Dissertation

Efficient Pruning Methods for Relational Learning

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines *Doktors der technischen Wissenschaften*

eingereicht an der Technisch-Naturwissenschaftlichen Fakultät
der *Technischen Universität Wien*

von

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

OEFAI-TR-94-28

Kurzfassung

Diese Dissertation beschäftigt sich mit effizienten Methoden für fehlertolerantes maschinelles Lernen unter Berücksichtigung von relationalem Hintergrundwissen. Während sich klassische Algorithmen auf das Lernen von propositionalen Konzepten in der Form von Entscheidungsbäumen oder Entscheidungslisten beschränken, sind relationale Lernalgorithmen in der Lage, nicht nur Wissen über Datenattribute und deren Werte, sondern auch über Relationen zwischen einzelnen Attributen in den Lernprozeß miteinzubeziehen. Aufgrund der mächtigeren Repräsentationssprache dieser Algorithmen — im Prinzip lernen sie PROLOG Programme zur Klassifikation der Lernbeispiele — werden sie dem relativ jungen Gebiet der induktiven logischen Programmierung zugerechnet, einer neuen Forschungsrichtung an der Schnittstelle von maschinellem Lernen und logischer Programmierung.

In dieser Arbeit werden einerseits verschiedene bekannte Methoden zur Erreichung von Fehlertoleranz untersucht und in ein einheitliches Schema gebracht, andererseits jedoch drei neue, verbesserte Algorithmen vorgestellt. Die beiden grundlegenden Ansätze zum sogenannten Prunen sind entweder zu versuchen, Datenfehler während des Lernvorgangs zu erkennen, oder sie vorerst zu ignorieren und danach die daraus resultierenden Fehler im gelernten Konzept zu entdecken und zu korrigieren. Beide Methoden haben ihre Vorzüge, weshalb im Zuge dieser Dissertation Möglichkeiten untersucht werden, diese beiden Verfahren zu kombinieren bzw. zu integrieren. Die Nützlichkeit der neuen Algorithmen wird anhand praktischer Experimente in verschiedenen künstlichen und natürlichen Domänen demonstriert.

Abstract

This thesis is concerned with efficient methods for achieving noise-tolerance in Machine Learning algorithms that are capable of using relational background knowledge. While classical algorithms are restricted to learn propositional concepts in the form of decision trees or decision lists, relational learning algorithms are able to include into the learning process not only knowledge about data attributes and values, but also about relations between the attributes. As these algorithms use a more powerful representation language — they learn PROLOG programs for classification — they are part of the recent field of Inductive Logic Programming, a new research area at the intersection of Machine Learning and Logic Programming.

In this work we first review several known methods for achieving noise-tolerance and put them into a unified framework and then introduce three new and improved algorithms. The two basic approaches to pruning are either to try to recognize noise in the data during the learning process or to first learn a theory from the data as they are and subsequently try to detect and correct the resulting mistakes in this theory. Both approaches having their advantages, the major part of this thesis is devoted to trying to combine and integrate them into new powerful algorithms. A series of experiments with artificial and natural data sets demonstrates the usefulness of these approaches.

Acknowledgements

Parts of this work were sponsored by the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (FWF) under grant number P8756-TEC. The Austrian Research Institute for Artificial Intelligence (ÖFAI) is financially supported by the Federal Ministry of Science and Research. Having said what must be said I want to take the opportunity and thank the many people without whom this thesis would not be the same.

Most importantly, my advisor, Gerhard Widmer, read numerous preliminary and final versions of this work over and over again, and patiently improved each one of them with insightful comments and fruitful discussions. He must be as glad as I am that we are finally done.

Robert Trappl has provided me with the opportunity to pursue my research interests at the ÖFAI. The unique atmosphere of this institute has contributed to this research in many ways. Special thanks go to my colleagues Bernhard Pfahringer for many helpful suggestions, Christian Holzbaur for leading me into the abyss of PROLOG, and to paolo petta and John Matiasek for answering a lot of untimely questions.

I'm also very grateful to Ross Quinlan and Sašo Džeroski for letting me use their programs. Mike Cameron-Jones and William Cohen gave me some particularly helpful hints.

Thanks are also due to the guys at the University of Chicago who taught me enough about AI to understand the Yale robot joke. I had a wonderful year in Chicago and truly miss you all.

Finally, I want to thank my parents who continue to believe in me and whose support I can always count on. I hope this time you let *me* send out the announcements.

In the last years I have also profitted immensely from the love and moral support of a dear friend of mine. Unfortunately she does not like acknowledgements and prefers to stay anonymous.

Contents

Chapter 1

Introduction

Your name is Bond, James Bond. You are on a mission, and enemy robots are on their way to stop you. The secret service was able to get the pictures of some of the robots. Special agents have managed to provide a classification of the robots as dangerous or harmless. However, due to counterespionage, their classification might be wrong in some cases. You look at the pictures and see robots with round, square, or octagonal heads or bodies. Each robot is wearing a jacket that is either red, yellow, green, or blue. Some of the robots are smiling and some are wearing ties. And they are holding things like swords, flags, or balloons. Based on the pictures you have to learn to recognize whether the robots are dangerous or harmless. The basic problem, however, is that you cannot get away with just remembering each robot's appearance and its classification, because although your secret service was able to get pictures of some of the robots, they have not been able to get them all. You will be likely to see new robots on your mission and you have to identify those correctly as well. So your goal is to come up with a rule of thumb to be able to discriminate between dangerous and harmless robots. Rules might be as simple as "Harmless robots are smiling." or "Dangerous robots are wearing a sword." However, they might also be as complicated as "Robots that do not have a square head and are wearing a blue jacket, robots that are holding a balloon and are wearing a tie, and robots that are not smiling and whose heads and bodies have the same shape are dangerous."

1.1 Propositional Learning

Machine Learning research has developed a number of powerful algorithms that are very well suited for this and similar kinds of problems. The input of these algorithms is a description of the known data objects in the form of known attributes (e.g. `head_shape`, `body_shape`, `is_smiling`, `holding`, `jacket_color`, `has_tie`). Each data object is described as a vector of values, one

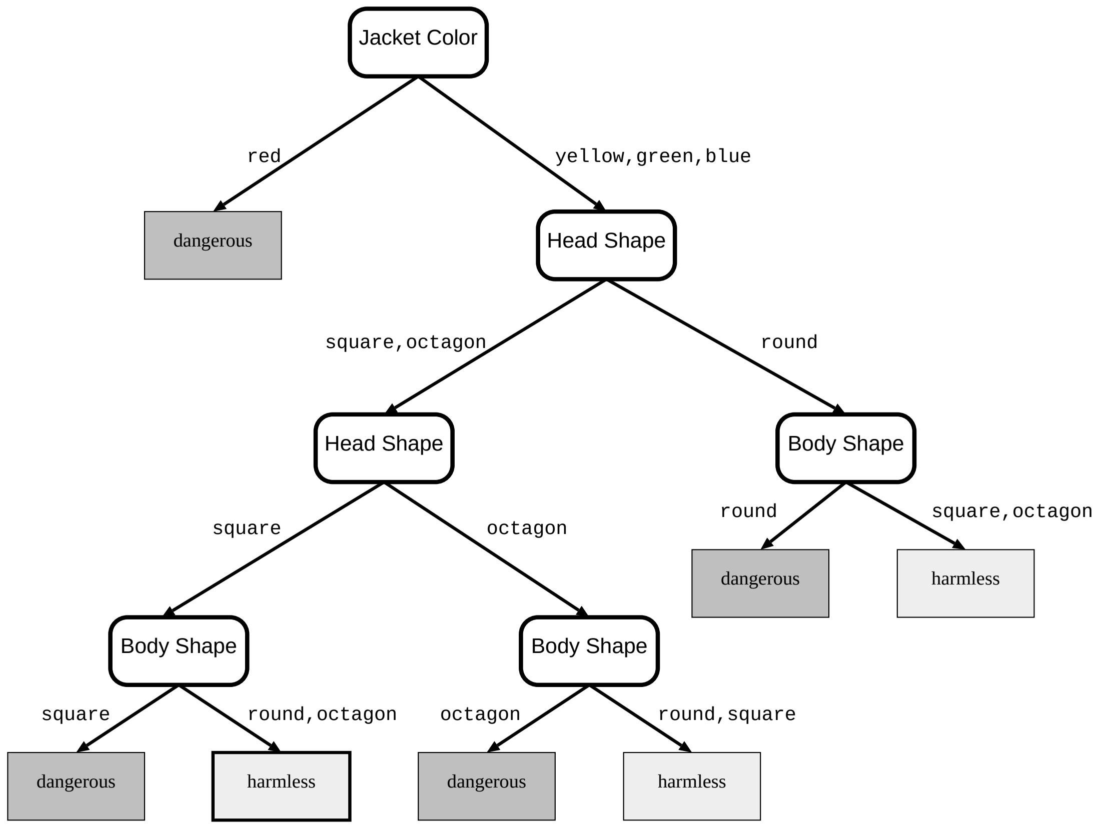


Figure 1.1: A Decision Tree

for each of the attributes, and a classification (e.g. `[robot_3: (square, round, yes, sword, red, no), dangerous]`). The Machine Learning algorithm's output will be a rule which will allow you to derive a (hopefully correct) classification for all objects, the ones you have already seen and the ones you have not seen yet.

The most prominent family of algorithms that attack these problems is based on the success of ID3 [Quinlan, 1983]. All algorithms of this family construct a concept description in the form of a *decision tree* (see figure 1.1). Each *node* of a decision tree typically corresponds to a test on one attribute of the input data, and each *edge* typically represents a subset of the values of the previous node's attribute. Each *leaf* is labelled with one of the different classification possibilities. Classification of an object with a decision tree starts with the root attribute and performs the corresponding test. Depending on the outcome it follows down the edge that has the appropriate label and performs the test of the next node. This is repeated until one arrives at a leaf of the tree, in which case its classification label is returned. The most prominent representa-

```
dangerous(Head,Body,Smiling,Holding,Jacket,Tie) :-
        red(Jacket).
dangerous(Head,Body,Smiling,Holding,Jacket,Tie) :-
        Head = Body.
```

Figure 1.2: PROLOG solution for problem 1

tives of this family are C4.5 [Quinlan, 1993], Assistant [Cestnik *et al.*, 1987] and CART [Breiman *et al.*, 1984]. [Fürnkranz, 1991] contains a summary of some basic methods for inducing decision trees from attribute-value data sets.

1.2 The Need for Relations in Learning

In [Thrun *et al.*, 1991] three problems are proposed in the robots domain. Each of them addresses a problem that is hard for decision tree learning algorithms. The first of these problems is the following:

Problem 1: *Robots whose head and body are equally shaped and those who are wearing a red jacket are dangerous.*

Propositional learning algorithms can only make use of the values of the attributes, but not of relations between these values. As the correct solution for problem 1 should test whether the equality relation holds between the attributes `head_shape` and `body_shape`, propositional algorithms are not able to find this description if they only know the given six attributes.

Figure 1.1 shows a binary decision tree solution for problem 1. It is easy to see that the program has to generate one rule for each special case of the equality between the shapes of the heads and bodies, because it is not able to directly compare two values of different attributes. If, however, the algorithm were able to learn a concept description in a first-order logic representation, the concept could be as simple and understandable as the PROLOG program of figure 1.2. Comparing the two, we not only realize that the PROLOG program is simpler than the decision tree solution, but we also see that the PROLOG solution is more general than the decision tree solution. If a robot with equal head and body shapes that are neither square, octagon, or round comes along we would not know how to classify it with the decision tree, but we would recognize it as dangerous with the PROLOG rule.

In principle it would be possible to adopt this problem for propositional learning by providing a new hand-coded attribute `head_equal_body` that contains the value `true` for all robots that have equal head and body shapes and the value `false` otherwise. However, inventing new relevant attributes requires a lot of data engineering and domain knowledge. It can also be a very tiresome process. If e.g. all of the 6 attributes were of the same type, then the knowledge engineer would have to consider $\binom{6}{2}$ new possible attributes for each binary relation. One relational learning algorithm, LINUS, tries to automate this process (see page 10).

Like the decision tree algorithms, other classic propositional learning algorithms (including CN2 [Clark and Niblett, 1989], AQ [Michalski, 1980, Michalski *et al.*, 1986], and others) are also limited to concept descriptions in propositional calculus. The algorithms we will discuss in this thesis will be able to make use of relations in order to get simpler and more understandable concept descriptions.

1.3 Noise Tolerance

Another important problem for decision tree algorithms is *noise tolerance*. Problem 3 of [Thrun *et al.*, 1991] illustrates this difficulty.

Problem 3: *Robots are dangerous when their jacket is green and they are holding a sword or when their jacket is not blue and the shape of their bodies is not octagon. Be aware that due to counterespionage the classification of 5% of the known robots may be wrong.*

The problem with learning this comparatively simple concept is that you have to find a good generalization, although you cannot entirely rely on the examples you are given. The purpose of this exercise is to simulate learning from real-world databases which very often contain *noise*, i.e. erroneous or incomplete instances. There may be many reasons for noise in databases: manual data collection and entry is error-prone, measuring devices may fail occasionally, some attribute values may not always be available, classification by human experts is often inconsistent, and many more.

Research in propositional learning algorithms has already developed a variety of methods for noise handling. Noise-tolerant decision tree learning algorithms like CART [Breiman *et al.*, 1984], C4.5 [Quinlan, 1993], ASSISTANT [Bratko and Kononenko, 1986], CN2 [Clark and Boswell, 1991] are well-known and often applied to real-world problems. However, many of the solutions employed in these programs are tailored for propositional learning algorithms, so that the developement of noise-tolerant relational learning algorithms is currently a very active research field.

1.4 Overview

The goal of this thesis is two-fold: First we put a variety of noise-tolerant relational learning algorithms into a unified framework and show how they are related to each other. Chapter 2 will give a short introduction into the new research area called *Inductive Logic Programming* (ILP) and will introduce a prototypical **F**irst **O**rder **I**nductive **L**earner — FOIL [Quinlan, 1990]. In the following two chapters 3 and 4 we will review well-known methods for noise handling in this relational learning system. Chapter 3 will also introduce our relational learner FOSSIL which uses new search and pre-pruning heuristics, and show that it compares very well to other approaches known from the literature.

The second goal of this research is to show how this framework can be naturally extended to incorporate two new approaches to noise-tolerance in relational learning. In chapter 5 we will introduce an algorithm that combines the two basic methods of chapters 3 and 4 in an elegant way. Chapter 6 will then show how a tight integration of these approaches naturally gives rise to another new and powerful algorithm for noise handling in Inductive Logic Programming. Chapter 7 then compares all of the discussed algorithms, in particular the three new methods, on a variety of natural and artificial test problems. The main results are summarized and discussed in chapter 8.

Chapter 2

Inductive Logic Programming

Inductive Logic Programming (ILP) can be viewed as research in the intersection of Logic Programming and Inductive Machine Learning. Informally speaking the field is concerned with the induction of PROLOG programs. As we have seen in chapter 1, being able to express the discovered knowledge in a first-order logic representation language can overcome some of the limitations of classical learning algorithms. Most of them learn from an attribute-value representation of the input data and their representational power thus is restricted to decision trees as in the ID3 family [Quinlan, 1983] or propositional Horn clauses as in AQ [Michalski *et al.*, 1986] or CN2 [Clark and Niblett, 1989]. ILP algorithms, on the other hand, can not only test attributes for specific values, but also make use of relations (like equality) between the different attributes.

Inductive Logic Programming has quickly developed to a very active research field. The expressiveness of first order logic has lead to many different research areas within ILP.

Relational Learning (also called *Empirical ILP*) will be the main emphasis of the research presented in this thesis. Relational learning algorithms learn classification rules for a concept. The program typically receives a large collection of positive and negative examples from real-world databases as well as background knowledge in the form of relations. The prototypical example for this research is FOIL [Quinlan, 1990] and its various successors, but there are several other approaches like LINUS [Lavrač *et al.*, 1991] and GOLEM [Muggleton and Feng, 1990].

Theory Revision systems are not so much concerned with the induction of a useful theory, but with the maintenance of a complete and consistent theory. Theory revision systems constantly monitor the performance of their theory and attempt to *generalize* it when is unable to explain an observation and *specialize* it when one of its predictions does not turn out to be correct. Typical revision systems are MIS [Shapiro, 1982], CIGOL [Muggleton, 1988], CLINT [De Raedt, 1992], KRT (MOBAL) [Wrobel, 1994], FORTE [Richards, 1992] and AUDREY [Wogulis, 1991].

Predicate Invention is an important research direction aiming at introducing new background knowledge in order to gain expressiveness or to make the resulting theory more understandable [Stahl, 1993]. Common methods to invent new predicates include detecting data dependencies [Flach, 1993] or clause completion heuristics [Kijsirikul *et al.*, 1992]. Many systems try to invert deductive reasoning and are able to invent new predicates that way. Some invert one step of a resolution proof [Muggleton and Buntine, 1988, Wrobel, 1989, Rouveirol and Puget, 1990], i.e. they are basically given a consequent and preconditions and try to guess a rule that can prove the consequent from the preconditions. To invent recursive predicates, typically all resolution steps involving the recursive predicate should be considered at once, a process that has been called inverting implication [Muggleton, in press, Lapointe *et al.*, 1993].

Discovery is a very new branch of ILP which differs from most other approaches in that there is no particular goal concept specified in advance, but the program tries to find interesting regularities by itself. The most prominent systems are CLAUDIEN [De Raedt and Bruynooghe, 1993] and LAGRANGE [Džeroski and Todorovski, 1993].

Of course, several systems would fit into more than one category. An excellent overview of the history of the field can be found in [Sammut, 1993], a selection of some of the most important papers in [Muggleton, 1992]. [Lavrač and Džeroski, 1993] is an introductory book on Inductive Logic Programming with a strong focus on relational learning systems, in particular on LINUS [Lavrač *et al.*, 1991] and mFOIL [Džeroski and Bratko, 1992a]. Other introductory texts include [De Raedt and Lavrač, 1993], [Muggleton, 1993], and [Muggleton and De Raedt, 1994].

2.1 Relational Learning

The main concern of this thesis will be *Relational Learning* or *Empirical Inductive Logic Programming*. Learning systems of this category typically are designed to learn classification rules from real-world databases. Main characteristics of real-world databases are that they are large and unreliable. Algorithms designed to work on them must therefore be efficient and noise-tolerant [Matheus *et al.*, 1993].

Given:

- a set of positive and negative training examples for the *target relation*
- background knowledge (typically in the form of relational tables)

Find:

- a logic program that is able to derive all of the positive and none of the negative training examples using the relations provided in the background knowledge.

Figure 2.1: A Generic Relational Learning Task

Relational Learning systems are designed for this purpose and have already been applied to various real-world problems, like finite-element mesh design [Dolšak and Muggleton, 1992, Džeroski and Bratko, 1992a], medical diagnosis [Lavrač *et al.*, 1993, Petta, 1994], chess endgames [Morales, 1992, Muggleton *et al.*, 1989, Quinlan, 1990, Džeroski and Bratko, 1992a], natural language understanding [Zelle and Mooney, 1993], document recognition [Esposito *et al.*, 1993b] etc. Some of the systems even produced new knowledge that was of considerable interest for researchers in the application domain and has been published in journals of their subject area [Muggleton *et al.*, 1992, Sternberg *et al.*, 1992, King *et al.*, 1992]. For an overview of applications of empirical ILP systems consult [Bratko and King, 1994].

The learning task for most relational learning systems is shown in figure 2.1. We will exemplify it with a short example. In figure 2.2 the learning system has the task of learning the concept `father`. It already knows about `female` and `male` persons and it knows that `christopher` and `penelope` are the `parents` of `arthur` and `victoria`.[1] The user now tells the system that `christopher` is the `father` of his children while `penelope` is not and that `christopher` is not the father of `penelope`. The learner's task is to find a definition for the *target relation*. In our example the result would hopefully be

```
father(A,B) :- male(A), parent(A,B).
```

Different methods have been found to solve this problem:

[1] The `known_literal` and `target` parts of the background knowledge tell the system what clauses of the background knowledge it should use and what the target concept is. They also contain information about types, modes and symmetries of the arguments, which will be explained in section 2.3.3. More details about the syntax of `known_literal` and `target` can be found in appendix B.

```
% Target relation

target(father(A,B),[A-person,B-person]).


% Training instances

pos_instance(father(christopher,arthur)).
pos_instance(father(christopher,victoria)).

neg_instance(father(penelope,arthur)).
neg_instance(father(christopher,penelope)).


% Relations in the background knowledge

known_literal(male(X),[X-person],[+],[]).
known_literal(female(X),[X-person],[+],[]).
known_literal(parent(X,Y),[X-person,Y-person],[+,+],[]).


% Definition of the background knowledge

parent(christopher,arthur).
parent(penelope,arthur).
parent(christopher,victoria).
parent(penelope,victoria).

male(christopher).
male(arthur).
male(colin).

female(victoria).
female(penelope).
```

Figure 2.2: Example input for a relational learner

Top-Down Induction: This method is most common and will be our main concern in this research. It tries to find a complete and consistent theory by iterative *specialization* of the most general (empty) theory. The prototype system of this approach is Foil [Quinlan, 1990]. We will discuss this approach in more detail in section 2.2.

Bottom-Up Induction: As opposed to top-down induction, the bottom-up approaches search for a complete and consistent theory by iterative *generalization* of clauses. The most common generalization operators are based on *inverse resolution* as in CIGOL [Muggleton and Buntine, 1988] or on *relative least general generalization* [Plotkin, 1971] as in Golem [Muggleton and Feng, 1990]. A generic framework for bottom-up induction can be found in [Rouveirol *et al.*, 1993].

Representation Change: The systems of this class approach the problem by reformulating it in a language that conventional Machine Learning systems can use. Prototypical for this approach is the LINUS system [Lavrač *et al.*, 1991] that transforms the relational learning problem into an attribute-value representation that can be used by propositional learning algorithms like CN2 [Clark and Niblett, 1989] and ASSISTANT [Cestnik *et al.*, 1987].

All three approaches impose different restrictions on the class of programs they can learn. LINUS e.g. is not able to learn recursive predicates, while Golem is limited to a restricted form of ground background knowledge. Top-down induction methods have similar constraints, which we will discuss in more detail in the next section.

2.2 Separate-and-Conquer Rule Learning Algorithms

Many ILP algorithms address the relational learning task specified in figure 2.1 with the so-called *separate-and-conquer* strategy which is well-known from propositional learning algorithms. It has its roots in the early days of Machine Learning in the covering algorithm of the famous AQ family [Michalski, 1980, Michalski *et al.*, 1986]. CN2 [Clark and Niblett, 1989, Clark and Boswell, 1991] eliminated AQ's dependence on an initial seed example and employed the separate-and-conquer strategy for the first time. The term separate-and-conquer has been coined by [Pagallo and Haussler, 1990] in the context of learning decision lists. Finally, it was first used in the Foil algorithm for efficiently inducing logic programs [Quinlan, 1990], which pioneered significant research in the field of relational learning. Although the work reported here will be mainly presented

```
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)
    Examples = Examples $-$ Cover
    Theory = Theory $\cup$ Clause
return(Theory)
```

Figure 2.3: A Separate-and-Conquer Rule Learning Algorithm

in the relational learning framework, we think that parts of this thesis are also very relevant for propositional learning algorithms that employ the same or a similar learning strategy.

2.2.1 The Basic Algorithm

Figure 2.3 shows the basic SeparateAndConquer rule learning algorithm. It constructs rules by successively adding conditions in the form of PROLOG literals with a certain variabilization to the right-hand side of the current rule. Each relation of the background knowledge with each possible variabilization is considered and the most promising one is selected with a greedy search heuristic. 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*).

2.2.2 Search Heuristics

The remaining question is how a literal with a certain variabilization is selected as a condition for a rule in the FindLiteral subroutine. There are several search heuristics known from decision tree learning, most of them perform similarly (see e.g. [Mingers, 1989b, Buntine and Niblett, 1992]). [Lavrač *et al.*, 1992b] and [Lavrač *et al.*, 1992a] list a selection of several search heuristics for Foil-like empirical ILP system (see figure 2.4).

Probability	$P(c) = \frac{p}{p+n}$
Laplace Estimate	$P(c) = \frac{p+1}{p+n+2}$
m**-Estimate**	$P(c) = \frac{p+m\frac{P}{P+N}}{p+n+m}$
Information	$I(c) = \log_2 \frac{1}{P(c)}$
Gain	$HG(c) = H(c) - H(c')$
Weighted	$WHG(c) = w \times H(c)$
	(e.g. $w = \frac{p}{p'}$ or $w = p$)

Figure 2.4: Search Heuristics for Foil-like ILP algorithms

The simplest heuristic is to estimate the *probability* that an example covered by the current clause c will be positive.[2] This probability is commonly estimated with the relative frequency of the positive examples covered by c, i.e. the proportion of positive (p) examples of the total number of examples ($p + n$) covered by c. If $P(c) = 1$ then only positive and no negative examples are covered and the clause can be completed with the current literal.

The *Laplace* and m-estimates try to improve the estimation of $P(c)$. In the Laplace estimate a correction term is introduced to improve the quality of the relative frequency probability estimate for low sample sizes, while the m-estimate tries to trade off between the *a posteriori* distribution ($\frac{p}{p+n}$) and the *a priori* distribution ($\frac{P}{P+N}$) of the positive and negative examples.[3] The improved probability estimates can be used in any of the other search heuristics. mFoil [Džeroski and Bratko, 1992a] uses them directly for the simple probability search heuristic. [Cestnik, 1990] gives a closer analysis of these two heuristics.

The *information* heuristic estimates the amount of information necessary to specify that an example covered by the clause is positive. This estimate can simply be calculated by taking the logarithm of the inverse of the probability $P(c)$.

While the *probability* and *information* heuristics estimate the quality of a clause using the population of examples covered by it, *gain* heuristics try to estimate the improvement that would be gained by adding a literal to an incomplete clause c'. They calculate the heuristic value (using any of the above heuristics) of c' and subtract it from the heuristic value of the clause c which results from adding the current literal to c'. The literal which maximizes this difference will be chosen.

[2] c is assumed to be the current incomplete clause plus the literal under scrutiny.

[3] The a priori distribution is estimated with the relative frequency of positive examples (P) in the training set ($P + N$) *before* learning.

Several approaches also tried to incorporate different search biases by assigning different *weights* to different candidate literals. Usually weights are chosen that give a higher value to literals that cover many positive examples in order to avoid very specific rules (i.e. rules that only cover a few examples). The example of section 2.2.3 shows why this can be very useful in some cases.

The heuristic used in FOIL itself is *weighted information gain*. It uses the difference in information between the clause with or without the current literal and multiplies it with the number of positive examples covered by the former.

2.2.3 Example

We will look at a short example (figure 2.5) that illustrates the behavior of a FOIL-like algorithm. It tries all literals with all possible variabilizations using the variables of the head of the clause (`father(A,B)`) and counts how many positive and negative examples would be covered by the clause consisting of the head and the literal added to the body. For example the clause

```
father(A,B) :- male(A).
```

would be true for both positive examples of the training set, but also for one of the negative examples. With a the simple probability heuristic the literals `male(A)`, `\+female(A)`[4], and `parent(A,B)` all three would be estimated with a heuristic value of 2/3. All three of them could be equally useful for constructing a definition for the concept `father`. In our example the first of them is (arbitrarily) chosen as the first literal in the clause and the three examples that are covered by this literal form the new training set. Note that one of the negative examples has already been ruled out by this literal.

With the new training set FOIL performs a second pass and again examines all literals[5]. We now have two literals that cover some positive examples and no negative examples — `male(B)`, and `parent(A,B)`. The former only covers 1 positive example, while the latter covers both. Using the simple probability heuristic, both literals would be equally valid choices (both of them have a heuristic value of 1), although the former would be a bad choice. The resulting clause

```
father(A,B) :- male(A), male(B).
```

would only "explain" one of the positive training examples, namely `father(christopher,arthur)`. In this case the program would remove this

[4]Read as `not female(A)`.

[5]It would not be necessary to re-examine the literals that already appear in the clause, i.e. the literal `male(A)`.

```
| ?- foil(father(A,B),Clauses).

2 positive and 2 negative instances left.

male(A):        covers 2 + and 1 -
\+male(A):      covers 0 + and 1 -
male(B):        covers 1 + and 1 -
\+male(B):      covers 1 + and 1 -
female(A):      covers 0 + and 1 -
\+female(A):    covers 2 + and 1 -
female(B):      covers 1 + and 1 -
\+female(B):    covers 1 + and 1 -
parent(A,A):    covers 0 + and 0 -
\+parent(A,A):  covers 2 + and 4 -
parent(A,B):    covers 2 + and 1 -
\+parent(A,B):  covers 0 + and 1 -
parent(B,A):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 4 -
parent(B,B):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 4 -

Chose literal male(A).

2 positive and 1 negative instances left.

male(A):        covers 2 + and 1 -
\+male(A):      covers 0 + and 1 -
male(B):        covers 1 + and 0 -
\+male(B):      covers 1 + and 1 -
female(A):      covers 0 + and 0 -
\+female(A):    covers 2 + and 1 -
female(B):      covers 1 + and 1 -
\+female(B):    covers 1 + and 0 -
parent(A,A):    covers 0 + and 0 -
\+parent(A,A):  covers 2 + and 4 -
parent(A,B):    covers 2 + and 0 -
\+parent(A,B):  covers 0 + and 1 -
parent(B,A):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 4 -
parent(B,B):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 4 -

Chose literal parent(A,B).

Found clause: father(A,B):-male(A),parent(A,B)


Clauses = [(father(A,B):-male(A),parent(A,B))] ?

yes
| ?-
```

Figure 2.5: Trace of a FOIL-like algorithm

example from the training set and would learn a second (presumably equally meaningless) rule for the remaining example.

However, if we choose the weighted probability heuristic for our program, it would give preference to the second literal, `parent(A,B)`, because it covers 2 positive examples instead of only 1 for the literal `male(B)`. In this case the algorithm has discovered a complete and consistent program.

2.3 Problems

The *separate-and-conquer* learning strategy has proven to be very efficient and powerful in learning simple logic programs. However, the simplicity of this approach has its limitations and causes several problems that have to be solved.

2.3.1 Introducing New Variables

As the reader may have noticed, in example 2.5 only the variables that already appeared in the head of the clause were used for the literals in the body. For the current example this was sufficient to induce a correct concept description. But if the goal were to learn the concept

```
grandparent(A,B) :- parent(A,C), parent(C,B).
```

the program would need to be able to introduce new intermediate variables like `C`. In fact this particular example would cause no problems for FOIL. It is highly characteristic for grandparents that they are parents as well. So the feature of being a parent is quite likely to discriminate between grandparents and some (but not all) people who are not grandparents. When the literal `parent(A,C)` is selected, adding `parent(C,B)` would perfectly discriminate between grandparents and other people.[6]

However, if we consider the seemingly equivalent task of learning the definition

```
grandchild(A,B) :- child(A,C), child(C,B).
```

[6]A prerequisite for this is that the values for `C` can be determined from the background knowledge. Usually background knowledge is represented as PROLOG ground facts, so that the values to which `A` will be bound in the literal `parent(A,C)` determine the values of the new variable `C`. There are two different approaches how subsequent literals are evaluated: FOIL *counts the proofs*, i.e. whenever a literal using one of the new variables is added to the clause, all possible instantiations of this variable will be considered and it is counted how many of them will prove the current instantiation of the head of the clause. A more efficient method is to merely *count the instances*, i.e. it is only checked whether there is a substitution that would prove the head or not. Both variants seem to be reasonable choices.

a problem occurs: *everybody* is the child of somebody. Being a child does not discriminate between grandchildren and other people. Adding the literal `child(A,C)` to the body of the clause would therefore not exclude any negative examples and the heuristic value of adding the literal to the body of the clause would be very low for all of the greedy heuristics listed in figure 2.4.

In [Quinlan, 1991] a solution is proposed that works for a powerful subclass of the problematic cases, the so-called *determinate literals*. Whenever no literals with a heuristic gain above a certain threshold can be found, FOIL tries to improve the situation by adding determinate literals and thus introducing new variables. A literal containing new variables is called determinate if it has exactly one instantiation that proves each of the positive training instances, and if at most one instantiation proves each of the negative training examples. Determinate literals include commonly used PROLOG predicates such as `plus(X,Y,Z)`, `multiply(X,Y,Z)`, `append(X,Y,Z)` and many more. However, the `grandchild` example could not be solved with determinate literals either, because each child usually has two parents, i.e. `child(A,C)` would have more than one instantiation for each example for a grandchild and would therefore not be determinate.

The algorithms discussed here will in general ignore this problem and only use heuristic gain as criterion for adding literals to a clause.

2.3.2 Recursion and Cut

Learning recursive predicate definitions is no fundamental limitation for FOIL. It is able to learn recursive concepts like

```
ancestor(A,B) :- parent(A,B).
ancestor(A,B) :- parent(A,C), ancestor(C,B).
```

This is handled easily by including the instances of the target relation into the background knowledge that can be added to the right hand side of a clause. However, one has to take care that not all variabilizations of recursive predicates are valid. Otherwise the program would very elegantly discover the "solution"

```
ancestor(A,B) :- ancestor(A,B).
```

There are many other pitfalls for a program that learns recursive definitions. [Cameron-Jones and Quinlan, 1993a] extensively discuss this problem and provide a fairly general solution. However, we feel that being able to learn recursive predicates is not a major issue in learning in real-world domains and have restricted our programs to non-recursive concept definitions.

Similarly, we are not concerned with learning cuts in our PROLOG programs. However, contrary to recursion, learning cuts seems to be a very hard problem. The main reason for this is that a cut only has a procedural meaning and therefore cannot be easily induced from declarative data [Bergadano *et al.*, 1993]. In particular FOIL-like algorithms that learn one clause at a time will have difficulties in learning cuts, because many cuts only makes sense in the context of the whole program.

2.3.3 Search Space

The more relations there are in the background knowledge and the more variables there are in the target relation, the more possible literals have to be examined at each step. Although both the number of background relations and the number of variables are usually beyond the control of the user, several provisions can be made to cut down the size of the search space.

Types: One way of restricting the number of literals that have to be evaluated at each step is to use type restrictions. For each relation (including the target) the type of the arguments is specified. Only variables of the same[7] type are considered as possible variabilizations for the arguments of the background relations. With this mechanism it could e.g. be enforced that only arguments of the type `number` are tried for the `<` relation. In the example of figure 2.2 the second arguments of the `target` and `known_literal` predicates are used to specify that all arguments of the specified relation are of the type `person`.

Modes: Mode declarations are a simple way of restricting the places at which new variables can be introduced in the relations of the background knowledge. In figure 2.2 the third argument of the `known_literal` predicate is used to specify the mode of the argument of the predicate. A `+` indicates that only old variables should be used for the corresponding argument, while a `-` would allow a new variable at this place.

Symmetries: A third way for reducing the search space is to specify symmetries in the arguments of a background relation. It would be unnecessary work e.g. to examine `plus(Y,X,Z)` if we already had evaluated `plus(X,Y,Z)`, because both literals can be considered identical. The fourth argument of a `known_literal` declaration can be used to specify a list of pairs of variables that are symmetrical (like `[X-Y]` in our example).

As relational learners are targeted for learning from large amounts of real-world data, efficiency is a major issue. Therefore the above mechanisms for

[7]A similar mechanism could be proposed for incorporating type *hierarchies*.

restricting the search space have been included into our implementations of all learning algorithms discussed later on. A description of the interface to these methods can be found in appendix B.

2.3.4 Noise

The simple SEPARATEANDCONQUER algorithm of figure 2.3 has a severe drawback for learning from real-world data. As already discussed in section 1.3, real-world data may be imperfect. Some instances may be erroneously classified, or some of the attribute values may be missing or wrong. In short, the 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 2.3 tries to explain all of the positive examples and none of the negative examples, i.e. in the presence of noise it will find explanations for negative examples that are erroneously considered to be positive and try to exclude positive examples that are believed to be negative. Typically this results in a very complex theory that is hard to understand and is not very predictive on unseen examples.[8]

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

Pre-Pruning means that during concept generation some training examples are deliberately ignored, so that the final concept description does not classify all training instances correctly. In decision tree learning these pre-pruning approaches include ID3 [Quinlan, 1986] or ASSISTANT [Bratko and Kononenko, 1986].

Post-Pruning means that first a complete and consistent concept description is generated from a proportion of the training data. The resulting concept is then analyzed with the remaining, unseen examples and, if necessary, is generalized to improve the accuracy on these examples. In decision tree learning post-pruning approaches have e.g. been used in *Reduced Error Pruning* [Quinlan, 1987], CART [Breiman *et al.*, 1984] or ASSISTANT [Cestnik *et al.*, 1987].

The main concern of the research reported in this thesis will be efficient pruning methods for rule learning algorithms. In chapters 3 and 4 we will review some of the pruning methods that have been adopted for relational concept learning

[8]Figure 4.2 shows an example in the KRK chess endgame domain with artificial noise added. A correct domain definition and some good approximations can be found in Appendix A.

systems. We also present our relational learner FOSSIL that employs a new search heuristic based on statistical correlation and effectively combines it with a very simple and efficient pre-pruning heuristic. In chapter 5 we then show how pre-pruning can be used for improving the efficiency and accuracy of post-pruning by relaxing the condition of generating a complete and consistent theory first. Finally, in chapter 6, we introduce a method that actually integrates both approaches by using post-pruning methods as a pre-pruning criterion.

Chapter 3

Pre-Pruning

Being able to deal with noisy domains is a must for learning algorithms that are meant to learn concepts from real-world data (see section 2.3.4). *Pre-pruning* deals with this problem by relaxing the constraint of being complete and consistent with the training data and thus admitting over-general concept definitions.

In this chapter we will first introduce several pre-pruning approaches known from the ILP literature (section 3.1). Then we will present FOSSIL, our own FOIL-like algorithm that uses a new search heuristic based on statistical correlation (section 3.2). One of the nice features of this heuristic is that it gives a reliable measure of the heuristic value of a literal on a uniform scale. We show empirically that this feature can advantageously be used for dealing with noise by pre-pruning all literals that have a heuristic value below a certain threshold. We also present empirical evidence that this threshold is robust in the sense that a good value for this parameter is independent of the number of training examples and of the amount of noise in the data (section 3.3).

Parts of this chapter have been previously published in a somewhat different form in [Fürnkranz, 1993a] and [Fürnkranz, 1994b].

3.1 Pre-Pruning in Relational Learning

As we have discussed in section 2.3.4, noise in the data is a problem for the simple SEPARATEANDCONQUER rule learning algorithm of figure 2.3, because it tries to find explanations for every single example in the training set, including the erroneous examples. Explanations for noisy examples typically are very complicated and will decrease predictive accuracy when used to classify unseen examples. It has been suggested by [Holte *et al.*, 1989] that a large proportion of the overall classification error of a theory can be attributed to clauses that cover only a small number of positive examples. This problem is known as the *Small*

```
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)
    Examples = Examples − Cover
    Theory = Theory ∪ Clause
return(Theory)
```

Figure 3.1: A Rule Learning Algorithm Using Pre-Pruning

Disjuncts Problem . In the particular context of learning in noisy domains it is also 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. It 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.

Figure 3.1 shows an adaptation of the simple SeparateAndConquer algorithm that includes a *pre-pruning* heuristic. It can easily be seen that the algorithm is identical to the one of figure 2.3 except that in the inner `while` loop contains a call to the subroutine StoppingCriterion. A *stopping criterion* is a heuristic that determines when to stop adding conditions to a rule, and when to stop adding rules to the concept description. If the current clause with the new literal added fulfills the stopping criterion the inner `while` loop is terminated and the incomplete clause will be added to the concept description. Note that if this clause is empty, no positive (and no negative) examples are covered and the outer loop will terminate as well. In addition (not shown in figure 3.1) many pre-pruning rule learning algorithms use a simple mechanism to monitor the quality of the over-general clauses. Foil e.g. requires that a certain minimum percentage (usually 80%) of the covered examples has to be positive. Otherwise the clause is rejected and no further clauses will be learned.

Most Foil-like algorithms employ stopping criteria for noise handling. The most commonly used among them are

- *Encoding Length Restriction*: This heuristic used in Foil itself

[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, Clark and Boswell, 1991] and later on in the relational learner mFOIL [Džeroski and Bratko, 1992a]. 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[2] to a χ^2 distribution with 1 degree of freedom at the desired significance level. Insignificant rules are rejected.

Both of these stopping criteria require additional computation to determine when to stop learning. In section 3.2.3 we will introduce a much simpler stopping criterion, the *cutoff*, that has been realized in our new relational learning system FOSSIL. But first we will look at some of the features of FOSSIL's correlation search heuristic (section 3.2.1) that are a prerequisite for the use of the cutoff stopping criterion (section 3.2.2).

3.2 FOSSIL

FOSSIL is a SEPARATEANDCONQUER rule learning algorithm that uses a search heuristic based on statistical correlation. In this section we will introduce this heuristic and point out some of the features that distinguish it from other search heuristics used in empirical ILP (see figure 2.4).

3.2.1 The Correlation Heuristic

FOSSIL's evaluation function is based on the concept of statistical *correlation*. The *correlation coefficient* of two random variables X and Y is defined as

$$corr(X,Y) = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \times \sigma_Y} = \frac{E(X \times Y) - \mu_X \times \mu_Y}{\sigma_X \times \sigma_Y} \tag{3.1}$$

[1]The number of bits needed to encode the training instances is $log_2(n) + log_2(\begin{pmatrix} n \\ p \end{pmatrix})$ 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.

[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 μ and σ are *expected value* and *standard deviation*, respectively, of the random variables X and Y. This *correlation coefficient* measures the degree of dependence of two series of points on a scale from -1 (*negative correlation*) to $+1$ (*positive correlation*). In the following we will describe its adaptation as a search heuristic for a SEPARATEANDCONQUER rule learning algorithm.

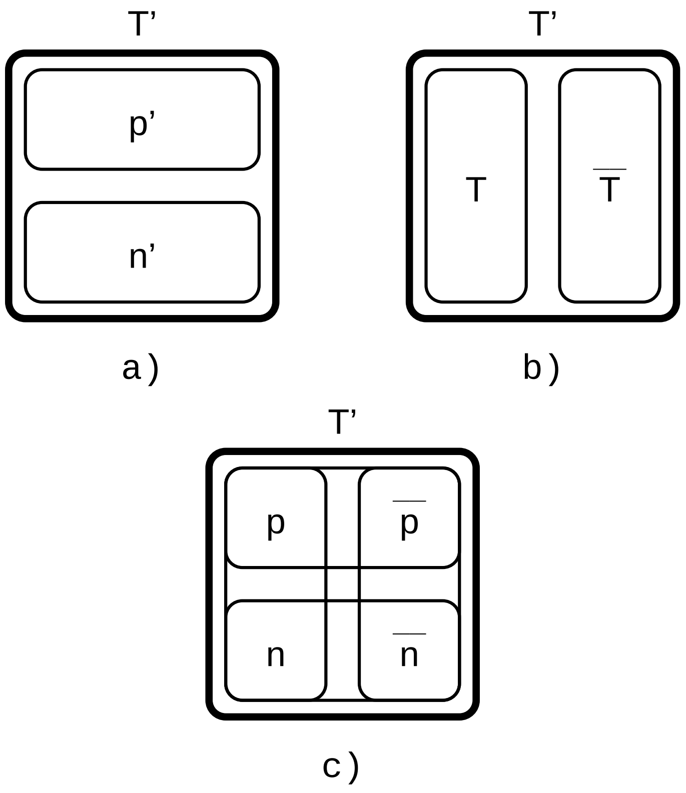


Figure 3.2: Illustration to the Correlation Heuristic

Suppose FOSSIL has learned a partial clause c', which currently covers the example set T', containing p' positive and n' negative instances (see figure 3.2.a).[3] We now assume to have a candidate literal L that may be appended to c' to form a new clause c. c will then cover a set of $T \subseteq T'$ instances, p of them positive and n of them negative. The set $\bar{T} = T' \setminus T$ on the other hand contains $\bar{p}$ positive and $\bar{n}$ negative examples (figure 3.2.b and c). The ideal situation would occur when all positive instances and none of the negative instances are covered by c, i.e. T is equal to the set of positive examples covered by c' ($n = 0$) and $\bar{T}$ is equal to the set of negative examples covered by c' ($\bar{p} = 0$). In short, p is the number of

[3]As in figure 2.4 we denote variables that describe the incomplete clause that is currently examined with a $'$.

the *true positive* examples in T', and $\bar{n}$ the number of the *true negatives*. $\bar{p}$ and n denote the number of *false positives* and *false negatives* respectively.

For each instance we now arbitrarily assign the numeric values $+1$ to positive examples and -1 to negative examples and substitute the series S' of the signs of the examples in T' for the variable X in (3.1). Similarly we form a series S by assigning $+1$ to all instances of T and -1 to all instances in $\bar{T}$ and substitute it for the variable Y. We now can compute the correlation of these two series of values.[4]

The expected values in (3.1) will be estimated by the mean values of S' and S respectively. Standard deviation will be approximated by the empirical variance. Thus we get

$$\begin{aligned} \mu' &= \frac{p'-n'}{|T'|}, \qquad \sigma'^2 = E(S'^2) - E(S')^2 = 1 - \mu'^2, \\ \mu &= \frac{|T|-|\bar{T}|}{|T'|}, \qquad\qquad \sigma^2 = 1-\mu^2 \end{aligned} \tag{3.2}$$

The last remaining term to be computed is $E(S' \times S)$. The p positive examples that are covered by c contribute a $+1$ to this expected value, as do the $\bar{n}$ negative examples that are not covered c. Positive examples that are not covered ($\bar{p}$) and negative examples that are covered (n) contribute a -1. Thus

$$E(S' \times S) = \frac{p+\bar{n}-n-\bar{p}}{|T'|} \tag{3.3}$$

The partial results of above now only need to be substituted into the formula for the correlation coefficient (3.1). As μ' and σ' only need to be evaluated once for each example set T', the evaluation of this formula is not as complicated as it may seem at first sight. Also notice that with this approach no separate calculation for negated literals has to be performed, as a high negative correlation indicates a high dependence on the negated literal.[5]

The literal L with the highest absolute value of the correlation coefficient (or \+L if the sign of the coefficient is negative) is finally chosen to extend c' to form the new clause c. This is based on the assumption that its high correlation with the instances in the current training set T' indicates some form of causal relationship between the target concept and L.

[4]Of course the two series of signs assigned to the examples have to be in the same order, e.g. the order in which the examples appear on the data file.

[5]When new variables are introduced in the body of the clause this symmetry will only hold if the algorithm counts the proofs and not only the instances (see the footnote on page 15).

3.2.2 Interesting Properties of the Correlation Heuristic

Figure 3.3 shows a trace of Fossil on the example file of figure 2.2. Following this trace we will shortly discuss some interesting properties of the correlation heuristic.

```
| ?- fossil(father(A,B),Clauses).

2 positive and 2 negative instances left.

male(A):     Correlation: 0.5774 (covers 2 + and 1 -)
male(B):     Correlation: 0.0000 (covers 1 + and 1 -)
\+female(A): Correlation: 0.5774 (covers 2 + and 1 -)
female(B):   Correlation: 0.0000 (covers 1 + and 1 -)
parent(A,A): Correlation: 0.0000 (covers 0 + and 0 -)
parent(A,B): Correlation: 0.5774 (covers 2 + and 1 -)
parent(B,A): Correlation: 0.0000 (covers 0 + and 0 -)
parent(B,B): Correlation: 0.0000 (covers 0 + and 0 -)

Chose literal male(A).

male(A):     Correlation: 0.0000 (covers 2 + and 1 -)
male(B):     Correlation: 0.5000 (covers 1 + and 0 -)
female(A):   Correlation: 0.0000 (covers 0 + and 0 -)
\+female(B): Correlation: 0.5000 (covers 1 + and 0 -)
parent(A,A): Correlation: 0.0000 (covers 0 + and 0 -)
parent(A,B): Correlation: 1.0000 (covers 2 + and 0 -)
parent(B,A): Correlation: 0.0000 (covers 0 + and 0 -)
parent(B,B): Correlation: 0.0000 (covers 0 + and 0 -)

Chose literal parent(A,B).

Found clause: father(A,B):-male(A),parent(A,B)


Clauses = [(father(A,B):-male(A),parent(A,B))] ?

yes
| ?-
```

Figure 3.3: Fossil trace

- The first thing that becomes obvious when comparing figure 3.3 to the behavior of Foil (figure 2.5) is that Fossil only examines a literal or its negation, but not both. In Foil, the heuristic value of each literal and of its negation have to be calculated separately. FOSSIL, however, is able to do this in one computation: A high positive correlation indicates that the literal under scrutiny is good in reproducing the signs of the positive and negative training examples, while a high negative correlation indicates that it is good in reproducing the inverse signs. The correct signs can in that case be obtained by negating the literal and the heuristic value of this negated literal is the absolute value of the computed correlation. This can be seen clearly from the beginning of the example trace, where FOSSIL finds high correlation with `male(A)` and an equally high negative correlation with `female(A)`. Consequently it will only consider the negated literal `\+female(A)`.

- FOSSIL's correlation coefficient — after taking absolute values and choosing the appropriate (positive or negative) literal — allows to compare the candidate literals on a uniform scale from 0 to 1. This is different from e.g. FOIL, where the weight used in its information gain heuristic directly depends on the number of the training examples and therefore does not have an absolute upper bound (see section 2.2.2).

- The correlation function is symmetric and gives equal consideration to covering many positive and excluding many negative examples. As we have seen in figure 2.5 the unweighted probability heuristic would not be able to decide between the literals `male(B)` and `parent(A,B)` in the second pass of the algorithm. We had to use a weighted heuristic to get the desired behavior. FOSSIL accomplishes the same by not only considering the $p + n$ examples that the new clause c covers, but also taking into account the $\bar{p} + \bar{n}$ examples that it no longer covers.

- The correlation between an example set and a literal that has at least one true instantiation for each example is undefined, because μ will be 1 and thus σ will be 0. Note that this will happen for the problematic cases discussed in section 2.3.1. The problem was that literals that only serve the purpose of computing a dependent value often compute a value for *all* input values. Thus the truth value of the literal is the same for positive and negative examples and it is therefore not able to discriminate between them. FOSSIL offers a flexible way of handling this problem: Defining the heuristic value of these literals as 1 puts all of them into the clause body. Irrelevant literals could be removed later in a post-processing phase. Values between 0 and 1 result in a behavior similar to that described in [Quinlan, 1991]: until a literal with a correlation above a preset value is found, these problematic literals will be added to the clause body in the hope that the new variables they introduce will improve the situation. In most of the experiments reported in this thesis introducing new variables was turned off with mode declarations (see section 2.3.3) and therefore the undefined cases would have been useless and were consequently treated as having correlation 0.

The next section describes how FOSSIL's correlation heuristic can be used for a new and simple, but nevertheless powerful stopping criterion.

3.2.3 The Cutoff Stopping Criterion

FOSSIL is able to judge the relevance of all literals on the same uniform scale from 0 to 1. This allows the user to require the literals that are considered for clause construction to have a certain minimum correlation value — the *cutoff parameter*.

This 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. If no literal with a correlation above the cutoff can be added to the current clause, this clause is considered to be complete. Similarly, if no literal can be found that can start a new clause, the concept definition is considered to be complete. Note that it may happen that FOSSIL "refuses" to learn anything in cases where no predicate in the background knowledge has a significant correlation with the training data.[6]

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 (like e.g. `parent(A,B) :- child(B,A).`). In noisy domains, it can be expected that FOSSIL will over-generalize at high cutoff values, while it will overfit the noise in the data at low settings of this parameter. Thus FOSSIL's cutoff parameter may be viewed as a means for directly controlling the *Overfitting Avoidance Bias* [Schaffer, 1993, Wolpert, 1993]. Finding a good value for the cutoff is therefore an important problem. Section 3.3 will show some empirical evidence that setting the cutoff to 0.3 is a good heuristic, while chapter 5 will show ways for automatically adjusting the cutoff parameter.

One problem that may arise with permitting incomplete clauses (the basic principle of all pre-pruning heuristics), is that they can still be very inaccurate. FOIL's strategy in these cases is to ensure that a certain user-definable percentage — usually 80% — of the examples covered by a clause is positive (see page 21). If the current clause does not fulfill this criterion, FOIL rejects the clause and

[6]This has actually happened several times, and is evident in the result with 50% Noise (i.e. random classification) in table 3.2, where FOSSIL did not learn a single clause in any of the 10 training sets.

assumes that no other reasonable clause can be learned. FOSSIL takes a more cautious approach to solve this problem: If a clause covers more negative than positive examples, it will not be added to the concept description. If a clause passes this criterion it is guaranteed that — on the training set — the accuracy of the concept with the clause will be better than the accuracy of the concept without the clause. The second difference is that if a clause is pruned because of the above criterion, FOSSIL will not stop, but continue to learn from the remaining examples, i.e. from the positive examples that have not been covered by the pruned clause.

3.3 Experimental Evaluation of FOSSIL

For most of the experiments in this thesis we have used the domain of recognizing illegal chess positions in the king-rook-king (KRK) endgame which is described in detail in appendix A.

3.3.1 Setup of the Experiments

The experiments in this section were performed with artificial noise added to the data following the *Classification Noise Process* described in [Angluin and Laird, 1988]. In this model a noise level of η means that the sign of each example is reversed with a probability of η. This means that $\eta\%$ of the data have been wrong at training time. Note that this may differ from other results in the ILP literature, where a noise level of η means that, with a probability of η, the sign of each example is randomly chosen. Thus a noise level of η in our experiments is roughly equivalent to a noise level of 2η in the results reported in [Lavrač and Džeroski, 1992, Džeroski and Bratko, 1992b]. Noise was added incrementally, i.e. instances which had a reversed sign at a noise level η_1 also had a reversed sign at a noise level $\eta_2 > \eta_1$. Similarly, training sets with n examples were fully contained in training sets with $m > n$ examples. In all experiments the induced rules were tested against sets of 5000 randomly chosen noise-free instances.

For the experiments used in this section, typing constraints were used to speed up the search, but no information about modes and symmetries was given, so that the programs could introduce new variables.[7] We have also recorded the number of clauses in the induced concept and the average number of literals per clause to measure the complexity of the learned concept description.

[7]The correct domain theory, given in figure A.2, however, does not require the introduction of new variables. The reason for this setup is that experiments in this section were performed with a preliminary version of FOSSIL that was not yet able to use mode and symmetry information.

3.3.2 Finding a Good Cutoff Value

The first series of experiments were aimed at determining an appropriate value for the cutoff parameter for further experimentation. 10 training sets of 100 instances each were used at three different noise levels (5%, 10% and 20%). We examined 6 different settings for the *Cutoff* parameter. The average results from the 10 runs are reported in table 3.1 and plotted in figure 3.4.

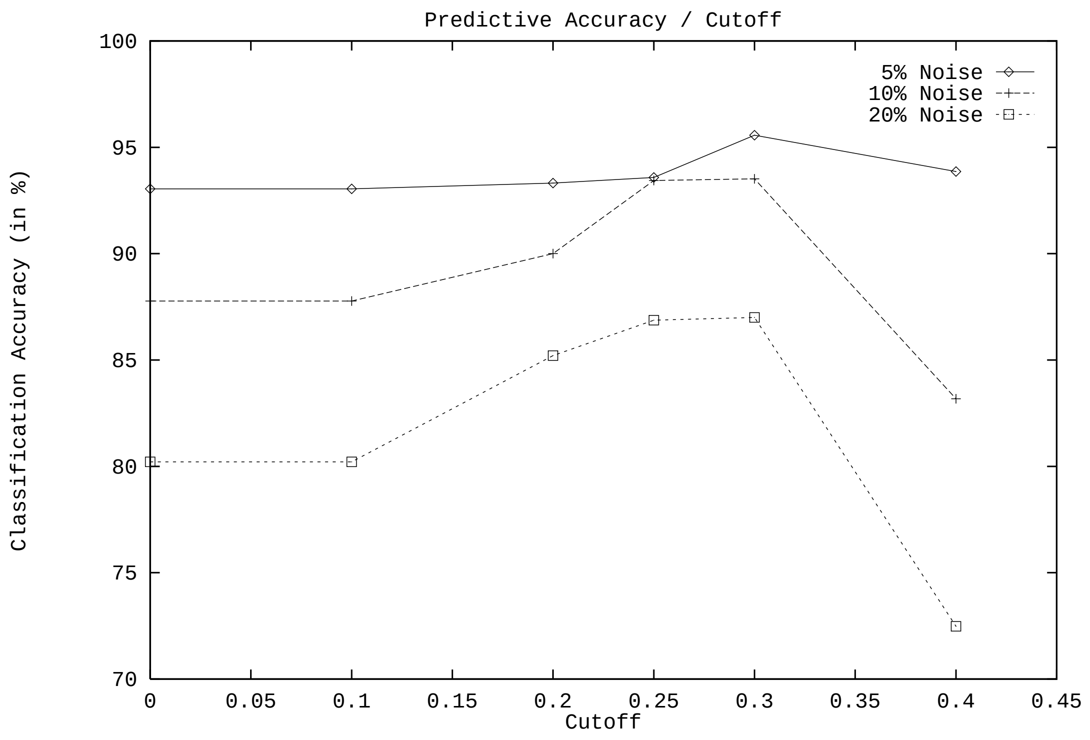
		Cutoff					
Noise		0.0	0.1	0.2	0.25	0.3	0.4
5%	Accuracy	93.05	93.05	93.32	93.58	95.57	93.86
	Clauses	6.3	6.3	6.2	5.8	4.2	2.7
	Lits/Clause	2.25	2.25	2.25	2.19	2.02	1.87
10%	Accuracy	87.77	87.77	90.0	93.44	93.52	83.18
	Clauses	8.2	8.2	6.3	4.5	3.8	1.8
	Lits/Clause	2.74	2.74	2.52	2.24	2.24	1.53
20%	Accuracy	80.21	80.21	85.21	86.87	87.00	72.48
	Clauses	11.4	11.4	6.0	4.1	3.2	0.7
	Lits/Clause	3.09	3.09	2.80	2.76	2.67	0.85

Table 3.1: Experiments with different settings for the *Cutoff*.

The following observations can be made from these graphs:

- A good setting for *Cutoff* in this domain seems to be somewhere around 0.3 for all three noise levels. Coincidentally, the learned concepts are of about equal complexity at this point.
- The curve for the predictive accuracy is U-shaped, similar to some results on pruning decision trees to various degrees (see e.g. [Breiman *et al.*, 1984]).
- There is a transition from overfitting the noise to over-generalizing the rules. A low setting of *Cutoff* has a tendency to fit the noise, because most of the literals will have a correlation above the threshold. Conversely, too optimistic a setting of *Cutoff* results in over-generalization as only a few literals will have a correlation above the threshold.
- The complexity of the learned concepts monotonically decreases with an increase of the cutoff parameter.
- The influence of a bad choice of the cutoff is more significant in data containing a larger amount of noise.

All of this seems to confirm our hypothesis that the cutoff is a means of controlling the amount of allowed overfitting. In section 3.3.3 we will compare the noise tolerance of Fossil to that of Foil.



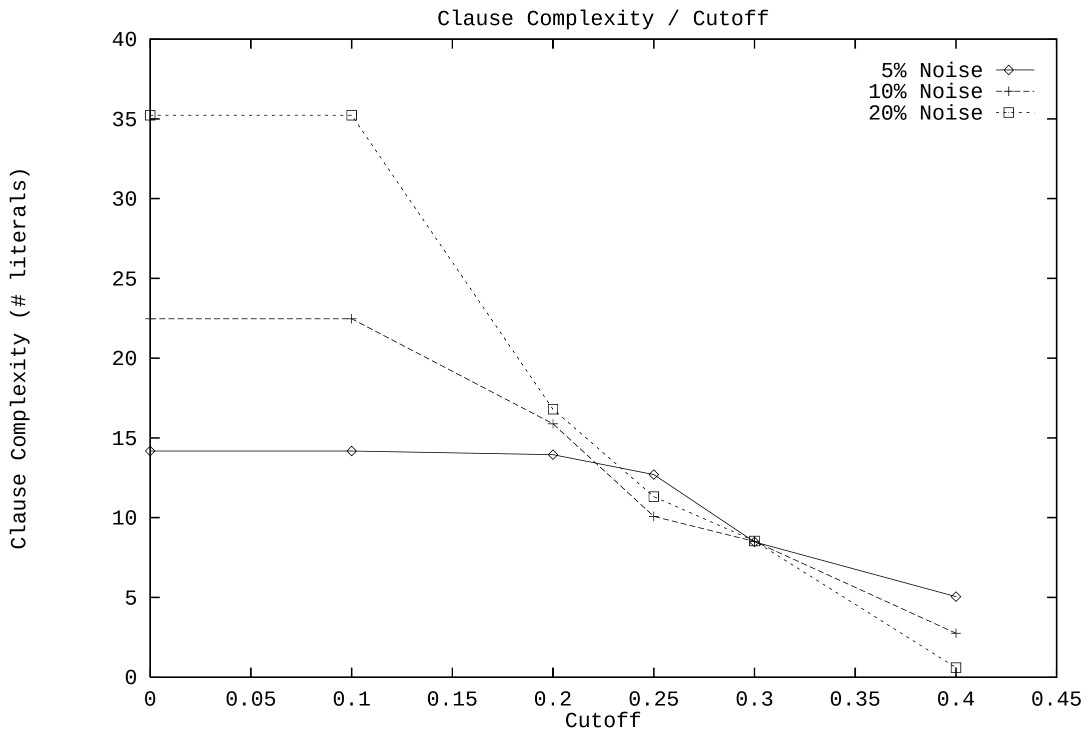


Figure 3.4: Experiments with different settings for the *Cutoff*.

3.3.3 Comparison with FOIL

We conducted two experiments to compare FOSSIL's performance to the performance of FOIL. In the first series we compared the behavior of the two systems on 10 training sets of 100 instances each at different noise levels, which has been the standard procedure for evaluating many ILP systems [Quinlan, 1990, Džeroski and Lavrač, 1991, Džeroski and Bratko, 1992b, Muggleton *et al.*, 1989]. In the second experiment we evaluated both programs at a constant noise level of 10%, but with an increasing number of training instances. According to the results of section 3.3.2 we set $Cutoff = 0.3$ and never changed this setting.

Comparison at different noise levels

In this experiment we compared FOIL4[8] to FOSSIL at different noise levels. In order to have a fair comparison to FOSSIL, we used two versions of FOIL, regular FOIL4 and a new version, FOIL-NBT, where some of FOIL4's enhancements[9] were not allowed. Surprisingly this version performed better than the original FOIL4 in noisy data as can be seen from the results of table 3.2.

An analysis of the result shows that FOSSIL performs best in most of the tests. However the differences are probably not statistically significant. A comparison of the average number of induced clauses and of the average literals per clause shows evidence that FOSSIL over-generalized at the high noise levels. A lower value of the cutoff parameter may result in better performance in the case of 30% noise, although it is unlikely that a useful theory would be learned. An interesting detail is that FOSSIL did not learn anything at a noise level of 50%, i.e. with totally random data. Thus the cutoff mechanism seems to be a primitive, but efficient means of distinguishing noise from useful information.

On the other hand, FOIL4 seems to perform worse than both FOIL-NBT and FOSSIL. The complexity of the concepts learned by FOIL4 increases with the amount of noise in the data, which is clear evidence for over-fitting noise in the data. The next experiment was designed to confirm this hypothesis.

Comparison at different training set sizes

In this series of experiments we compared FOIL without backtracking to FOSSIL at training sets of different sizes with 10% noise. We decided to use FOIL-NBT

[8]FOIL is available by anonymous `ftp` from `ftp.cs.su.oz.au (129.78.8.1)`. The current version (as of September 1994) is 6.2.

[9]FOIL4 has several features that improve the basic algorithm. In particular it is able to save backup points and restart the search at these whenever the greedy search leads to poor definitions. It also relearns entire clauses in some cases. A detailed description of these improvements can be found in [Quinlan and Cameron-Jones, 1993].

Different Noise Levels		Noise 0%	5%	10%	15%
FOIL4	Accuracy	98.32	95.26	92.12	90.26
	# Clauses	3.5	4.2	5.4	5.9
	Lits/Clause	1.64	1.98	2.41	2.47
FOIL-NBT	Accuracy	98.11	95.00	92.98	91.76
	# Clauses	3.5	4.1	4.2	4.2
	Lits/Clause	1.64	1.98	2.34	2.48
FOSSIL (0.3)	Accuracy	98.54	95.57	93.52	92.83
	# Clauses	3.7	4.3	3.8	4.2
	Lits/Clause	1.62	2.02	2.24	2.29
		20%	25%	30%	50%
FOIL4	Accuracy	85.21	79.83	71.53	53.00
	# Clauses	5.7	6.6	8.0	7.9
	Lits/Clause	2.66	2.98	3.03	3.45
FOIL-NBT	Accuracy	87.12	79.42	76.32	55.33
	# Clauses	4.5	5.4	5.0	5.2
	Lits/Clause	2.67	2.80	2.79	3.08
FOSSIL (0.3)	Accuracy	87.00	81.63	70.59	(67.07)
	# Clauses	3.2	2.7	0.7	0.0
	Lits/Clause	2.67	2.69	0.85	0.0

Table 3.2: A comparison of FOIL and FOSSIL on different levels of noise.

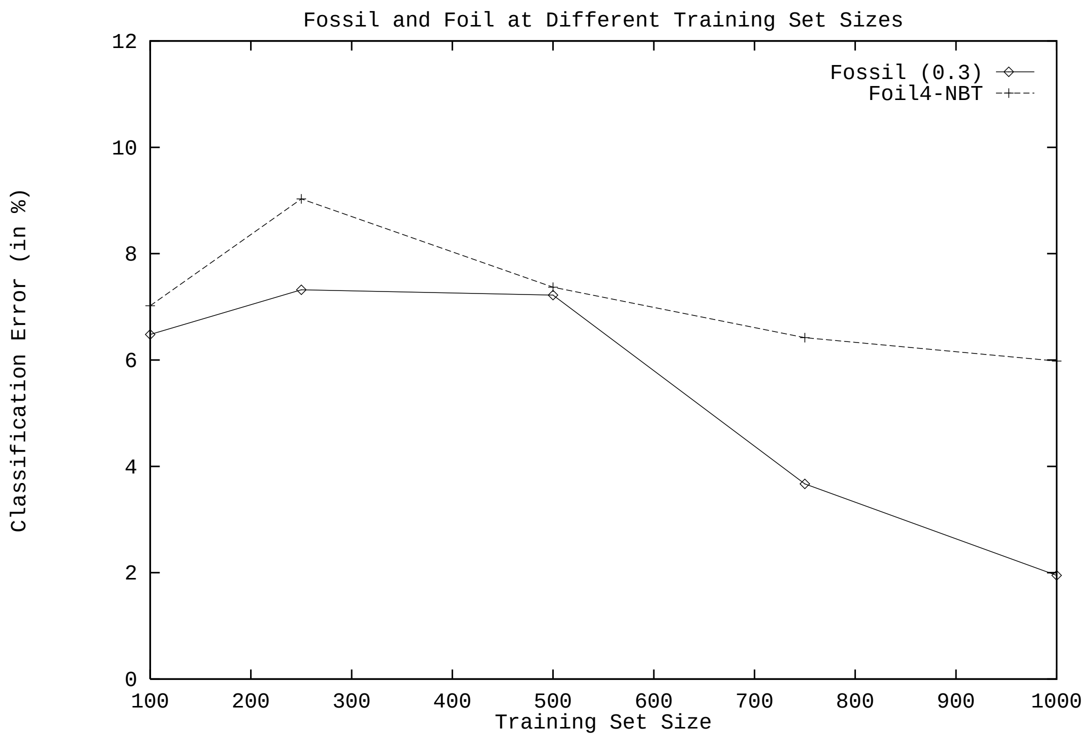
instead of FOIL4, because it performed better in the previous series of tests. Besides, the version without backtracking naturally runs faster, which proved to be important. However, we have done a few sample runs with FOIL4 to confirm that its results would not be qualitatively different from those of FOIL-NBT.

Again, we used 10 different training sets and averaged the results. The outcomes of these experiments are summarized in table 3.3 and figure 3.5.

Different Training Set Sizes (10% Noise)		Training Set Size 100	250	500	750	1000	2000
FOIL-NBT	Accuracy	92.98	90.97	92.63	93.58	94.02	—
	Clauses	4.2	7.7	11.5	16.7	22.0	—
	Lits/Clause	2.34	3.31	3.61	3.89	4.15	—
FOSSIL (0.3)	Accuracy	93.52	92.68	92.79	96.33	98.05	98.41
	Clauses	3.8	3.7	3.1	3.0	3.0	3.0
	Lits/Clause	2.24	3.01	2.63	1.94	1.5	1.4

Table 3.3: A Comparison of FOIL and FOSSIL with different training set sizes

The most important finding is that FOIL clearly fits the noise, while FOSSIL avoids this and learns a slightly over-general, but much more useful theory



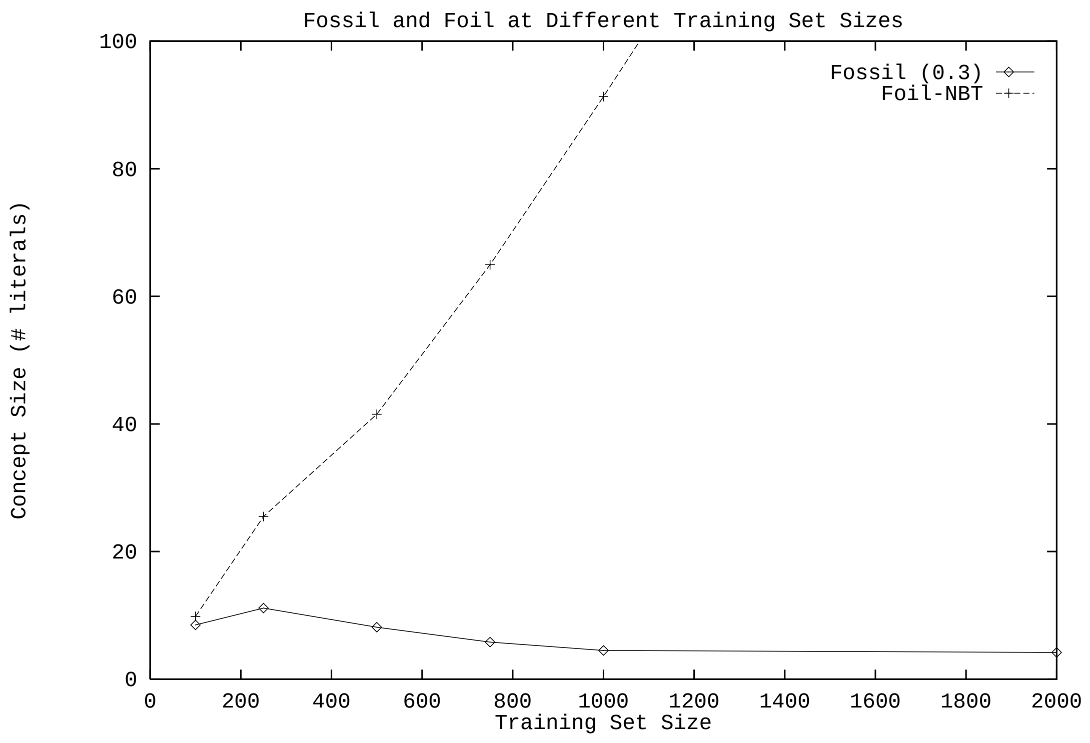


Figure 3.5: A Comparison of FOIL and FOSSIL with different training set sizes

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

Figure 3.6: An approximate theory for the KRK domain (98.54% correct).

instead. FOIL's fitting the noise has several disadvantages:

Accuracy: The more examples there are in the noisy training set, the more specialized are the various clauses in the concept description, which decreases the predictive ability of each clause learned by FOIL. We have already recognized this as the *Small Disjuncts Problem* (page 21).

Efficiency: FOIL grows an increasing number of clauses with an increasing number of literals. Also, several of the literals chosen to fit the noise introduce new variables, which leads to an explosion of the size of the tuple set. In fact, the C implementation of FOIL could complete none of the ten experiments with 2000 training examples within 500 minutes of CPU time on a SUN SPARCstation IPX, while the PROLOG implementation of FOSSIL only needed about 15 minutes of CPU time for each of the training sets.

Understandability: It is a widely acknowledged principle that the more complex a concept definition is, the less understandable it will be, in particular when both definitions describe the same data set. While the descriptions induced by FOIL for large training sets often consisted of 20 or more clauses and were totally incomprehensible to the author, FOSSIL converged towards a simple, approximate theory. From 8 of 10 training sets with 2000 examples it learned the theory given in figure 3.6 (theory B from appendix A) which is 98.45% correct (see table A.2), while from the other two sets it learned theory F which is still 97.98% correct.

What seems to be responsible for the drastic increase in the complexity of the learned clauses is that FOIL's stopping criterion, the *Encoding Length Restriction* (p. 21), is dependent on the size of the training set. In the KRK domain it performs very well on sample sizes of 100 training examples. The more this number increases, the more bits are allowed for the theory to explain the data. However, more examples do not necessarily originate from a more complex theory. In fact, FOIL very often chooses the same literals as FOSSIL for the first clauses of its concept definition, but then continues to add literals and clauses, where FOSSIL stops.

FOSSIL uses a statistical stopping criterion based on the assumption that each literal in an explanation must have a significant correlation with the set of training examples. Statistical measures usually improve with the size of the training sets and so does the quality of the rules induced by FOSSIL. While both FOIL and FOSSIL successively improve their predictive accuracy with increasing training set sizes, only FOSSIL converges towards a useful theory.

3.3.4 Comparison with mFOIL

mFOIL [Džeroski and Bratko, 1992a] is an algorithm based on FOIL that has adapted several features from the CN2 learning algorithm, such as the use of the Laplace and m-estimate as a search heuristic (see figure 2.4) and the use of significance testing as a stopping criterion (see section 3.1). These methods have proved very effective for noise handling. In addition mFOIL uses beam search (default beam width 5) and can make use of mode and type information to constrain the search space (see section 2.3.3). The following experiments used mFOIL to its full capacity, but were performed with an early version of FOSSIL that did not support search space restrictions with modes and symmetry information.[10]

The values of the m parameter were increased until a maximum performance was reached in the sets of 100 training examples. We then used the same values for testing with 1000 training examples. The results can be found in table 3.4.

	mFOIL					FOSSIL
	$m = 0.01$	Laplace	$m = 8$	$m = 16$	$m = 32$	*Cutoff* = 0.3
100	89.77	89.84	93.03	93.06	91.46	93.52
1000	91.54	92.51	95.70	97.10	98.48	98.05

Table 3.4: Comparison with mFOIL

FOSSIL seems to be at least equal at an example size of 100, unless a considerably better theory has been missed somewhere around $m = 16$. However, mFOIL's strengths come to bear at an example size of 1000. The results reported here are probably not yet the peak of its performance, as with $m = 32$ mFOIL has learned some theories with a predictive accuracy of above 99% which FOSSIL has not achieved so far. Increasing the m further might well improve the bad theories learned, while keeping the good ones.

However, one of the points to make here is that a good value of the m parameter is not only dependent on the amount of noise (as can be seen from the results given in [Džeroski and Bratko, 1992a] and [Džeroski and Bratko, 1992b]), but also on the size of the example set. The higher the noise level is, the higher

[10]The current version achieves 98.44% on the same data sets with 1000 examples.

one should choose the value of m. Our experiments on the other hand suggest that the value of m should also increase with training set size. Assuming that a certain level of noise in a certain domain requires a certain value of m, the following illustrates why larger training set sizes may require larger values of m to produce the same heuristic value of the m-estimate (see figure 2.4).

Assume that we apply the learning algorithm to a certain data set. Then we repeat the experiment by presenting each example *twice*. Ideally the result of the heuristic evaluation for each literal should not change in that case. However, for establishing

$$\frac{p + m_1 \times \frac{P}{P+N}}{p + n + m_1} = \frac{2p + m_2 \times \frac{2P}{2P+2N}}{2p + 2n + m_2}$$

you also have to adjust the m parameter accordingly ($m_2 = 2m_1$). Similarly, FOIL's information gain heuristic would not produce the same value. FOSSIL's correlation heuristic on the other hand would calculate the same heuristic value in both cases as can be easily seen from formulas (3.1) to (3.3). It should therefore not be surprising that the same setting of the cutoff parameter does reasonably well at different levels of noise and at different training set sizes.

3.4 Summary

In this chapter we have been mainly concerned with pre-pruning methods for *separate-and-conquer* rule learning algorithms. Pre-pruning deals with noise in the data by relaxing the constraint that a complete and consistent program has to be learned. With pre-pruning heuristics (also called *stopping criteria*), the program can decide when a clause can be considered complete although it still covers negative examples and when to stop adding clauses to a theory although not all of the positive examples are explained yet.

In section 3.2 we have described a new system, FOSSIL. Its new search heuristic based on statistical correlation allows the use of a very simple and efficient stopping criterion, the *cutoff*. We see the main advantages of this approach in its

Efficiency: There is no separate calculation of a heuristic function for negated literals and the amount of computing involved in calculating the stopping criterion is reduced to a mere comparison.

Robustness: A good value of the cutoff parameter seems to be independent of the amount of noise and the number of training examples. It is nevertheless domain-dependent.

Simplicity: The close relation of the cutoff to the correlation search heuristic makes it easy to adjust the cutoff to an appropriate value. The cutoff can be seen as a direct means of controlling the *Overfitting Avoidance Bias*. We will further investigate this in chapter 5.

A comparison with the most common relational learning system, FOIL, has shown that FOSSIL's noise handling capabilities are superior, because FOIL's stopping criterion is dependent on the number of examples in the training set. However, mFOIL, another relational learner, seems to do a little better in terms of classification error, but a good value of its m-parameter depends on the size of the training set and on the amount of noise contained in it. FOSSIL's cutoff parameter on the other hand invariably gives good results at $\mathit{Cutoff} = 0.3$.

Chapter 4

Post-Pruning

Contrary to the pre-pruning approaches to achieving noise tolerance in ILP that 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 result 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 *et al.*, 1984], ID3 [Quinlan, 1987] and ASSISTANT [Niblett and Bratko, 1986]. An overview and comparison of various approaches can be found in [Mingers, 1989a] and [Esposito *et al.*, 1993a]. This chapter will first review how *Reduced Error Pruning* [Quinlan, 1987] can be adapted for a rule learning algorithm (section 4.1). We will then outline several problems with this simple method (section 4.2) and present an improved algorithm known from the literature (section 4.3) that solves some of them. However, we will see that even this improved version still has its shortcomings that have motivated the research presented in the following chapters (section 4.4).

4.1 Reduced Error Pruning

Post-pruning algorithms typically consist of two phases:

1. *Generate an overly specific concept description*
2. *Generalize it to an appropriate level*

While phase 1. basically is identical for all algorithms, there are two principal methods for phase 2. Several algorithms (like *Minimal Error Pruning* [Niblett and Bratko, 1986] or *Pessimistic Error Pruning* [Quinlan, 1987]) analyze the performance of decision trees on the same data set from which they have

```
procedure PostPruning(Examples, SplitRatio)

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

Figure 4.1: A Post-Pruning algorithm

been learned. For simplifying the resulting over-specific theory it is more common, however, to use a different set of examples that were not known in the initial learning phase (e.g. *Error Complexity Pruning* [Breiman *et al.*, 1984], *Iterative Pruning* [Gelfand *et al.*, 1991] and *Critical Value Pruning* [Mingers, 1989a]).

4.1.1 The Algorithm

The most common among these methods is *Reduced Error Pruning (REP)* [Quinlan, 1987]. This simple algorithm has been adapted for a propositional rule learning framework by [Pagallo and Haussler, 1990] and subsequently been introduced to noise handling in Inductive Logic Programming by [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 measured on the pruning set. A pseudo-code version of this algorithm can be found in figure 4.1.

The subroutine SimplifyTheory usually tries several *simplification operators* at all applicable points of the overfitting theory and selects the one that produces the theory with the highest accuracy on the pruning set. This is repeated with the new theory until the accuracy of the best pruned theory is below that of its predecessor.

Different implementations have used different simplification operators. The most commonly used operators are:

`delete-any-literal`: This operator tries to delete each single literal from the current theory. It is very expensive to use, as at each step the algorithm has to evaluate one new theory for each literal of the old theory.

`delete-last-literal`: This operator only tries to delete the last literal of each clause. [Brunk and Pazzani, 1991] argue that this more efficient operator is suitable for separate-and-conquer rule learning algorithms, because the order in which the literals of a clause are considered for pruning is inverse to the order in which they have been learned.

`delete-last-sequence`: This operator, used e.g. in [Cohen, 1993], selects the best of all theories that result from deleting a sequence of literals from the end of the clause. Each iteration is equally expensive as with the `delete-any-literal` operator, but one may arrive faster at the final theory, because this operator can prune several literals in the same iteration.

`delete-clause`: This simple operator examines all theories that result from deleting a clause from the old theory. It is very efficient and needs only few iterations. However, it should only be used in connection with one of the three other operators that are able to do the fine-tuning of the individual clauses.

Reduced Error Pruning in its original form [Brunk and Pazzani, 1991] uses `delete-last-literal` and `delete-clause` for pruning the overly specific theories generated in phase 1.

4.1.2 Experiments

[Brunk and Pazzani, 1991] show in a series of experiments in the KRK domain that REP improves upon pre-pruning with FOIL's Encoding Length Stopping Criterion. We have performed similar experiments in the KRK domain which is described in detail in appendix A. The setup of the experiments was basically the same as described in section 3.3.1. All training sets contained 10% misclassified examples in order to simulate noise and the learned theories were tested on 5000 noise-free examples.

Each row of table 4.1 shows the average performance and standard deviation of each algorithm in a series of 10 experiments with different training sets of the same size. Only 6 experiments were performed for the sets of size 1000. We have compared the currently newest version of FOIL with a PROLOG implementation of Reduced Error Pruning that used our own implementation of FOIL as the basic induction module. Of course, no stopping criterion was implemeted for the latter version in order to be able to overfit the data as described in [Brunk and Pazzani, 1991]. REP made use of all search optimization of section 2.3.3. The results are comparable to those reported in [Brunk and Pazzani, 1991]: REP performs better than FOIL. We have also included the accuracies of the intermediate unpruned theories, because they show the significant increase that post-pruning brings in terms of accuracy.[1]

Training	REP		FOIL6.1
Set Size	Unpruned	Pruned	
100	85.29 ± 4.76	91.77 ± 9.05	89.48 ± 6.02
250	83.79 ± 2.33	96.29 ± 1.94	90.78 ± 1.74
500	84.29 ± 2.72	97.62 ± 0.95	92.88 ± 1.42
750	85.17 ± 2.87	97.47 ± 1.44	93.55 ± 0.77
1000	85.65 ± 2.04	98.01 ± 1.38	94.70 ± 1.65

Table 4.1: Reduced Error Pruning in the noisy KRK domain.

An interesting detail is that in the *Unpruned* column, the results with only a few (100) examples are superior to those using bigger training sets (and thus more information) for learning. This phenomenon also appeared in figure 3.5. On the other hand there is big increase in accuracy on the pruned theories of sizes > 100. Apparently the theories generated from 100 examples did not capture as much information as the theories learned from bigger training set sizes, but generalized better nevertheless, because the little information that was captured is not as much obfuscated by unnecessary rules as when learning from more examples.

4.2 Problems with Reduced Error Pruning

REP has been shown to be quite effective in raising predictive accuracy in noisy domains. Nevertheless, this method has several shortcomings, which we will discuss in this section.

[1]Strictly speaking we should have included a column of accuracies for an algorithm that uses no pruning at all and learns from the *entire* set of training data. The numbers we have given are the results from the initial rule growing phase of REP and thus only used the growing set (2/3 of the training set) as input data. Using the entire set for input might give a better accuracy, but it will still be below that of REP. This can be seen when we consider that e.g. the *Unpruned* column for size 750 actually contains results for learning an unpruned theory from about 500 examples. Comparing these values to the values of the other algorithms shows that pruning is necessary in this domain.

4.2.1 Efficiency

In [Cohen, 1993] it was shown that the worst-case time complexity of REP is as bad as $\Omega(n^4)$ on random data (n is the number of examples). The growing of the initial concept, on the other hand, is only $\Omega(n^2 \log n)$. The derivation of these numbers as given in [Cohen, 1993] rests on the following assumptions:

1. Random data are incompressible. Therefore each example has to be explained by a separate rule. The resulting concept description before pruning will therefore contain $O(n)$ rules altogether.
2. Each rule has $O(\log n)$ conditions, because each literal will cover about half of the random instances.
3. The costs of adding 1 literal to a rule are $O(n)$, because each of the constant number of candidate relations in the background knowledge[2] has to be tested once against each of the n instances in the growing set.[3]
4. The size of the final (pruned) theory will not depend on the number of training examples, i.e. will be constant. As the best explanation for random training examples should be the empty theory, this is a reasonable assumption, provided the algorithm works.
5. Each rule in the intermediate concept description will be modified or deleted at least once until the final theory of constant size has been found.

From the first two assumption we see that there will be about $O(n \log n)$ literals in the concept description, each of them costs $O(n)$ (assumption 3). Therefore we have a total cost of the order of $O(n^2 \log n)$ for the growing phase.

In each step of the pruning phase each of the $O(n)$ clauses can be simplified by deleting the last literal or deleting the whole clause[4]. The $O(n)$ rules of each of these $O(n)$ simplifications have to be tested against the $O(n)$ examples in the pruning set[5]. According to 5. this simplification loop has to be performed at least n times. Therefore we get a total cost of $\Omega(n^4)$. A more detailed proof can be found in [Cohen, 1993].

[2]While the number of relations is constant, the number of variabilizations for each literal might be increasing with the introduction of new variables. In this case we have to assume that the number of variables is bounded by a constant.

[3]By remembering which examples are covered so far, one can avoid to test the clause against all n instances after adding a new literal. However, the clause has to be tested against all of the negative instances, and so we still have costs of $O(n)$.

[4]Using more expensive pruning operators like `delete-any-literal` would further increase the cost.

[5]As the complexity of the concepts steadily decreases, the number of rules and the number of simplifications decrease as well. However they are still $O(n)$, when assuming that in the first half of the iterations the size of the concepts is greater than a constant fraction of n (e.g. $n/2$).

```
illegal(A,B,C,D,E,F):-adj(A,E),adj(B,F),\+lt(E,A).
illegal(A,B,C,D,E,F):-C==E,lt(D,F).
illegal(A,B,C,D,E,F):-D==F,\+lt(B,D),\+A==C.
illegal(A,B,C,D,E,F):-D==F,lt(A,E),\+A==C.
illegal(A,B,C,D,E,F):-A==C,adj(B,F),\+B==F.
illegal(A,B,C,D,E,F):-\+lt(D,F),adj(A,E),adj(B,F).
illegal(A,B,C,D,E,F):-D==F,\+lt(B,D),\+lt(A,E).
illegal(A,B,C,D,E,F):-C==E,\+lt(D,B).
illegal(A,B,C,D,E,F):-\+adj(B,F),\+adj(A,C),lt(E,A),\+adj(C,E),
                      lt(B,F),\+lt(D,B),B==D.
illegal(A,B,C,D,E,F):-D==F,\+adj(A,C),\+adj(B,D),\+lt(A,C).
illegal(A,B,C,D,E,F):-\+adj(B,F),\+lt(D,F),lt(D,B),lt(A,E),
                      \+adj(B,D),\+A==C.
illegal(A,B,C,D,E,F):-\+adj(C,E),\+adj(B,F),\+lt(D,F),\+lt(A,C),
                      \+adj(A,C),B==D.
illegal(A,B,C,D,E,F):-\+adj(C,E),D==F,lt(A,C).
illegal(A,B,C,D,E,F):-\+adj(C,E),adj(A,E),lt(E,A),\+lt(B,D).
illegal(A,B,C,D,E,F):-\+adj(B,F),lt(C,E),D==F.
illegal(A,B,C,D,E,F):-\+adj(B,F),lt(C,E),\+adj(C,E),\+lt(A,E),
                      \+adj(B,D),\+adj(A,E).
illegal(A,B,C,D,E,F):-\+lt(D,B),\+adj(B,F),\+adj(C,E),lt(C,E),
                      adj(B,D),lt(B,F),lt(A,C).
illegal(A,B,C,D,E,F):-A==E,lt(B,D),\+adj(A,C).
illegal(A,B,C,D,E,F):-lt(F,B),lt(B,D),lt(E,A),\+adj(A,E).
```

Figure 4.2: Theory learned from noisy KRK examples without pruning.

It has also been pointed out there that this result for random data generalizes to data containing noise, i.e. a constant fraction of random and thus incompressible data. Figure 4.2 shows a theory that has been learned by the first phase of REP in one of the experiments of section 4.1 in the KRK domain with 250 examples. To get a useful rule most of the rules and conditions of this theory have to be pruned. In fact the best that REP can do in this case is to use the first two literals of the first rule and the first literal of rules 2 and 3 and to remove all other rules. This way it constructed the theory of figure 3.6. The same theory has also often been found by Fossil in the experiments of section 3.3 without the overhead of generating and pruning the complicated theory of figure 4.2.

From the above analysis it can be concluded that in the long run the costs of pruning will by far outweigh the costs of generating the initial concept description, which already are higher than the costs of using a pre-pruning algorithm that entirely avoids overfitting.

4.2.2 Split of the Training Data

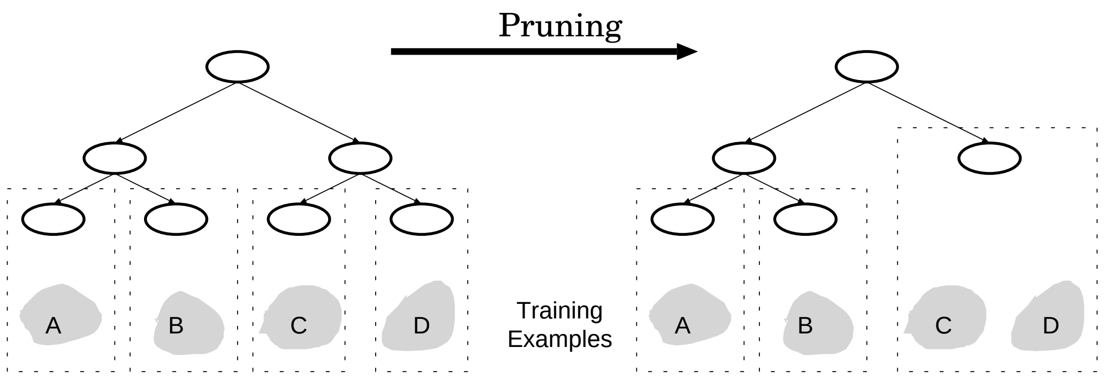
Another disadvantage of REP is that the training data have to be split into two sets, a *growing set* (usually 2/3) and a *pruning set* (usually 1/3). A two-fold problem results from this: As learning occurs from examples in the growing set only, the algorithm might miss to learn important rules if some or most of the examples for this rule are in the pruning set and not in the growing set. On the other hand, each learned rule has to have support in the pruning set, because otherwise some relevant rules — although learned correctly — might be pruned or deleted altogether. Thus a bad split of the given examples can have a negative influence on the behavior of both the learning and the pruning algorithm. As mentioned in section 4.1 there are several alternative post-pruning algorithms that do not use a separate pruning set and thus are not subject to this problem. However, there is some evidence that algorithms using a separate pruning set perform better than those that have to estimate the amount of over-fitting exclusively from the training data [Mingers, 1989a].

4.2.3 Separate-and-Conquer Strategy

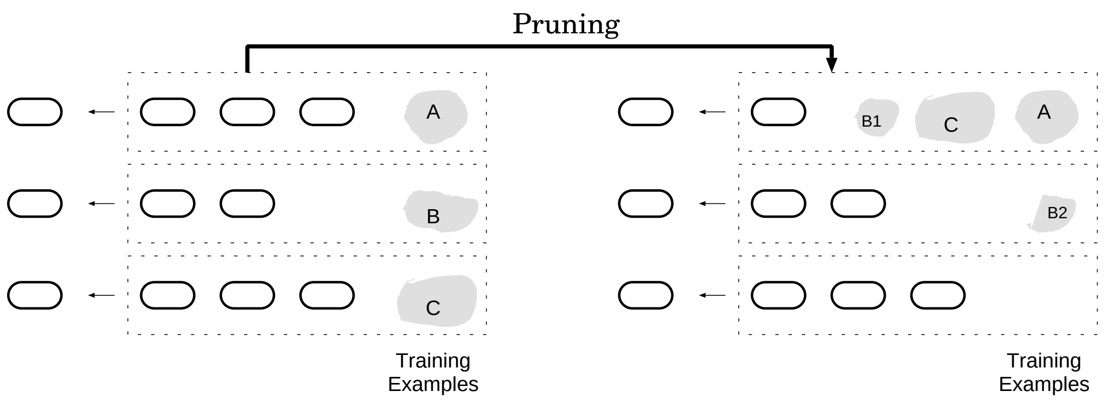
In decision tree learning usually a *divide-and-conquer* strategy is used. This means that the training set is split into disjoint sets according to the outcome of the test chosen for the top level decision. After this, the algorithm is recursively applied to each of these sets independently. Greedy covering algorithms like FOIL follow a *separate-and-conquer* strategy (see section 2.2). This method first learns a rule from the whole training set and subsequently removes all examples that are covered by this rule. Then the algorithm recursively tries to find rules that explain the remaining examples.

Although the separate-and-conquer approach shares many similarities with the divide-and-conquer strategy, there is one important difference: Pruning of branches in a decision tree will never affect the neighboring branches, whereas pruning of literals of a rule will affect all subsequent rules. Figure 4.3 (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.

Pruning a literal from a clause on the other hand 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. However, the initial growing phase of REP does not know



(a)



(b)

Figure 4.3: Post-Pruning in (a) Divide-and-Conquer and (b) Separate-and-Conquer learning algorithms.

which of the instances are noisy and will henceforth carry along instances that should already be covered by one of the previous clauses.

In the example of figure 4.3 (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 the pruning algorithm, in general it need not be the case that the second rule or one of its pruned versions will be 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 an explanation for B2 needs a totally different set of literals than an explanation for its superset B.

Another way of looking at this problem may be to view a PROLOG program as a *decision list*.[6] Each body of a clause of the program corresponds to a node in the decision list. If the body is true, the head is proven and we arrive at a leaf node. Otherwise we try the next node in the list, i.e. the next clause in the program. Classical decision tree pruning would only allow to prune the nodes bottom up, i.e. only apply a `delete-last-clause` operator. Reduced Error Pruning, however, not only allows to prune any (instead of only the last) node, but also to prune the conditions of the rules associated with each node with the `delete-literal` operators. Changing the test associated with a node in a decision tree will in general change the split it induces on the examples and thus could lead to the generation of different subtrees for its children. However, as the `delete-literal` operators change the test at pruning time (*after* learning), REP has to keep the subtree that has been previously learned from a different set of examples.

Thus it is clear that the initial overfitting phase of post-pruning algorithms may in the best case only lead to the generation of some additional clauses that will be pruned in the pruning phase (like the third rule in the example). In the worst case, however, the instances that will be covered by a pruned rule, but are not covered by its unpruned original (the sets C and B1 in our example) may lead the learner down a garden path. They may change the evaluation of the candidate literals in subsequent learning and thus the "correct" literals might not be selected. A wrong choice of a literal cannot be undone by pruning.

4.2.4 Bottom-Up Hill-Climbing

REP employs a greedy hill-climbing strategy: Literals and clauses will be deleted from the concept definition so that predictive accuracy on the pruning set is 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 it can be expected that the theory that has been generated in the growing phase is much too specific, i.e. REP will have to do a lot of pruning and therefore has ample opportunity to go wrong (compare e.g. the theory of figure 4.2 with the theory of figure 3.6 which will (ideally) be left after pruning). REP's specific-to-general search can be expected to be slow and imprecise for noisy data, because it has to prune a significant portion of the theory previously generated in the growing phase and is likely to prematurely stop at a lower local maximum during this process.

For several problems, including the KRK examples of section 4.2.1 it would be much more suitable to search for the final, pruned theory in a top-down, general-

[6]A decision list is a binary decision tree, where each node's children contain at least one leaf.

to-specific fashion in order to avoid over-specialization. Of course, a top-down method would risk to over-generalize the same way that a bottom-up algorithm risks to over-specialize, but the argument is that in many cases the final theory is closer to the empty theory than to the theory generated by the overfitting phase of REP (see also figure 5.3). Section 4.3 will further discuss this issue and present an alternative post-pruning algorithm that searches in a top-down fashion.

4.3 The Grow Algorithm

In [Cohen, 1993] some of the problems of section 4.2 — in particular efficiency — have been recognized. Cohen has then proposed Grow, a post-pruning algorithm based on a technique used in [Pagallo and Haussler, 1990]. 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. For a detailed description of the Grow algorithm see [Cohen, 1993].

This algorithm solves some of problems of section 4.2:

- Under the assumptions of section 4.2.1 the costs of pruning on random data are reduced to $O(n^2 \log n)$: Each clause contains about $\log n$ literals (assumption 2.), each of which can be the last literal in a generalized clause. So in the worst case we get a total of $O(n \log n)$ clauses in the augmented intermediate theory, because there are $O(n)$ clauses (assumption 1.). Each of them is tentatively added to the initially empty final theory and the resulting set of clauses is tested on the $O(n)$ training examples. This is repeated until all clauses in the final concept description (which is assumed to be of constant size by 4.) have been found.[7] Therefore the costs of this algorithm are $O(n^2 \log n)$. Again, consult [Cohen, 1993] for a detailed proof. However, it should be intuitively clear that it will be less work to grow the theory of figure 3.6 from the empty theory (as Grow does) than to get there by successively pruning the theory of figure 4.2 (as REP does).

[7] However, in [Cameron-Jones, 1994] it has recently been shown that this assumption is not quite correct and that the asymptotic time complexity of the Grow post-pruning method is also above the complexity of the initial rule growing phase. We will discuss this in a little more detail in section 6.3. In all practical experiments, however, the run-time of Grow has been small compared to the run-time of the over-fitting phase.

- Grow replaces the bottom-up hill-climbing search of REP (see section 4.2.4) by a top-down approach. It does not remove the most useless clause or literal from the specific theory, but instead adds the most promising generalization of a rule to an initially empty theory. This results in a significant gain in efficiency, along with a slight gain in accuracy as the experiments in [Cohen, 1993] show. As we have already discussed, an explanation for this could be that top-down hill-climbing starts from the empty theory, which in many domains is much closer to the correct theory than the most specific one. Thus it is more likely that the bottom-up approach will over-specialize than that the top-down approach will over-generalize.

Table 4.2 shows a comparison of REP and Grow on the same data sets that have been used for the experiments reported in table 4.1. Along with the average accuracies and the standard deviations that the algorithms were able to achieve on these sets, we also report the run-times for the initial rule-growing phase, which is identical for both, REP and Grow, and the time needed by each of the algorithms for post-pruning. So the total run-times for REP and Grow can be obtained by adding the column *Overfitting* to the columns *REP* and Grow. Run-times are measured in CPU seconds on a SPARCstation IPX. All programs were implemented by the author in SICStus PROLOG (see appendix B).

Training	Accuracy		Run-time		
Set Size	REP	Grow	*Overfitting*	REP	Grow
100	91.77 ± 9.05	91.60 ± 10.71	8.63	2.44	1.66
250	96.29 ± 1.94	95.91 ± 4.44	91.31	104.98	19.81
500	97.62 ± 0.95	98.17 ± 0.72	456.56	1578.16	100.81
750	97.47 ± 1.44	98.31 ± 0.79	1142.78	7308.84	361.41
1000	98.01 ± 1.38	98.30 ± 0.77	2129.89	23125.34	806.89

Table 4.2: REP and Grow in the noisy KRK domain.

As in [Cohen, 1993] the basic findings are that Grow's top-down search strategy prunes significantly faster than REP and is also able to gain a little accuracy, although these differences are not statistically significant. However, Grow does not attempt to solve the problems with bad splits (section 4.2.2) and with the separate-and-conquer strategy (section 4.2.3). [Srinivasan *et al.*, 1992] use an algorithm similar to Grow to select a subset of clauses with an evaluation criterion based on the *Minimum Description Length Principle* [Rissanen, 1978] which does not require a separate test set for evaluating theories. But no matter what criterion it uses, the overall costs of the Grow algorithm still include the cost of overfitting the data and thus are much higher than the costs of pre-pruning approaches that directly construct the final theory.

4.4 Summary

In this chapter we have reviewed several post-pruning approaches to noise-tolerant rule learning algorithm. *Reduced Error Pruning* (REP) [Brunk and Pazzani, 1991] is the best-known among these approaches and has been shown to achieve good results in terms of accuracy. However, this simple algorithm has several shortcomings as we have pointed out in section 4.2, most notably efficiency and incompatibility with the separate-and-conquer learning strategy. We have further presented a modified pruning algorithm by [Cohen, 1993] that prunes more efficiently and is also a little more accurate than REP. However the GROW algorithm still suffers from the fact that it has to generate an overly specific theory first.

[Cohen, 1993] has tried to improve his GROW algorithm by adding two stopping heuristics to the initial stage of overfitting, and thus achieved a further speed-up of the algorithm. We will introduce an alternative way of *combining* pre-pruning and post-pruning methods in chapter 5 in order to avoid excessive overfitting. Chapter 6 will then proceed to present an alternative approach that *integrates* pre-pruning and post-pruning in a way that prevents the expensive initial phase of overfitting altogether.

Chapter 5

Combining Pre- and Post-Pruning

In chapter 4 we have discussed several shortcomings of typical post-pruning algorithms, in particular of *Reduced Error Pruning (REP)*. Most importantly we have seen from figure 4.2 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 almost all of the work done by 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 to speed up the learning process. The goal of pre-pruning in this context is not to entirely prevent overfitting (as it has been the case in all approaches discussed in section 3.1), but to reduce the amount of overfitting so that a subsequent post-pruning phase has to do less work and is thus less likely to go wrong. However, there is always the danger that a predefined stopping criterion will over-generalize the theory.

In this chapter we will discuss an alternative approach based on FOSSIL that combines pre- and post-pruning. Its advantage is that it can automatically discover the right amount of pre-pruning by systematically varying FOSSIL's cutoff parameter (section 5.1). We then suggest *Top-Down Pruning*, an algorithm that exploits this approach by searching for a theory that is a little too specific, but can be efficiently generalized with post-pruning methods (section 5.2). Again we report some experiments in the KRK domain (section 5.3) and subsequently summarize our findings (section 5.4).

Parts of this chapter have been previously published in [Fürnkranz, 1994d].

5.1 Generating a Series of Concept Descriptions

One advantage of the simple and efficient cutoff stopping criterion described in section 3.2.3 is its close relation to the search heuristic. While FOIL (*encoding length restriction*) and *m*FOIL (*significance test*) have to do separate calculations to determine when to stop learning, FOSSIL needs to do a mere comparison between the heuristic value of the best candidate literal and the cutoff value. This property makes it easy to experiment with FOSSIL, because we can directly examine the heuristic values of literals that are known to be relevant in order to determine appropriate values for the cutoff parameter. This would be much harder in other relational learning systems, where the search heuristics and the stopping criteria are usually independent of each other.

5.1.1 The Algorithm

However, when there is no information about the relevance of literals at hand (which usually will be the case) FOSSIL can be extended nicely to generate *all* theories that could be learned by FOSSIL with any setting of the cutoff parameter (see figure 5.1).

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

Figure 5.1: Algorithm to generate all theories learnable by FOSSIL

The basic idea behind this algorithm is the following: Assume that you are 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.

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 FOSSILwith 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. However, as our experience with shows, this is very often not the only change. Usually several more literals that have a correlation value higher than the new cutoff will be added, because adding a literal to the current concept definition will change the example distribution and thus the heuristic values for subsequent literals.

At this new setting of the cutoff parameter a new theory will be learned and 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 cut off (i.e. MaxPrunedCorr = 0.0) and thus the most specific theory has been reached.

5.1.2 An Example

In figure 5.2 we see an example of how Fossil generates a series of theories from 1000 noise free examples in the KRK domain (see Appendix A). It is interesting to see how the quality of the learned concept steadily improves until it arrives at a 99.32% correct theory. At this point, clauses (1), (5) and (6) try to categorize positions with white rook and black king on the same file. Clause (1) is correct, while clauses (5) and (6) only fit the examples in the training set.

Clause (2) on the other hand, says that all positions with white rook and black king on the same rank are illegal, which is too general, because it misses the positions where the white king blocks the check of the white rook (see figure A.1 c) for an example of this type of position). At the next refinement, Fossil finds a new rule (2) which is symmetric to rule (1) and incorporates the literal `not A=C` that was previously not considered because of its relatively low correlation of 0.3871. Because of the addition of this literal Fossil "forgets" the already learned clauses (4) to (6) of the last theory, as the starting literals of those rules do not have a high enough correlation under the new circumstances. This goes hand in hand with a decrease in predictive accuracy.

Lowering the cutoff once again, however, rediscovers the rules that have been "forgotten" in the previous theory. The final theory completely explains all of the training examples. No literal has been cut off in this theory, and no further refinement is possible; thus MaxPrunedCorr = 0.0. As these theories have been learned from noise-free data, the most specific theory is also the most accurate one. In noisy domains it is more likely the the final theory overfits the noise and that one of its predecessors is the most accurate.

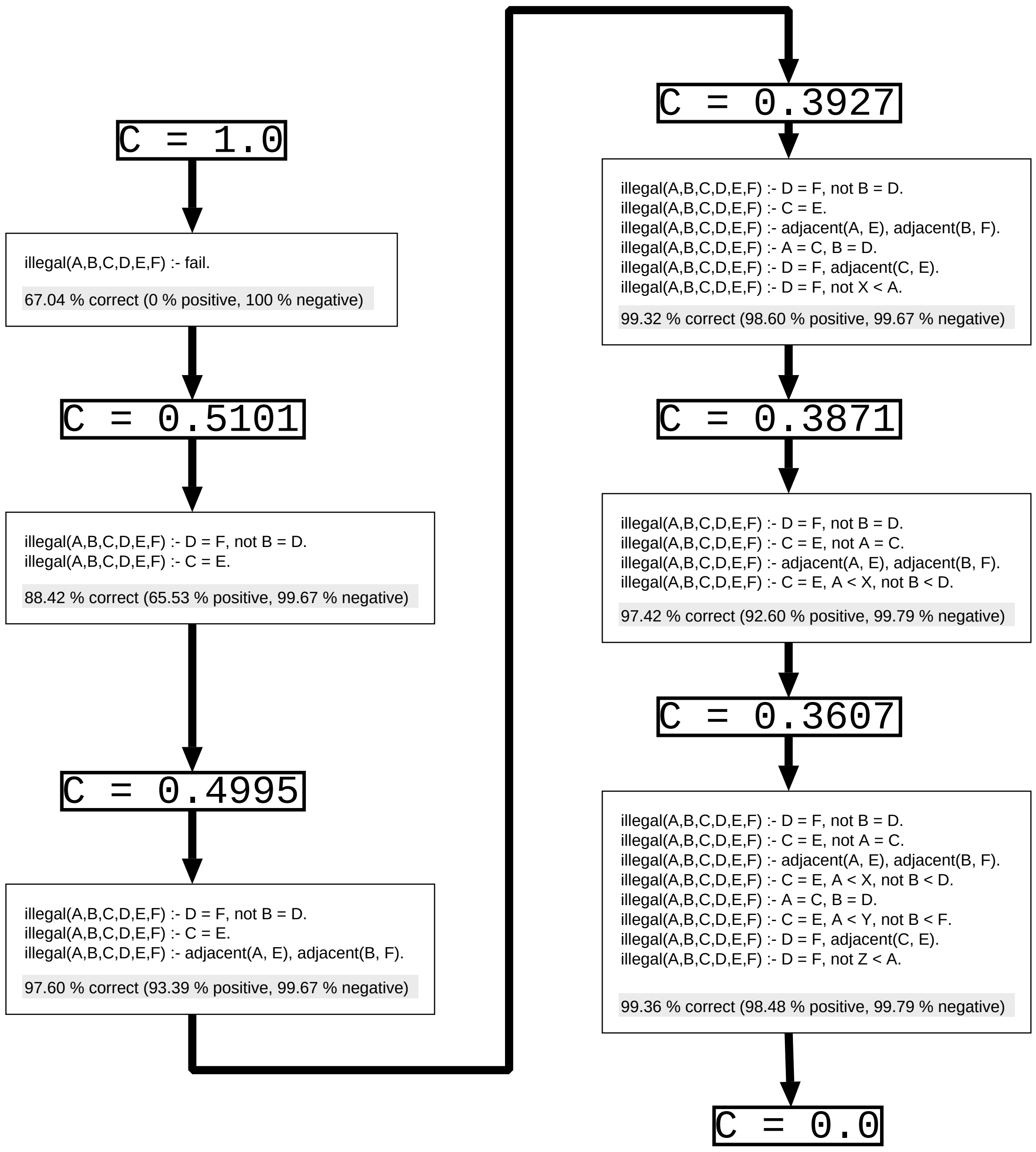

C = 1.0
illegal(A,B,C,D,E,F) :- fail.
67.04 % correct (0 % positive, 100 % negative)
C = 0.5101
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E.
88.42 % correct (65.53 % positive, 99.67 % negative)
C = 0.4995
illegal(A,B,C,D,E,F) :- D = F, not B = D.
illegal(A,B,C,D,E,F) :- C = E.
illegal(A,B,C,D,E,F) :- adjacent(A, E), adjacent(B, F).
97.60 % correct (93.39 % positive, 99.67 % negative)
C = 0.3927
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)
C = 0.3871
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)
C = 0.3607
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)
C = 0.0

Figure 5.2: Generating a series of theories in the KRK domain

A very important property of the described algorithm is that it will generate a series of theories in a (roughly) general to specific order:[1] High values of the cutoff will cause over-generalization, while lower values are likely to fit the characteristics of the training set which may be the cause for overfitting in noisy domains (see also sections 3.2.3 and 3.3.2). However, we have already seen in our example that this order is not strict: Lowering the cutoff does not necessarily result in a more specific theory, because adding a literal to the current concept definition will change the example distribution for subsequent literals and thus change the correlation values of the literals at all subsequent choice points.

5.1.3 Experiments

In a preliminary series of experiments we have used the simple algorithm of figure 5.1 in the following way: The same training sets with 10% noisy used in the experiments described in section 3.3.3 were randomly split into two sets of equal size. The first set was used for learning all theories down to a cutoff of 0.15 with the algorithm described in the last section.[2] From these theories the one with the best predictive accuracy measured on the examples of the second set was selected as the final theory. The results are shown in table 5.1.

Different Training Set Sizes (10% Noise)		Training Set Size					
		100	250	500	750	1000	2000
FOSSIL (0.3)	Accuracy	**93.52**	**92.68**	**92.79**	**96.33**	**98.05**	**98.41**
	Clauses	3.8	3.7	3.1	3.0	3.0	3.0
	Lits/Clause	2.24	3.01	2.63	1.94	1.5	1.4
Minimal Error Selection	Accuracy	**88.96**	**92.48**	**95.73**	**95.16**	**95.14**	**98.25**
	Clauses	4.3	3.5	4.2	4.8	3.9	3.2
	Lits/Clause	1.81	2.37	2.31	2.63	2.48	1.51

Table 5.1: Selecting the theory with minimal error

This simple method of generating all theories FOSSIL can learn and selecting the one that has the lowest error on a separate test set performs a little worse than FOSSIL with a cutoff of 0.3. It is better at example size 500 and worse

[1]Note that we use the terms "general" and "specific" in an intuitive way. We consider the empty theory to be most general, because "Everything is false." is a very general statement. However, our "most specific" theory will cover more ground instances than the empty theory, and thus may be considered (extensionally) more general. See [Flach, 1992] for a discussion of related matters.

[2]The restriction for admitting only theories with a cutoff above 0.15 was only made for reasons of efficiency. From the results of table 3.1 we already know that theories below 0.15 are likely to overfit the noise and will thus have a low classification accuracy.

at sizes 100 and 1000. At other training set sizes it performs about the same, although it practically only learns from half of the training examples.

On the other hand, the Minimal Error Selection method invariably selects a theory at least as good as the theory that FOSSIL with a fixed cutoff would have learned from a training set of the same size. To see this we have to compare the values of the Minimal Error Selection method with the corresponding values of FOSSIL at half the training set size.

Our conclusion from these experiments is that setting the cutoff to 0.3 gives a very good average performance for FOSSIL, but there is some advantage that can be gained by adjusting the cutoff parameter to the particular characteristics of the training set at hand.

5.1.4 Important Properties

While the naive approach of section 5.1 might be too crude to be applied in practice, the experiments (table 5.1) have convinced us that there is some potential for refinement. There are several characteristics of this algorithm which can be used for further improvements:

Series of Rules: A major advantage of our algorithm is that we get a series of concept descriptions pruned to different degrees from which a theory with an appropriate level of generality can be selected. Research in decision tree learning has developed several methods for automating this selection [Breiman *et al.*, 1984, Mingers, 1989a, Weiss and Indurkhya, 1994]. Most of these methods can be easily adapted for rule learning algorithms, once we have series of theories with different degrees of generality.

Completeness: With the above algorithm we can examine all theories that FOSSIL can generate with any setting of the cutoff parameter.

Top-Down Search: Our algorithm generates the theories in a general to specific order (*top-down*). This is contrary to the *bottom-up* approach of most post-pruning methods (including those for decision tree pruning) that generate a most specific theory first and then successively generalize it. We have already discussed similar issues in section 4.2.4.

Efficiency: There are two reasons why we believe that the above approach can be very efficient:

- With increasing example set sizes and increasing noise levels, generating a most specific starting theory for pruning becomes more and more expensive (see the run-times for initial growing phase of REP and GROW in table 4.2). Generating a simple general theory is much less expensive. In the experiments described in section 5.1.3, typically the best theory has been found within the first five theories that have been generated. A lot of work can be saved by not generating the more specific theories.

- Efficiency can be further increased, because a clever implementation doesn't have to learn an entirely new theory. It can use the part of the last theory up to the point where the cutoff of the literal with the maximum correlation has occurred and continue learning from there.

In the next section we will show how to exploit these characteristics for a simple algorithm that combines pre- and post-pruning by selecting a better starting theory for post-pruning from the space of pre-pruned theories generated by different settings of FOSSIL's cutoff parameter.

5.2 Top-Down Pruning

As we have seen, the algorithm of figure 5.1 generates a series of different concept descriptions in a — roughly — general to specific order (*top-down*). Figure 5.3 shows the accuracies and complexities of a complete series of theories learned by this algorithm from 500 training examples of the noisy (10 %) KRK domain. It can be clearly seen that

- the most accurate theories are learned after a few iterations
- the lower the cutoff will be the more complex will the theories be and the smaller will be the distance between the cutoff values of two neighbouring theories

If we could find a way to realize when the learned theories get worse, we could stop earlier and avoid learning many of the specific theories. This may save a lot of work, as figure 5.3 indicates. We have also noted in the last section that usually several clauses — up to the point where the highest cutoff has occurred — can be reused from the previous run, so that the total cost of generating a series of concept description may not be much more than the cost of generating the most specific theory only.

Based on the above ideas, we have implemented the algorithm shown in figure 5.4. It tries to find the most specific among all reasonably good theories, and subsequently generalizes it with *Reduced Error Pruning*. Because of the initial general-to-specific search for a good theory, we have named the method *Top-Down Pruning* (TDP). Basically the algorithm does the following:

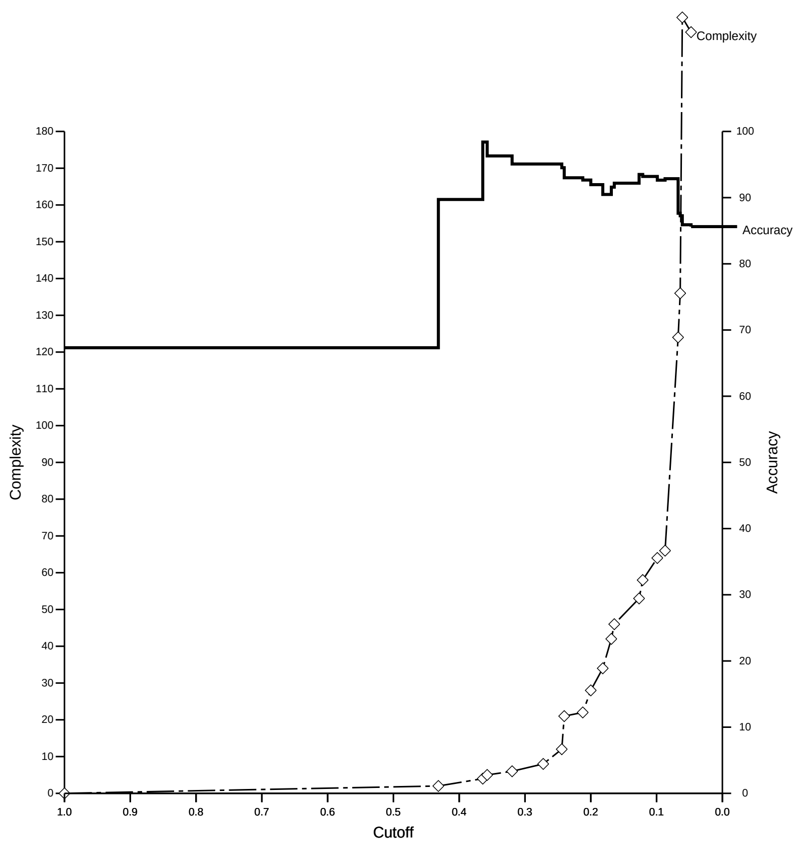


Figure 5.3: Accuracy and Complexity vs. Cutoff

1. *Split the training set into a* growing set *and a* pruning set *(usually* 2/3 *and* 1/3*).*

2. *Generate a series of concept descriptions from the examples in the growing set.*

3. *Evaluate each theory on the pruning set.*

4. *When the measured accuracy of one of the theories falls below the measured accuracy of the best theory so far minus the* Standard Error *for classification*[3]*, stop generating theories and return the last theory within the 1 SE margin.*

[3]This is based on an idea in CART [Breiman *et al.*, 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 mis-classification (estimated on the pruning set) and N is the number of examples in the pruning set.

```
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 = SimplifyTheory(Theory, PruningSet)
      if Accuracy(NewTheory, PruningSet) < Accuracy(Theory, PruningSet)
         exit loop
      Theory = NewTheory
return(Theory)
```

Figure 5.4: Combining Pre- and Post-Pruning with *Top-Down Pruning*.

5. *Prune the theory obtained in step 4. using* Reduced Error Pruning *as described in chapter 4.*

If this algorithm succeeds in finding a starting theory that it close to the final theory, we can expect our algorithm to be faster than *Reduced Error Pruning*. The reason is that it will

- *speed up the growing phase*, because the most expensive theories will not be generated,[4]
- *speed up the pruning phase*, because pruning starts from a simpler theory and thus the number of possible pruning operations is much smaller.

In the example of figure 5.5 that was taken from a run of TDP in the KRK endgame domain with 10% noise and 1000 training examples, the best theory is the third theory examined (including the empty theory). This theory is a reasonable approximation of the target concept (98.45% accurate on noise-free

[4]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 slow, because it has to generate all theories down to a cutoff of 0.0.

test data), although the first two rules are over-generalizations and one important rule is missing (see appendix A).[5] However, the theory that FOSSIL generates next already starts to fit the noise in the data. Therefore the 3-rule theory found is the best. This theory does not require any further changes in the subsequent post-pruning process. If we compare figure 4.2 with figure 5.5 we see that TDP generates fewer clauses and literals than REP. This difference would be much bigger when REP's starting theory would have been learned from 1000 training examples as well (and not only from 250 as in figure 4.2).

```
CUTOFF has been set to 1.0000

New Best Theory: 63.66%
CUTOFF has been set to 0.4256

illegal(A,B,C,D,E,F):-C==E
illegal(A,B,C,D,E,F):-D==F
New Best Theory: 81.98%
CUTOFF has been set to 0.3737

illegal(A,B,C,D,E,F):-C==E
illegal(A,B,C,D,E,F):-D==F
illegal(A,B,C,D,E,F):-adj(B,F),adj(A,E)
New Best Theory: 89.19%
CUTOFF has been set to 0.2460

illegal(A,B,C,D,E,F):-C==E,\+A==C,\+lt(B,F)
illegal(A,B,C,D,E,F):-D==F
illegal(A,B,C,D,E,F):-C==E,\+adj(B,F),\+adj(A,C)
illegal(A,B,C,D,E,F):-adj(B,F),adj(A,E),\+lt(A,C),\+A==E
illegal(A,B,C,D,E,F):-adj(B,F),adj(A,E)
Theory (86.79%) below Standard Error (1.70%) from Best (89.19%)
```

Figure 5.5: Trace of the Top-Down Pruning Algorithm

Unfortunately TDP doesn't always work as well as shown in figure 5.5. In preliminary experiments it turned out that sometimes one over-general clause will specialize to several more specific clauses which together cover most of the examples the general clause had covered, but exclude some of the negative examples (compare e.g. theories 4 and 6 of figure 5.2). However, if there are only a few of the specific clauses found in the next theory, the overall accuracy may drop

[5]In figure 5.5 the theory is estimated as being 89.19% correct. The reason for this is that the examples in the pruning set are noisy, because TDP of course separates them form its original (noisy) training set.

drastically, because too few examples are covered (as at theory 5 of figure 5.2). To avoid this problem we have added the requirement 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. 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. This modification is similar to the common criterion that only clauses covering a certain percentage of the training data are permitted (see section 3.2.3), but it deals with the whole theory and not with single clauses. However, the criterion is somewhat ad hoc, and we are thinking about improving the system in that respect.

5.3 Experiments

We compared *Top-Down Pruning* (TDP) to *Reduced Error Pruning* (REP) in terms of accuracy and run-time on the chess king-rook-king endgame domain (appendix A). 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 section 4.1 for their post-pruning phase. All programs were implemented in SICStus PROLOG 2.1 and the run-times were measured on a SUN SPARCstation IPX.

The experiments in the KRK domain followed the setup described in section 3.3. Experiments were performed with 10% of the examples having their classification reversed. Testing was done on sets of 5000 noise-free examples. REP and TDP were both given the same 10 training sets for each of the 4 different training set sizes. Mode, type and symmetry information (see section 2.3.3) was used to reduce the search space.

In order to exclude possible influences from the underlying learning algorithm, we did not use the version of REP that was used in the experiments of section 4.1.2. Instead of generating the initial theory for REP with an implementation of FOIL we used FOSSIL with a cutoff of 0.0 for that purpose, so that both, REP and TDP, had to rely on the same underlying learning algorithm. The version using FOSSIL did better than the version using FOIL (compare tables 4.1 and 5.2).

Table 5.2 confirms 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

Average Accuracy		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 5.2: Accuracy in the KRK domain with 10% noise.

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 5.2). 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 (see section 4.2.4 for a discussion of the merits of a top-down search).

The top-down search for a good theory (without pruning) could even yield better results if we did not use the most specific theory within one standard error of the best theory, but the best itself as we have done in section 5.1.3. However, this could lead to over-generalization. Starting with an overly specific theory is less dangerous, because overfitting can be corrected by the subsequent post-pruning process.

Comparing the run-times of REP and TDP (table 5.3), it can be seen that TDP is much faster than REP. In fact it is even faster than REP's initial phase of overfitting alone. The reason for this is that TDP only has to find a few fairly general theories, while TDP generates huge theories that fit all the noisy examples. With increasing training set sizes, the costs of REP are dominated by the pruning process. This result is consistent with the analysis of section 4.2.1. 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.

Average Run-time		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 5.3: Run-time in the KRK domain with 10% noise.

For the remaining 9 sets the average run-time was 116.74 CPU secs. for growing and 12.88 CPU secs. for pruning.

The explanation for this surprising result can be found in two reasons:

1. TDP converges faster towards good theories. With increasing training set sizes the top-down search is less likely to return overfitting starting theories, because the estimation of their accuracy improves and the size of the allowed Standard Error decreases (see footnote 3). Decreasing the amount of allowed overfitting with increasing training set sizes is useful because FOSSIL's learning and noise-handling capabilities increase with training set sizes.

2. Because of 1. the starting theories learned by FOSSIL become increasingly more accurate as the training set grows, so that less and less pruning has to be done table 5.2). This explains why TDP's pruning times may decrease with increasing training set sizes.

This supports the two hypotheses stated in the last section. TDP will decrease learning time (1.) *and* pruning time (2.).

5.4 Summary

We have shown that the relational learning algorithm FOSSIL (described in section 3.2) is able to generate a series of theories in a top-down fashion. This method can be extended to find a good starting theory for a subsequent post-pruning process. Ideally this theory will be more specific than the most accurate theory found so far (to avoid over-generalization), but will nevertheless be close to the final theory, so that only a limited amount of post-pruning has to be performed. Thus this method — *Top-Down Pruning* — effectively combines pre-

and post-pruning. TDP brings a significant speed-up compared to *Reduced Error Pruning* without losing accuracy. Contrary to the speed-up gained by GROW (section 4.3) which concentrated on reducing the pruning costs only, the entire process of TDP may be significantly faster than REP's growing phase alone, because TDP can avoid excessive overfitting.

It should also be mentioned that the approach described above could not be taken by other relational learning systems that use pre-pruning, because it crucially depends on the use of the cutoff stopping criterion which in turn requires some important properties of FOSSIL's correlation heuristic. mFOIL (see section 3.3.4), another system that uses a parameter to adjust itself to the noise level in the data, requires some experimentation to determine an appropriate value of the m parameter. Also, a good value for m depends on the training set size as well as on the amount of noise in the data. The easiest approach is to try the standard settings used in the literature and choose the m that results in the best theory according to an independent test set. However, this approach does not guarantee that one does not miss a better theory with a different m. An automated approach that generates all theories for all possible settings of the m parameter is not as easy as it is for FOSSIL's cutoff parameter, because there is no upper bound for the m and, more importantly, the relation between the values of m, the heuristic values of the literals and mFOIL's significance-based stopping criterion is not obvious. On the other hand, FOIL e.g. could be easily adapted to use some form of FOSSIL's cutoff stopping criterion (only literals with an information gain above a certain percentage of the maximum gain will be added to a clause). Whether this is useful or not remains to be investigated.

In the next chapter we will introduce a method for tightly integrating pre- and post-pruning. This algorithm — *Incremental Reduced Error Pruning* — is independent of the used search heuristic and most relational learning systems can be adapted to make use of this method.

Chapter 6

Integrating Pre- and Post-Pruning

In chapter 5 we have presented an approach that combines pre- and post-pruning. An initial top-down search was able to find theories that are close to the intended final theory and could be used in a subsequent post-pruning phase. In this chapter we will investigate an alternative approach that will tightly integrate pre- and post-pruning into an efficient new algorithm that solves the problems mentioned in section 4.2. Contrary to Top-Down Pruning, the applicability of which to search heuristics other than FOSSIL's correlation heuristic is not yet clear, the algorithm introduced in this chapter will interleave learning and pruning in a way that is independent of the underlying search heuristic.

Excerpts of this chapter have been previously published in [Fürnkranz and Widmer, 1994].

6.1 Incremental REP

The new algorithm that we will present in this chapter was motivated by the observation that post-pruning is incompatible with the separate-and-conquer learning strategy as we have discussed in section 4.2.3. The problem is that pruning literals from a clause and thus generalizing it changes the situation for subsequent clauses in the theory. Some of them may become redundant, others may contain irrelevant conditions that have been selected based on examples that are now covered by the generalized clause (see figure 4.3). While redundant clauses are no problem (they will be removed in the next iteration of the post-pruning loop), a wrong choice of a literal cannot be undone by 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

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

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

Figure 6.1: Integrating Pre- and Post Pruning with *Incremental Reduced Error Pruning*

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.

A pseudo-code version of the algorithm can be found in figure 6.1. Before learning a clause, the current set of training examples is split into a growing (usually 2/3) and a pruning set (usually 1/3) as in many post-pruning algorithms. After learning a clause from the growing set, literals will be deleted from this clause in a greedy fashion until any further deletion would decrease the accuracy of this clause on the pruning set. The resulting rule will then be added to the concept description and all covered positive and negative examples will be removed from the training — growing *and* pruning — set. The remaining 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 will be learned. When the predictive accuracy of the pruned clause is below the predictive accuracy of the empty clause (i.e. the clause with the body `fail`), the clause will not be added to the concept description and I-REP returns the learned clauses. Thus the accuracy of the pruned clauses on the pruning set also serves as a stopping criterion. Post-pruning methods are used as pre-pruning heuristics.

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 the post-pruning algorithms REP (section 4.1) and GROW (section 4.3), because it is aimed at solving the problems of section 4.2:

Efficiency: I-REP does not generate an intermediate concept description. Thus the costs of I-REP are roughly the costs of generating the final theory, while REP and GROW have to generate a more specific theory first. As in REP, growing one clause from purely random data costs $n \log n$ (see section 4.2.1). I-REP considers *every* literal in the clause for pruning, i.e. each of the $\log n$ literals has to be tested against n examples until the final clause has been found, i.e. at most $\log n$ times. Thus the costs of pruning one clause are $n \log^2 n$. As the size of the final theory is assumed to be constant, the overall costs are also of the order $n \log^2 n$. This is significantly lower than the complexity of growing an overfitting theory which has been shown to be $\Omega(n^2 \log n)$ in section 4.2.1. Thus I-REP can be expected to be faster than both GROW and REP.

Split of Training Data: I-REP redistributes its pruning and growing sets after a clause has been found. This ensures that the examples are split according to the user-specified proportions (usually 2/3 of the examples are in the growing and 1/3 in the pruning set). Thus the scope of the problems discussed in section 4.2.2 is limited to the learning of a single clause. However, it may happen that a clause learned from a bad growing set or evaluated on a bad pruning set appears worse than the empty clause, which might cause I-REP to prematurely stop learning, while REP would continue learning and prune the bad clause later on.

Separate-and-Conquer Strategy: I-REP learns the clauses in the order in which they will be used by a PROLOG interpreter. Before subsequent rules will be learned, each clause is completed (learned *and* pruned) and all covered examples are removed. For this reason the problems discussed in section 4.2.3 cannot appear in I-REP.

Bottom-Up Hill-Climbing: Similarly to GROW, I-REP uses a top-down approach instead of REP's bottom-up search: Final programs are not found by removing unnecessary clauses and literals from an overly specific theory, but by repeatedly adding clauses to an initially empty theory. However, GROW still has to generate an intermediate, specific concept description, while I-REP directly constructs the final theory. In cases where the correct concept definition is fairly simple, the top-down approach can be expected to be less sensitive to local optima as discussed in section 4.2.4.

```
illegal(A,B,C,D,E,F):-adj(A,E),adj(B,F),\+lt(E,A)
Found clause: illegal(A,B,C,D,E,F):-adj(A,E),adj(B,F)

illegal(A,B,C,D,E,F):-D==F,lt(C,E),\+adj(B,D)
Found clause: illegal(A,B,C,D,E,F):-D==F

illegal(A,B,C,D,E,F):-C==E,lt(B,F)
Found clause: illegal(A,B,C,D,E,F):-C==E

illegal(A,B,C,D,E,F):-B==D,A==C
Found clause: illegal(A,B,C,D,E,F):-B==D,A==C

illegal(A,B,C,D,E,F):-lt(F,B),\+lt(A,E),\+lt(D,B),\+adj(C,E),A==E
Found clause: illegal(A,B,C,D,E,F):-fail


Found Theory (99.57% correct):
illegal(A,B,C,D,E,F):-adj(A,E),adj(B,F)
illegal(A,B,C,D,E,F):-C==E
illegal(A,B,C,D,E,F):-D==F
illegal(A,B,C,D,E,F):-B==D,A==C
```

Figure 6.2: I-REP learning from noisy KRK examples.

Most of the efficiency of the I-REP algorithm comes from the integration of pre-pruning and post-pruning by defining a stopping criterion based on the accuracy of the pruned clause on the pruning set. Thus I-REP does not need REP's `delete-clause` operator (see section 4.1.1), because the clauses of the final theory are constructed directly and learning stops when no more useful clauses can be found. However, this may also cause problems: Whenever the pruned clause does not have an accuracy above the accuracy of the empty clause, no more clauses will be learned. If this accuracy is not estimated accurately, either because there are not enough remaining examples or because of a bad split, I-REP will be prone to over-generalization. Using the terminology of [Schaffer, 1993, Wolpert, 1993], I-REP has a strong *Overfitting Avoidance Bias*, which can be detrimental in some domains (see [Murphy and Pazzani, 1994] for experiments along related lines).

6.2 An Example

Figure 6.2 shows an example run of I-REP in the KRK domain. The data it has learned from are the same 250 examples that have been used to generate the overly specific theory from figure 4.2. As the same initial split into growing and pruning sets has been used, the first clause that has been learned is the same in both examples. However, I-REP generalizes this clause appropriately by immediately pruning its last literal (`\+lt(E,A)`). After this it redistributes the remaining training examples into a new growing and a new pruning set to ensure that the next clause can be learned from the specified percentage of the remaining examples. The next three clauses are learned with the same method. The last learned clause contains only useless literals. Consequently its accuracy is not higher than that of the empty clause and learning stops at this point without adding this clause to the theory.

The resulting 4-clause theory is 99.57% correct (see also figure A.3). If we compare the resulting theory with the theory of figure 4.2 we see that the first 3 clauses are contained in clauses of the overfitting theory and thus could be uncovered by a post-pruning algorithm like REP or GROW. The last clause, however, could not be found by a post-pruning algorithm, because the pair of literals `(B==D,A==C)` is not contained in a single clause of the theory in figure 4.2. Therefore the best theory a post-pruning algorithm can find is the 98.45% correct theory consisting only of the first 3 clauses (see section A.2). There are two possible explanations for this: Either REP could not find this clause (or a specialization of it) because the previous clauses have distorted the space of training examples (see section 4.2.3) or the original split of the training examples has been unfortunate so that none or too few of the examples supporting this clause have been in the growing set (see section 4.2.2). Whatever the problem has been, I-REP seems to have solved it.

Of course, this example was specifically chosen to illustrate the possible gain of using I-REP. However, the next section will demonstrate that this example was nevertheless characteristic, and that I-REP is able to gain efficiency as well as accuracy compared to post-pruning algorithms.

6.3 Experiments

We have tested I-REP on the KRK chess endgame domain used in the previous chapters and described in detail in appendix A. We used 5 different training set sizes containing 10% of artificial class noise, i.e. 10% of the training examples were misclassified. For each training set size we used 10 different example sets except for the sets with 1000 examples, which were only tested on 6 sets because of the high run-times of this task. Accuracies were measured on 5000 noise-free examples. The training and test data were the same for all algorithms.

6.3.1 Implementation of the Algorithm

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

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

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

For comparison, REP and GROW were implemented as described in [Cohen, 1993] with the exception that `delete-last-literal` was used as a clause pruning operator (as in [Brunk and Pazzani, 1991]) instead of Cohen's `delete-last-sequence` operator that deletes a final sequence of literals from a clause (see section 4.1.1).[1] I-REP uses `delete-any-literal` for pruning a clause. In order to have a comparison to the original version of REP we used information gain as a search heuristic. Consequently, we also implemented this heuristic in the I-REP algorithms to make sure that all differences can be attributed to the used pruning method. There is, however, no fundamental requirement for using this method. Any other heuristic, in particular FOSSIL's correlation heuristic, could have been used as well.

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 into growing (2/3) and pruning (1/3) sets. All programs made use of mode, type and symmetry information about the background relations to restrict the search space. Run-times were measured in CPU secs. on a SUN SPARCstation IPX.

6.3.2 Results

Table 6.1 shows a comparison of the run-times of the different algorithms. The column *Initial Rule Growth* refers to the initial growing phase that REP and

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

GROW have in common[2], while the columns REP and GROW give the results for the pruning phases only. Thus the total run-time of REP (GROW) is the run-time of *Initial Rule Growth* plus the run-time of REP (GROW). Figure 6.3 illustrates the effect of training set size on the total run-time of all algorithms in the KRK domain.

Domain	Initial Rule Growth	REP	GROW	I-REP	I-REP-2
KRK-100 (10%)	8.36	2.44	1.66	4.20	4.37
KRK-250 (10%)	91.31	104.98	19.81	17.30	18.09
KRK-500 (10%)	456.56	1578.16	100.81	46.32	57.05
KRK-750 (10%)	1142.78	7308.84	361.41	83.64	118.99
KRK-1000 (10%)	2129.89	23125.34	806.89	115.35	178.26

Table 6.1: Average Run-Time

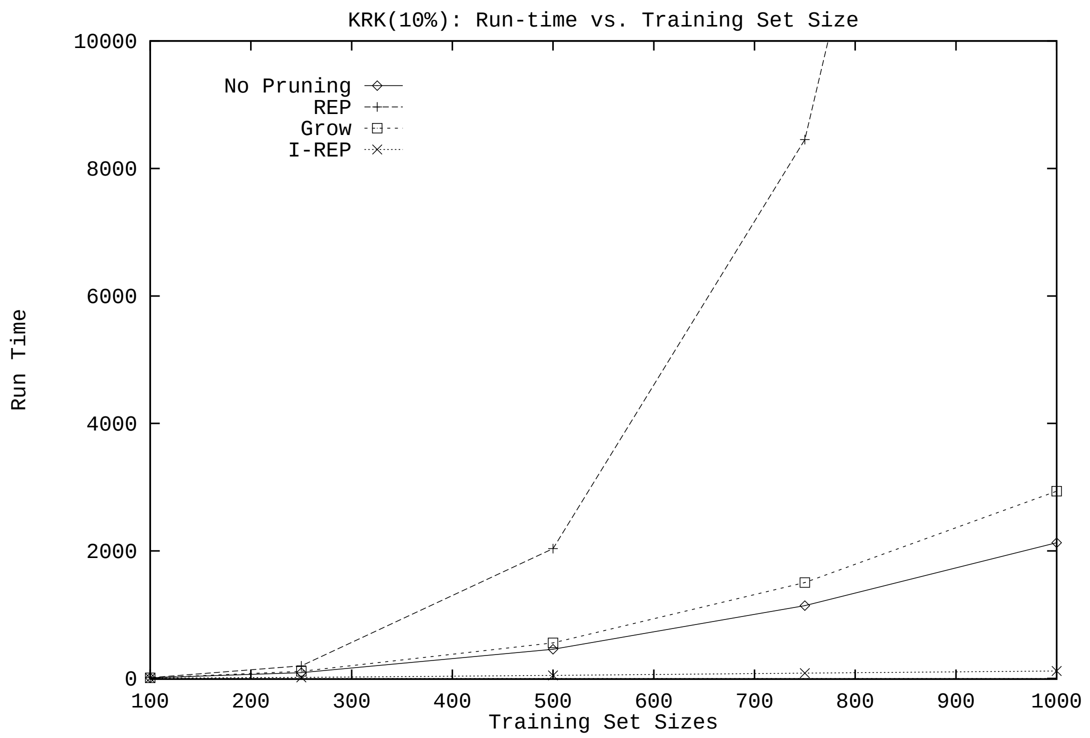
It is obvious that I-REP is usually significantly faster than the post-pruning algorithms. In fact, it is always faster than the initial growing phase that both REP and GROW have in common, because I-REP does not have to learn an intermediate overfitting theory. It can also be seen that GROW's pruning algorithm is much faster than REP's, which confirms the results of section 4.3.

Domain	Initial Rule Growth	REP	GROW	I-REP	I-REP-2
100-250	2.61	4.11	2.71	1.54	1.55
250-500	2.32	3.91	2.35	1.42	1.66
500-750	2.26	3.78	3.15	1.46	1.81
750-1000	2.16	4.00	2.79	1.12	1.41

Table 6.2: Log-log analysis of the run-times on noisy KRK data.

In order to get an idea on the asymptotic complexity of the various algorithms we have performed a log-log analysis as has been done in [Cameron-Jones, 1994]. Dividing the logarithm of the run-time by the logarithm of the training set size

[2]As in the overfitting phase only 2/3 of the training data are used for learning, the *Initial Rule Growth* column should only be used with caution for comparing pruning to non-pruning approaches, in particular with respect to the accuracy results. However, one could choose to compare e.g. the *Initial Rule Growth* results for 750 examples with the results of the pruning algorithms for 500 examples to get an idea how much improvement in terms of accuracy pruning brings in this domain.



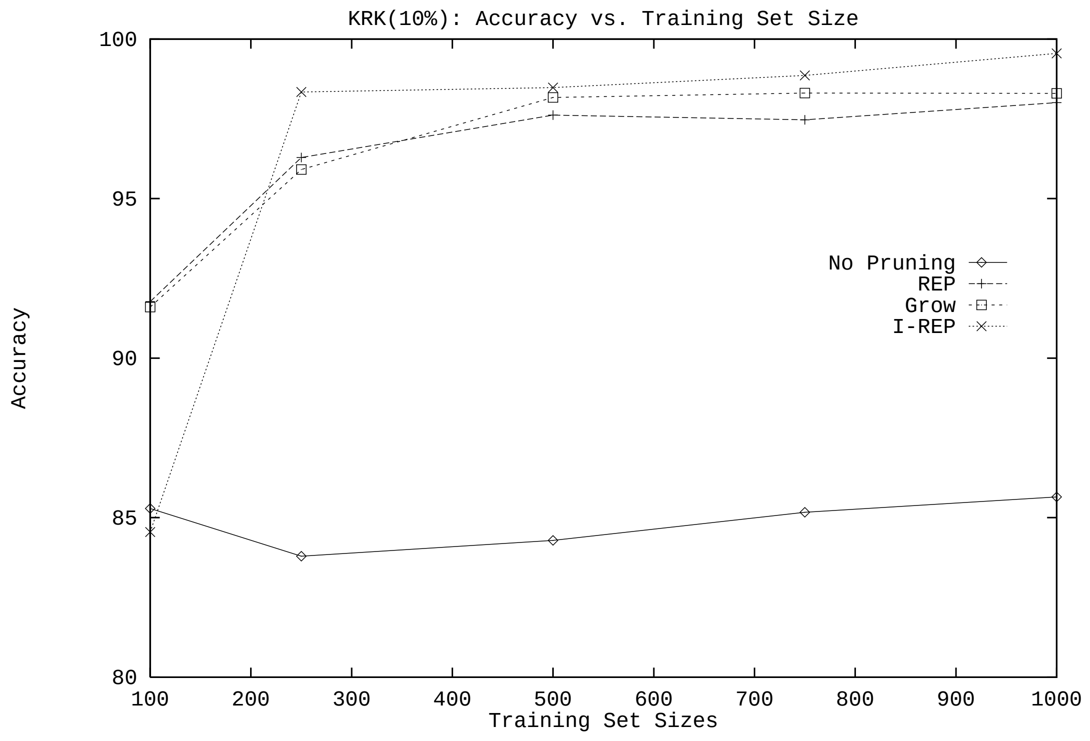


Figure 6.3: Different Training Set Sizes in the KRK Domain.

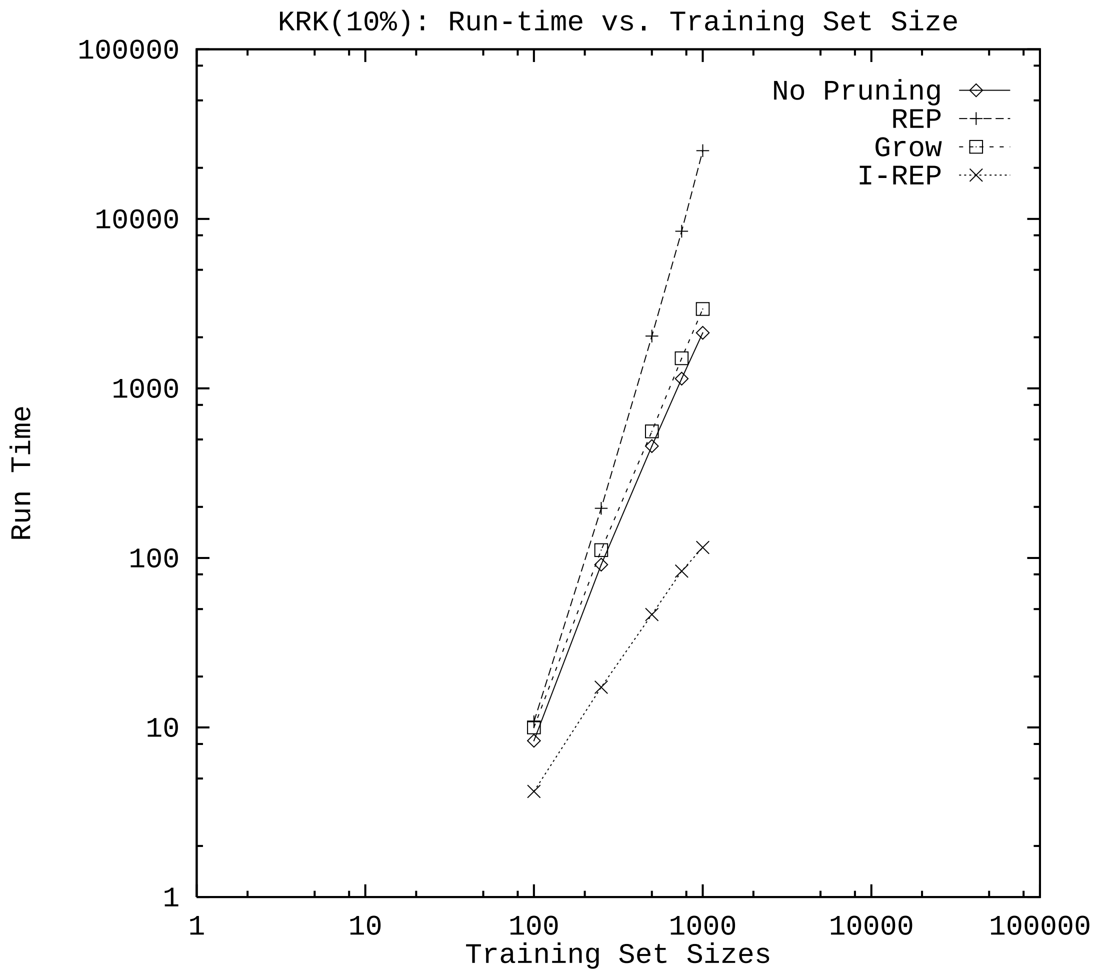


Figure 6.4: Log-log Plot of the Run-Times.

results in the slope of the log-log plot. Calculating this value for a polynomial function would yield the degree of the highest order term in this polynomial. We have tabulated the slopes for adjacent training set sizes in table 6.2. Note that contrary to figure 6.4, where the logarithms of the complete algorithms (initial overfitting and post-pruning) are plotted, the analysis of REP and Grow in table 6.2 was performed for the post-pruning phases only.

The main result for our study is that I-REP in fact seems to have a sub-quadratic time complexity. This is consistent with our conjecture of section 6.1, where we estimated I-REP to have a time complexity of $O(n \log^2 n)$. Thus it is significantly faster than the initial overfitting phase of both REP and Grow. In general the results we get are consistent with the analysis performed in [Cameron-Jones, 1994] for random data. In particular the evidence supports 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] (see also section 4.1). 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 claimed in [Cohen, 1993]. However, in all experiments of this section, the absolute values for the run-time of Grow has been negligible compared to the run-time of the over-fitting phase.

In terms of accuracy (table 6.3) I-REP also is superior to the post-pruning algorithms, although it seems to be more sensitive to small training set sizes (see also figure 6.3). 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 does better than the other algorithms.

Domain	Initial Rule Growth	REP	Grow	I-REP	I-REP-2
KRK-100 (10%)	85.29	91.77	91.60	84.55	85.37
KRK-250 (10%)	83.79	96.29	95.91	98.34	97.93
KRK-500 (10%)	84.29	97.62	98.17	98.48	95.67
KRK-750 (10%)	85.17	97.47	98.31	98.86	98.49
KRK-1000 (10%)	85.65	98.01	98.30	99.55	98.30

Table 6.3: Average Accuracy

REP often gets caught in local maxima and is not able to generalize to the right level. Interestingly, despite its top-down search strategy, Grow also occasionally overfits the noise in the data: Some of the highly specialized clauses in the intermediate theory sometimes also fit a few noisy examples in the pruning set and thus will be added to the concept description. This argument has been formalized in [Cameron-Jones, 1994]. In I-REP this is less likely to happen, because for each individual clause there is a high chance that if it fits noisy examples in the growing set it will not fit noisy examples in the pruning set. In REP and Grow, however, a large number of these clauses are investigated, which makes it more probable that this will happen for one of them. This is also the main reason why Grow has a higher asymptotic time complexity than the initial overfitting phase [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. Contrary to REP and Grow, I-REP can be expected to have very fast run-times on purely random data, 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.

In general it seems to be the case that Grow outperforms REP and that I-REP is better than REP and Grow. I-REP-2 seems to be worse than I-REP. Its purity criterion for evaluating a clause seems to have a preference for more specific clauses than I-REP (which can also be seen from the higher run-times).

6.4 Summary

In this chapter we have introduced a new method of integrating pruning and learning — *Incremental Reduced Error Pruning* (I-REP). This algorithm improves upon well-known post-pruning methods in the following ways:

Efficiency: The complexity of our algorithm, due to its new method of integrating pruning into learning, is of the order $n \log^2 n$. Experiments confirm the significant run-time improvement over REP and Grow, although I-REP uses the more expensive, but presumably more powerful pruning operator `delete-any-literal`.[3]

Separate-and-Conquer Strategy: In section 4.2.3 we argued that the separate-and-conquer strategy of many relational learning algorithms may lead to problems when used for overfitting noisy data. Our algorithm avoids this problem because the rules are pruned right after they are generated. Thus they are immediately adjusted to the right level of generality and the learning of subsequent clauses cannot be disturbed by the influence of an overly specific first clause.

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

I-REP's efficiency stems from the tight integration of post-pruning and pre-pruning. Whenever the algorithm learns a clause that is worse than the empty

[3]Preliminary experiments using `delete-any-literal` in the Grow algorithm indicate that its usage may not only result in an increase in run-time, but surprisingly also in a decrease in accuracy. [Brunk and Pazzani, 1991] also claim that in Foil and similar systems the more expensive `delete-any-literal` operator is not needed, because of the order in which the literals are added to the body of a clause. In addition in the example of section 6.2 we have seen that REP could not have found I-REP's theory with any pruning operator, because it was not part of its intermediate concept description.

clause, learning stops. However, at small training set sizes this may cause the algorithm to over-generalize because in those cases there is low redundancy in the data and chances that an unfortunate split of the training data might cause I-REP to stop prematurely are high. Similar problems can be expected in a domains with a rather specific underlying theory. Confirmation for this hypothesis in the form of additional experimental evidence can be found in [Fürnkranz and Widmer, 1994] and in chapter 7.

Chapter 7

Experimental Evaluation

After having presented an overview of new and old pruning methods for relational concept learning, we will now try to empirically evaluate and compare these methods in various test domains. We will mostly concentrate on comparing the approaches we have discussed in this thesis to each other, but will occasionally also include other learning systems into the tests to have a comparison to some well-known algorithms.

Parts of this chapter have previously been presented in [Fürnkranz, 1994a] and [Fürnkranz, 1994c].

7.1 Summary of the Experiments in the KRK Domain

We have tested several algorithms in the domain of recognizing illegal chess positions in the KRK chess endgame [Muggleton *et al.*, 1989]. This domain has become a standard benchmark problem for relational learning systems, because it cannot be solved in a trivial way by propositional learning algorithms. A detailed description of this domain can be found in appendix A.

In this series of experiments the signs of 10% of the training instances were deliberately reversed to generate noise in the data. The learned concepts were evaluated on test sets with 5000 noise-free examples. Although the training sets used are the same as in all previous chapters, some results may differ. In particular we used the up-to-date version (6.1) of FOIL[1], and the version of FOSSIL was also newer than the one used for the experiments of chapter 3. The most important difference is that the old version of FOSSIL was not able to make use

[1]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`.

of mode, type and symmetry information (see section 2.3.3) and that it counted the proofs for the covered instances (see the footnote on page 15), while the new version counts the instances itself (which is more efficient, but causes some problems with new variables).

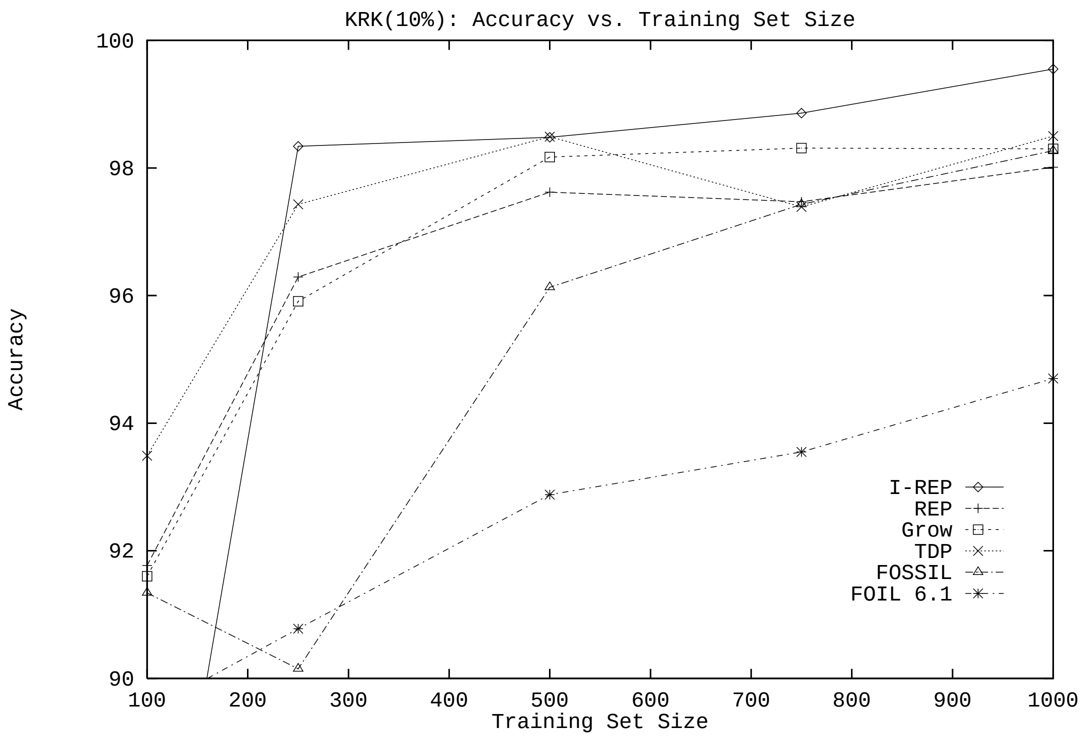
The algorithms were trained on identical sets of sizes from 100 to 1000 examples. All reported results were averaged over 10 runs, except for the training set size 1000, where only 6 runs were performed, because of the complexity of this task for some algorithms.

The tested algorithms were

- the pre-pruning systems Foil 6.1 [Quinlan and Cameron-Jones, 1993] and Fossil with a cutoff of 0.3 (section 3.2),
- the post-pruning systems REP (section 4.1) and Grow (section 4.3),
- the combined system TDP (section 5.2),
- and the integrated system I-REP (section 6.1).

All algorithms were implemented by the author in PROLOG (see appendix B) except for Foil 6.1 which is written in C. Foil 6.1 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. The PROLOG systems had all of their argument modes declared as input, which has the same effect. The only difference in search space between the PROLOG systems and Foil 6.1 was that the former did not consider recursive literals for efficiency reasons. Foil 6.1 would probably have been faster if this had been enforced, but no significant difference in accuracy can be expected as can be seen from the experiments in chapter 3.3.3 where the code of Foil 4 was modified to prevent recursion. Run-times for most algorithms were measured in CPU seconds for SUN SPARCstations ELC. The experiments with Fossil, Foil 6.1 and TDP, however, were performed on a SUN SPARCstation IPX, which gave the other algorithms a 5 : 6 advantage (empirically estimated on a few test runs on both machines).

Figure 7.1 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 is dependent on the training set size and thus too weak to effectively prevent overfitting the noise as we already have discussed in section 3.3.3. From 1000 examples Foil learns concepts that have more than 20 rules and are incomprehensible. I-REP on the other hand consistently produces the 99.57% correct, understandable 4-rule approximation of the correct concept description of figure A.3. This theory correctly identifies all illegal positions, except the ones where the white king is between the black king and the white rook



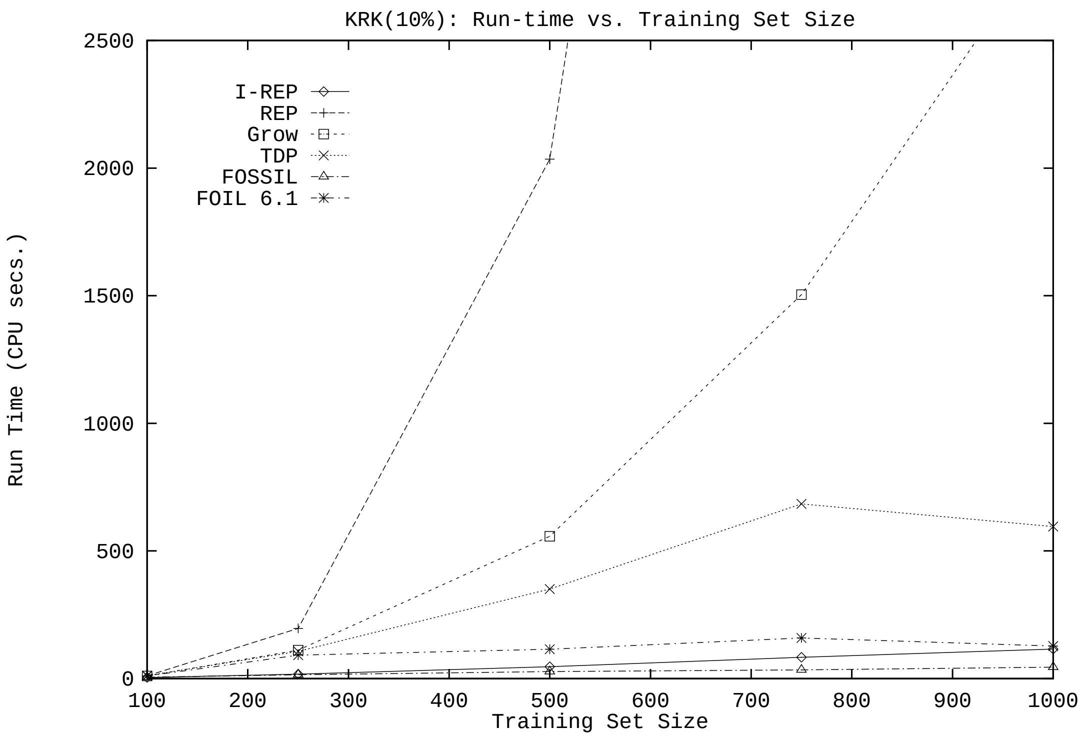


Figure 7.1: KRK domain (10% noise), different training set sizes

and thus blocks a check that would make the position illegal, because white is to move (see figure A.1 c)). The post-pruning approaches REP and GROW both are about equal, and TDP does not lose accuracy compared to them. All three, however, only rarely find the 4th rule of figure A.3. 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.

	Accuracy		Run-time		
Algorithm	Pre	Post	Pre	Post	Total
FOIL 6.1	**94.70**	—	127.17	—	**127.17**
FOSSIL (0.3)	**98.27**	—	44.39	—	**44.39**
REP	85.65	**98.01**	2129.89	23125.34	**25255.23**
GROW	85.65	**98.30**	2129.89	806.89	**2936.78**
TDP	96.56	**98.50**	433.54	162.25	**595.79**
I-REP	—	**99.55**	—	—	**115.35**

Table 7.1: KRK domain (10% noise), 1000 examples

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 as we have seen in section 3.3.3. REP proves that its pruning method is very inefficient. GROW has an efficient pruning algorithm, but still suffers from the expensive overfitting phase. TDP brings a little speed-up compared with REP and GROW, which is mainly due to the fact that it is able to start post-pruning with a much better theory than REP or GROW, as can be seen from table 7.1. 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 7.2, where accuracy (with the standard deviations observed in the different runs) is plotted against the logarithm of the run-time.

In summary, 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. I-REP integrates pre- and post-pruning into one criterion. 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.

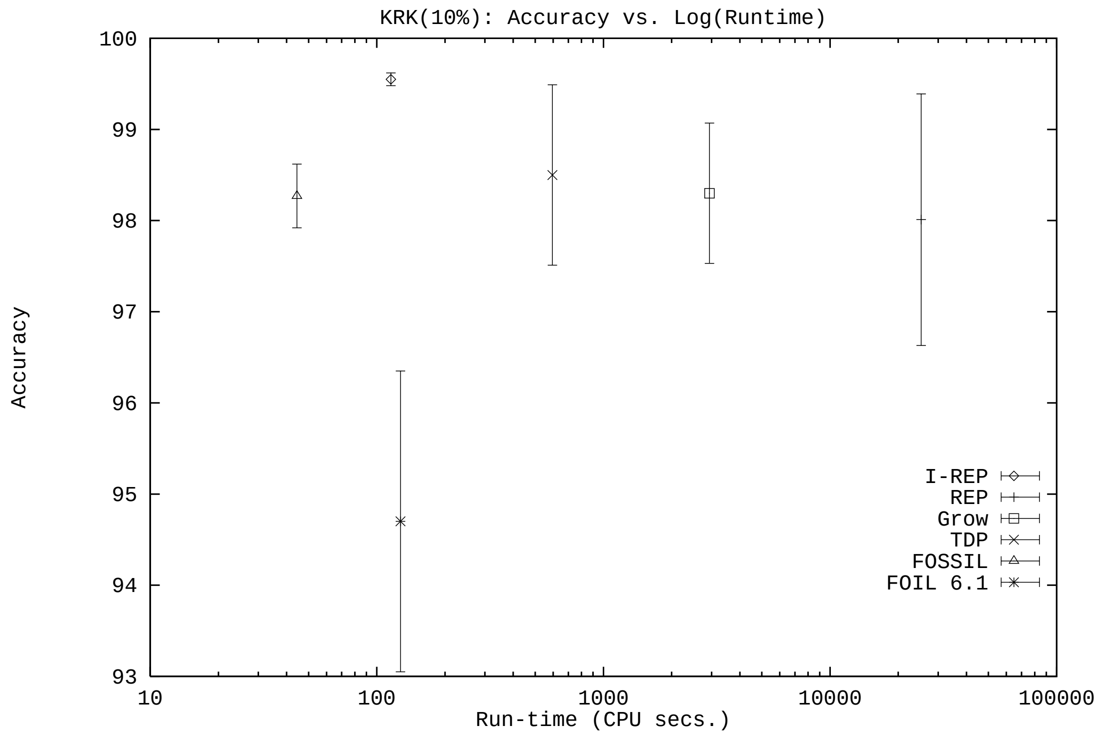


Figure 7.2: KRK domain (10% noise), 1000 examples

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 the correct 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, 1992a, 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 (A – E) 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, 1992a]: 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, 1992a] tested the target predicate with an unbound value for the number of finite elements for positive examples. The two procedures yield the same result when we assume that the rules are not overlapping, which, of course, cannot be guaranteed. The five results from testing on each of the five objects (and learning from the other four) are presented in tables 7.2 and 7.3. In table 7.2 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.

Algorithm	A		B		C	
	D		E		*Average*	
Fossil (0.05)	43.64	88.92	30.95	87.01	25.00	87.76
	28.07	83.92	28.12	86.77	**31.16**	**86.88**
Fossil (0.3)	0.00	91.17	0.00	90.91	0.00	90.48
	0.00	90.92	0.00	91.36	**0.00**	**90.97**
No Pruning	38.18	88.76	33.33	88.96	32.14	86.05
	19.30	86.94	34.38	86.41	**31.47**	**87.42**
REP	40.00	92.13	33.33	89.61	21.43	88.10
	28.07	87.90	11.46	85.96	**26.86**	**88.74**
Grow	38.18	92.13	26.19	90.04	21.43	88.44
	29.82	88.06	3.12	87.67	**23.75**	**89.27**
No Pruning (TDP)	40.00	92.13	28.57	88.74	21.43	86.73
	26.32	87.42	28.12	89.92	**28.89**	**88.99**
TDP	43.64	92.46	21.43	88.74	21.43	88.44
	29.82	87.74	3.12	88.21	**23.89**	**89.12**
I-REP	36.36	89.57	14.29	91.77	28.57	89.12
	29.82	92.20	3.12	88.03	**22.43**	**90.14**
I-REP-2	0.00	90.85	14.29	91.77	17.86	91.84
	29.82	92.20	2.08	91.54	**12.81**	**91.64**

Table 7.2: Accuracies in the mesh domain

Some of the results reported here differ a little from the ones we have reported previously [Fürnkranz, 1993b, Fürnkranz, 1994d, Fürnkranz and Widmer, 1994], because the previous results are from different phases of our research which used no or different constraints on the search space. However, the general picture remains the same: I-REP is clearly faster and a little more accurate than all of the other algorithms (with the exception of Fossil (0.3)). The advantage in terms of accuracy of I-REP is not as significant as in [Fürnkranz and Widmer, 1994]. However, I-REP-2, the version of I-REP that prefers purity over accuracy (see section 6.3.1), is better than I-REP in this domain. It usually learns only a few simple rules, and is the only algorithm that is able to improve upon the default accuracy (which is the result of Fossil (0.3) for reasons that we will discuss below). Fossil with a cutoff of 0.05 (this value was empirically established in

Algorithm	A	B	C	D	E	*Average*
Fossil (0.05)	13126.42	3990.41	9427.06	1977.22	3616.08	**6427.44**
Fossil (0.3)	15.94	13.84	21.63	13.02	12.54	**15.99**
No Pruning	9284.98	3712.14	7982.03	7541.57	3257.74	**6355.69**
REP	29669.36	19757.67	42305.26	38442.16	11144.54	**28263.80**
Grow	14437.17	6649.63	12225.26	11769.10	4320.45	**9880.32**
No Pruning (TDP)	2364.68	3833.84	6743.36	4521.52	1351.28	**3762.94**
TDP	3686.85	15622.16	18592.36	11104.58	1550.42	**10111.27**
I-REP	892.70	290.77	512.35	443.83	216.60	**471.25**
I-REP-2	132.01	293.15	234.09	443.51	352.84	**291.12**

Table 7.3: Run-times in the mesh domain

[Fürnkranz, 1993b]) achieves a relatively good performance on unseen positive examples. Its overall accuracy, however, is very bad.

The post-pruning algorithms are very inefficient[2]. Grow is more efficient and more accurate than REP. TDP clearly outperforms REP because it can start post-pruning with simpler and better theories. TDP's pruning time is not as low as in [Fürnkranz, 1994d], but still significantly lower than REP's. The time needed for finding the starting theory is also lower than the time needed for overfitting. It might be worthwhile to further speed up TDP by using Grow for the post-pruning phase. An interesting phenomenon is that pruned theories are more accurate, but cover fewer positive examples. The reason is that the post-pruning process usually removes a lot of rules that cover a few positive examples, but also an equal or greater number of negative examples. In some domains, however, this simple evaluation of the quality of a clause might be undesirable. [Kononenko and Bratko, 1991] discuss some alternatives to an accuracy-based evaluation of classifiers that could easily be adapted for pruning algorithms. Cost-sensitive pruning has recently been tried in [Knoll *et al.*, 1994].

The only algorithm faster than I-REP is Fossil with a cutoff of 0.3. The reason for this is simply that 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). However, it is still second best in terms of accuracy which again shows how poorly ILP algorithms do in this domain.

One of the reasons why ILP algorithms do not perform well in this domain is that there are not enough training examples. Certain numbers of finite elements on an edge appear in only one of the five objects. Thus rules for these classes cannot be learned from the other objects. Only examples for classes 1 and 2 are present in all five objects. For this reason recently five new objects have been

[2]The given run-times are the total run-times (learning *and* pruning).

included into the database. Preliminary results reported in [Dolšak *et al.*, 1994] show significant improvements compared to previous results using only five objects.

Another reason for the bad performance of ILP algorithms on this problem is that many of them (including our implementations) are not really able to make use of the provided topological relations, because of the problems discussed in section 2.3.1. For algorithms like GOLEM [Muggleton and Feng, 1990] and FOIL [Quinlan and Cameron-Jones, 1993] the topological relations have been made determinate, i.e. it has been enforced that each topological relation has at most one output value for each set of input values. [Džeroski and Bratko, 1992a] report experiments that forced mFOIL to use one of the topological relations at the beginning of the clause. This did not help much for the first four objects, but significantly improved the quality of the learned rules for object E. In section 3.2.2 we have discussed that FOSSIL's correlation heuristic is undefined for many of the problematic cases and we have sketched a method how we can exploit this characteristic in order to deal with the problem. We hope to significantly improve our results in this domain by working out these ideas and by using them on the new data set [Dolšak *et al.*, 1994] which contains a total of 10 objects (and thus provides more redundancy).

7.3 Propositional Data Sets

The purpose of the series of experiments reported in this section is that we want to compare our algorithms on a variety of natural data sets in order to get more information about their applicability to real-world problems. Therefore, in addition to the experiments performed in the relational mesh domain (section 7.2) we have experimented with data sets from the UCI repository of Machine Learning databases that have previously been used to compare propositional learning algorithms. Propositional data do not challenge the underlying relational learning algorithms to their full extent. However, we are primarily concerned with the comparison of pruning methods, and for these it should not make much difference what sort of conditions — relational or propositional — are pruned. This holds in particular for those algorithms that clearly separate the learning and pruning phases.

A side effect of using propositional data is that we can compare propositional and relational learning algorithms and confirm that the quality of the concepts learned by the latter is not below the quality of the theories learned by the former (see [Cameron-Jones and Quinlan, 1993b] for more experiments along these lines). 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 cannot be handled by our current 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 and one constant argument. Wherever appropriate, comparisons between two different variables of the same data type were allowed as well. Introduction of new variables was not allowed. 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 7.4, 7.5, and 7.6. 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 state-of-the-art 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 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. The explanation for this probably lies in the fact that all algorithms except for FOSSIL only learn from 2/3 of the training data and use the remaining 1/3 for pruning. If not much pruning has to be done, it can be expected that FOSSIL is slower than the other algorithms.

Most differences in accuracies are not statistically significant.[3] Significant differences can be found in the *KRKPa7* chess endgame domain, where TDP and

[3]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

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
I-REP-2	70.89	5.23	19.58	27.69
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
I-REP-2	78.66	2.80	7.34	62.07
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
I-REP-2	97.40	0.65	2.02	1114.92

Table 7.4: Results in the Breast Cancer, Hepatitis, and Sick Euthyroid domains.

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
I-REP-2	76.04	4.16	14.00	68.18

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
I-REP-2	94.94	1.52	4.81	30.57

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
I-REP-2	87.21	2.77	10.01	43.53

Table 7.5: Results in the Glass and Votes domains.

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
I-REP-2	97.58	0.99	3.26	3148.28
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
I-REP-2	80.27	6.45	25.06	10.39
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
I-REP-2	99.97	0.04	0.11	2482.93

Table 7.6: Results in the Chess (KRKPa7), Lymphography, and Mushroom domains.

Fossil performed significantly (1%) worse than all other algorithms. Fossil was significantly (5%) better than TDP in the *Votes (VI)* domain[4] and outperformed (5%, sometimes 1%) all other algorithms 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 relational algorithms seem to be competitive.

To allow a more structured analysis we have grouped the 9 domains into 3 subclasses: Table 7.4 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.[5] Table 7.5 contains domains where pruning does not make a significant difference and finally table 7.6 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 algorithms. The *Mushroom* and *KRKPa7* domains are known to be free of noise, while the medical domains of table 7.4 are noisy. Therefore we suspect that our grouping of the domain corresponds to the amount of noise contained in the data.

Summary	Table 7.4	Table 7.5	Table 7.6	Total
C4.5	83.63	86.40	92.24	87.42
Fossil	82.33	87.25	94.12	87.90
No Pruning	78.43	85.46	93.72	85.87
REP	81.49	86.77	93.22	87.16
Grow	80.81	86.25	93.05	86.70
No Pruning (TDP)	80.56	86.71	93.33	86.87
TDP	82.88	85.66	92.75	87.10
I-REP	82.34	86.10	92.29	86.91
I-REP-2	82.33	86.06	92.61	87.00

Table 7.7: Average performances of the algorithms

Table 7.7 shows the average performances of the algorithms in each of the three different groups and a total comparison. Fossil seems to a good choice for

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 7.4 – 7.6.)

[4]This is the *Votes* domain with the most significant attribute removed.

[5]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).

all domains. I-REP's comparative performance seems to decrease when the noise level in the data decreases. This is probably due to its very strong *Overfitting Avoidance Bias*, which may lead to over-generalizations in noise-free domains. REP is better with lower noise levels for the opposite reason: Its specific-to-general search strategy may get stuck in a local optimum, which can lead to over-specializations some cases.

In summary, it has been confirmed that I-REP provides a significant speed-up and is the most efficient algorithm. However, in terms of accuracy it only proved useful in noisy domains. The pre-pruning approach taken by FOSSIL exhibits considerable stability. In general, the relational learning systems seem to be competitive with the well-known decision tree learning system C4.5 in propositional domains.

Chapter 8

Conclusion

This thesis has been concerned with pruning in relational learning, a subarea of the rapidly growing field of *Inductive Logic Programming.* In many real-world problems not only the values of the attributes, but also relations between them may be relevant for the definition of the target concept. Relational learning algorithms extend classical propositional learning systems like CN2 and ID3 with the ability to include this kind of knowledge. From a set of positive and negative examples and background knowledge in the form of PROLOG relations, relational learners will try to find a PROLOG program that can derive the positive, but not the negative examples. As real-world problems very often contain erroneous values and incomplete or misclassified examples, noise handling is an important problem that has to be faced. *Pruning* is the common framework for avoiding this problem of *overfitting* the noise in the data. The basic idea is to incorporate a bias towards more general and simpler theories in order to avoid fitting 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, they use heuristics — so-called *stopping criteria* — to relax this constraint by stopping the learning process although some positive examples might not yet be explained by the current theory and it might still explain some of the negative examples. We have reviewed some of the basic algorithms that use this approach, most importantly FOIL which is the ancestor of most relational learning algorithms. Thereafter we have introduced a new system, FOSSIL. Although it uses the same basic *separate-and-conquer* learning strategy as FOIL, FOSSIL employs a new search heuristic based on statistical correlation which has several interesting properties that distinguish it from other approaches. Most importantly, we show how the correlation heuristic can effectively deal with noise when used with the simple *cutoff* stopping criterion. This new simple and efficient criterion does not depend on the number of training examples or on the amount of noise in the data and thus seems to be more robust than alternative approaches.

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 clauses and literals that only explain characteristics of the particular training set used and thus do not reflect true regularities of the domain are discarded. The quality of the found clauses and literals is commonly evaluated on a separate set of training examples that have not been seen during learning. We have reviewed two common post-pruning methods, *Reduced Error Pruning* (REP) and a variant that uses a top-down search, GROW. While both methods prove to be very effective in noise-handling, they are also very inefficient.

One of the reasons for the inefficiency of post-pruning methods is that the intermediate theory resulting from the initial overfitting phase can be much more complex than the final theory. Thus a lot of time has to be wasted in generating and subsequently simplifying this intermediate theory. One way for reducing 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. Our method, *Top-Down Pruning* (TDP), uses FOSSIL's simple cutoff stopping criterion to systematically vary the overfitting avoidance bias. Theories pruned to different degrees are generated 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, we can also achieve a significant speed-up in the post-pruning phase along with a slight gain in accuracy.

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) 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 and thus significant speedups 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.

In an extensive series of experiments in natural domains we have tried to confirm the quality of our algorithms. In all noisy domains, I-REP was superior in terms of both, accuracy and efficiency. However, this advantage was not as clear in propositional domains as it was in domains that require relations. In noise-free propositional domains, I-REP's strong overfitting avoidance bias often over-generalizes. Pre-pruning as used in FOSSIL appears to be the most stable method and should be used when nothing is known about the domain. A setting of 0.3 for the cutoff parameter has yielded good results in almost all tested domains, which illustrates the robustness of this algorithm against varying noise levels and varying training set sizes. REP counterintuitively compares well to other approaches at low noise levels. The reason for this is that at higher noise levels, where the initial theory has to be pruned a lot, REP's specific-to-general search strategy may get stuck at too specific theories. GROW effectively avoids this problem by replacing REP's bottom-up pruning algorithm with a top-down version. The search for a good starting theory as performed in TDP leads to a stable behavior. It brings speed-up and a high accuracy at higher noise levels, while it will still yield good results at low noise levels, although its general-to-specific search strategy is inappropriate then and may lead to higher run-times.

A severe deficiency of all approaches that incorporate post-pruning methods is that they usually separate a certain percentage of the available training data in order to have an independent test set on which the learned concepts can be evaluated and pruned. In particular in domains where training instances are expensive or rare, this method is wasteful. One approach to avoid this problem could be to use cross-validation or bootstrap methods for evaluating the accuracy of the theories (see e.g. [Bailey and Elkan, 1993]). In particular for the efficient I-REP the additional computational costs caused by this method might still be bearable. The well-known *Minimum Description Length* (MDL) principle [Rissanen, 1978] that trades off the complexity of a theory against its classification performance may prove to be an efficient alternative that allows to assess the quality of a theory without evaluating it on a separate test set.

However, the experiments in the mesh domain have exhibited another possible deficiency of many pruning algorithms: Their evaluation criterion — accuracy on the pruning set — gives equal consideration to covering positive and not covering negative examples. This may cause the percentage of correctly classified positive examples to go down, while the overall accuracy increases. [Kononenko and Bratko, 1991] discusses why this can be problematic and give an alternative to an accuracy-based evaluation of classifiers that could easily be adapted for pruning algorithms. Methods for including misclassification costs into pruning have already been incorporated into propositional learning systems [Knoll *et al.*, 1994].

We also feel that FOSSIL's ability to generate logic theories of various degrees of generality should be further explored. One possible application might be not to select a single theory, but to use all of them simultaneously for classification. This approach — *averaging* — has already been tried on propositional learning systems and has yielded encouraging results (see e.g. [Oliver and Hand, 1994]).

During the writing of this thesis the implementation of the learning systems has been continuously improved. In fact, the current version is already able to handle multiple classes by incorporating an approach similar to that suggested in the relational learner HYDRA [Ali and Pazzani, 1993]: For each class a separate concept description will be learned considering all examples for other classes as negative examples for the class to learn. Then the program orders the clauses of the resulting theories according to some heuristic in order to avoid conflicts caused by overlapping rules. A closer evaluation of the approach and a comparison to other approaches (e.g. the MDL based method used in C4.5 for generating rules from decision trees [Quinlan, 1993]) still has to be performed. Two other improvements that have been on the agenda since the first days of this project are to incorporate a simple beam search into FOSSIL and to elaborate the ideas for avoiding the problem with the introduction of new variables. In particular in the mesh domain that contains non-determinate background relations it can be expected that the latter will lead to a further improvement of the performance of FOSSIL. However, these issues are not directly relevant to pruning methods for relational learning and will therefore be discussed elsewhere.

Bibliography

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

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

[Bailey and Elkan, 1993] Timothy L. Bailey and Charles Elkan. Estimating the accuracy of learned concepts. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 895–900, Chambery, France, 1993.

[Bain, 1991] Michael Bain. Experiments in non-monotonic learning. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 380–384, Evanston, Illinois, 1991.

[Bergadano *et al.*, 1993] Francesco Bergadano, Daniele Gunetti, and Umberto Trinchero. The difficulties of learning logic programs with cut. *Journal of Artificial Intelligence Research*, 1:91–107, 1993.

[Bratko and King, 1994] Ivan Bratko and Ross King. Applications of Inductive Logic Programming. *SIGART Bulletin*, 5(1):43–49, 1994.

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

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

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

[Buntine and Niblett, 1992] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[Cameron-Jones and Quinlan, 1993a] R. M. Cameron-Jones and John Ross Quinlan. Avoiding pitfalls when learning recursive theories. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

[Cameron-Jones and Quinlan, 1993b] R.M. Cameron-Jones and J.R. Quinlan. First order learning, zeroth order data. In *Proceedings of the 6th Australian Joint Conference on AI*. World Scientific, 1993.

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

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

[Cestnik, 1990] Bojan Cestnik. Estimating probabilities: A crucial task in Machine Learning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, Stockholm, August 1990.

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

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

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

[De Raedt and Bruynooghe, 1993] Luc De Raedt and Maurice Bruynooghe. A theory of clausal discovery. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063, Chambery, France, 1993. Morgan Kaufmann.

[De Raedt and Lavrač, 1993] Luc De Raedt and Nada Lavrač. The many faces of Inductive Logic Programming. In J. Komorowski, editor, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993.

[De Raedt, 1992] Luc De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, 1992.

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

[Dolšak *et al.*, 1994] Bojan Dolšak, Ivan Bratko, and Anton Jezernik. 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, pages 305–320, Bad Honnef, Germany, 1994.

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

[Džeroski and Bratko, 1992b] Sašo Džeroski and Ivan Bratko. Using the m-estimate in Inductive Logic Programming. In *Logical Approaches to Machine Learning, Workshop Notes of the 10th European Conference on AI*, Vienna, Austria, 1992.

[Džeroski and Lavrač, 1991] Sašo Džeroski and Nada Lavrač. Learning relations from noisy examples: An empirical comparison of LINUS and FOIL. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 399–402, Evanston, Illinois, 1991.

[Džeroski and Todorovski, 1993] Sašo Džeroski and Ljupčo Todorovski. Discovering dynamics: From Inductive Logic Programming to machine discovery. In *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 125–137, Washington, DC, 1993. AAAI Press.

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

[Esposito *et al.*, 1993b] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and Michael Pazzani. A Machine Learning approach to document understanding. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 276–292, 1993.

[Flach, 1992] Peter A. Flach. Generality revisited. In *Logical Approaches to Machine Learning, Workshop Notes of the 10th European Conference on AI*, Vienna, Austria, 1992.

[Flach, 1993] Peter A. Flach. Predicate invention in inductive data engineering. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, number 667 in Lecture Notes in Artificial Intelligence, pages 83–94, Vienna, Austria, 1993. Springer-Verlag.

[Fürnkranz and Widmer, 1994] Johannes Fürnkranz and Gerhard Widmer. Incremental Reduced Error Pruning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 70–77, New Brunswick, NJ, 1994.

[Fürnkranz, 1991] Johannes Fürnkranz. Induktives Lernen durch Generieren von Decision Trees. Master's thesis, Vienna University of Technology, 1991. In German.

[Fürnkranz, 1993a] Johannes Fürnkranz. Avoiding noise fitting in a FOIL-like learning algorithm. In *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 14–23, 1993.

[Fürnkranz, 1993b] Johannes Fürnkranz. FOSSIL: A robust relational learner. Technical Report OEFAI-TR-93-28, Austrian Research Institute for Artificial Intelligence, 1993. Extended version.

[Fürnkranz, 1994a] Johannes Fürnkranz. A comparison of pruning methods for relational concept learning. In *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 371–382, 1994.

[Fürnkranz, 1994b] Johannes Fürnkranz. FOSSIL: A robust relational learner. In *Proceedings of the European Conference on Machine Learning*, pages 122–137, Catania, Italy, 1994. Springer-Verlag.

[Fürnkranz, 1994c] Johannes Fürnkranz. Pruning methods for rule learning algorithms. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, number 237 in GMD–Studien, pages 321–336, 1994.

[Fürnkranz, 1994d] Johannes Fürnkranz. Top-down pruning in relational learning. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 453–457, Amsterdam, The Netherlands, 1994.

[Gelfand *et al.*, 1991] S.B. Gelfand, C.S. Ravishankar, and E.J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):163–174, 1991.

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

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

[Kijsirikul *et al.*, 1992] Boonserm Kijsirikul, Masayuki Numao, and Masamichi Shimura. Discrimination-based constructive induction of logic programs. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 44–49, 1992.

[King *et al.*, 1992] R. King, S. Muggleton, R. Lewis, and M. Sternberg. Drug design by Machine Learning: The use of Inductive Logic Programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23), 1992.

[Knoll *et al.*, 1994] Ulrich Knoll, Gholamreza Nakhaeizadeh, and Birgit Tausend. Cost-sensitive pruning of decision trees. In *Proceedings of the European Conference on Machine Learning (ECML-94)*, pages 383–386, Catania, Italy, 1994.

[Kononenko and Bratko, 1991] Igor Kononenko and Ivan Bratko. Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6:67–80, 1991.

[Lapointe *et al.*, 1993] Stéphane Lapointe, Charles Ling, and Stan Matwin. Constructive Inductive Logic Programming. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1030–1036, Chambery, France, 1993. Morgan Kaufmann.

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

[Lavrač and Džeroski, 1993] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1993.

[Lavrač *et al.*, 1991] Nada Lavrač, Sašo Džeroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the European Working Session on Learning*, Porto, Portugal, 1991.

[Lavrač *et al.*, 1992a] Nada Lavrač, Bojan Cestnik, and Sašo Džeroski. Search heuristics in empirical Inductive Logic Programming. In *Logical Approaches to Machine Learning, Workshop Notes of the 10th European Conference on AI*, Vienna, Austria, 1992.

[Lavrač *et al.*, 1992b] Nada Lavrač, Bojan Cestnik, and Sašo Džeroski. Use of heuristics in empirical Inductive Logic Programming. In S.H. Muggleton, editor, *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan, 1992.

[Lavrač *et al.*, 1993] Nada Lavrač, Sašo Džeroski, Vladimir Pirnat, and Viljem Križman. The utility of background knowledge in learning medical diagnostic rules. *Applied Artificial Intelligence*, 7:273–293, 1993.

[Matheus *et al.*, 1993] Christopher J. Matheus, Philip K. Chan, and Gregory Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):903–913, December 1993.

[Michalski *et al.*, 1986] R. S. Michalski, I. Mozetič, J. Hong, and N. Lavrač. 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*, pages 1041–1045, Philadelphia, PA, 1986.

[Michalski, 1980] Ryszard S. Michalski. Pattern recognition and rule-guided inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:349–361, 1980.

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

[Mingers, 1989b] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[Mittenecker, 1977] Erich Mittenecker. *Planung und statistische Auswertung von Experimenten*. Verlag Franz Deuticke, Vienna, Austria, 8th edition, 1977. In German.

[Morales, 1992] E. Morales. Learning chess patterns. In Stephen Muggleton, editor, *Inductive Logic Programming*, pages 517–538. Academic Press Ltd., London, 1992.

[Muggleton and Buntine, 1988] Stephen H. Muggleton and Wray L. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352, 1988.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive Logic Programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[Muggleton and Feng, 1990] Stephen H. Muggleton and Cao Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 1–14, Tokyo, Japan, 1990.

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

[Muggleton *et al.*, 1992] S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based Machine Learning. *Protein Engineering*, 5(7):647–657, 1992.

[Muggleton, 1988] Stephen H. Muggleton. A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*, pages 123–130, 1988.

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

[Muggleton, 1993] Stephen H. Muggleton. Inductive Logic Programming: Derivations, successes and shortcomings. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, number 667 in Lecture Notes in Artificial Intelligence, pages 21–37, Vienna, Austria, 1993. Springer-Verlag.

[Muggleton, in press] Stephen Muggleton. Inverting implication. *Artificial Intelligence*, in press.

[Murphy and Pazzani, 1994] Patrick M. Murphy and Michael J. Pazzani. Exploring the decision forest: An empirical investigation of Occam's Razor in decision tree induction. *Journal of Artificial Intelligence Research*, 1:257–275, 1994.

[Niblett and Bratko, 1986] Tim Niblett and Ivan Bratko. Learning decision rules in noisy domains. In *Proceedings of Expert Systems 86*. Cambridge University Press, 1986.

[Oliver and Hand, 1994] Jonathan J. Oliver and David Hand. Averaging over decision stumps. In *Proceedings of the European Conference on Machine Learning (ECML-94)*, pages 231–241, Catania, Italy, 1994.

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

[Petta, 1994] Paolo Petta. The value of background knowledge for ILP: A case study in the interpretation of TI-201 myocardial SPECT polar maps. Technical Report OEFAI-TR-94-15, Austrian Research Institute for Artificial Intelligence, 1994.

[Plotkin, 1971] G. D. Plotkin. *Automatic methods of inductive inference*. PhD thesis, University of Edinburgh, Scotland, 1971.

[Quinlan and Cameron-Jones, 1993] John Ross Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pages 3–20, Vienna, Austria, 1993.

[Quinlan, 1983] John Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning. An Artificial Intelligence Approach*, pages 463–482. Tioga Publishing Co., 1983.

[Quinlan, 1986] John Ross Quinlan. The effect of noise on concept learning. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 149–166. Morgan Kaufmann, 1986.

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

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

[Quinlan, 1991] John Ross Quinlan. Determinate literals in Inductive Logic Programming. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 442–446, 1991.

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

[Quinlan, 1994] John Ross Quinlan. The minimum description length principle and categorical theories. In *Proceeding of the 11th International Conference on Machine Learning*, pages 233–241, New Brunswick, NJ, 1994.

[Richards, 1992] Bradley L. Richards. *An Operator-Based Approach to First-Order Theory Revision.* PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, 1992.

[Rissanen, 1978] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Rouveirol and Puget, 1990] Cèline Rouveirol and Jean François Puget. Beyond inversion of resolution. In *Proceedings of the 7th International Conference on Machine Learning*, pages 122–130, 1990.

[Rouveirol *et al.*, 1993] Céline Rouveirol, Hilde Adé, and Luc de Raedt. Bottom up generalization in I.L.P. In *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 59–70, 1993.

[Sammut, 1993] Claude Sammut. The origins of Inductive Logic Programming: A prehistoric tale. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 177–216, Bled, Slovenia, 1993.

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

[Shapiro, 1982] Ehud Y. Shapiro. *Algorithmic Program debugging.* The MIT Press, Cambridge, 1982.

[Srinivasan *et al.*, 1992] A. Srinivasan, S. H. Muggleton, and M. E. Bain. Distinguishing noise from exceptions in non-monotonic learning. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan, 1992.

[Stahl, 1993] Irene Stahl. Predicate Invention in ILP — an overview. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, number 667 in Lecture Notes in Artificial Intelligence, pages 313–322, Vienna, Austria, 1993. Springer-Verlag.

[Sternberg *et al.*, 1992] M. Sternberg, R. Lewis, R. King, and S. Muggleton. Modelling the structure and function of enzymes by machine learning. *Proceedings of the Royal Society of Chemistry: Faraday Discussions*, 93:269–280, 1992.

[Thrun *et al.*, 1991] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowitz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, December 1991.

[Weiss and Indurkhya, 1994] Sholom M. Weiss and Nitin Indurkhya. Small sample decision tree pruning. In *Proceedings of the 11th Conference on Machine Learning*, pages 335–342, Rutgers University, New Brunswick, NJ, 1994.

[Wogulis, 1991] James Wogulis. Revising relational domain theories. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 462–466, Evanston, Illinois, 1991.

[Wolpert, 1993] David H. Wolpert. On overfitting avoidance as bias. Technical Report SFI TR 92-03-5001, The Santa Fe Institute, Santa Fe, NM, 1993.

[Wrobel, 1989] Stefan Wrobel. Demand-driven concept formation. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*, number 347 in Lecture Notes in Artificial Intelligence, pages 289–319. Springer-Verlag, Berlin, 1989.

[Wrobel, 1994] Stefan Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–191, 1994.

[Zelle and Mooney, 1993] John M. Zelle and Raymond J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on AI*, pages 817–822, Washington, DC, 1993.

Appendix A

The KRK Domain

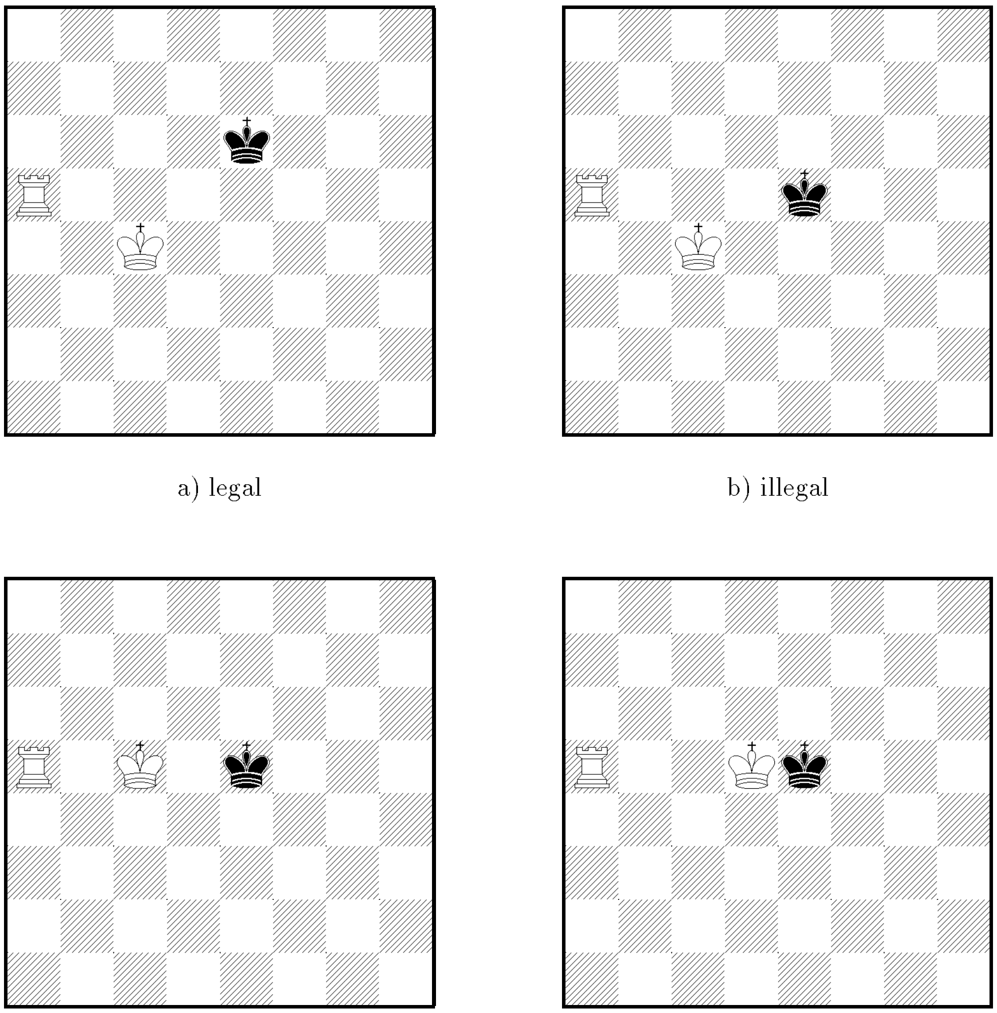
The KRK domain was first used in [Muggleton *et al.*, 1989]. Since then it has become a standard test bed for ILP algorithms such as Foil [Quinlan, 1990], Golem [Muggleton and Feng, 1990], Linus [Lavrač *et al.*, 1991], mFoil [Džeroski and Bratko, 1992a], NM-Golem [Bain, 1991], and Fossil. The goal is to distinguish legal from illegal positions in a chess endgame with white king, white rook and black king on the board. In all positions white is to move. Therefore all positions where the black king is in check are illegal. Figure A.1 a) is a legal position and therefore a negative example. Position b) on the other hand is illegal, because the white rook is on the same row as the black king who therefore is in check. In position c) the white king blocks the check of the white rook which makes the position legal again. However, if he does so on a square that is adjacent to the black king (thus both being in check) the position is illegal again as in figure A.1 d). Also illegal are positions where two or more pieces are on the same square.

This problem has become a standard benchmark problem for relational learning algorithms, because its solution requires the use of relations like equal or adjacent, and thus could not be solved by propositional learning systems (see also section 1.2).

A.1 A Correct Domain Theory

Technically speaking the goal is to find a correct definition of the predicate `illegal(A,B,C,D,E,F)`, the six arguments of which are the file and row resp. coordinates of the three chess pieces in the above order. Background knowledge usually consists of the predicates `X < Y`, `X == Y` and `adjacent(X,Y)`[1]. A correct

[1]Our definition of `adjacent` actually was `adjacent_or_equal`. In the text we usually have abbreviated the predicates with `adj`, `lt`, and `==`.



a) legal b) illegal

c) legal d) illegal

Figure A.1: Example positions for the KRK domain

```
%%%
%%% Correct KRK Program
%%% (including the between relation)
%%%

illegal( A, B, C, D, E, F) :- A == C, B == D.
    %  (1) WK and WR on same square.
illegal( A, B, C, D, E, F) :- C == E, D == F.
    %  (2) BK and WR on same square.
illegal( A, B, C, D, E, F) :- adjacent( A, E), adjacent( B, F).
    %  (3) WK and BK on adjacent (or equal) squares.
illegal( A, B, C, D, E, F) :- C == E, A \== C.
    %  (4) WR gives check and WK on different row
illegal( A, B, C, D, E, F) :- D == F, B \== D.
    %  (5) WR gives check and WK on different file
illegal( A, B, C, D, E, F) :- C == E, \+ between( B, D, F).
    %  (6) WR gives check and WK on same row
illegal( A, B, C, D, E, F) :- D == F, \+ between( A, C, E).
    %  (7) WR gives check and WK on same file

%%%
%%% Background Knowledge
%%%

adjacent( X, Y) :- X is Y + 1.
adjacent( X, Y) :- X = Y.
adjacent( X, Y) :- X is Y - 1.

between( Z, X, Y) :- X < Z, Z < Y.
between( Z, X, Y) :- Y < Z, Z < X.
```

Figure A.2: A correct theory

domain theory in PROLOG clauses is given in Figure A.2. Note that this theory is somewhat simplified by using the intermediate predicate `between` which is usually not available. Instead the appearances of `between` have to be expressed in terms of the relation `<`. In that case the negated `between` literal will unfold rules (6) and (7) to several other rules. Each of them will only cover a couple of examples and thus is very hard to detect.

Of course this is only one possibility of defining a theory of this domain. A close examination of the rules e.g. reveals that rule (2) is redundant, as the cases where the black king and the white rook are on the same square are already covered by rules (1), (4) and (5): Either all three pieces are on the same square (1) or only the black king and the white rook are on the same square and the white king is on a different row (4) or on a different file (5). Nevertheless we stick with this representation, because we feel it is more intuitive (and maybe easier to analyze). In some other domain definitions in the literature, `adjacent` has been defined without its second clause.

We will use the above domain theory for a numerical analysis of the domain, because the basic structure remains the same in other formulations of the theory. The seven clause theory of figure A.2 is easier to analyze.

Number of possible examples (chess positions): As each of the 6 variables can have 8 different values there are $8^6 = 262,144(100\%)$ different instances in this domain.

3 Pieces on same square: Naturally, there are 64 (0.024%) possibilities for having all three pieces on the same square, because there are 64 squares on a chess board.

2 Pieces on same square: There are 64 squares for the first piece, 1 possible square for the second piece and 63 different squares for the third piece. So we have altogether $64 \times 1 \times 63 = 4032(1.538\%)$ possibilities for two fixed pieces being on the same square and the third piece being on a different square.

Rule (1): From the above results follows that rule (1) of the domain theory covers $64 + 4032 = 4096(1.563\%)$ positions.

Rule (2): As the positions with all three pieces on one square are already covered by rule (1) there are only 4032(1.538%) positions left for rule (2).

Adjacent Kings: Assume you place the white king on a random square of the board. If it lands on one of the 36 squares on rows 2 to 7 and files `b` to `g` there are 8 adjacent squares where the black king can be placed. This adds up to $36 \times 8 = 288$ possibilities. If the white king is at one of the 4 corner squares, which have 3 adjacent squares each, we get another $4 \times 3 = 12$ possibilities for adjacent kings. The other 24 border squares have 5 adjacent squares each, i.e. $24 \times 5 = 120$. Therefore we have $288 + 12 + 120 = 420$ possibilities altogether for placing two kings on adjacent, but not identical squares on a chess board.

Rule (3): For each of the adjacent king positions there are 62 open squares for placing the white rook. Including the cases where the kings are on identical squares, which are also covered by this rule, we have rule (3) covers $420 \times 62 + 4032 = 30,072(11.472\%)$ positions.

Rule	Covered	Percentage
(1)	4,096	1.563%
(2)	4,032	1.538%
(3)	30,072	11.472%
(4)	22,932	8.748%
(5)	22,932	8.748%
(6)	1,456	0.555%
(7)	1,456	0.555%
illegal	86,976	33.179%
legal	175,168	66.821%
total	262,144	100.000%

Table A.1: Rule Coverage

Rules (4) and (5): For each position of the black king there are 7 possibilities to place the white rook on the same file/row. Depending on whether the black king is in the corner, on the border file/row, border row/file or in the center, we have 54, 53, 52 or 50 possible squares for the white king, not counting cases that have already been covered by rules (1) to (3). Thus each of the rules (4) and (5) covers $7 \times (54 \times 4 + 53 \times 12 + 52 \times 12 + 50 \times 36) = 22,932(8.748\%)$ positions.

White King not between Black King and Rook: We assume that all three pieces are on the same file/row, that the two kings are not adjacent and that all pieces are on different squares. If the black king is immediately adjacent to the white rook, there are 5 squares remaining for the white king under the above assumption, except for the case where the black king is on the border, where we have 6 possibilities. So we get $6 \times 5 + 1 \times 6 = 6 \times 6 = 36$ possible positions. The same number results when we swap black king and white rook. If we now increase the distance between black king and white rook by 1, we analogously get $2 \times (5 \times 5) = 50$ more positions. This is repeated until the distance between black king and white rook is 5 squares, when only 1 possible square is left for the white king. So we have a total of $2 \times (6^2 + 5^2 + 4^2 + 3^2 + 2^2 + 1^2) = 2 \times 91 = 182$ positions.

Rules (6) and (7): As the above calculations are the same for each file or row, each of the two rules covers $8 \times 182 = 1,456(= 0.555\%)$ positions.

A summary of the results is given in table A.1.

```
illegal( A, B, C, D, E, F) :- A == C, B == D.
    %  (a) same as (1).
illegal( A, B, C, D, E, F) :- adjacent( A, E), adjacent( B, F).
    %  (b) same as (3)
illegal( A, B, C, D, E, F) :- C == E.
    %  (c) generalization of (2), (4), (6).
illegal( A, B, C, D, E, F) :- D == F.
    %  (d) generalization of (2), (5), (7).
```

Figure A.3: Approximate Theory A

A.2 Approximate Theories

Sometimes ILP programs do not learn the complete theory, but instead learn an approximation. In this section we will examine some simple theories that give a very good approximation of the desired concept. As already mentioned, it is very hard for programs to find clauses (6) and (7) which are mostly concerned with the rare cases when the white king is between his rook and the opponent's king, thus preventing the check. So we will mostly examine good approximative theories when those 2 clauses are not present. Figure A.3 gives an approximate theory which has been reported in a similar form in [Quinlan, 1990], [Bain, 1991], and [Srinivasan *et al.*, 1992].

Theories like this are often learned in the presence of noise or from only a few training examples. In the first case it is very hard to discriminate between noisy and rare examples, while in the second case chances are high that the examples are missing that are representative for the missing rules with low coverage.

Theory A: The approximate theory of figure A.3 is an over-generalization of the theory in figure A.2, in a way that all positions with white rook and black king on the same row/file are treated as illegal. It is wrong for all cases where the white king is between the white rook and the black king, thus blocking the check.

We already have covered the case where the white pieces are on the same square and the cases where the two kings are adjacent will be handled by rule (b). Assume that all three pieces are on the same file/row. The maximum distance between white rook and black king is when both are on opposite sides of the file/row. The white king then has 5 possible squares for blocking the check, i.e. rules (c) or (d) resp. will consider 5 illegal positions as legal. We get another 5 cases when swapping white rook and black king. If the white rook and the black king are moved towards each other, the number of squares for the white king decreases, but there are more possibilities for putting the black king and the white rook on the board. So e.g. there are 2 possibilities to put the black king and the white rook on one file/row with 5 squares inbetween. Each of them yields 4 valid squares for the white king. Thus we get $2 \times (1 \times 5 + 2 \times 4 + 3 \times 3 + 4 \times 2 + 5 \times 1) = 70$ possibilities for each file/row. Considering that there are 8 rows and 8 files, rules (c) and (d) together erroneously classify a total of $70 \times 8 \times 2 = 560 \times 2 = 1,120 (= 0.427\%)$ of all examples.

Theory B: In the presence of 10% noise FOSSIL converges towards theory A except rule (1) is usually not found (see section 3.3). This is not surprising, because as we can see from table A.1 an example exclusively covered by rule (1) only occurs in 1.56% of the training data. The regularity in these examples can easily be overlooked when 10% of the examples are erroneously classified. Besides, many of the examples for rule (1) are covered by the three remaining rules, so that the error for leaving out this rule is not that big, as we will see in the following.

Of the 4096 positions with white king and white rook on the same square, $\frac{1}{8}$th $(= 512)$ have the black king on the same row. Another 512 have the black king on the same file. Subtracting the 64 positions where all three pieces are on the same square, which we have counted twice now, we get 960 positions that are covered by rules (c) and (d). In addition rule (b) covers positions where the white rook and king are one square diagonal of the black king. This are $36 \times 4 + 24 \times 2 + 4 \times 1 = 196$ positions. So rules (b), (c) and (d) of theory A cover $196 + 960 = 1156$ positions that would otherwise be covered by rule (a). This means that in addition to the error of Theory A, dropping rule (a) misclassifies another $4,096 - 1,156 = 2,940$ positions. The total error of Theory B thus is $2940 + 1120 = 4060 (= 1.549\%)$.

Theory C: Another good approximation results when rules (c) and (d) are replaced with rules (4) and (5). This means instead of generalizing rules (2) (4) and (6) to rule (c) the most common of the three rules is chosen as a representative. The cases that are not correctly classified by this approximation are those, where the white king is on the same row as his rook and the enemy king, but is not inbetween them, the black king thus being in check. We have already seen that rule (2) is redundant, so the only cases that will be misclassified by this approximation are those that would originally have been covered by rules (6) and (7), i.e. $1,456 \times 2 = 2,912 (= 1.111\%)$.

Theory D: Theories B and C can also be interleaved, e.g. only rule (c) is replaced by rule (4). In this case in addition to the 1,456 errors made by missing out rule (6), we have the 560 mistakes by overgeneralizing rules (2), (5) and (7) to rule (c). This means we have a total error $560 + 1,456 = 2,016 (= 0.769\%)$.

Theory	Rules	Error	Accuracy
A	(a), (b), (c), (d)	0.427%	99.573%
D	(a), (b), (c), (5) or (a), (b), (4), (d)	0.769%	99.231%
C	(a), (b), (4), (5)	1.111%	98.889%
B	(b), (c), (d)	1.549%	98.451%
F	(b), (c), (5) or (b), (4), (d)	2.019%	97.981%
E	(b), (4), (5)	2.489%	97.511%

Table A.2: Accuracy of Approximate Theories

Theory E: Of course, dropping rule (a) from Theory C yields another approximation. This amounts to dropping rules (1), (2), (6) and (7) from the correct theory of figure A.2. We have already seen that there are $420 + 64 = 484$ possibilities for placing the two kings on adjacent or identical squares. These cases are still covered by rule (b), but the remaining $4096 - 484 = 3612$ positions will be erroneously considered as legal by this theory. In addition the $2,912$ mistakes made by Theory D, this yields a total error of $2,912 + 3,612 = 6,524 (= 2.489\%)$ for Theory E.

Theory F: The last approximation we consider is dropping literal (a) from the two possible Theories D. Let us assume we have the theory consisting of rules (b), (c) and (5). Of the 4096 positions with white king and white rook on the same square, $\frac{1}{8}$th $(= 512)$ have the black king on the same file. These instances are now covered by rule (5). In addition rule (b) covers all positions with the black king one square above or below this file. Considering different numbers of neighboring squares we get $36 \times 6 + 12 \times 4 + 12 \times 3 + 4 \times 2 = 308$ covered positions. So $4096 - 512 - 308 = 3276$ positions with white king and rook on the same square are not covered by other rules. Also replacing rule (d) with (5) misclassifies 2,016 examples as we have seen above. So we get a total error of $2,016 + 3,276 = 5,292 (= 2.019\%)$.

A summary of the approximation errors for approximate theories ordered according to their approximation accuracy can be found in table A.2.

Appendix B

Implementation of the Algorithms

All of the relational learning systems have been implemented by the author in SICStus PROLOG 2.1. To be able to make fair runtime comparisons, all systems had major parts of their implementations in common. In particular they shared a common basic search engine that enumerates all possible literals that can be added to a clause and — for the case of pruning — the literals that can be removed from a clause. Heuristic evaluation and selection of appropriate literals or pruning operators was up to the individual programs.

All programs also had the same interface to the data. This interface allows to specify the training and test instances, the valid background relations and their mode type and symmetry constraints as discussed in section 2.3.3. We will briefly illustrate the main features of the interface here with definitions of the background knowledge for the KRK domain of appendix A.

Training Instances: Training instances are usually specified as a collection of ground facts. Positive instances are denoted with

`pos_instance(`*Instance*`).`

and negative instances with

`neg_instance(`*Instance*`).`

However, the definitions of these two predicates do not have to be extensional. The predicates can also have intensional definitions as long as backtracking over the predicates will produce a finite set of positive and negative training instances for the target predicate.

The example positions of figure A.1 can be specified as

```
pos_instance(c,4,a,5,e,6).
```

```
neg_instance(c,4,a,5,e,5).
pos_instance(c,5,a,5,e,5).
neg_instance(d,5,a,5,e,5).
```

Target predicate: The target predicate is specified with

`target(`*Predicate,Types*`).`

Predicate will be used as a template for the head of the clauses learned by the programs. For each variable that appears in the template a type has to specified in the list *Types*. A declaration for the target predicate `illegal/6` can look like this:

```
target(illegal(WKf,WKr,WRf,WRr,BKf,BKr),
   [WKf-file,WKr-rank,WRf-file,WRr-rank,BKf-file,BKr-rank]).
```

Two types, `file` and `rank`, specify the two different coordinates that are used to annotate a square on a chess board.

Background knowledge: Relational learning algorithms also have to know which relations they can use for the construction of a definition for the target concept. The literals available for this purpose can be specified with `known_literal/4`:

`known_literal(`*Literal,Types,Modes,Symmetries*`).`

Literal is used to specify a template for a literal of the background knowledge and *Types*, as in `target/2`, lists the types of its arguments. In addition one can also specify *Modes* — `+` means that only old variables can appear for the corresponding argument, while `-` also allows a new variable to be introduced at this place — and *Symmetries*. The background knowledge for the KRK domain used in most of the experiments of this thesis was specified as follows:

```
known_literal( ==(X,Y),        [X-FR,Y-FR], [+,+], [X-Y]).
known_literal( adjacent(X,Y), [X-FR,Y-FR], [+,+], [X-Y]).
known_literal( <(X,Y),         [X-FR,Y-FR], [+,+], []   ).
```

Note that the types of the relations are specified by a variable. This means that any type — `rank` or `file` — can appear at this place. However, the use of the same variable for both arguments ensures that both arguments must be of the same type and thus unnecessary literals that e.g. try to compare ranks to files will not be generated. The mode declaration specifies that no new variables can be introduced, while the symmetry declaration prevents that an equality or adjacency relation with two specific arguments is considered again with the arguments swapped. This, of course, would not make sense with the `>` relation which is not symmetric.

Constants: In all background relations it is possible to specify places where not only variables, but also constants can appear. This is done by appending an identifier to the type, changing the mode declaration to `=` and adding an appropriate `constants/3` statement:

`constants(`*Type,Identifier,Constants*`).`

This was not used in the KRK domain experiments, but was extensively used in the experiments in propositional domains (section 7.3). For an example we look at the declarations

```
known_literal( ==(X,Y),[X-file,Y-file+1], [+,=], [X-Y]).
constants( file, 1, [a,b,c,d,e,f,g,h]).
```

This pair of statements would specify that conditions that check for the value of the file of a certain piece — like `KRf == c` — are admissible.

In addition to listing the admissible constants, there are two special declarations that can appear as the *Constants* argument: The keyword `all` forces the program to collect all constants that appear in the training data at all arguments of *Type*. The keyword `dynamic` recalculates this list during learning from the training set of each particular literal. `dynamic` therefore may often try fewer constants, but has the overhead of recalculating the constants each time the literal is considered as a possible condition.

The above declarations for the background knowledge are the same for all of the implemented programs:

- `foil(`*Predicate,Theory*`)`: a simple version of FOIL without a stopping criterion.
- `fossil(`*Predicate,Theory*`)`: FOSSIL as described in section 3.2. The value of the cutoff parameter can be changed with `set_cutoff(`*Cutoff*`)`. The default value is `0`.
- `fossil_rep(`*Predicate,SplitRatio,Theory*`)`: REP as described in section 4.1 using a starting theory learned by FOSSIL with *Cutoff* $= 0.0$.
- `foil_rep(`*Predicate,SplitRatio,Theory*`)`: REP using the simple version of FOIL for generating the starting theory.
- `foil_grow(`*Predicate,SplitRatio,Theory*`)`: GROW as described in section 4.3 using FOIL for learning an initial theory.
- `foil_rep_and_grow(`*Predicate,SplitRatio,Theory1,Theory2*`)`: generates a starting theory and prunes the same theory first using REP and then using GROW.

- `fossil_series(`*Predicate,MinCutoff,Theories*`)`: Generates all theories that Fossil can learn with any setting of the cutoff parameter in the range from 1.0 down to *MinCutoff* (see section 5.1).

- `td_pruning(`*Predicate,MinCutoff,Theories*`)`: TDP as described in section 5.2.

- `irep(`*Predicate,SplitRatio,Theory*`)`: I-REP as described in section 6.1.

- `irep2(`*Predicate,SplitRatio,Theory*`)`: an alternative version of I-REP that was used in some of the experiments.

The programs were developed for experimental purposes only and are not very user-friendly, nicely programmed or sufficiently documented. If the reader is not deterred by this, s/he can obtain the latest version of these programs from the author upon request.[1]

[1]Send e-mail to `juffi@ai.univie.ac.at`.