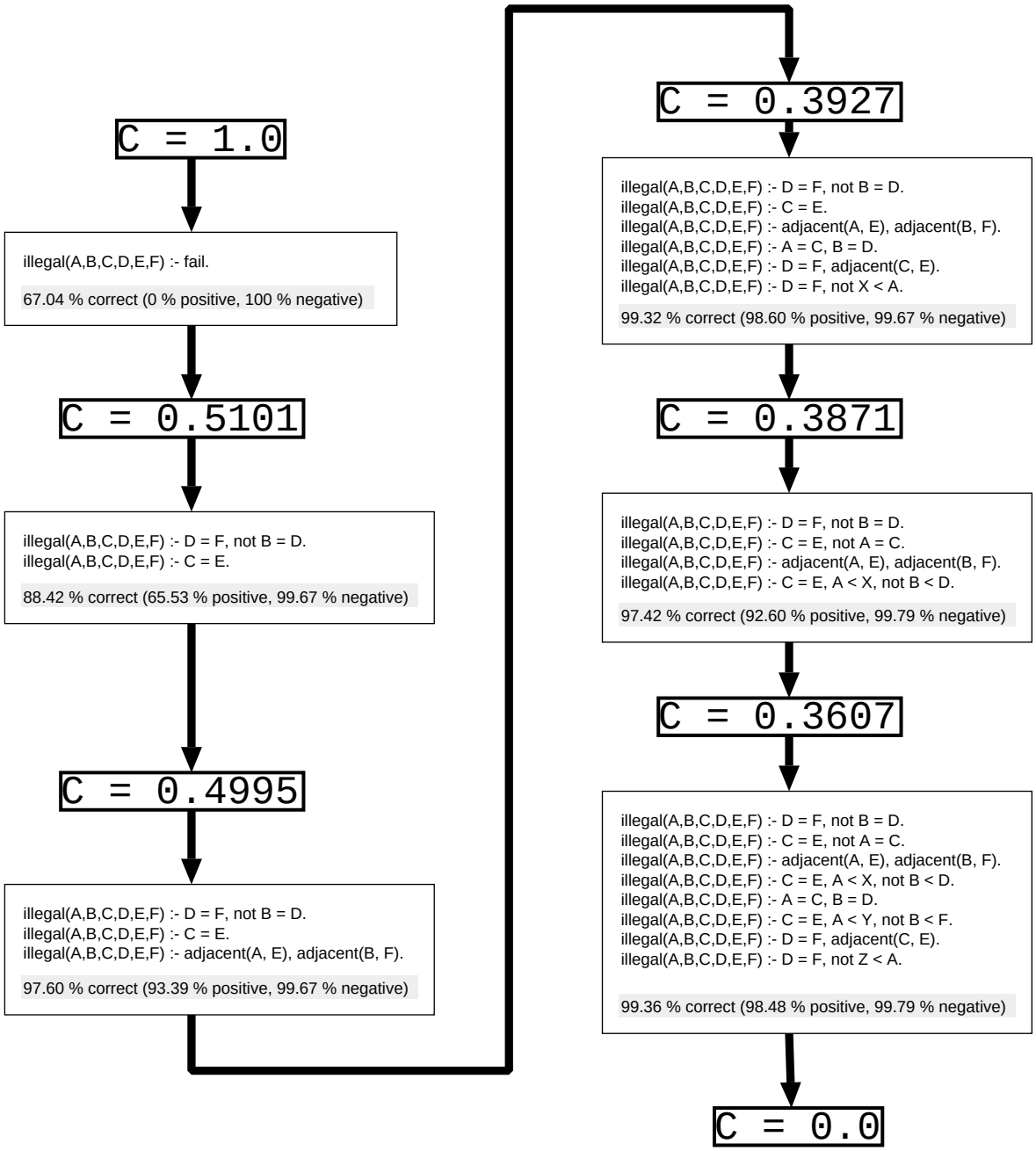


Figure 8: Generating a series of theories in the KRK domain

```
procedure TDP(Examples, SplitRatio)

Cutoff = 1.0
BestTheory = ∅
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 = PruneTheory(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*.

but not too general, will subsequently be generalized using *Reduced Error Pruning*. Because of the initial general-to-specific 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,[5]
- *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 sometimes a cutoff happens at a point where only a small fraction of the available positive examples are covered. Clearly such theories are useless. Therefore we added the constraint that only theories that cover more than 50% of the positive examples in the growing set will be evaluated on the pruning set. If a theory does not fulfill 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.[6]

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 mis-classification (estimated on the pruning set) and N is the number of examples in the pruning set.

[5]This argument, of course, only apply to noisy domains. In non-noisy domains the most specific 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.

[6]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.

5.2 Experimental Results

We compared *Top-Down Pruning* (TDP) to *Reduced Error Pruning* (REP) in terms of accuracy and run-time on the KRK endgame domain with 10% artificial noise added. The setup of the experiments will be described in more detail at the beginning of section 7.1. Both algorithms split the supplied data sets into the same growing (ca. 2/3) and pruning sets (ca. 1/3). Both algorithms used *Reduced Error Pruning* as described in (Brunk and Pazzani 1991) for their post-pruning phase. 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.[7]

Average Accuracy (10 runs)		100	250	500	750
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: Accuracy in the KRK domain with 10% noise.

Table 1 shows that TDP is not worse than REP in terms of predictive accuracy. REP was only better at a training set size of 250, where TDP heavily over-pruned 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 probably too small to be statistically significant.

Comparing the accuracies of the intermediate theories shows that TDP starts with significantly better theories than REP (see the first line of table 1). 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.

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 general 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 growing 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 specific 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.

[7] 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.

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**

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

These results confirm that TDP exhibits a fast convergence towards good theories and is faster than REP 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 done.

6 Integrating Pre- and Post-Pruning

The algorithm that we will present in this section was 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 we attempted to solve is that post-pruning approaches do not take into account that pruning a clause will generalize it so that it eventually covers more examples of the training set, which may influence 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. Thus it can be avoided that these examples influence the learning of the following clauses.

Figure 10 shows a pseudo-code version of the 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. The resulting rule 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 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.

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

Theory = ∅
while Positive(Examples) ≠ ∅
      Clause = ∅
      SplitExamples(SplitRatio, Examples, GrowingSet, PruningSet)
      Cover = GrowingSet
      while Negative(Cover) ≠ ∅
            Clause = Clause ∪ FindLiteral(Clause, Cover)
            Cover = Cover(Clause, Cover)
      loop
           NewClause = SimplifyClause(Clause, PruningSet)
           if Accuracy(NewClause, PruningSet) < Accuracy(Clause, PruningSet)
              exit loop
           Clause = NewClause
      if Accuracy(Clause, PruningSet) ≤ Accuracy(fail, PruningSet)
         exit while
      Examples = Examples − Cover
      Theory = Theory ∪ Clause
return(Theory)
```

Figure 10: Integrating Pre- and Post Pruning with *Incremental Reduced Error Pruning*

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). 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:

Efficiency: I-REP's asymptotic complexity is of the order $\Omega(n \log^2 n)$, n being the size of the training set. This is significantly lower than the complexity of growing an overfitting theory which has been shown to be $\Omega(n^2 \log n)$ under the same assumptions (Cohen 1993). As in REP, growing one clause from purely random data costs $n \log n$ (each of the approximately $log(n)$ literals has to be tested against n examples). I-REP considers *every* literal in the clause for pruning, i.e. each of the $\log n$ literals has to be evaluated on n examples until the final clause has been found, i.e. at most $\log n$ times. Thus the costs of pruning one clause are $n \log^2 n$. Assuming that the size of the final theory is constant, the overall costs are also $\Omega(n \log^2 n)$.

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, overly specific concept description, while I-REP directly constructs the final theory.

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 this problem of the incompatibility of the learning strategy with the pruning strategy cannot appear in I-REP.

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. 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.

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

Table 3: Average Run-Time

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 to learn an intermediate overfitting theory. It can also be seen that GROW's pruning algorithm is much faster than REP's, which confirms the results of (Cohen 1993).

In order to get an idea on the asymptotic complexity of the various algorithms we have performed a log-log analysis as in (Cameron-Jones 1994). We estimate the asymptotic complexity by dividing the differences between the logarithms of two run-times by the differences of the logarithms of the corresponding training set sizes and thus estimating the slope of a log-log-plot. We have tabulated the slopes for adjacent training set sizes in table 4.

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

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

In fact, the table suggests that I-REP has a sub-quadratic time complexity. This is consistent with our conjecture that I-REP's time complexity is $\Omega(n \log^2 n)$. In general the results we get are consistent with the analysis performed in (Cameron-Jones 1994) for random data, which is not surprising when we view the noise-level as the degree of randomness in the data. In particular the evidence supports that result that REP has a complexity of $\Omega(n^4)$ and that the initial rule growing phase is $O(n^2 \log n)$ as shown in (Cohen 1993). It also

confirms the main result of (Cameron-Jones 1994), namely that the asymptotic complexity of GROW is *not* below the asymptotic complexity of the initial rule growing phase as has been originally suggested in (Cohen 1993). However, in all our experiments the absolute values for the run-time of GROW's pruning phase were negligible compared to the initial over-fitting phase.

REP often gets caught in local maxima and is not able to generalize 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 in (Cameron-Jones 1994). 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.

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

Table 5: Average Accuracy

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 here, as there is little redundancy in the data because of the small sample size. However, at larger example set sizes I-REP outperforms the other algorithms.

7 Experimental Evaluation

We have tested the algorithms presented in this paper in a variety of domains. All algorithms 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.

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, and 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)`.

The signs of 10% of the training instances were deliberately reversed to generate artificial noise in the data. The learned concepts were evaluated on test sets with 5000 noise-free examples. We used the state-of-the-art relational learner FOIL(Quinlan 1990) as a benchmark.[8] FOIL 6.1, which is implemented in `C`, was used with its default settings except that the `-V 0` option was set 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. to prevent recursion. All algorithms were trained on identical sets of sizes from 100 to 1000 examples. All reported results were averaged over 10 runs, except for the training set size 1000, where only 6 runs were 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 1994). 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, except the ones where the white king is between the black king and the white rook and thus blocks a check that would make the position illegal, because 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.

FOSSIL, on the other hand, is the fastest algorithm. FOIL, although implemented in C, is slower, because with increasing training set sizes it learns more clauses than FOSSIL (see also (Fürnkranz 1994)). 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, where post-pruning is integrated into a pre-pruning criterion, 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 becomes also apparent in figure 12, where accuracy (with the standard deviations observed in the different runs) is plotted against the logarithm of the run-time.

[8] 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`. The experiments were performed with version 6.1.

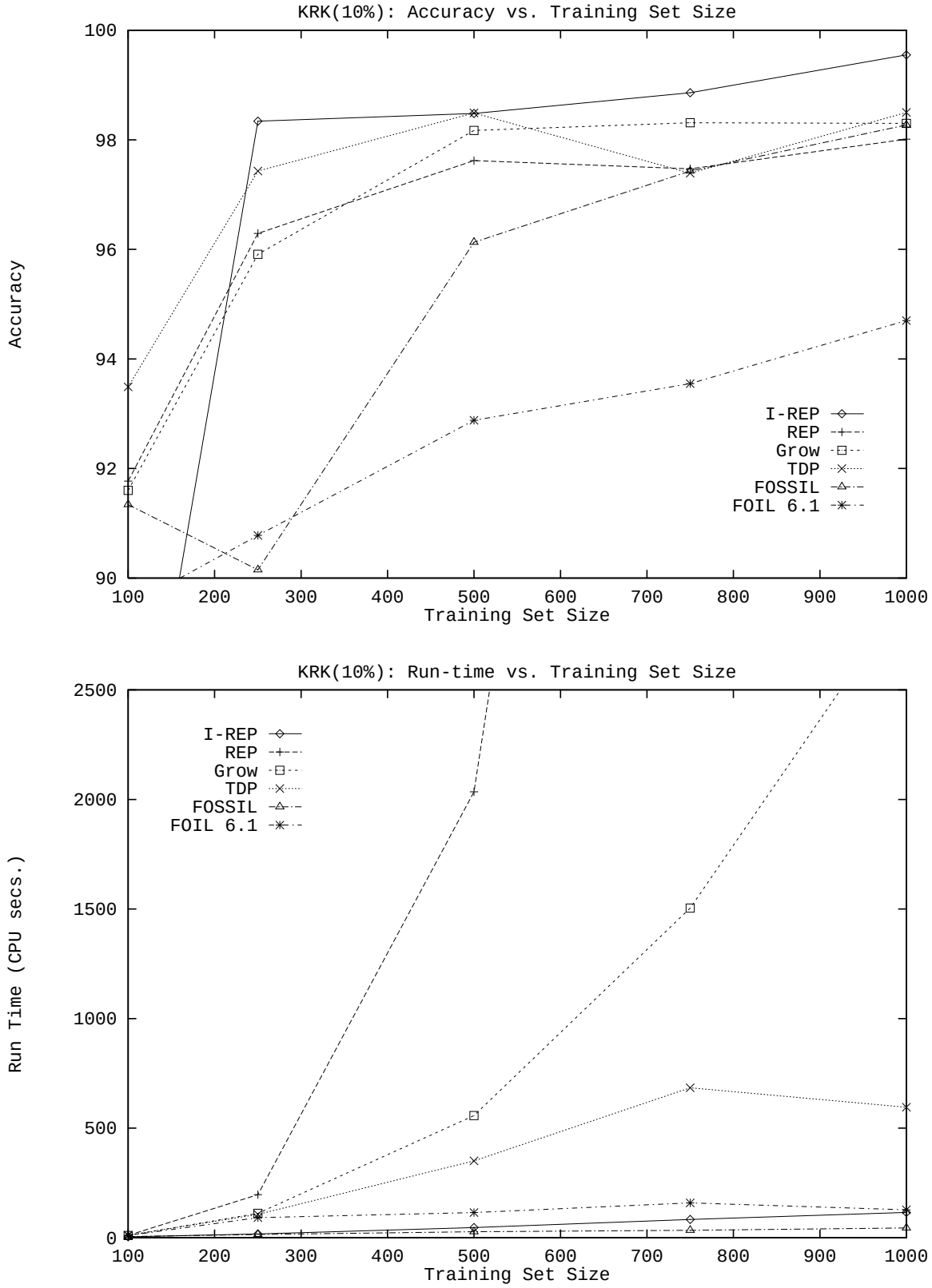


Figure 11: KRK domain (10% noise), different training set sizes

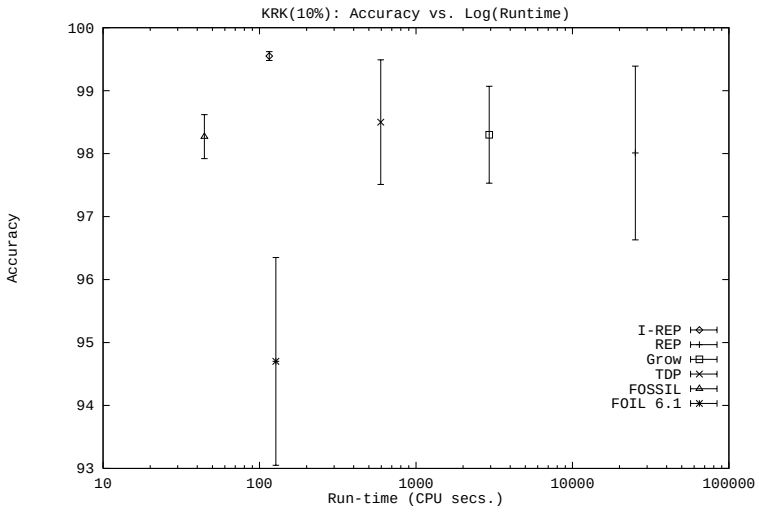


Figure 12: KRK domain (10% noise), 1000 examples

7.2 The Mesh Domain

We have also tested our algorithms on the finite element mesh design problem first studied and described in detail in (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 and Muggleton 1992; Džeroski and 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 (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. The learned theories were tested as in (Quinlan 1994), which is a little different from the setup used in (Džeroski and Bratko 1992): Instead of actually predicting a value for the number of finite elements on an edge, we merely checked for all possible values whether this value could be derived from the learned rules or not. The basic difference is that we tested on ground instances, whereas (Džeroski and Bratko 1992) tested the target predicate with an unbound value for the number of finite elements for positive examples. In our setting we also had to test the learned theories on negative examples to make sure that they are not over-general. In table 6 two numbers are given for each of the five sets: the first number is the accuracy on the positive examples only, while the second number shows the accuracy when testing on the negative examples as well. 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. However, TDP is not faster than GROW, although it starts the pruning phase

Algorithm	Accuracy	Only +	Run-time
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

Table 6: Experiments in the mesh domain

with a simpler theory. The reason for this is that our implementation of TDP uses REP to prune the theory that results from the initial search for a good starting theory. It might be worthwhile to further improve TDP by using the GROW algorithm for its post-pruning phase. But this also indicates that in this domain TDP's initial top-down search was not as effective as in the KRK domain, because more work was left for the post-pruning phase.

The only algorithm faster and more accurate than I-REP is FOSSIL with a cutoff of 0.3. 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 which shows how poorly all algorithms do in this domain. We hope to be able to improve our results in this domain by trying the faster algorithms on the new data set (Dolšak, Bratko, and Jezernik 1994) which contains a total of 10 objects (and thus hopefully provides more redundancy). However, for this comparative study the new data set was too big.

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 did not improve accuracy as much as removing the entire rule did. Therefore the overall accuracy of the theory was primarily optimized by deleting many rules that cover a few positive examples, but also an equal or greater number of negative examples. 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. The appendix of (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 9 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 the implementation of the learning algorithms. In the *Lymphography* 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 as described in (Holte 1993). For all data sets the task was to learn a definition for the minority class.

In all datasets the background knowledge consisted of `<` and `=` relations with one variable

Breast Cancer	Accuracy	Stnd. Dev.	Range	Time
C4.5	71.96	4.36	—	—
FOSSIL	73.33	4.56	17.66	19.68
No Pruning	65.39	4.21	13.27	169.70
REP	69.97	3.80	12.16	257.29
GROW	68.46	4.72	15.39	183.67
No Pruning (TDP)	67.98	5.56	20.62	154.05
TDP	71.74	3.79	12.43	173.31
I-REP	70.89	5.23	19.58	28.97
Hepatitis	Accuracy	Stnd. Dev.	Range	Time
C4.5	81.23	5.12	—	—
FOSSIL	76.07	5.77	23.43	217.40
No Pruning	73.66	4.99	17.12	101.66
REP	76.96	3.93	10.80	102.28
GROW	76.45	4.24	11.14	102.39
No Pruning (TDP)	76.33	3.40	10.92	115.41
TDP	79.42	3.88	11.87	116.24
I-REP	78.66	2.80	7.34	60.40
Sick Euthyroid	Accuracy	Stnd. Dev.	Range	Time
C4.5	97.69	0.40	—	—
FOSSIL	97.58	0.40	1.35	891.40
No Pruning	96.25	0.51	1.70	4554.65
REP	97.55	0.32	1.06	5040.23
GROW	97.52	0.47	1.64	4635.26
No Pruning (TDP)	97.37	0.51	1.78	2965.51
TDP	97.49	0.43	1.21	3010.97
I-REP	97.48	0.50	1.70	970.70

Table 7: Results in the Breast Cancer, Hepatitis, and Sick Euthyroid domains.

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. Run-times for all datasets were measured in CPU seconds for SUN SPARCstations ELC except for the *Mushroom* and *KRKPa7* datasets which are quite big and thus had to be run on a considerably faster SPARCstation S10. All experiments followed the setup used in (Holte 1993), 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 can be found in tables 7, 8, and 9. Each line shows the average accuracy on the 10 sets, its standard deviation and range (difference between the maximum and the minimum accuracy encountered), and the run-time of the algorithm. The results of C4.5, a decision tree learning system with extensive noise-handling capabilities (Quinlan 1993), are taken from the experiments performed in (Holte 1993) and are meant as an indicator of the performance of state-of-the-art decision tree learning algorithms on these data sets.

A short look shows that the results vary in terms of accuracy, but are quite consistent in

Glass (G2)	Accuracy	Stnd. Dev.	Range	Time
C4.5	74.26	6.61	—	—
FOSSIL	77.32	4.79	15.96	216.42
No Pruning	75.24	5.26	18.15	91.89
REP	77.76	4.31	14.73	93.31
GROW	75.63	4.69	16.97	93.11
No Pruning (TDP)	77.23	4.01	12.64	85.56
TDP	75.90	6.18	20.51	87.39
I-REP	76.31	4.89	15.95	63.01
Votes	Accuracy	Stnd. Dev.	Range	Time
C4.5	95.57	1.31	—	—
FOSSIL	95.35	1.17	3.34	105.22
No Pruning	94.69	1.89	6.55	50.45
REP	95.84	1.39	3.92	57.41
GROW	95.63	1.36	3.92	53.84
No Pruning (TDP)	95.33	1.22	4.48	60.88
TDP	95.22	1.54	4.49	62.17
I-REP	94.75	1.75	6.95	22.43
Votes (VI)	Accuracy	Stnd. Dev.	Range	Time
C4.5	89.36	2.45	—	—
FOSSIL	89.07	2.64	8.13	88.94
No Pruning	86.46	2.01	7.36	124.47
REP	86.72	3.46	10.78	163.26
GROW	87.49	3.35	10.93	137.49
No Pruning (TDP)	87.57	1.36	4.29	105.67
TDP	85.85	2.62	9.21	113.05
I-REP	87.25	3.27	10.75	38.78

Table 8: Results in the Glass and Votes domains.

run-times: I-REP is the fastest algorithm in 6 of the 9 test problems, while it is second-best in 2 of the remaining 3. The tables also confirm 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 datasets, but by far the slowest on other data sets.

Most differences in accuracies are not statistically significant.[9] Significant differences can be found in the *KRKPa7* chess endgame domain, where TDP and FOSSIL performed significantly (1%) worse than all other algorithms. FOSSIL was significantly (5%) better than TDP in the *Votes (VI)* domain and outperformed (5%, sometimes 1%) all other algorithms

[9]We have used a range test which can be used to quickly determine significant differences between medium values for small ($N < 20$) sample sizes (Mittenecker 1977). For $N = 10$ the value of $L = \frac{\mu_1 - \mu_2}{R_1 + R_2}$ has to be > 0.152 for a significance level of 5% and > 0.210 for a significance level of 1%. (μ_i are medium values and R_i are ranges. Both can be found in tables 7 – 9.)

KRKPa7	Accuracy	Stnd. Dev.	Range	Time
C4.5	99.19	0.27	—	—
FOSSIL	95.17	2.66	8.63	2383.61
No Pruning	97.92	0.58	1.85	4063.80
REP	97.84	0.54	2.01	4243.08
GROW	97.48	0.41	1.06	4219.00
No Pruning (TDP)	96.26	1.85	4.74	2368.28
TDP	96.41	1.87	4.74	2376.28
I-REP	97.74	0.36	1.32	1785.50
Lymphography (2 classes)	Accuracy	Stnd. Dev.	Range	Time
C4.5 (on all 4 classes)	77.52	4.46	—	—
FOSSIL	87.22	4.39	17.23	20.79
No Pruning	83.25	4.79	16.03	17.05
REP	81.85	4.86	16.83	18.81
GROW	82.10	5.28	17.53	18.42
No Pruning (TDP)	83.73	5.50	17.53	18.66
TDP	81.86	4.39	12.39	20.27
I-REP	79.17	4.42	15.30	10.14
Mushroom	Accuracy	Stnd. Dev.	Range	Time
C4.5	100.00	0.00	—	—
FOSSIL	99.96	0.03	0.11	3538.19
No Pruning	100.00	0.01	0.04	1878.51
REP	99.97	0.05	0.15	1931.75
GROW	99.57	0.66	1.56	2088.81
No Pruning (TDP)	100.00	0.01	0.04	4595.23
TDP	99.97	0.05	0.15	4656.31
I-REP	99.97	0.04	0.11	2493.77

Table 9: Results in the Chess (KRKPa7), Lymphography, and Mushroom domains.

in the *Lymphography* domain. In general C4.5 seems to be a little superior to the other algorithms (one cannot count the results on *Lymphography* where the rule learning algorithms had a presumably easier 2-class task.). However, the rule learning algorithms seem to be competitive.

To allow a more structured analysis we have grouped the 9 domains into 3 subclasses: Table 7 contains all domains where overfitting seems to be harmful, i.e. where REP's post-pruning phase significantly (at least 5%) improves upon the concepts learned by the initial overfitting phase.[10] Table 8 contains domains where pruning does not make a significant difference and finally table 9 contains all domains where pruning cannot be recommended as exemplified by the *Mushroom* data, where the overfitting phases learned 100% correct concept descriptions that were significantly better (5%) than those learned by all pruning

[10]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).

algorithms. The *Mushroom* and *KRKPa7* domains are known to be free of noise, while the medical domains of table 7 are noisy. Therefore we assume that our grouping of the domain corresponds to the amount of noise contained in the data.

8 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 is overfitting the training data, but is still fairly general. This theory is then used as a starting theory for a subsequent post-pruning phase that tries to generalize 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 a general-to-specific order so that a good starting theory can often be found in considerably less time than would be needed for generating the most specific 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 most specific theory.

I-REP integrates pre- and post-pruning into one algorithm. Instead of post-pruning entire theories, each rule is pruned after 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 domains with high redundancy. As real-world databases typically are 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.

I-REP and TDP were 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 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 efficient 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. For instance, it has been pointed out in (Cohen 1995) that accuracy estimates for low-coverage rules will have a high variance and therefore I-REP is likely to stop prematurely and to over-generalize in domains that are susceptible to the *Small Disjuncts Problem* (Holte, Acker, and Porter 1989). (Cohen 1995) also points out some deficiencies of the accuracy-based pruning criterion and shows how a stopping criterion based on description length and a better pruning criterion can significantly improve I-REP's accuracy without a loss in efficiency.

Another way for improving I-REP has been tried in (Fürnkranz 1995). Just as I-REP tries to improve upon REP by pruning at the rule level instead of the theory level, we have investigated a way of taking this further and tried to improve upon I-REP with an algorithm

that prunes at the literal level. The resulting algorithm — I^2-REP — seemed to be a little more stable at low training set sizes, but no significant differences in run-time could be observed (Fürnkranz 1995). Besides, I^2-REP appeared to be a little slower than I-REP, although asymptotically both algorithms are clearly subquadratic.

Currently we are investigating the merits of avoiding the loss of information that is caused by the need of splitting the training set into separate growing and pruning sets. In particular techniques based on the well-known *Minimal Description Length* principle could provide valuable alternatives.

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 and Research. I would like to thank Gerhard Widmer for patiently reading and improving numerous versions of this paper.

References

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

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

Cameron-Jones, R. (1994, May). The complexity of Cohen's grow method. Unpublished draft for comments.

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

Clark, P. & T. Niblett (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. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, pp. 988–994.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*.

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

Dolšak, B. & S. Muggleton (1992). The application of Inductive Logic Programming to finite-element mesh design. In S. Muggleton (Ed.), *Inductive Logic Programming*, pp. 453–472. London: Academic Press Ltd.

Džeroski, S. & I. Bratko (1992). Handling noise in Inductive Logic Programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan.

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

Fürnkranz, J. (1994). Fossil: A robust relational learner. In *Machine Learning: ECML-94*, pp. 122–137. Springer-Verlag.

Fürnkranz, J. (1994). Top-down pruning in relational learning. In *Proceedings of the 11th European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, pp. 453–457.

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

Fürnkranz, J. (1995). A tight integration of pruning and learning (extended abstract). In N. Lavrač and S. Wrobel (Eds.), *Machine Learning: ECML-95*, Lecture Notes in Artificial Intelligence 912, pp. 291–294. Springer Verlag.

Fürnkranz, J. & G. Widmer (1994). Incremental Reduced Error Pruning. In *Proceedings of the 11th International Conference on Machine Learning*, New Brunswick, NJ, pp. 70–77.

Holte, R., L. Acker, & B. Porter (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, MI.

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

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

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

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

Mittenecker, E. (1977). *Planung und statistische Auswertung von Experimenten* (8th ed.). Vienna, Austria: Verlag Franz Deuticke. In German.

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

Niblett, T. & I. Bratko (1986). Learning decision rules in noisy domains. In *Proceedings of Expert Systems 86*. Cambridge University Press.

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

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach*, pp. 463–482. Tioga Publishing Co.

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. In *Proceeding of the 11th International Conference on Machine Learning*, New Brunswick, NJ, pp. 233–241.

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

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

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