Artificial Intelligence Review **13:** 3–54, 1999.

Separate-and-Conquer Rule Learning

JOHANNES FÜRNKRANZ*
Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Wien, Austria;
*E-mail: juffi@ai.univie.ac.at; *Current address: Carnegie Mellon University,*
Computer Science Department, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891, U.S.A.;
E-mail: juffi@cs.cmu.edu

Abstract. This paper is a survey of inductive rule learning algorithms that use a *separate-and-conquer* strategy. This strategy can be traced back to the AQ learning system and still enjoys popularity as can be seen from its frequent use in inductive logic programming systems. We will put this wide variety of algorithms into a single framework and analyze them along three different dimensions, namely their search, language and overfitting avoidance biases.

Key words: covering, inductive logic programming, inductive rule learning

1. Introduction

In this paper we will give an overview of a large family of symbolic rule learning algorithms, the so-called *separate-and-conquer* or *covering* algorithms. All members of this family share the same top-level loop: basically, a separate-and-conquer algorithm searches for a rule that explains a part of its training instances, separates these examples, and recursively conquers the remaining examples by learning more rules until no examples remain. This ensures that each instance of the original training set is covered by at least one rule.

It is well-known that learning algorithms need an appropriate bias for making an "inductive leap". Mitchell (1980) defined *bias* as

> any basis for choosing one generalization over another, other than strict consistency with the observed training instances.

A learning algorithm can thus be characterized with the biases it employs. While the basic top-level loop is invariant for all algorithms of the separate-and-conquer family, their method for learning single rules can vary considerably for different members of this family. We will characterize separate-and-conquer algorithms along three dimensions:

Language Bias: The search space for a learning algorithm is defined by its hypothesis language. Certain concepts may not be expressible or may have an awkward representation in certain hypothesis languages. An appropriate choice of the hypothesis language thus constitutes an important form of bias.

Search Bias: The term search bias refers to the way the hypothesis space is searched. It includes the search algorithm (hill-climbing, beam search, etc.), its search strategy (top-down or bottom-up), and the search heuristics that are used to evaluate the found hypotheses.

Overfitting Avoidance Bias: Many algorithms employ heuristics for avoiding overfitting of noisy data. They may prefer simpler rules to more complex rules, even when the accuracy of the simpler rules on the training data is lower, in the hope that their accuracy on unseen data will be higher. Such a bias for simpler rules has recently been termed *overfitting avoidance bias* (Schaffer 1993; Wolpert, 1993).[1]

We will start with the description of a generic separate-and-conquer algorithm that can be instantiated to various existing (and new) learning algorithms by specifying different biases.

2. The Separate-and-Conquer Strategy

The separate-and-conquer strategy has its origins in the AQ family of algorithms (Michalski 1969) under the name *covering* strategy. The term *separate-and-conquer* has been coined by Pagallo and Haussler (1990) because of the way of developing a theory that characterizes this learning strategy: learn a rule that covers a part of the given training set (the *separate* part) and recursively learn another rule that covers some of the remaining examples (the *conquer* part) until no examples remain. The terminological choice is a matter of personal taste, both terms can be found in the literature. We will use the term *separate-and-conquer* learning.

Separate-and-conquer algorithms have been developed for a variety of different learning tasks. Figure 1 shows a collection of well-known algorithms grouped by the types of concepts they learn. The classical separate-and-conquer algorithms induce *rule sets* for attribute-value based concept learning problems. Variants generalize this approach to inducing ordered rule sets (also called *decision lists*) for multi-class problems. Problems with continuous class variables can be solved by learning *regression rules*. Research in the field of *inductive logic programming* (Bergadano and Gunetti 1995; Muggleton 1992; De Raedt 1995) has developed a variety of separate-and-conquer algorithms that can solve the above tasks in a richer representation language by inducing

propositional rule sets (DNF):

AQ (Michalski, 1969), AQ15 (Michalski et al., 1986), AQ17 (Bloedorn & Michalski, 1991; Wnek & Michalski, 1994; Bloedorn et al., 1993), PRISM (Cendrowska, 1987), SWAP-1 (Weiss & Indurkhya, 1991, 1993a), POSEIDON (Bergadano et al., 1992), PFOIL (Mooney, 1995), JoJo (Fensel & Wiese, 1993; Wiese, 1996) ATRIS (Kononenko & Kovačič, 1992; Mladenić, 1993), CIPF (Pfahringer, 1994a, 1994b), DLG (Webb, 1992; Webb & Agar, 1992), CLASS (Webb, 1993), SIA (Venturini, 1993), GROW (Cohen, 1993), RIPPER (Cohen, 1995), BEXA (Theron & Cloete, 1996)

CNF:

PFOIL-CNF (Mooney, 1995), ICL (De Raedt & Van Laer, 1995)

decision lists:

(Rivest, 1987), CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991), CN2-MCI (Kramer, 1994), GREEDY3 (Pagallo & Haussler, 1990), PREPEND (Webb & Brkič, 1993; Webb, 1994), FOIDL (Mooney & Califf, 1995)

logic programs:

INDUCE (Michalski, 1980), FOIL (Quinlan, 1990; Quinlan & Cameron-Jones, 1995a), mFOIL (Džeroski & Bratko, 1992), SFOIL (Pompe et al., 1993), MILP (Kovačič, 1994b), HYDRA (Ali & Pazzani, 1993), CHAMP (Kijsirikul et al., 1991, 1992), FOSSIL (Fürnkranz, 1994b), REP (Brunk & Pazzani, 1991), TDP (Fürnkranz, 1994c), I-REP (Fürnkranz & Widmer, 1994), I^2-REP (Fürnkranz, 1995), MDL-FOIL (Quinlan, 1994), MARKUS (Grobelnik, 1992), ML-SMART (Bergadano & Giordana, 1988), GA-SMART (Giordana & Sale, 1992), SMART+ (Botta & Giordana, 1993), FOCL (Pazzani & Kibler, 1992), GRENDEL (Cohen, 1994), GOLEM (Muggleton & Feng, 1990), NINA (Adé et al., 1995), PROGOL (Muggleton, 1995)

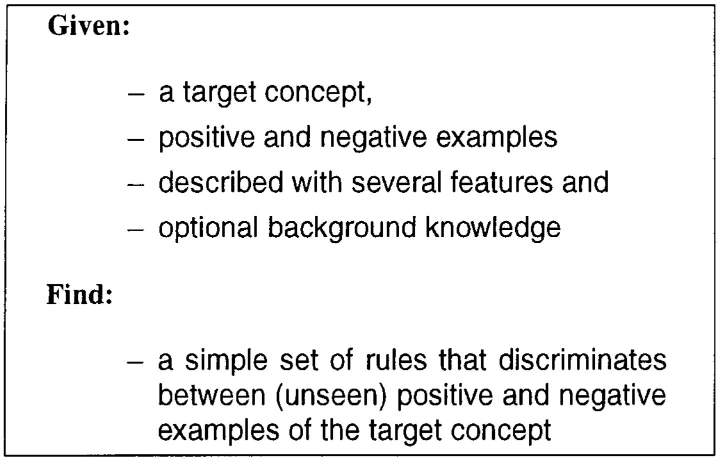
functional relations:

FILP (Bergadano & Gunetti, 1993), FFOIL (Quinlan, 1996)

regression rules:

IBL-SMART (Widmer, 1993), RULE (Weiss & Indurkhya, 1993b, 1995), FORS (Karalič, 1995)

Figure 1. Separate-and-conquer algorithms grouped by concept type.



Given:

- a target concept,
- positive and negative examples
- described with several features and
- optional background knowledge

Find:

- a simple set of rules that discriminates between (unseen) positive and negative examples of the target concept

Figure 2. The inductive concept learning problem.

logic programs for classification or for predicting output values in *functional relations*.

In this paper we will mainly concentrate on *concept learning* tasks, as they seem to be the most common application of separate-and-conquer algorithms in the literature. We will start with a brief formalization of this learning problem (section 2.1), proceed to formalize a generic separate-and-conquer algorithm that can address this problem (section 2.2) and briefly discuss important variants that handle related learning problems (section 2.3).

2.1. *The learning problem*

Figure 2 shows the inductive concept learning problem. Given are positive and negative examples of a target concept, described with a fixed number of attributes, maybe enriched with additional background knowledge. The goal of the algorithm is to discover a description for the target concept in the form of explicit rules formulated in terms of tests for certain values of the attributes or the background knowledge. The resulting rule set should be able to correctly recognize instances of the target concept and discriminate them from objects that do not belong to the target concept.

There are various approaches for tackling this problem. The most commonly used alternative is decision tree learning via the *divide-and-conquer* strategy (Quinlan 1986). Much of the popularity of decision tree learning stems from its efficiency in learning and classification (Boström 1995). Moreover, decision trees can easily be turned into a rule set by generating one rule for each path from the root to a leaf. However, there are several aspects which make rule learning via the separate-and-conquer strategy attractive:

- Decision trees are often quite complex and hard to understand. Quinlan (1993) has noted that even pruned decision trees may be too cumbersome, complex, and inscrutable to provide insight into the domain at hand and has consequently devised procedures for simplifying decision trees into pruned production rule sets (Quinlan 1987a, 1993). Additional evidence for this comes from Rivest (1987) who shows that decision lists (ordered rule sets) with at most *k* conditions per rule are strictly *more expressive* than decision trees of depth *k*. A similar result has been proven in (Boström 1995).
- The restriction of decision tree learning algorithms to non-overlapping rules imposes strong constraints on learnable rules. One problem resulting from this constraint is the *replicated subtree problem* (Pagallo and Haussler 1990): It often happens that identical subtrees have to be learned at various placed in a decision tree, because of the fragmentation of the example space imposed by the restriction to non-overlapping rules. Separate-and-conquer learners do not make such a restriction and are thus less susceptible to this problem. An extreme example for this problem can be found in (Cendrowska 1987), where it is shown that the minimal decision tree for the concept `x` defined as

  ```
  x :– a = 3, b = 3.
  x :– c = 3, d = 3.
  ```

  has 10 interior nodes and 21 leafs assuming that each attribute `a` ... `d` can be instantiated with three different values.
- Propositional separate-and-conquer algorithms extend naturally to the first-order *inductive logic programming* framework, where the goal is basically the induction of a PROLOG program. First-order background knowledge can also be used for decision-tree induction (Watanabe and Rendell 1991; Lavrač, Džeroski and Grobelnik 1991; Kramer 1996; Blockeel and De Raedt 1997), but once more, Watanabe and Rendell (1991) have noted that first-order decision trees are usually more complex than first-order rules.

In particular the last issue has contributed to a recent revival of separate-and-conquer learning strategies, which has been a source of motivation for this systematic overview.

2.2. *The algorithm*

Figure 3 shows a simple separate-and-conquer algorithm, which has been implemented in a more or less equivalent form in the PRISM learning system

```
procedure SIMPLESEPARATEANDCONQUER(Examples)

Theory = $\emptyset$
while POSITIVE(Examples) $\neq \emptyset$
    BestRule = {true}
    Rule = BestRule
    Covered = COVER(Rule,Examples)
    while NEGATIVE(Covered) $\neq \emptyset$
        for Condition $\in$ Conditions
            Refinement = Rule $\cup$ Condition
            if PURITY(Refinement,Examples) >
                    PURITY(BestRule,Examples)
                BestRule = Refinement
        Rule = BestRule
    Theory = Theory $\cup$ Rule
    Examples = Examples $\setminus$ Covered
return(Theory)
```

Figure 3. A simple separate-and-conquer algorithm.

(Cendrowska 1987). It starts with an empty theory and successively adds rules to it until all positive examples are covered. The learning of single rules starts with a rule whose body is always true. As long as its still covers negative examples the current rule is specialized by adding conditions to its body. Possible conditions are tests on the presence of certain values of various attributes. In order to move towards the goal of finding a rule that covers no negative examples, the algorithm selects a test that optimizes the purity of the rule, i.e., a test that maximizes the percentage of positive examples among all covered examples. When a rule has been found that covers only positive examples, all covered examples are removed and the next rule is learned from the remaining examples. This is repeated until no examples remain. Thus it is ensured that the learned rules together cover all of the given positive examples (*completeness*) but none of the negative examples (*consistency*).

All separate-and-conquer algorithms share the basic structure of this simple algorithm. However, many learning tasks require modifications of this procedure. For example, if the data are noisy, the induction of complete and consistent theories can lead to overfitting. Thus many algorithms relax this constraint and use stopping criteria or post-processing methods to be able to learn simpler theories, which are not complete and consistent but often more predictive on unseen data. Other algorithms replace the top-down search of the inner `while`-loop with a bottom-up search, where rules are successively generalized starting with a most specific rule (e.g. consisting of one of the positive examples itself). Yet other algorithms do not use hill-climbing, but employ less myopic search algorithms like beam search or best-first search. Another issue is the type of conditions that can be used in the formulation of the hypotheses. The algorithm of Figure 3 can only formulate rules in propositional logic, but research in inductive logic programming has developed algorithms that can learn rules in first-order logic.

Figure 4 shows a generic separate-and-conquer rule learning algorithm that calls various subroutines which can be used to instantiate the generic algorithm into specific algorithms known from the literature. SEPARATEANDCONQUER starts with an empty theory. If there are any positive examples in the training set it calls the subroutine FINDBESTRULE for learning a rule that will cover a subset of the positive examples. All covered examples are then separated from the training set, the learned rule is added to the theory, and another rule is learned from the remaining examples. Rules are learned in this way until no positive examples are left or until the RULESSTOPPINGCRITERION fires. Often the resulting theory undergoes some POSTPROCESSing.

The procedure FINDBESTRULE searches the hypothesis space for a rule that optimizes a given quality criterion defined in EVALUATERULE. The value of this heuristic function usually is the higher the more positive and the less negative examples are covered by the candidate rule. FINDBESTRULE maintains *Rules*, a sorted list of candidate rules, which is initialized by the procedure INITIALIZERULE. New rules will be inserted in appropriate places (INSERTSORT), so that *Rules* will always be sorted in decreasing order of the heuristic evaluations of the rules. At each cycle, SELECTCANDIDATES selects a subset of these candidate rules, which are then refined using REFINERULE.[2] Each refinement is evaluated and inserted into the sorted *Rules* list unless the STOPPINGCRITERION prevents this. If the evaluation of the *NewRule* is better than the best rule found previously, *BestRule* is set to *NewRule*. FILTERRULES selects the subset of the ordered rule list that will be used in subsequent iterations. When all candidate rules have been processed, the best rule will be returned.

Different choices of these functions allow the definition of different biases for the separate-and-conquer learner. The search bias is defined by the choice of a search strategy (INITIALIZERULE and REFINERULE), a search algorithm (SELECTCANDIDATES and FILTERRULES), and a search heuristic (EVALUATERULE). The refinement operator REFINERULE constitutes the language bias of the algorithm. An overfitting avoidance bias can be implemented via the two stopping criteria and/or in a post-processing phase.

As an example, assume we want to instantiate the generic algorithm into the simple algorithm of Figure 3. SIMPLESEPARATEANDCONQUER searches the hypothesis space in a top-down fashion. INITIALIZERULE will thus return

```
procedure SeparateAndConquer(Examples)

Theory = $\emptyset$
while Positive(Examples) $\neq \emptyset$
    Rule = FindBestRule(Examples)
    Covered = Cover(Rule,Examples)
    if RuleStoppingCriterion(Theory,Rule,Examples)
        exit while
    Examples = Examples $\setminus$ Covered
    Theory = Theory $\cup$ Rule
Theory = PostProcess (Theory)
return(Theory)


procedure FindBestRule(Examples)

InitRule = InitializeRule(Examples)
InitVal = EvaluateRule(InitRule)
BestRule = <InitVal,InitRule>
Rules = {BestRule}
while Rules $\neq \emptyset$
    Candidates = SelectCandidates(Rules, Examples)
    Rules = Rules $\setminus$ Candidates
    for Candidate $\in$ Candidates
        Refinements = RefineRule(Candidate, Examples)
        for Refinement $\in$ Refinements
            Evaluation = EvaluateRule(Refinement, Examples)
            unless StoppingCriterion(Refinement, Examples)
                NewRule = <Evaluation,Refinement>
                Rules = InsertSort(NewRule, Rules)
                if NewRule > BestRule
                    BestRule = NewRule
    Rules = FilterRules(Rules, Examples)
return(BestRule)
```

Figure 4. A generic separate-and-conquer rule learning algorithm.

the most general rule, i.e., the rule with the body {true}. RefineRule will specialize a given rule by adding a condition to it. The rules will be evaluated by the percentage of covered examples that are positive, i.e., EvaluateRule will implement the Purity subroutine used in Figure 3. FilterRules will only let the best refinement pass to the next iteration, so that SelectCandidates will always have only one choice. Together these two procedures implement the hill-climbing search. As only the first (and best) element of the sorted list of all refinements will be used, this part of the code is equivalent to the corresponding part in SIMPLESEPARATEANDCONQUER. Both stopping criteria will always be false, and the learned rules will not be post-processed. Together, these choices instantiate SEPARATEANDCONQUER into SIMPLESEPARATEANDCONQUER.

2.3. *Rule ordering*

Note that we assume a binary classification task: the goal of the induced concept is to discriminate between positive and negative examples of a target concept. Many separate-and-conquer learning algorithms, in particular the algorithms used in inductive logic programming, are based on this assumption. In this case, the order in which the rules are used for classification does not matter, because the rules only describe one class, the positive class. Negative examples will be classified using *negation as failure*, i.e., when no rule fires for a given example, it will be classified as negative. This is equivalent to assuming a default rule for the negative class at the end of an ordered rule list.

However, many real world problems are concerned with multi-valued or even continuous class variables. In such *multi-class* or *regression problems* the order of the rules is very important, because each example could be covered by several rules that make different predictions. A different rule order can thus make a different prediction for an example. This problem is known as the *overlap problem*. Segal and Etzioni (1994) address it by allowing only *homogeneous rules*. A homogeneous rule is a rule with the property that all its specializations have the same heuristic evaluation as the rule itself. Segal and Etzioni (1994) have shown that for each decision list there exists a logically equivalent homogeneous decision list. The latter has the advantage that the evaluation of its rules does not change with their position in the list, but the rule can be considerably more complex.

Theories that impose a fixed evaluation order on their rules as commonly referred to as *decision lists* (Rivest 1987). They can be viewed as a PROLOG program where each rule ends with a cut (!) (Mooney and Califf 1995). CN2 (Clark and Niblett 1989) is able to handle multi-class problems by learning decision lists. For this purpose, it uses an evaluation function that gives preference to class distributions where examples of one class are dominating (see section 4.3). Each learned rule predicts the class that is dominant among the examples it covers. Learning stops when *all* examples are covered by at least one rule. To handle clashes (when multiple rules fire) CN2 orders rules in the order they have been learned. This seems to be a natural strategy, because most search heuristics tend to learn more general rules first. However, it has been pointed out in (Webb and Brkič 1993) that prepending a new rule to the previously learned rules can produce simpler concepts. The intuition behind this argument is that there are often simple rules that would cover many of the positive examples, but also cover a few negative examples that have to be excluded as exceptions to the rule. Placing such a simple general rule near the end of the rule list allows to handle these exceptions with rules that are placed before the general rule thus keeping it simple. This hypothesis has been empirically confirmed in (Webb 1994) and (Mooney and Califf 1995).

Another method for inducing multi-class concepts is to learn a separate concept description for each class, taking all examples of other classes as negative examples for the class to learn. Then the program assigns a weight to each rule according to some heuristic. Examples will be classified with the class predicted by the rule with the highest weight. This method is used in HYDRA (Ali and Pazzani 1993) where the *ls*-content of a rule (see section 4.3) is used as a weighting heuristic. In AQ-15 (Michalski et al. 1986) each rule is weighted by the percentage of positive examples in the set of examples covered by it. The weights of rules of the same class are combined to a weight for the entire class and the class with the highest weight will be returned. Quinlan (1987a) sorts the rules by their *advantage*, i.e., by the number of examples that would be erroneously classified after deleting the rule. In later work, Quinlan (1993) replaced this algorithm by a scheme that groups the rules according to the classes they predict and orders these groups using a heuristic based on the *minimum description length principle* (Rissanen 1978). This increases the comprehensibility of the learned concepts. Pazzani, Merz, Murphy, Ali, Hume and Brunk (1994) present an algorithm that orders a set of learned rules with respect to a minimization of expected misclassification cost.

Similar problems have to be tackled in first-order *function learning* (Quinlan 1996), where the learned rules do not check the validity of a given ground instance but derive ground values for its unbound variables. Different rules might derive different values and some ordering of the rules is needed to handle such clashes. Finally, the ordering of the rules is also important when learning *recursive concepts*, where it has to be ensured that the base case of the recursion is placed before the recursive rule as, e.g., in FOIL (Quinlan 1990).

In the remainder of this paper we will neglect the aspect of rule ordering and simply assume that rules are used in the same order in which they are learned.

3. Language Bias

Before the user invokes a certain learning algorithm he already has to make a choice of a suitable representation language for the hypotheses to learn. This choice naturally has a considerable influence on the result of the learning procedure (Mitchell 1980). Although most options for hypothesis languages that we will discuss in this section are also available or could probably be adapted for other learning approaches like divide-and-conquer decision tree learning algorithms, many of them have been developed for separate-and-conquer rule learning systems. In particular in inductive logic programming, the ability to restrict the possibly infinite space of first-order logic clauses[3] proved to be crucial. Flexible mechanisms that use an explicit representation of the language bias are particularly useful.

We will first discuss static approaches, where the user has to fix the language bias before the induction task, and then shortly summarize several techniques that allow a learning system to autonomously change the langauge bias when it proves to be inadequate.

3.1. *Static language bias*

There is a wide variety of condition types that can be made available for a classification learning algorithm. The spectrum reaches from simple selectors that relate the value of an attribute to one of its domain values (section 3.1.1) to rule models that offer a flexible way to restrict the huge search space for first-order classification rules (section 3.1.5).

3.1.1. *Selectors*

Selectors are the most commonly used form of representation language in inductive learning. The term *selector* was introduced by Michalski (1973). A selector is a condition of the form

$$\mathtt{A_i\ \#\ constant}$$

where an example's value of a certain attribute $\mathtt{A_i}$ is related to a constant value of its domain via relations like $=$, $>$, or $<$. The equality relations are used for symbolic attributes, while the inequalities are more often used for numeric, in particular for continuous attributes. Often the negations of these relations (i.e., $\neq$, $\leq$, $\geq$) are also available.

Algorithms of the AQ-family (Michalski 1980; Michalski et al. 1986; Bergadano et al. 1992) are able to extend these elementary selectors to using attribute sets (*internal conjunctions*), value sets (*internal disjunctions*), and intervals (*range operators*). Moreover, they can also make use of *tree-structured attributes*. A description of these extensions can be found in Michalski (1980).

3.1.2. *Literals*

Research in *inductive logic programming (ILP)*, in particular on FOIL and related systems (Quinlan and Cameron-Jones 1995a), has produced algorithms for solving classification problems in first-order logic. In these cases the target concept is usually represented as a PROLOG relation in the form

$$\texttt{concept\_name}(\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_n)$$

where `concept_name` is an *n*-ary PROLOG predicate denoting the concept to learn. Its *n* arguments represent the attributes that have been measured for this concept. Thus each propositional learning problem of the type discussed in section 3.1.1 can be transformed into a first-order learning problem by turning examples into PROLOG literals using the class variable as the predicate symbol and the attributes' values as arguments. All selectors discussed in the previous sections can be trivially used as possible conditions in the learned rules.

However, in addition to these selectors, the user can also specify relations between several attributes as background knowledge in the form of PROLOG predicates. These (or their negations) can then be used as additional conditions in the final rules. A typical example is the king-rook-king (KRK) chess endgame learning task (Muggleton et al. 1989) that has developed into a standard benchmark problem for ILP algorithms. The goal concept is to recognize illegal white-to-move positions in this endgame. These can be positions where two or more pieces are on the same square or positions where the black king is in check. The target predicate is `illegal` (`WKf, WKr, WRf, BKf, BKr`) where the six arguments specify the file and row coordinates of the squares of three pieces. When the hypothesis language is restricted to the use of selectors, no meaningful concept representation can be learned in this task (Muggleton et al. 1989). However, if additional background relations like `adjacent/2`, $<$`/2`, or $=$`/2` can be used as possible conditions, the final rules are able to check whether two pieces are on the same file or rank, on adjacent squares, etc. With the help of these predicates, simple rules can be learned for this concept. Note that here the equality (and inequality) relations are used in a different way as in section 3.1.1: while selectors can only compare the value of one attribute with a certain domain constant, general literals are able to compare the values of two attributes with each other.

In many cases these background relations can not only make use of the available attributes, but also introduce new variables that can be used for subsequent tests in this rule. For example, the `adjacent/2` relation can be added as a condition with one old and one new variable. The condition `adjacent` (`WKf, X`) binds the new variable `X` to one of the files that are adjacent to the file WKf on which the white king is located. Subsequent conditions of this rule can then use the variable X in their arguments. New variables also allow the construction of *recursive rules*.

A problem often caused by conditions introducing new variables is that they have no discriminatory power. The literal adjacent (WKf, X) is true for KRK positions, be they legal or illegal, because for every file WKf, on which the white king can possibly be placed, there will always be a file X adjacent to WKf. Top-down hill-climbing algorithms that learn rules by adding one condition at a time will thus attribute low importance to these conditions, because they appear to be unable to discriminate between positive and negative examples. This is also the main reason why propositional problems are usually translated in the way we specified above. Alternative methods, like using a target relation with one argument – an index to the examples – and one additional binary background relation for each attribute that takes the index and the attribute's value as arguments, may easily run into the above problems (see also section 4.3.6).

3.1.3. *Syntactic restrictions*

Considering general first-order literals as conditions in the body of a rule may lead to huge, even infinite search spaces. The original version of FOIL (Quinlan 1990) for example allowed all possible combinations of variables in the arguments of the used literals. Thus the search space grew exponentially with the number of attributes in the data set, which severely handicapped the program in terms of both efficiency and accuracy.

One observation that has been made is that in most applications the available variables have different types. Similarly, many of the available background literals only make sense when their arguments are instantiated with variables of certain types. For example, list-processing predicates need list-valued variables, arithmetic operations need numbers and so forth. These *argument types* can be specified in advance so that attributes are only used in appropriate places. The places where new variables can appear can be specified by so-called *mode declarations*. Arguments where only bound variables can appear are called *input variables*, whereas arguments where new variables can be used as well are called *output variables*. Type and mode declarations have already been used in early ILP systems such as MIS (Shapiro 1981). Many predicates are also symmetric in some of their input arguments, i.e., they will produce the same result no matter in which order these arguments are given. Such *symmetries* can also be exploited by various programs.

The programs used in (Fürnkranz 1994a, 1997) for example can specify background knowledge with statements like

```
known_literal(adjacent(X,Y), [X-file, Y-file], [+,+], [X—Y]).
```

This declaration specifies that the literal `adjacent/2` can be used as a rule condition with two input variables (+) of the type `file`. The literal is symmetric in the variables `X` and `Y`. A more detailed description of this syntax can be found in the appendix of (Fürnkranz 1994a). Similar declarations can be made in *m*FOIL (Džeroski and Bratko 1992), recent versions of FOIL (Quinlan and Cameron-Jones 1995a), PROGOL (Muggleton 1995), and others. Restrictions of this type can significantly reduce the hypothesis space. Similar effects can also be achieved by restricting the domains of the selectors. Several systems allow to declare certain values of an attribute as constants that can be used in specified places like the right-hand side of a selector. For example, it may be useful to specify that selectors for list-valued attributes will only be used with the empty list, but not with arbitrary lists (Quinlan and Cameron-James 1995a).

Many rule learning systems also place upper bounds on the complexity of the learned rules in order to restrict the search space. PROGOL, e.g., has a parameter that can be used to specify a *maximum rule length*. FOIL allows the specification of a *maximum variable number* and a *maximum variable depth*.[4] It also makes sense to allow only *linked* literals, i.e., literals that share variables with the head of the clause or with another linked literal (Helft 1989). Severe restrictions have to be used when learning recursive programs in order to avoid infinite recursions. This problem has been discussed at length in (Cameron-Jones and Quinlan 1993).

3.1.4. *Relational clichés*

In some cases it may prove useful for top-down algorithms to not only add one condition at a time, but to add a conjunction of conditions at once. The classical example is to avoid myopic behavior in hill-climbing algorithms. The simplest approach to achieve this goal is to declare specific conjunctions as if they were single background literals. For example in the `known_literal/4` declaration shown above, the first argument does not have to be a single literal; it could also be a conjunction that for example consists of one literal that introduces a new variable (a *generator*) and one condition that uses this variable. Such declarations can be made in the programs described in (Fürnkranz 1994a, 1997; Kramer 1996; Blockeel and De Raedt 1997).

A similar idea has been explored further in (Silverstein and Pazzani 1991). *Relational clichès* are conjunctions of the type explained above with the difference that the predicate does not have to be exactly specified. Instead the user can provide a place-holder that stands for a certain class of predicates. For this purpose, the predicates in the background knowledge can often be organized into a symbol hierarchy, in which the leaves are the predicates and the interior nodes represent certain predicate classes that can be used as *predicate variables*. A typical example – the so-called *threshold comparator clichè* – is a conjunction consisting of a literal that introduces a new measurement and a selector that compares this new variable to a domain value. Silverstein and Pazzani (1993) also demonstrate a method for learning useful clichès from experience. Similar ideas for improving the learning behavior by adding more than one literal at a time can be found in ML-SMART (Bergadano and Giordana 1988) and FOCL (Pazzani and Kibler 1992). These systems allow to replace conditions of a rule with the body of their definitions in the background knowledge.

In a later version of ML-SMART (Bergadano, Giordana and Ponsero 1989) one could also specify so-called *predicate sets*. A predicate set is a set of literals that are known to be relevant for the definition of a certain predicate. Although the precise definition of the predicate is not available, it is known that it will contain a subset of these literals, which will be identified by the program.

3.1.5. *Rule models*

Building upon previous work in ML-SMART, Giordana and Sale (1992) describe a simple method for modeling the entire hypothesis space with a single clause consisting of multiple predicate sets, a so-called *template*. These have later been generalized to *clause sets* (Bergadano and Gunetti 1995), enhanced PROLOG programs, where sets of variables, literals and clauses can appear in the definition. The task of the learner is to find the best rules defined by appropriate subsets of these clause sets.

Similarly, the RDT rule learning system has expanded the relational clichès idea to modeling the entire hypothesis space instead of only certain conjunctions (Kietz and Wrobel 1992). Hierarchically organized predicate variables can be used for writing down *rule models* that can be instantiated by replacing all predicate variables in a rule model with suitable predicates. These rule models can be organized into a lattice that can be efficiently searched in a top-down fashion.

Adé et al. (1995) introduce a framework for comparing certain language bias mechanisms in bottom-up separate-and-conquer learning algorithms. This framework combines the advantages of rule models and clause sets by recognizing that predicate variables implicitly define predicate sets. It has later been generalized into the DLAB declarative langauge bias formalism (Dehaspe and De Raedt 1996), which can also specify the number of items that have to be chosen from an explicitly or implicitly defined predicate set.

The most flexible and most expressive framework for explicitly modeling hypothesis spaces are *antecedent description grammars* (Cohen 1994) as used in the top-down separate-and-conquer learning algorithm GRENDEL. An antecedent description grammar is a context-free grammar whose symbols are logical literals. Its terminal symbols are literals from the background knowledge. The grammar has a designated starting symbol which will be successively expanded using the rules of the grammar. Sentences, i.e., rules consisting only of terminal symbols, can be evaluated as in conventional algorithms. Sentential forms, i.e., rules where not all non-terminal symbols have been expanded into terminals, can be evaluated by replacing non-terminal symbols with a disjunction of all possible terminal symbols that can be derived from them. In many cases non-terminal symbols will expand to a disjunction that will always be true. These special cases can often be efficiently recognized and efficiently computed.

Although antecedent description grammars are certainly the most flexible and most expressive rule modeling technique, they are harder to understand than other rule models. While it is usually obvious whether a certain clause can be generated from a certain clause set, it is considerably harder to determine whether a certain clause can be derived from an antecedent description grammar.

More details on declarative bias formalisms along with an evaluation of their respective strengths and weaknesses can be found in (Nédellec, Rouveirol, Adé, Bergadano and Tausend 1996; Adé et al. 1995; Bergadano and Gunetti 1995).

3.2 *Dynamic language bias*

While the approaches discussed in the last section offer a considerable flexibility for defining a language bias, they are all static. Once a language bias is defined it cannot be changed without user intervention. In this section we will discuss approaches that can dynamically adjust their language bias to the problem at hand via a so-called *bias shift* (Utgoff 1986): If the hypothesis space does not include an acceptable concept description, these techniques allow the learner to shift to a weaker bias, i.e., they allow to express concepts in a more expressive representation language.

3.2.1. *Language hierarchies*

A simple approach for implementing a procedure for dynamically shifting the language bias has been first proposed for the first-order theory revision system CLINT (De Raedt 1992). CLINT makes use of a predefined (possibly infinite) series of hypothesis languages with increasing expressive power. Whenever it cannot learn a complete and consistent concept with the given language it tries again with the next, more expressive language (De Raedt and Bruynooghe 1990). Although CLINT is not a separate-and-conquer learning algorithm, this technique could be easily adapted for members of this family.

In fact this idea has already been used in the NINA separate-and-conquer framework for analyzing bottom-up inductive logic programming algorithms (Adé et al. 1995). Furthermore, techniques for explicitly modeling the langauge bias (see section 3.1.5) could be conveniently used for defining a series of hypothesis languages with increasing expressiveness like the ones used in CLINT. Other separate-and-conquer learning systems like FOIL or PROGOL have parameters for controlling syntactic aspects of the hypothesis language (see section 3.1.3) that could be systematically varied to allow expressing concepts in more complex languages. Kohavi and John (1995) describe an approach how such parameters could be automatically adjusted.

3.2.2. *Constructive induction*

In its original use (Utgoff 1986) the term *bias shift* referred to the process of automatically extending the representation language by constructing new features, a process that is commonly known as *constructive induction* (Matheus 1989). Contrary to the approaches of section 3.2.1 where the user has to predefine a series of hypothesis languages, constructive induction techniques can automatically extend the representation language by constructing new useful features or predicates on the basis of the given information. This will not necessarily extend the space of representable concepts, but adjusting the vocabulary to the task at hand may make the learner's task easier.

The simplest approach to constructive induction with separate-and-conquer learning algorithms is to apply a predefined set of arithmetic and logical operators to certain attributes and compute new attributes from them. For example AQ17-DCI (Bloedorn and Michalski 1991) – and to some extent AQ15 (Michalski et al. 1986) – can compute equality and inequality relations, use addition and multiplication operators, and can determine optima, averages and frequencies in sets of features, reminiscent of some ideas previously used in the BACON discovery system (Langley, Simon and Bradshaw 1987). The generated attributes are then evaluated with an *attribute quality function* which basically computes the number of attribute values that only occur in instances of single target classes.

However, a better evaluation for the quality of generated attributes might result from an analysis of the performance of the learning algorithm that uses the new features. Many approaches to constructive induction implement this idea with a so-called *wrapper* approach.[5] They use a stand-alone induction algorithm and wrap around it an algorithm that analyzes the performance of the induction algorithm by estimating predictive accuracies, looking for co-occurrences of certain conditions in the rules etc. Based on these observations, the wrapper changes the input representation or certain parameters of the learning algorithm to improve the result. The prototypical system for this approach is the AQ17-HCI algorithm (Wnek and Michalski 1994), which incorporates several constructive induction operators based on ideas used earlier in the INDUCE system (Michalski 1980). AQ17-HCI scans the rules generated by the AQ15 induction algorithm (Michalski et al. 1986) for patterns of co-occurring values of a single attribute, co-occurring conjunctions, or even subsets of the induced rule set that have a high *pattern strength*. Pattern strength is evaluated by computing the ratio of positive examples over negative examples covered by the pattern (with the possible addition of a factor proportional to the number of positive examples uniquely covered by this pattern). Patterns that have a high strength will be turned into additional attributes. Feature subset selection and discretization algorithms can also be cast into this framework of constructive induction as a bias shift operation. AQ17-MCI (Bloedorn et al. 1993) integrates the approaches taken by AQ17-DCI and AQ17-HCI into a single system, which is able to learn meta-rules that specify which types of constructive operators are suitable for which type of tasks.

Similar ideas are implemented in CIPF (Pfahringer 1994a, 1994b) which uses a propositional top-down separate-and-conquer algorithm as the basic induction module and in CN2-MCI (Kramer 1994) which introduces a new powerful constructive induction operator for CN2-like algorithms. This operator constructs a new attribute from the cross-product of attributes that often occur together in different rules. Both systems evaluate the quality of generated features by estimating the predictive accuracy of the concepts learned with the new representation and stop when no further improvement is possible.

All approaches mentioned above either invoke constructive induction before the learning process or after a careful analysis of the result of previous learning episodes. A few other approaches that directly invoke constructive induction operators during the learning process have been developed in inductive logic programming. In this context, constructive induction is often called *predicate invention*, as it is not concerned with the construction of new features, but of new predicates.

CHAMP (Kijsirikul et al. 1992) reverts to inventing a new predicate whenever the top-down first-order separate-and-conquer learner CHAM (Kijsirikul et al. 1991) cannot complete the current clause by appending a literal from the background knowledge without excluding all positive examples or without exceeding a certain clause length.[6] In such a case CHAMP constructs a new predicate. The arity of this predicate is determined by a search for a minimal set of variables that is able to form a discriminating relation between the covered positive and negative instances. A definition of the invented predicate is then induced from these examples by recursively calling CHAMP with the invented predicate as the target predicate.

A very similar approach – called *closed-world specialization* – is taken in (Bain 1991) and (Srinivasan, Muggleton and Bain 1992). Here over-general clauses learned by the bottom-up first-order separate-and-conquer learner GOLEM (Muggleton and Feng 1990) are specialized by adding a *negated* new predicate. The intuition behind this approach is that the original rule will already cover more positive than negative examples and the new predicate will only be used for denoting the few exceptions to the rule. A definition for this predicate will be learned by reverting the role of the positive and negative examples covered by the original rule. Note that generalizing (specializing) the definition of the newly invented predicate will specialize (generalize) the rules containing its negation.

4. Search Bias

There are several options for searching a given hypothesis space for acceptable rules. First, different search algorithms can be employed, ranging from a greedy hill-climbing approach to an exhaustive search of the complete hypothesis space (section 4.1). Second, the hypothesis space can be searched in different directions: hypotheses can be refined either by specialization or by generalization (section 4.2). Finally, various heuristic evaluation functions can be used to compare different candidate clauses (section 4.3). All these search options can be implemented with the help of the various subroutines of the procedure FINDBESTRULE (Figure 4).

4.1. *Search algorithms*

This section will discuss various options for searching the space of possible hypotheses that can be implemented into the FINDBESTRULE procedure (see Figure 4). The simplest method would be to systematically generate all possible rules and check each of them for consistency. Rivest (1987) discusses such an algorithm that examines all rules up to a maximum rule length *k*. Whenever it finds a rule that covers only positive examples, it adds this rule to the current rule set, removes the covered examples, and continues the search until all examples are covered. This simple algorithm will find a complete and consistent concept description if there is one in the search space. For a fixed *k* the time complexity of the algorithm is polynomial in the number of tests to choose from.

However, this algorithm was only developed for theoretical purposes. Its severe drawback is that it does not use any heuristics for guiding the search, which makes the algorithm very inefficient. Rivest (1987) notes that it would be advisable for practical purposes to generate the rules ordered by simplicity. Thus simple consistent rules will be found first and will therefore be preferred. For similar reasons most practical algorithms have access to a preference criterion that estimates the quality of a found rule as a trade-off between simplicity and accuracy (see also section 4.3) and use this heuristic to guide its search through the hypothesis space.

4.1.1. *Hill-climbing*

The most commonly used search algorithm in separate-and-conquer learning systems is *hill-climbing*, which tries to find a rule with an optimal evaluation by continuously choosing the refinement operator that yields the best refined rule and halting when no further improvement is possible. Hill-climbing tries to discover a *global* optimum by performing a series of *locally* optimal refinements. The simple algorithm Figure 3 employs such an approach. Hill-climbing can be trivially implemented in the procedure FINDBESTLITERAL of Figure 4 by specifying that FILTERRULES will only return the first and best rule of the list of all refined rules.

The basic problem of this method is its *myopia* due to the elimination of all one-step refinements but one. If a refined rule is not locally optimal, but one of its refinements is the global optimum, hill-climbing will not find it (unless it happens to be a refinement of the local optimum as well). A typical example of this myopia are first-order literals that introduce new variables and have no discriminatory power as discussed in sections 3.1.2 and 4.3.6. Nevertheless, most first-order top-down separate-and-conquer algorithms, like FOIL and its many relatives (Quinlan and Cameron-Jones 1995a), use hill-climbing because of its efficiency.

A simple technique for decreasing search myopia in hill-climbing is to look further ahead. This can be done by choosing the best rule resulting from performing *n* refinement steps at once instead of only 1. This approach has been implemented in the ATRIS rule learning shell (Mladenić 1993). Its major deficiency is its inefficiency, as the search space for each refinement step grows exponentially with *n*.

4.1.2. *Beam search*

Many algorithms try to alleviate the myopic behavior of hill-climbing by using *beam search*. In addition to remembering the best rule found so far, beam search also keeps track of a fixed number of alternatives, the so-called *beam*. While hill-climbing has to decide upon a single refinement at each step, beam search can defer some of the choices until later by keeping the *b* best rules in its beam. It can be implemented by modifying the FILTERRULES procedure of a hill-climbing algorithm so that it will return the *b* best elements of the refinements of the previous beam. Setting $b = 1$ results in hill-climbing.

Beam search effectively maintains hill-climbing's efficiency (reduced by a constant factor), but can yield better results because it explores a larger portion of the hypothesis space. Thus many separate-and-conquer algorithms use beam search in their FINDBESTRULE procedures. Among them are AQ (Michalski et al. 1986), CN2 (Clark and Niblett 1989), *m*FOIL (Džeroski and Bratko 1992), and BEXA (Theron and Cloete 1996). Extreme cases of myopia, like indiscriminative literals, are nevertheless a problem for beam search algorithms.

4.1.3. *Best-first search*

Hill-climbing and beam search algorithms are both limited by myopia that results from their restriction to storing only a fixed number of candidate rules and immediately pruning the others. Best-first search, on the other hand, selects the best candidate rule (SELECTCANDIDATES) and inserts *all* its refinements into the sorted *Rules* list unless unpromising rules are pruned by the STOPPINGCRITERION. FILTERRULES will not remove any rules. Thus best-first search does not restrict the number of candidate rules and may be viewed as a beam search with an infinite beam size $b = \infty$. ML-SMART (Bergadano, Giordana and Saitta 1988) implements such a strategy with several coverage-based pruning heuristics that discard unpromising rules. In (Botta, Giordana and Saitta 1992) this approach has been shown to compare favorably to hill-climbing in an artificial domain.

When no pruning heuristics are used (i.e., STOPPINGCRITERION always returns *false*) the search space will be completely exhausted and it is guaranteed that an optimal solution will be found. Nevertheless, the A* algorithm (Hart, Nilsson and Raphael 1968) allows to prune large portions of the search space without losing an optimal solution. This optimality can be guaranteed if the search heuristic is *admissible*. An admissible search heuristic usually consists of two components, one for evaluating the quality of a rule and one for computing an estimate for the quality of the rule's best refinement. It has to be ensured that the latter estimate will always return an optimistic value, i.e., it has to overestimate the quality of the best refinement of the rule. If this optimistic estimate is already worse than the evaluation of the best rule found so far, the refinements of the current rule need not be further investigated, because their true evaluation can only be worse than the optimistic estimate, which in turn is already worse than the best rule.

The ILP system PROGOL (Muggleton 1995) implements an A* best-first search. It generates the most specific clause in the hypothesis space that covers a randomly chosen example and searches the space of its generalizations in a top-down fashion. It guides this search by a heuristic that computes the number of covered positive examples minus the covered negative examples minus the length of the rule. Incomplete rules are evaluated by subtracting an estimate for the number of literals that are needed to complete the rule and adding the number of covered negative examples to this heuristic value. By doing so it is assumed that the completed rule will cover all positive instances the incomplete rules covers, but none of the negative instances which clearly is an optimistic assumption. With this admissible search heuristic PROGOL performs an exhaustive search through the hypothesis space. However, for longer rules (>4 conditions) this exhaustive search is too inefficient for practical problems (Džeroski et al. 1996). A similar heuristic is used in FOIL's hill-climbing algorithm for safely pruning certain branches of the search space (Quinlan 1990). It might be worth-while to try a best-first search with this heuristic.

However, there is considerable evidence that exhausting a search space can lead to worse results because the chances that rules are encountered that fit the training data by chance are increased. For example, Webb (1993) has used the efficient best-first search algorithms OPUS (Webb 1995) for inducing decision lists in a covering framework and has surprisingly found that the generalizations discovered by the beam search with CN2 are often superior to those found by an exhaustive best-first search. Quinlan and Cameron-Jones (1995b) have later studied this problem of *over-searching* by studying the behavior of a CN2-like algorithm at different beam widths. They have empirically verified in a variety of domains that too large a beam width may lead to worse results. This is also confirmed by early results from statistical learning theory (Vapnik and Chervonenkis 1971, 1981), where it has been observed that larger hypothesis spaces can lead to poorer generalization behavior (see also Saitta and Bergadano (1993)).

4.1.4. *Stochastic search*

Another approach to escape the danger of getting stuck in local optima is to use a *stochastic search*, which can be implemented in the framework of Figure 4 by allowing randomness in the REFINERULE procedure. In that case this procedure will not refine a given rule step by step, but may (with a certain probability) also perform bigger leaps, so that the learner has the chance to focus on entirely new regions of the hypothesis space. In the simplest case, each call to REFINERULE will return a random rule of the search space. More elaborate methods employ randomized generalization and specialization operators. The probability for selecting an operator is often correlated with the quality of the resulting rule, so that better rules are selected with a higher chance, but seemingly bad candidates also get a fair chance to be improved with further refinement steps (*stochastic hill-climbing*). The probability for selecting a suboptimal rule may also decrease over time so that the algorithm will eventually stabilize (*simulated annealing* (Kirkpatrick, Gelatt and Vecchi 1983)).

Kononenko and Kovačič (1992) and Mladenić (1993) present and compare a variety of such algorithms, ranging from an entirely random search to an approach based on Markovian neural networks (Kovačič 1991). The latter algorithm has later been generalized into a first-order framework (Kovačič 1994b). The resulting system, MILP, performs a stochastic hill-climbing search with simulated annealing. Whenever it reaches a local optimum, it backtracks to a previous rule, whose successors have not yet been examined, in order to get a new starting point. A similar approach is implemented in the SFOIL algorithm (Pompe et al. 1993).

Another family of stochastic separate-and-conquer rule learning algorithms choose a *genetic algorithm* (Goldberg 1989) for finding good rules. One such system, SIA (Venturini 1993), selects a random starting example and searches for a suitable generalization in a bottom-up fashion. It maintains a set of candidate rules – a *generation* – which is initialized with random generalizations of the selected examples. The next generation is obtained by randomly generating new rules, randomly generalizing old rules, or randomly exchanging conditions between rules. The resulting rules are evaluated using a weighted average of their accuracy and complexity, and a fixed number of them are retained to form the next generation. This process is repeated until the best rule remains stable for a certain number of generations. A similar approach was used in GA-SMART for learning first-order rules in a top-down fashion (Giordana and Sale 1992).

4.2. *Search strategy*

An important decision that has to be made is in which direction the hypothesis space will be searched. Rules can be organized into a *generality lattice*, where rule *A* is considered to be more general than rule *B* iff *A* covers all instances that are covered by *B*. *B* is then said to be more specific than *A*. A separate-and-conquer rule-learning algorithm can employ different strategies for searching this lattice: *top-down* (general-to-specific), *bottom-up* (specific-to-general), and *bidirectional*. These options can be implemented into the REFINERULE procedure of Figure 4. After a rule has been initialized appropriately with INITIALIZERULE, REFINERULE will specialize it when a top-down strategy is used and generalize it in the case of a bottom-up search strategy.

4.2.1. *Top-down search*

Top-down search is most commonly used in separate-and-conquer learning algorithms. The hypothesis space of possible rules is searched by repeatedly specializing candidate rules. Typically, the list of candidate rules is initialized with the rule with the sole condition *true*, which usually is the most general rule in the hypothesis space. Candidate rules are then refined using specialization operators, most typically by adding conditions as in Figure 3. Most declarative bias formalisms like DLAB and ADG (see section 3.1.5) return a specialization refinement operator that can be used in the REFINERULE procedure.

AQ, for example, selects a random example and repeatedly specializes the most general rule until it still covers the selected example, but none of the negative examples. Later algorithms, like CN2 and FOIL do not aim at covering a specific positive example, but specialize the most general rule with the goal of covering as many positive examples as possible without covering a negative example. Most recently, the ILP system PROGOL (Muggleton 1995) has returned to using a starting example for defining a lower bound on the search space. Starting with the most general clause, PROGOL searches in a top-down fashion for a good rule which is more general than the most specific clause in the hypothesis space that still covers the selected positive starting example.

4.2.2. *Bottom-up search*

In bottom-up search, the hypothesis space is examined by repeatedly generalizing a most specific rule. In the propositional case this can simply be a randomly chosen positive example, while first-order systems usually construct a *starting clause* that is more specific than any other clause that entails this example. Usually this clause is constructed by adding all ground facts that can be deduced from the background knowledge as conditions to the body of a rule that has a generalization of a randomly chosen example as its head. This starting rule is then generalized to increase the number of covered examples.

GOLEM (Muggleton and Feng 1990), for example, forms a starting clause by computing the *relative least general generalization* (Plotkin 1971) of a randomly chosen pair of positive examples. This starting clause is successively generalized by greedily selecting additional positive examples that will be used for building the *rlgg*. ITOU (Rouveirol 1992) constructs a starting clause by adding all conditions that can be proved from the background knowledge (*saturation*). A representation change called *flattening* that removes all function symbols from the examples and the background knowledge allows to implement generalization with a single operator that drops literals from rules (*truncation*) (Rouveirol 1994). NINA (Adé et al. 1995) is a bottom-up first-order separate-and-conquer algorithm that unifies several bottom-up ILP algorithms such as GOLEM (Muggleton and Feng 1990), ITOU (Rouveirol 1992), and CLINT (De Raedt 1992). As the most specific clauses can be exponentially large, even infinite in the case of general first-order horn-clause logic (Plotkin 1971), the hypothesis space has to be restricted to a subset of first-order logic using syntactic (section 3.1.3) or semantic (section 4.3.6) restrictions.

There are only a few propositional bottom-up separate-and-conquer learning algorithms. One such example is SIA (Venturini 1993) which uses a genetic algorithm for searching the space of generalizations of a randomly selected example. However, the stochastic search does not progress in a strictly bottom-up fashion. Another propositional rule learning system, DLG (Webb 1992), successively generalizes a starting example by constructing a propositional least generalization of the current rule and the next positive example. If the resulting rule covers more positive examples without covering any negative examples it is retained.

4.2.3. *Bidirectional search*

the third option for searching the hypothesis space is to combine the previous approaches into a *bidirectional* search algorithm, which can employ both specialization and generalization operators during the search for good rules. For example, the basic induction algorithm of the SWAP-1 rule learning system (Weiss and Indurkhya 1991) checks whether dropping or replacing a previously learned condition can improve a rule's purity before it tries to improve it by adding a new condition. Similarly, IBL-SMART (Widmer 1993) can perform a generalization step by dropping a condition whenever its top-down search leads to a rule that covers too few positive examples (according to some predefined threshold). However, both algorithms preserve an overall top-down tendency in their search.

The JOJO algorithm (Fensel and Wiese 1993) on the other hand starts the search at an arbitrary point in the hypothesis space (e.g., a randomly generated rule) and improves it by applying generalization and specialization operators, i.e., by adding or dropping conditions. Recent additions allow the system to directly replace conditions in rules (Fensel and Wiese 1994) and to use general first-order literals (Wiese 1996). The ATRIS rule learning shell (Mladenić 1993) allows to perform a similar bidirectional search, but replaces JOJO's hill-climbing search with less myopic stochastic search procedures as in (Kononenko and Kovačič 1992) or a generalized hill-climbing technique that allows to perform a fixed number of refinement operations at a time.

4.3. *Search heuristics*

The most influential bias is the search heuristic, which estimates the quality of rules found in the search space and ideally guides the search algorithms into the right regions of the hypothesis space. In this section we describe several commonly used heuristics, which can be implemented into the EVALUATERULE subroutine, and the intuitions behind them.

In general the search heuristics used in separate-and-conquer rule learning are similar to the heuristics used in other inductive learning algorithms like those discussed in (Mingers 1989; Buntine and Niblett 1992). The major difference between heuristics for rule learning and heuristics for decision tree learning is that the latter evaluate the average quality of a number of disjoint sets (one for each value of the attribute that is tested), while rule learning approaches only evaluate the quality of the set of examples that is covered by the candidate rule.

All common heuristics are based on determining several basic properties of a candidate rule, like the number of positive and negative examples that it covers. Minor variations in counting are possible, for example in FOIL (Quinlan 1990) which does not rely on *counting instances* but instead counts the number of different instantiations of the rule body that allow to infer a given example (*counting proofs*). The relative merits of these approaches have not yet been evaluated.

In subsequent sections we will use the following notational conventions:

P . . . the total number of positive examples
N . . . the total number of negative examples
r . . . the candidate rule
r' . . . the predecessor of r, i.e., $r \in$ REFINERULE(r')
p . . . the number of positive examples covered by r
n . . . the number of negative examples covered by r
l . . . number of conditions in r
x' . . . denotes the value of x for r'

4.3.1. *Basic heuristics*

As the goal of the FINDBESTRULE procedure is to find a rule that covers as many positive examples while covering as few negative examples as possible, most search heuristics try to find a trade-off between these two conditions. The most commonly used among them are:

Accuracy:

$$A(r) \; = \; \frac{p + (N - n)}{P + N} \cong p - n$$

This measure evaluates the accuracy of *r* as a theory containing only one rule. It computes the percentage of correctly classified examples, i.e., the positive examples covered by the rule plus the negative examples not covered by the rule. As *P* and *N* are constant for all candidate rules, maximizing accuracy amounts to maximizing $p - n$. In this form it is part of the admissible search heuristic used in PROGOL (Muggleton 1995). It is also used in I-REP (Fürnkranz and Widmer 1994), which will be discussed in section 5.4, where we will also mention some deficiencies of this measure.

Purity:

$$P(r) \; = \; \frac{p}{p + n}$$

The simplest approach is to evaluate rules with their purity, i.e., the percentage of positive examples among the examples covered by the rule. This measure will attain its optimal value when no negative examples are covered. However, it does not aim at covering many positive examples. It is used in the GREEDY3 (Pagallo and Haussler 1990) and SWAP-1 (Weiss and Indurkhya 1991) algorithms.

Information content:

$$IC(r) \; = \; -\log \frac{p}{p + n}$$

Sometimes the logarithm of the rule's purity is used, which measures the amount of information contained in the classification of the covered examples. This estimate is essentially used in PRISM (Cendrowska 1987). It is basically equivalent to the purity estimate in the sense that a set of rules ordered by ascending information content will exhibit the same order as when ordered by descending purity. Thus its disadvantages apply here as well. The main advantage of using a logarithmic scale is that it tends to assign higher penalties to less frequent events.

Entropy:

$$E(r) \; = \; -\frac{p}{p + n} \log \frac{p}{p + n} - \frac{n}{p + n} \log \frac{n}{p + n}$$

The entropy is the weighted average of the information content of the positive and negative class. Originating from the ID3 decision tree learning system (Quinlan 1983), this measure has been used in early versions of the CN2 learning algorithm (Clark and Niblett 1989). However, it suffers from similar deficiencies as purity and information content and has later been replaced by the Laplace estimate (Clark and Boswell 1991).

Cross entropy:

$$CE(r) \;=\; -\frac{p}{p+n}\log\frac{\frac{p}{p+n}}{\frac{P}{P+N}} - \frac{n}{p+n}\log\frac{\frac{n}{p+n}}{\frac{N}{P+N}}$$

The cross entropy is an information theoretic measure for the distance between the *a priori* distribution of examples and the *a posteriori* distribution of the examples that are covered by *r*. It has been used in the *J*-measure (Goodman and Smyth 1988) and in the significance tests for rules used in CN2 (Clark and Niblett 1989). Both will be discussed below.

Laplace estimate:

$$LAP(r) \;=\; \frac{p+1}{p+n+2}$$

The Laplace estimate penalizes rules with low coverage. If a rule covers no examples, its Laplace will be $\frac{1}{2}$ (random guessing). On the other hand, if the rule's coverage goes to infinity, *LAP*(*r*) converges towards *P*(*r*).[7] Because of its simplicity this heuristic is quite popular and is used in CN2 (Clark and Boswell 1991), *m*FOIL (Džeroski and Bratko 1992), CLASS (Webb 1993), BEXA (Theron and Cloete 1996) and several others.

***m*-estimate**:

$$M(r) \;=\; \frac{p+m\frac{P}{P+N}}{p+n+m}$$

The *m*-estimate generalizes the Laplace so that rules with 0-coverage will be evaluated with the *a priori* probability of the positive examples in the training set instead of $\frac{1}{2}$. The parameter *m* can be used to control the influence of the *a priori* probability. The Laplace estimate can be obtained from the *m*-estimate for *m* = 2 in problems with an equal number of positive and negative examples. Both, the Laplace and the *m*-estimate can also be used for estimating probabilities in more complicated formulas. The *m*-estimate is primarily used in CN2 (Clark and Boswell 1991) and *m*FOIL (Džeroski and Bratko 1992).

ls-**content**:

$$LS(r) = \frac{\frac{p+1}{P+2}}{\frac{n+1}{N+2}} \cong \frac{p+1}{n+1}$$

In its general form the *ls*-content divides the proportion of positive examples that are covered by the current rule by the proportion of covered negative examples, both estimated with the Laplace correction. As the denominators $P + 2$ and $N + 2$ remain constant inside the FINDBESTRULE procedure, they can be omitted without changing the behavior of the program. In its more general form, the *ls*-content is used in HYDRA (Ali and Pazzani 1993) for assessing the quality of rules in multi-class problems.

ϕ-**coefficient**:

$$PHI(r) = \frac{p \cdot \bar{n} - \bar{p} \cdot n}{\sqrt{(p+\bar{p}) \cdot (n+\bar{n}) \cdot (p+n) \cdot (\bar{p}+\bar{n})}}$$

The ϕ-coefficient measures the correlation between two events with two outcomes, tabulated into a $2{\times}2$ contingency table. Its use has been suggested in (Fürnkranz 1994b, 1997)[8] for computing the correlation between the true classification of the positive and negative examples covered by rule r' and the classification suggested by its refinement $r = r' \cup c$. Of the examples covered by r', r will correctly classify p as positive and will rightly leave $\bar{n} = n' - n$ negative examples uncovered. On the other hand, it will falsely classify n negative examples as positive, while leaving $\bar{p} = p' - p$ positive examples uncovered. The result of $PHI(r)$ is a value between -1 and $+1$. Negative values indicate a negative correlation, which suggests to consider to add the negation of the condition c to r'. For a discussion of the advantages of this heuristic we have to refer the reader to (Fürnkranz 1994b).

4.3.2. *Complexity estimates*

There are a variety of heuristics for measuring the complexity of candidate rules. Among them are:

Rule length:

$$L(r) = l$$

This simple measure estimates the complexity of a rule with the number of its conditions. For example it is used in components of the search heuristics of PROGOL (Muggleton 1995).

Positive coverage:

$$C(r) \; = \; p$$

This measure for the complexity of a rule is based on the assumption that shorter rules are more general and thus cover a higher number of positive examples. DLG (Webb 1992) employs it as a search heuristic, but it is more often used as a weighting function (as e.g. in FOIL (Quinlan 1990)).

Abductivity:

$$ABD(r) \; = \; 1 - \frac{l_G}{l_S}$$

This measure has been used in various versions of the SMART family of algorithms (Botta and Giordana 1993). It aims at measuring how well *r* is explained by the available background knowledge. For this purpose, it computes r_G, a generalization of *r* that can be obtained by repeatedly *replacing* conditions of *r* that match the body of a rule in the background knowledge with the head of this rule (*absorption*). Similarly, it computes r_S, a specialization of *r* that can be obtained by *adding* these rule heads to *r* (*saturation*). *ABD*(*r*) is the percentage of conditions that appear in r_S, but not in r_G. If no background knowledge is available l_G = l_S = *l* and thus *ABD*(*r*) = 0. For details we refer to (Botta and Giordana 1993). In some versions l_G is approximated by *l*. l_S can be approximated with the length of the *language template* used in GA-SMART (see section 3.1.5). Venturini (1993) uses a propositional version of this estimate that computes the proportion of attributes that are not tested in the body of the rule.

Minimum description length:

$$MDL(H) \; = \; I(H) + I(E|H)$$

The minimum description length principle (Wallace and Boulton 1968; Rissanen 1978) has recently gained popularity in inductive learning as a heuristic that aims at finding a trade-off between the complexity and the accuracy of a hypothesis (Georgeff and Wallace 1984). It is defined as the amount of information needed to transmit a hypothesis *H* and the amount of information needed to transmit a set of examples *E* with the help of this hypothesis. The latter task is usually reduced to transmitting only the classification of the examples which basically amounts to transmitting the exceptions to the theory. The goal is to minimize this measure. Unfortunately, both terms are not computable and have to be approximated. For various computable approximations for rule learning we refer to (Kovačič 1994a, 1994b) and (Pfahringer 1995a, 1995b). FOIL (Quinlan 1990) uses a variant of the MDL principle as a stopping criterion (see section 5.1), and another variant, in an experimental version, as a pruning criterion (Quinlan 1994).

With the exception of various *MDL* measures, which already incorporate classification accuracy, complexity heuristics are rarely used on their own, but usually form components of more complex heuristics that combine various measures into one evaluation function so that different aspects of a candidate rule can be taken into account. The most popular methods for combining heuristics are described in the following sections.

4.3.3. *Gain heuristics*

Gain heuristics compute the difference in the heuristic estimates between the candidate rule r and its predecessor r', i.e., they compute the heuristic gain that can be achieved by refining r' to r. This difference is often multiplied with a weighting function,[9] so that its general form is the following:

$$G(r) \;=\; W(r)(H(r) - H(r'))$$

Some examples for gain heuristics are:

Weighted information gain:

$$WIG(r) \;=\; -C(r)(IC(r) - IC(r'))$$

The classical example for a gain heuristic is the *weighted information gain* heuristic used in FOIL (Quinlan 1990). Here the basic heuristic is information content and the difference is weighted with the number of covered positive examples. The sign has to be reversed as $IC(r)$ is a heuristic that has to be minimized so that $IC(r) - IC(r') < 0$ will usually hold. In its original formulation (Quinlan 1990), $IC(r)$ and $IC(r')$ are computed by counting proofs, while $C(r)$ is computed by counting instances.

Coverage gain:

$$CG(r) \;=\; \frac{p - p'}{P} - \frac{n - n'}{N}$$

This heuristic is used for pruning in the POSEIDON algorithm (Bergadano et al. 1992). In the case of specialization operators, where $p' \geq p$ and $n' \geq n$, it measures the increase in uncovered negative examples minus the decrease in covered positive examples, for generalization operators vice versa.

ls-**Gain**:

$$LSG(r) \;=\; C(r)LS(r) - C(r')LS(r')$$

HYDRA (Ali and Pazzani 1993) uses the gain in *ls*-content times coverage between a rule and its refinement as a search heuristic.

4.3.4. *Weighted heuristics*

Many heuristics use weighting functions for combining several basic heuristics or for adjusting the behavior of a single heuristic in a certain direction (as for example in the weighted information gain heuristic discussed above). The general form of weighted heuristics is the following:

$$WH(r) \;=\; \sum_{i=1}^{n} W_i(r)H_i(r)$$

Some of the best-known weighted heuristics are:

J-**measure**:

$$J(r) \;=\; C(r)CE(r)$$

The *J*-measure has first been described in (Goodman and Smyth 1988) and later used in the stochastic search experiments of (Kononenko and Kovačič 1992). Its weights the cross entropy measure with the positive coverage complexity estimate.

SMART+:

$$SMART(r) \;=\; aWIG(r) + b\frac{C(r)}{P}P(r) + cABD(r)$$

This and similar measures have been used in various versions of the ML-SMART and GA-SMART algorithms (Botta et al. 1992; Giordana and Sale 1992) and their successor, the SMART+ learning tool (Botta and Giordana 1993). It computes the weighted average of three terms, the weighted information gain used in FOIL, a weighted consistency measure based on purity, and the abductivity measure. The weights a, b, and c can be used to trade off the relative importance of the three factors of the evaluation function. Venturini (1993) uses a similar measure with $a = 0$.

4.3.5. *Lexicographic evaluation functionals*

Many heuristics are quite likely to evaluate different rules with the same or almost the same value, so that additional criteria have to be used in order to determine the best candidate. Lexicographic evaluation functionals (*lef*s) (Michalski 1983) are a general mechanism for using a hierarchy of evaluation functions. A *lef* is an ordered set of pairs ($H_i(r)$, t_i), where the H_i are heuristic functions and the $t_i \in [0,1]$ are tolerance thresholds. The heuristics are evaluated in order. All candidate rules that have an optimal evaluation ($H_i(r) = H_i(r_{opt})$) or are within the tolerance threshold ($H_i(r) \geq t_i H_i(r_{opt})$) are evaluated by the next pair ($H_{i+1}(r)$, t_{i+1}). This is continued until a unique best candidate is determined.

In their general form, *lef*s are primarily used in the AQ-family of algorithms. Michalski (1983) suggests the use of not covered negative examples and covered positive examples as the basic heuristics for a *lef*. The special case where $t_i = 1$ is often used for tie-breaking. PRISM (Cendrowska 1987) for example evaluates rules with a variant of the information content heuristic and breaks ties using positive coverage.

4.3.6. *Determinate literals*

Several inductive logic programming algorithms, like GOLEM (Muggleton and Feng 1990), restrict the conditions that may be used in the body of a rule to *determinate literals*, i.e., to literals that have at most one valid ground substitution for each combination of input variables. In this case the counting-instances and counting-proofs methods discussed at the beginning of this section produce identical results. In FOIL, such determinate literals are added to the body of the rule when no other condition is given a high evaluation by the search heuristic (Quinlan 1991). This is necessary, because determinate literals usually have a low heuristic evaluation, because they will typically have a valid ground instantiation for all positive and negative training examples. Thus they will not exclude any negative examples and a rule that is refined by adding such a literal consequently receives a low heuristic value by most common heuristics.

This problem is also addressed by the *merit heuristic* used in CHAM (Kijsirikul et al. 1991), which computes a weighted average of information gain and a measure that computes the similarity of the instantiations of the variables used in the body of the rule with the output variables for a randomly chosen seed example. This results in higher heuristic values for conditions that introduce new variables whose values are similar to those of the output variables.

5. Overfitting Avoidance Bias

The SIMPLESEPARATEANDCONQUER algorithm of Figure 3 has a severe drawback: real-world data may be noisy. Noisy data are a problem for many learning algorithms, because it is hard to distinguish between rare exceptions and erroneous examples. The algorithm forms a complete and consistent theory, i.e., it tries to cover all of the positive and none of the negative examples. In the presence of noise it will therefore attempt to add literals to rules in order to exclude positive examples that have a negative classification in the training set and add rules in order to cover negative examples that have erroneously been classified as positive. Thus complete and consistent theories generated from noisy examples are typically very complicated and exhibit low predictive accuracy on classifying unseen examples. This problem is known as *overfitting the noise*.

One remedy for the overfitting problem is to try to increase the predictive accuracy by considering not only complete and consistent theories, but also approximate, but simpler theories. A simple theory that covers most positive examples and excludes most negative examples of the training set will often be more predictive than a complete and consistent, but very complex theory. Such a bias towards simpler theories has been termed *overfitting avoidance bias* (Schaffer 1993; Wolpert 1993).

Several algorithms, such as CLASS (Webb 1993), rely on the noise handling capabilities of search heuristics like the Laplace-estimate, which can prefer rules that cover only a few negative examples over clauses that cover no negative examples if the former cover more positive examples. Other algorithms such as PROGOL (Muggleton 1995) can also rely on a severely constrained hypothesis language which is unlikely to contain overfitting hypotheses. On the other hand, a wide variety of algorithms employs techniques that are specifically designed for overfitting avoidance. The remainder of this section is devoted to a discussion of such *pruning* heuristics and algorithms.

5.1. *Pre-pruning*

Pre-pruning methods deal with noise during concept generation. They are implemented into the STOPPINGCRITERION subroutine of Figure 4. Their basic idea is to stop the refinement of rules although they may still be over-general.

Thus, rules are allowed to cover a few negative examples if excluding the negative examples is esteemed to be too costly by the stopping criterion.

The most commonly used stopping criteria are

- *Minimum Purity Criterion*: This simple criterion requires that a certain percentage of the examples covered by the learned rules is positive. It is for example used in the SFOIL algorithm (Pompe et al. 1993) as a termination criterion for the stochastic search. In FOIL (Quinlan 1990) this criterion is used as a RULESTOPPINGCRITERION: When the best rule is below a certain purity threshold (usually 80%) it is rejected and the learned theory is considered to be complete.
- *Encoding Length Restriction*: This heuristic use in the ILP algorithm FOIL (Quinlan 1990) is based on the Minimum Description Length principle (Rissanen 1978). It tries to avoid learning complicated rules that cover only a few examples by making sure that the number of bits that are needed to encode a clause is less than the number of bits needed to encode the instances covered by it. For encoding p positive and n negative training instances one needs at least $\log_2(p+n) + \log_2(\binom{p+n}{p})$ bits. Literals can be encoded by specifying one out of r relations ($\log_2(r)$ bits), one out of v variabilizations ($\log_2(v)$ bits) and whether it is negated or not (1 bit). The sum of these terms for all l literals of the clause has to be reduced by $\log_2(l!)$ since the ordering of literals within a clause is in general irrelevant.
- *Significance Testing* was first used as rule stopping criterion in the propositional CN2 induction algorithm (Clark and Niblett 1989) and later on in the relational learner *m*FOIL (Džeroski and Bratko 1992). It tests for significant differences between the distribution of positive and negative examples covered by a rule and the overall distribution of positive and negative examples. For this test is exploits the fact that the likelihood ratio statistic that can be derived from the *J*-measure as $LRS(r) = 2(P + N)J(r)$ is *approximately* distributed χ^2 with 1 degree of freedom. Insignificant rules can thus be rejected. In BEXA (Theron and Cloete 1996) this test is also used for comparing the distribution of instances covered by a rule to that of its direct predecessor. If the difference is insignificant, the rule is discarded.
- *The Cutoff Stopping Criterion* compares the heuristic evaluation of a literal to a user-set threshold and only admits literals that have an evaluation above this *cutoff*.[10] It has been used in the relational separate-and-conquer learning system FOSSIL (Fürnkranz 1994b). Under the assumption that the search heuristic returns values between 0 and 1, FOSSIL will fit all of the data at *cutoff* = 0 (no pre-pruning). On the other hand, *cutoff* = 1 means that FOSSIL will learn an empty theory (maxi-

mum pre-pruning). Values between 0 and 1 trade off the two extremes. For the correlation heuristic, a value of 0.3 has been shown to yield good results at different training set sizes and at differing levels of noise (Fürnkranz 1994b) as well as across a variety of test domains (Fürnkranz 1997).

5.2. *Post-pruning*

While pre-pruning techniques try to account for the noise in the data while constructing the final theory, post-pruning methods attempt to improve the learned theory in a post-processing phase (subroutine POSTPROCESS of Figure 4). A commonly used post-processing technique aims at removing redundant conditions from the body of a rule and removing unnecessary rules from the concept. The latter technique has already been used in various versions of the AQ algorithm (Michalski et al. 1986). The basic idea is to test whether the removal of a single condition or even of an entire rule would lead to a decrease in the quality of the concept description, usually measured in terms of classification accuracy on the training set. If this is not the case, the condition or rule will be removed.

This framework has later been generalized in the POSEIDON system (Bergadano et al. 1992). POSEIDON can simplify a complete and consistent concept description, which has been induced by AQ15 (Michalski et al. 1986), by removing conditions and rules and by contracting and extending intervals and internal disjunctions. POSEIDON successively applies the operator that results in the highest coverage gain (see section 4.3.3) as long as the resulting theory increases some quality criterion.

This method can be easily adopted for avoiding overfitting of noisy data. A frequently used approach is to maximize the predictive accuracy measured on a separate set of data that has not been available to the learner during theory construction. This method has been suggested in Pagallo and Haussler (1990) based on similar algorithms for pruning decision trees (Quinlan 1987b). Before learning a complete and consistent concept description, the training set is split into two subsets: a *growing set* (usually 2/3) and a *pruning* set (1/3). The concept description that has been learned from the growing set is subsequently 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.

Figure 5 shows this algorithm in pseudo-code. The subroutine BEST-SIMPLIFICATION selects the theory with the highest evaluation on the pruning set from the set of simplifications of the current theory. Simplifications that are usually tried are deleting an entire rule, or deleting the last condition of a rule as in *reduced error pruning* (REP) (Brunk and Pazzani 1991). Other algo-

```
procedure REP(Examples, SplitRatio)

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

Figure 5. A post-pruning algorithm.

rithms employ additional simplification operators like deleting each condition of a rule (Fürnkranz and Widmer 1994), deleting a final sequence of conditions (Cohen 1993), finding the best replacement for a condition (Weiss and Indurkhya 1991), and extending and contracting internal disjunctions and intervals (Bergadano et al. 1992). If the evaluation of the best simplification is not below the evaluation of the unpruned theory, REP will continue to prune the new theory. This is repeated until the evaluation of the best simplification is below that of its predecessor.

Brunk and Pazzani (1991) have empirically shown that REP can learn more accurate theories than FOIL, which uses pre-pruning. However, post-pruning has also several disadvantages, most notably efficiency. Cohen (1993) has shown that REP has a time complexity of $\Omega(n^4)$ on purely random data. Therefore he proposed GROW a new pruning algorithm based on a technique used in the GROVE learning system (Pagallo and Haussler 1990). Like REP, GROW first finds a theory that overfits the data. But instead of pruning the intermediate theory until any further deletion results in a decrease of accuracy on the pruning set, generalizations of rules from this theory are successively selected to form the final concept description until no more rules will improve predictive accuracy on the pruning set. Thus GROW performs a top-down search in the space of pruned theories instead of REP's bottom-up search. However, Cameron-Jones (1996) has shown that for noisy data, the asymptotic costs of this pruning algorithm are still somewhat higher than the costs of the initial overfitting phase.

5.3. *Combining pre- and post-pruning*

The advantages of pre- and post-pruning are typically complementary: While pre-pruning is more efficient, post-pruning will often produce better results. Although the GROW algorithm as described in the last section can drastically reduce the costs of pruning an overly complex theory, its overall costs are still unnecessarily high. The reason is that GROW, like REP, has to learn an overfitting intermediate theory. An obvious improvement would therefore be to limit the amount of overfitting by using pre-pruning heuristics inside SeparateAndConquer program that is called in the algorithm of Figure 5. Cohen (1993) improves the GROW algorithm in such a way by using two weak MDL-based stopping criteria. These are not intended to entirely prevent overfitting like the pre-pruning approaches of section 5.1, but to reduce the amount of overfitting, so that the post-pruning phase can start with a better theory and has to do less work. Similarly, the BEXA algorithm (Theron and Cloete 1996) uses significance testing as a pre-pruning criterion and performs an additional post-pruning phase where conditions and rules are pruned in the way described in Quinlan (1987a).

However, there is always the danger that a predefined stopping criterion will over-simplify the theory. To avoid this, Fürnkranz (1994c) has developed an algorithm called *Top-Down Pruning* (TDP). This algorithm generates all theories that can be learned with different settings of the cutoff parameter of FOSSIL’s cutoff stopping criterion (Fürnkranz 1994b). This series of theories is generated in a top-down fashion. The most complex theory within one standard error of classification of the most accurate theory is selected as a starting point for the post-pruning phase.[11] The hope is that this theory will not be an over-simplification (it is more complex than the most accurate theory found so far), but will also be close to the intended theory (its accuracy is still close to the best so far). Thus only a limited amount of pruning has to be performed. TDP’s implementation made use of several optimizations, so that finding this theory is often cheaper than completely fitting the noise.

5.4. *Integrating pre- and post-pruning*

There is another fundamental problem with post-pruning in separate-and-conquer algorithms, which was first pointed out in Fürnkranz and Widmer (1994). 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 conditions of a rule will affect all subsequent rules. Figure 6(a) illustrates how post-pruning works in decision tree learning. The right half of the overfitting tree covers the sets C and D of the training instances.

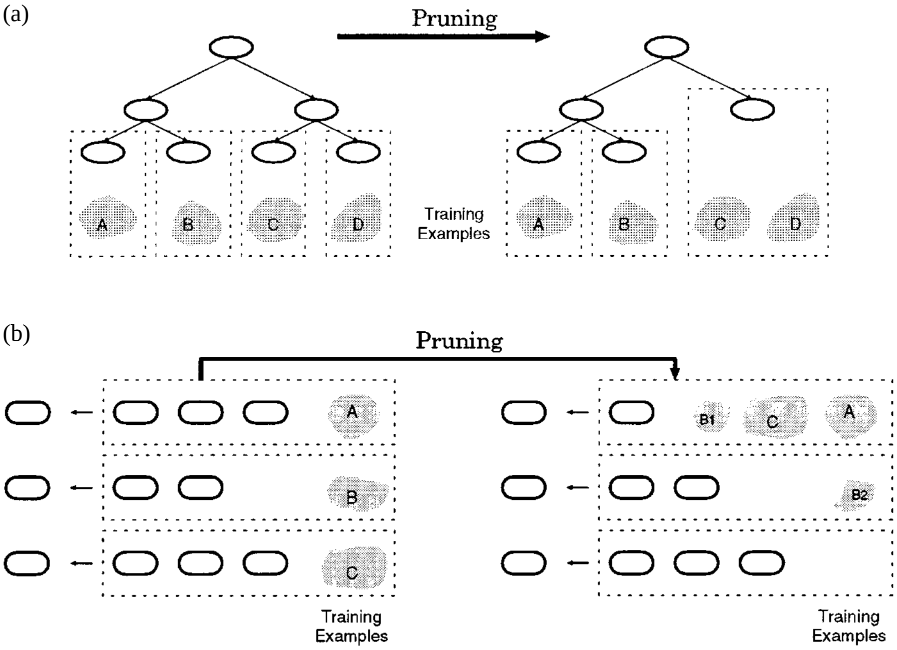


Figure 6. Post-pruning in (a) divide-and-conquer and (b) separate-and-conquer learning algorithms.

When the pruning algorithm decides to prune these two leaves, their ancestor node becomes a leaf that now covers the examples $C \cup D$. The left branch of the decision tree is not influenced by this operation.

On the other hand, pruning a condition from a rule means that it will be generalized, i.e., it will cover more positive and negative instances. Consequently those additional positive and negative instances should be removed from the training set so that they cannot influence the learning of subsequent rules. In the example of Figure 6(b), the first of three rules is simplified and now covers not only the examples its original version has covered, but also all examples that the third rule has covered and several of the examples that the second rule has covered. While the third rule could easily be removed by a post-pruning algorithm, the situation is not as simple with the remaining set of examples B2. The second rule will naturally cover all examples of the set B2, because it has been learned in order to cover the examples of its superset B. However, it might well be the case that a different rule could be more appropriate for discriminating the positive examples in B2 from the remaining negative examples. As pruning conditions from a rule can only generalize the concept, i.e., increase the set of covered examples, a post-pruning algorithm has no means for adjusting the second rule to this new situation. Thus the learner may be lead down a garden path, because the set of examples that remain uncovered by the unpruned rules at the beginning of a theory may yield a different evaluation of candidate conditions for subsequent rules than the set of examples that remain uncovered by the pruned versions of these rules.

Incremental reduced error pruning (I-REP) (Fürnkranz and Widmer 1994) addresses this problem by pruning each individual rule right after it has been learned. This ensures that the algorithm can remove the training examples that are covered by the pruned rule before subsequent rules are learned. Thus it can be avoided that these examples influence the learning of subsequent rules.

Figure 7 shows pseudo-code for this algorithm. As in REP the current set of training examples is split into a growing and a pruning set. However, not an entire theory, but only one rule is learned from the growing set. Then conditions are deleted from this rule in a greedy fashion until any further deletion would decrease the evaluation of this rule on the pruning set. Single pruning steps can be performed by submitting a one-rule theory to the same BESTSIMPLIFICATION subroutine used in REP. The best rule found by repeatedly pruning the original rule is added to the concept description and all covered positive and negative examples are removed from the training – growing *and* pruning – set. The remaining training instances are then redistributed into a new growing and a new pruning set to ensure that each of the two sets contains the predefined percentage of the remaining examples. The next rule is then learned from the new growing set and pruned on the new pruning set. When the evaluation of the pruned rule is below the evaluation of the empty rule (i.e. the rule with the body `fail`), the rule is not added to the concept description and I-REP returns the learned rules. Thus the evaluation of the pruned rules on the pruning set also serves as a stopping criterion. Post-pruning methods are used as pre-pruning heuristics.

I-REP has been shown to outperform various other pruning algorithms in a variety of noisy domains, in particular in terms of efficiency (Fürnkranz and Widmer 1994; Fürnkranz 1997). However, it has several weaknesses, which have been addressed in subsequent work (Cohen 1995). First Cohen (1995) has shown that I-REP's heuristic for evaluating rules, ACCURACY, has a high variance for low-coverage rules and therefore I-REP is likely to stop prematurely and to over-generalize in domains that are susceptible to the *small disjuncts problem* (Holte, Acker and Porter 1989). Second, the accuracy-based pruning criterion used in I-REP basically optimizes the difference between the positive and negative examples covered by a rule. Cohen (1995) points out that this measure can lead to undesirable choices. For example it would prefer a rule that covers 2000 positive and 1000 negative instances over a

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

$Theory = \emptyset$
while POSITIVE(Examples) $\neq \emptyset$
    $Rule = \emptyset$
    SPLITEXAMPLES(SplitRatio, Examples, GrowingSet, PruningSet)
    $Covered = GrowingSet$
    while NEGATIVE(Covered) $\neq \emptyset$
        $Rule$ = REFINERULE($Rule, Covered$)
        $Covered$ = COVER(Rule,Covered)
    loop
         NewRule = BESTSIMPLIFICATION(Rule,PruningSet)
         if EVALUATERULE(NewRule,PruningSet) <
                EVALUATERULE(Rule,PruningSet)
            exit loop
         Rule = NewRule
    if EVALUATERULE(Rule,PruningSet) $\leq$ EVALUATERULE(fail,PruningSet)
         exit while
    $Theory = Theory \cup Rule$
    $Examples = Examples - Covered$
return(Theory)
```

Figure 7. Integrating pre- and post-pruning.

rule that covers 1000 positive and only 1 negative instance. As an alternative, Cohen (1995) suggests to divide this difference by the total number of covered examples and shows that this choice leads to significantly better results. In addition he shows that an alternative rule stopping criterion based on theory description length and an additional post-pruning phase can further improve I-REP. The resulting algorithm, RIPPER, has been shown to be competitive with C4.5rules (Quinlan 1993) without losing I-REP's efficiency (Cohen 1995).

6. Related Work

In the literature one can also find a variety of separate-and-conquer learning algorithms that tackle slightly different tasks than the inductive concept learning problem as defined in Figure 2. Mooney (1995), for example, describes an algorithm that learns conjunctions of disjunctive rules by finding rules that cover all positive examples and exclude some of the negative examples. Such rules are added to the theory until all negative examples have been excluded. Thus the algorithm is a dual to conventional separate-and-conquer algorithms in the sense that it learns by covering on the negative examples. All algorithms and heuristics discussed in this paper can be applied to this task as well with only slight modifications (e.g., to swap the role of *p* and *n* in the heuristics). ICL (De Raedt and Van Laer 1995) is a similar approach that learns in a first-order logic framework. The basic entity that it covers are not single examples, but ground models of the target theory.

Separate-and-conquer algorithms have also been used to tackle *regression problems*, i.e., the prediction of real-valued instead of discrete class variables. The key issue here is the definition of an appropriate evaluation criterion. This is a non-trivial issue, because the real-valued prediction of a regression rule will in a general never be identical to the true value. In IBL-SMART (Widmer 1993), another algorithm from the ML-SMART family, this problem is solved by using a discretized class variable for learning. A numeric prediction for a new example is then derived by finding a rule that covers the example and using the numeric class variables of all other examples covered by that rule for deriving a prediction. RULE (Weiss and Indurkhya 1995) and FORS (Karalič 1995) are able to directly use real-valued class variables. They employ a top-down hill-climbing algorithm for finding a rule body that minimizes an error function (like the mean squared error) on the prediction of the covered examples. The problem here is that rules that cover only one example appear to commit no error when they predict this value. FORS tries to solve this problem of overfitting with the introduction of a variety of stopping criteria, like a maximum rule length or a minimum number of examples that a rule must cover.

There are also a variety of other rule learning approaches that do not use separate-and-conquer frameworks. In particular, several algorithms that use a bottom-up learning strategy start with a set of rules, each representing one example, and successively generalize the entire rule set. RISE (Domingos 1996b) is a particularly successful system that uses such an approach, where single rules are minimally generalized so that they cover the example that is most similar to them. Domingos (1996b) has named this method “conquering without separating”. A similar approach has also been taken in the inductive logic programming algorithm CHILLIN (Zelle, Mooney and Konvisser 1994), where rules are generalized by forming their *least general generalization* (Plotkin 1970) and, if necessary, successively specialized using top-down hill-climbing as in FOIL. CWS (Domingos 1996a) interleaves the induction of different rules by starting to induce the next rule in the same cycle as the second condition of the current rule is learned. ITRULE (Goodman and Smyth 1988) performs an efficient exhaustive search for a fixed number of rules with a fixed maximum length. BBG (Van Horn and Martinez 1993) learns a decision list by inserting a learned rule at appropriate places and eval-

Table 1. Biases of selected separate-and-conquer rule learning algorithms

	Language Bias							Search Bias							Overfitting Avoidance		
	Static					Dyn.		Algorithm				Strategy					
Algorithm	Selectors	Literals	Synt. Restr.	Rel. Clichés	Rule Models	Lang. Hier.	Constr. Ind.	Hill-Climbing	Beam Search	Best First	Stochastic	Top-Down	Bottom-Up	Bidirectional	Pre-Pruning	Post-Pruning	Integrated
AQ	×							×	×			×					
AQ15	×							×	×			×				×	
AQ17	×						×	×	×			×					
ATRIS	×							×			×			×		×	
BEXA	×							×	×			×			×	×	
CHAMP	×	×	×				×	×	×			×			×		
CIPF	×						×	×				×				×	
CN2	×							×	×			×			×		
CN2-MCI	×						×	×	×			×			×		
CLASS	×									×		×					
DLG	×							×	×				×				
FOCL	×	×		×				×				×			×		
FOIL	×	×	×					×				×			×		
FOSSIL	×	×	×	×				×				×			×		
GA-SMART	×	×		×	×						×	×			×		
GOLEM		×	×					×					×				
GREEDY3	×							×				×				×	
GRENDEL					×			×				×					
GROW	×							×				×				×	
HYDRA	×	×						×				×					
IBL-SMART	×	×		×						×				×	×		
INDUCE	×	×						×	×			×					
I-REP, I^2-REP	×	×	×	×				×				×					×
JoJo	×	×						×						×			
m-FOIL	×	×	×					×	×			×			×		
MDL-FOIL	×	×	×					×				×			×	×	
MILP	×	×	×								×	×			×		
ML-SMART	×	×		×				×	×	×		×			×		
NINA					×	×		×					×				
POSEIDON	×							×	×			×				×	
PREPEND	×							×				×					
PRISM	×							×				×					
PROGOL	×	×	×							×		×					
REP	×	×		×				×				×				×	
RIPPER	×							×				×				×	×
RDT					×			×				×					
SFOIL	×										×	×			×		
SIA	×										×		×			×	
SMART+	×	×		×	×			×	×	×	×	×			×		
SWAP-1	×							×						×		×	
TDP	×	×	×	×				×				×			×	×	

uating the quality of the resulting of the resulting decision list on the entire training set. Last but not least, the C4.5 decision tree induction algorithm has an option that allows to generate compact decision lists from decision trees by turning a decision tree into a set of non-overlapping rules, pruning these rules, and ordering the resulting overlapping rules into a decision list (Quinlan 1987a, 1993).

7. Conclusion

In this paper we surveyed the family of separate-and-conquer or covering learning algorithms. This learning strategy is now in use for almost 30 years. Despite its age, it still enjoys a great popularity, so that we feel a survey of this sort has been overdue.

We have mostly confined ourselves to binary concept learning tasks, which can be defined with a number of positive and negative examples for the target concept. A separate-and-conquer algorithm continues to learn rules until all (or most) of the positive examples are covered without covering any (or only few) negative examples. Table 1 shows the most important separate-and-conquer concept learning algorithms discussed in this paper, together with indicators for the most important biases they employ. References to papers that describe the individual algorithms had to be omitted because of space restrictions, but can be most quickly found in Figure 1.

Discussions of the pros and cons of each individual algorithm, empirical comparisons of various bias options, and recommendations are beyond the scope of this paper, and will depend on the task at hand. However, we hope that this paper can help the practitioner to pick the right algorithm based on its characteristics as well as guide the researcher to interesting research directions and suggest possible bias combinations that still have to be explored.

Acknowledgements

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P10489-MAT and J1443-INF. Financial support for the Austrian Research Institution for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Research. I would like to thank my colleagues Gerhard Widmer, Bernhard Pfahringer, Johann Petrak and Stefan Kramer for several enlightening discussions and suggestions.

Notes

[1] Strictly speaking an overfitting avoidance bias is another form of search bias. However, in particular in the context of separate-and-conquer rule learning, there are often two separate criteria for growing and simplifying hypotheses. Thus we feel that a separate treatment of these two issues is justified.

[2] Note that in the following we will use the term "refinement" for both specialization and generalization.

[3] Rules are often called (definite) clauses in logic programming terminology. In the remainder of this paper we will use these terms interchangeably. In general we will follow the logic programming terminology defined in Lloyd (1987).

[4] The original attributes have depth 0. A new variable has depth $i + 1$, where i is the maximum depth of all old variables of the literal where the new variable is introduced.

[5] The term *wrapper* is due to Kohavi (1995).

[6] This clause length is computed dynamically using FOIL's encoding length restriction (see section 5.1).

[7] In its general form the Laplace estimate has the number of classes c in its denominator, so that it will return $\frac{1}{c}$ for rules with no coverage.

[8] The derivation of the correlation coefficient given in the cited works is rather lengthy and the resulting formula a bit complicated, but it is equivalent to the simple definition of the ϕ-coefficient given here, which can be found in comprehensive statistical texts such as (Mittenecker 1977).

[9] $H(r')$ will often be constant for all candidate clauses (e.g. when using hill-climbing), so that optimizing $G(r)$ for a constant function $W(r)$ would produce the same behavior as directly optimizing $H(r)$.

[10] Note that it differs from the minimum purity criterion in the way that it directly thresholds the search heuristic instead of using purity as a separate criterion.

[11] This method is inspired by the approach taken in CART (Breiman, Friedman, Olshen and Stone 1984) where the most general decision tree within this standard error margin is selected as a final theory.

References

Adé, H., De Raedt, L. & Bruynooghe, M. (1995). Declarative Bias for Specific-to-General ILP Systems. *Machine Learning* **20**(1–2): 119–154. Special Issue on Bias Evaluation and Selection.

Ali, K. M. & Pazzani, M. J. (1993). HYDRA: A Noise-Tolerant Relational Concept Learning Algorithm. In Bajcsy, R. (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1064–1071. Morgan Kaufmann: Chambèry, France.

Bain, M. (1991). Experiments in Non-Monotonic Learning. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 380–384. Evanston, Illinois.

Bergadano, F. & Giordana, A. (1988). A Knowledge Intensive Approach to Concept Induction. In *Proceedings of the 5th International Conference on Machine Learning (ML-88)*, 305–317. Ann Arbor, Michigan.

Bergadano, F., Giordana, A. & Ponsero, S. (1989). Deduction in Top-Down Inductive Learning. In *Proceedings of the 6th International Workshop on Machine Learning (ML-89)*, 23–25.

Bergadano, F., Giordana, A. & Saitta, L. (1988). Automated Concept Acquisition in Noisy Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10**: 555–578.

Bergadano, F. & Gunetti, D. (1993). An Interactive System to Learn Functional Logic Programs. In Bajcsy, R. (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1044–1049. Morgan Kaufmann.

Bergadano, F. & Gunetti, D. (1995). *Inductive Logic Programming – From Machine Learning to Software Engineering*. Logic Programming Series. The MIT Press: Cambridge, MA.

Bergadano, F., Matwin, S., Michalski, R. S. & Zhang, J. (1992). Learning Two-Tiered Descriptions of Flexible Concepts: The POSEIDON System. *Machine Learning* **8**: 5–43.

Blockeel, H. & De Raedt, L. (1997). Top-Down Induction of Logical Decision Trees. Tech. rep. CW 247, Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium.

Bloedorn, E. & Michalski, R. S. (1991). Constructive Induction From Data in AQ17-DCI: Further Experiments. Tech. rep. MLI 91-12, Artificial Intelligence Center, George Mason University, Fairfax, VA.

Bloedorn, E., Michalski, R. S. & Wnek, J. (1993). Multistrategy Constructive Induction: AQ17-MCI. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, 188–203.

Boström, H. (1995). Covering vs. Divide-and-Conquer for Top-Down Induction of Logic Programs. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1194–1200.

Botta, M., Giordana, A. & Saitta, L. (1992). Comparison of Search Strategies in Learning Relations. In Neumann, B. (ed.) *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, 451–455. John Wiley & Sons: Vienna, Austria.

Botta, M. & Giordana, A. (1993). SMART+: A Multi-Strategy Learning Tool. In Bajcsy, R. (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 937–944. Morgan Kaufmann: Chambèry, France.

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

Brunk, C. A. & Pazzani, M. J. (1991). An Investigation of Noise-Tolerant Relational Concept Learning Algorithms. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 389–393. Morgan Kaufmann: Evanston, Illinois.

Buntine, W. & Niblett, T. (1992). A Further Comparison of Splitting Rules for Decision-Tree Induction. *Machine Learning* **8**: 75–85.

Cameron-Jones, R. M. (1996). The Complexity of Batch Approaches to Reduced Error Rule Set Induction. In Foo, N. & Goebel, R. (eds.) *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI-96)*, 348–359. Springer-Verlag: Cairns, Australia.

Cameron-Jones, R. M. & Quinlan, J. R. (1993). Avoiding Pitfalls When Learning Recursive Theories. In Bajcsy, R. (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1050–1057. Chambèry, France.

Cendrowska, J. (1987). PRISM: An Algorithm for Inducing Modular Rules. *International Journal of Man-Machine Studies* **27**: 349–370.

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

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

Cohen, W. W. (1993). Efficient Pruning Methods for Separate-and-Conquer Rule Learning Systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 988–994. Morgan Kaufmann: Chambèry, France.

Cohen, W. W. (1994). Grammatically Biased Learning: Learning Logic Programs Using an Explicit Antecedent Description Language. *Artificial Intelligence* **68**(2): 303–366.

Cohen, W. W. (1995). Fast Effective Rule Induction. In Prieditis, A. & Russell, S. (eds.) *Proceedings of the 12th International Conference on Machine Learning (ML-95)*, 115–123. Morgan Kaufmann: Lake Tahoe, CA.

De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press.

De Raedt, L. (ed.) (1995). *Advances in Inductive Logic Programming*, Vol. 32 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

De Raedt, L, & Bruynooghe, M. (1990). Indirect Relevance and Bias in Inductive Concept Learning. *Knowledge Acquisition* **2**: 365–390.

De Raedt, L. & Van Laer, W. (1995). Inductive Constraint Logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory (ALT-95)*, 80–94. Springer-Verlag.

Dehaspe, L. & De Raedt, L. (1996). DLAB: A Declarative Language Bias Formalism. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS-96)*, 613–622.

Domingos, P. (1996a). Linear-Time Rule Induction. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 96–101. AAAI Press.

Domingos, P. (1996b). Unifying Instance-Based and Rule-Based Induction. *Machine Learning* **24**: 141–168.

Džeroski, S. & Bratko, I. (1992). Handling Noise in Inductive Logic Programming. In *Proceedings of the International Workshop on Inductive Logic Programming (ILP-92)*, 109–125. Tokyo, Japan.

Džeroski, S., Schulze-Kremer, S., Heidtke, K. R., Siems, K. & Wettschereck, D. (1996). Applying ILP to Diterpene Structure Elucidation from ^{13}C NMR Spectra. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, 12–24.

Fensel, D. & Wiese, M. (1993). Refinement of Rule Sets with JoJo. In Brazdil, P. (ed.) *Proceedings of the 6th European Conference on Machine Learning (ECML-93)*, No. 667 in Lecture Notes in Artificial Intelligence, 378–383. Springer-Verlag.

Fensel, D. & Wiese, M. (1994). From JoJo to Frog: Extending a Bidirectional Strategy to a More Flexible Three-Directional Search. In Globig, C. & Althoff, K.-D. (eds.) *Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen*, Forschungbericht No. LSA-95-01, 37–44. University of Kaiserslautern. Zentrum für Lernende Systeme und Anwendungen.

Fürnkranz, J. (1994a). *Efficient Pruning Methods for Relational Learning*. Ph.D. thesis, Vienna University of Technology.

Fürnkranz, J. (1994b). FOSSIL: A Robust Relational Learner. In Bergadano, F. & De Raedt, L. (eds.) *Proceedings of the 7th European Conference on Machine Learning (ECML-94)*, Vol. 784 of *Lecture Notes in Artificial Intelligence*, 122–137. Springer-Verlag: Catania, Italy.

Fürnkranz, J. (1994c). Top-Down Pruning in Relational Learning. In Cohn, A. (ed.) *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, 453–457. John Wiley & Sons: Amsterdam, The Netherlands.

Fürnkranz, J. (1995). A Tight Integration of Pruning and Learning (Extended Abstract). In Lavrač, N. & Wrobel, S. (eds.) *Proceedings of the 8th European Conference on Machine Learning (EMCL-95)*, Vol. 912 of *Lecture Notes in Artificial Intelligence*, 291–294. Springer-Verlag: Heraclion, Greece.

Fürnkranz, J. (1997). Pruning Algorithms for Rule Learning. *Machine Learning* **27**(2): 139–171.

Fürnkranz, J. & Widmer, G. (1994). Incremental Reduced Error Pruning. In Cohen W. & Hirsh, H. (eds.) *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, 70–77. Morgan Kaufmann: New Brunswick, NJ.

Georgeff, M. P. & Wallace, C. S. (1984). A General Criterion for Inductive Inference. In O'Shea, T. (ed.) *Proceedings of the Sixth European Conference on Artificial Intelligence (ECAI-84)*, 473–482. Elsevier: Amsterdam.

Giordana, A. & Sale, C. (1992). Learning Structured Concepts Using Genetic Algorithms. In Sleeman, D. & Edwards, P. (eds.) *Proceedings of the 9th International Workshop on Machine Learning (ML-92)*, 169–178. Morgan Kaufmann: Edinburgh.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley: Reading, MA.

Goodman, R. M. & Smyth, P. (1988). Information-Theoretic Rule Induction. In Kodratoff, Y. (ed.) *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI-88)*, 357–362. Pitman: London.

Grobelnik, M. (1992). Markus – An Optimized Model Inference System. In Rouveirol, C. (ed.) *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*. Vienna, Austria.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* **4**(2): 100–107.

Helft, N. (1989). Induction as Nonmonotonic Inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, 149–156.

Holte, R., Acker L. & Porter, B. (1989). Concept Learning and the Problem of Small Disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, 813–818. Morgan Kaufmann: Detroit, MI.

Karalič, A. (1995). *First Order Regression*. Ph.D. thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science, Slovenia.

Kietz, J.-U. & Wrobel, S. (1992). Controlling the Complexity of Learning in Logic Through Syntactic and Task-Oriented Models. In Muggleton, S. H. (ed.) *Inductive Logic Programming*, chap. 16, 335–359. Academic Press Ltd.: London.

Kijsirikul, B., Numao, M. & Shimura, M. (1991). Efficient Learning of Logic Programs with Non-Determinate, Non-Discriminating Literals. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 417–421. Evanston, Illinois.

Kijsirikul, B., Numao, M. & Shimura, M. (1992). Discrimination-Based Constructive Induction of Logic Programs. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, 44–49.

Kirkpatrick, S., Gelatt, C. & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science* **220**: 671–680.

Kohavi, R. (1995). *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. Ph.D. thesis, Stanford University, Dept. of Computer Science.

Kohavi, R. & John, G. H. (1995). Automatic Parameter Selection by Minimizing Estimated Error. In Prieditis, A. & Russell, S. (eds.) *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, 304–312. Morgan Kaufmann.

Kononenko, I. & Kovačič, M. (1992). Learning as Optimization: Stochastic Generation of Multiple Knowledge. In Sleeman, D. & Edwards, P. (eds.) *Proceedings of the 9th International Workshop on Machine Learning (ML-92)*, 257–262. Morgan Kaufmann.

Kovačič, M. (1991). Markovian Neural Networks. *Biological Cybernetics* **64**: 337–342.

Kovačič, M. (1994a). MDL-Heuristics in ILP Revised. In *Proceedings of the ML-COLT-94 Workshop on Applications of Descriptional Complexity to Inductive, Statistical, and Visual Inference*.

Kovačič, M. (1994b). *Stochastic Inductive Logic Programming*. Ph.D. thesis, Department of Computer and Information Science, University of Ljubljana.

Kramer, S. (1994). CN2-MCI: A Two-Step Method for Constructive Induction. In *Proceedings of the ML-COLT-94 Workshop on Constructive Induction and Change of Representation.*

Kramer, S. (1996). Structural Regression Trees. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, 812–819. AAAI Press.

Langley, P., Simon, H. A. & Bradshaw, G. L. (1987). Heuristics for Empirical Discovery. In Bolc, L. (ed.) *Computational Models of Learning*. Springer-Verlag. Reprinted in Shavlik, J. W. & Dietterich, T. G. (ed.) *Reading in Machine Learning*. Morgan Kaufmann, 1991.

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

Lloyd, J. W. (1987). *Foundations of Logic Programming* (2nd, extended edition). Springer-Verlag: Berlin.

Matheus, C. J. (1989). A Constructive Induction Framework. In *Proceedings of the 6th International Workshop on Machine Learning*, 474–475.

Michalski, R. S. (1969). On the Quasi-Minimal Solution of the Covering Problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, Vol. A3 (Switching Circuits), 125–128. Bled, Yugoslavia.

Michalski, R. S. (1973). AQVAL/1 – Computer Implementation of a Variable-Valued Logic System VL_1 and Examples of Its Application to Pattern Recognition. In *Proceedings of the 1st International Joint Conference on Pattern Recognition*, 3–17.

Michalski, R. S. (1980). Pattern Recognition and Rule-Guided Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2**: 349–361.

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence* **20**(2): 111–162.

Michalski, R. S., Mozetič, I., Hong, J. & Lavrač, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, 1041–1045. Philadephia, PA.

Mingers, J. (1989). An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning* **3**: 319–342.

Mitchell, T. M. (1980). The Need for Biases in Learning Generalizations. Tech. rep., Computer Science Department, Rutgers University, New Brunswick, MA. Reprinted in Shavlik, J. W. & Dietterich, T. G. (eds.) *Readings in Machine Learning*. Morgan Kaufmann, 1991.

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

Mladenić, D. (1993) Combinatorial Optimization in Inductive Concept Learning. In *Proceedings of the 10th International Conference on Machine Learning (ML-93)*, 205–211. Morgan Kaufmann.

Mooney, R. J. (1995). Encouraging Experimental Results on Learning CNF. *Machine Learning* **19**: 79–92.

Mooney, R. J. & Califf, M. E. (1995). Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs. *Journal of Artificial Intelligence Research* **3**: 1–24.

Muggleton, S., Bain, M., Hayes-Michie, J. & Michie, D. (1989). An Experimental Comparison of Human and Machine Learning Formalisms. In *Proceedings of the 6th International Workshop on Machine Learning (ML-93)*, 113–118. Morgan Kaufmann.

Muggleton, S. H. (ed.) (1992). *Inductive Logic Programming*. Academic Press Ltd.: London.

Muggleton, S. H. (1995). Inverse Entailment and Progol. *New Generation Computing* **13**(3,4): 245–286. Special Issue on Inductive Logic Programming.

Muggleton, S. H. & Feng, C. (1990). Efficient Induction of Logic Programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, 1–14. Tokyo, Japan.

Nédellec, C., Rouveirol, C., Adé, H., Bergadano, F. & Tausend, B. (1996). Declarative Bias in ILP. In De Raedt, L. (ed.) *Advances in Inductive Logic Programming*, Vol. 32 of *Frontiers in Artificial Intelligence and Applications*, 82–103. IOS Press: Amsterdam.

Pagallo, G. & Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning* **5**: 71–99.

Pazzani, M. & Kibler, D. (1992). The Utility of Knowledge in Inductive Learning. *Machine Learning* **9**: 57–94.

Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T. & Brunk, C. (1994). Reducing Misclassification Costs. In Cohen, W. W. & Hirsh, H. (eds.) *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, 217–225. Morgan Kaufmann: New Brunswick.

Pfahringer, B. (1994a). Controlling Constructive Induction in CiPF: An MDL Approach. In Brazdil, P. B. (ed.) *Proceedings of the 7th European Conference on Machine Learning*

(ECML-94), Lecture Notes in Artificial Intelligence, 242–256. Springer-Verlag, Catania, Sicily.

Pfahringer, B. (1994b). Robust Constructive Induction. In Nebel, B. & Dreschler-Fischer, F. (eds.) *Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI-94)*, Lecture Notes in Artificial Intelligence, 118–129. Springer-Verlag.

Pfahringer, B. (1995a). A New MDL Measure for Robust Rule Induction (Extended Abstract). In Lavrač, N. & Wrobel, S. (eds.), *Proceedings of the 8th European Conference on Machine Learning (ECML-95)*, No. 912 in Lecture Notes in Artificial Intelligence, 331–334. Springer-Verlag: Heraclion, Greece.

Pfahringer, B. (1995b). *Practical Uses of the Minimum Description Length Principle in Inductive Learning*. Ph.D. thesis, Technische Universität Wien.

Plotkin, G. D. (1970). A Note on Inductive Generalisation. In Meltzer B. & Michie, D. (eds.) *Machine Intelligence* **5**, 153–163. Elsevier North-Holland/New York.

Plotkin, G. D. (1971). A Further Note on Inductive Generalisation. In Meltzer B. & Michie, D. (eds.) *Machine Intelligence* **6**, 101–124. Elsevier North-Holland/New York.

Pompe, U., Kovačič, M. & Kononenko, I. (1993). SFOIL: Stochastic Approach to Inductive Logic Programming. In *Proceedings of the 2nd Slovenian Conference on Electrical Engineering and Computer Science (ERK-93)*, Vol. B, 189–192. Portorož, Slovenia.

Quinlan, J. R. (1983). Learning Efficient Classification Procedures and Their Application to Chess End Games. In Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (eds.) *Machine Learning. An Artificial Intelligence Approach*, 463–482. Tioga: Palo Alto, CA.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning* **1**: 81–106.

Quinlan, J. R. (1987a). Generating Production Rules from Decision Trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, 304–307. Morgan Kaufmann.

Quinlan, J. R. (1987b). Simplifying Decision Trees. *International Journal of Man-Machine Studies* **27**: 221–234.

Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning* **5**: 239–266.

Quinlan, J. R. (1991). Determinate Literals in Inductive Logic Programming. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 442–446.

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

Quinlan, J. R. (1994). The Minimum Description Length Principle and Categorical Theories. In Cohen, W. & Hirsh, H. (eds.) *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, 233–241. Morgan Kaufmann: New Brunswick, NJ.

Quinlan, J. R. (1996). Learning First-Order Definitions of Functions. *Journal of Artificial Intelligence Research* **5**: 139–161.

Quinlan, J. R. & Cameron-Jones, R. M. (1995a). Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing* **13**(3,4): 287–312. Special Issue on Inductive Logic Programming.

Quinlan, J. R. & Cameron-Jones, R. M. (1995b). Oversearching and Layered Search in Empirical Learning. In Mellish, C. (ed.) *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1019–1024. Morgan Kaufmann.

Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica* **14**: 465-471.

Rivest, R. L. (1987). Learning Decision Lists. *Machine Learning* **2**: 229–246.

Rouveirol, C. (1992). Extensions of Inversion of Resolution Applied to Theory Completion. In Muggleton, S. H. (ed.) *Inductive Logic Programming*, 63–92. Academic Press Ltd.: London.

Rouveirol, C. (1994). Flattering and Saturation: Two Representation Changes for Generalization. *Machine Learning* **14**: 219–232. Special Issue on Evaluating and Changing Representation.

Saitta, L. & Bergadano, F. (1993). Pattern Recognition and Valiant's Learning Framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**(2): 145–154.

Schaffer, C. (1993). Overfitting Avoidance as Bias. *Machine Learning* **10**: 153–178.

Segal, R. & Etzioni, O. (1994). Learning Decision Lists Using Homogeneous Rules. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, 619–625. AAAI Press: Cambridge, MA.

Shapiro, E. Y. (1981). An Algorithm that Infers Theories from Facts. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-81)*, 446–451.

Silverstein, G. & Pazzani, M. J. (1991). Relational Clichès: Constraining Constructive Induction During Relational Learning. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, 203–207. Evanston, Illinois.

Silverstein, G. & Pazzani, M. J. (1993). Learning Relational Clichés. In *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, 71–82.

Srinivasan, A., Muggleton, S. H. & Bain, M. E. (1992). Distinguishing Noise from Exceptions in Non-Monotonic Learning. In *Proceedings of the International Workshop on Inductive Logic Programming (ILP-92)*, 97–107. Tokyo, Japan.

Theron, H. & Cloete, I. (1996). BEXA: A Covering Algorithm for Learning Propositional Concept Descriptions. *Machine Learning* **24**: 5–40.

Utgoff, P. E. (1986). Shift of Bias for Inductive Concept Learning. In Michalski R., Carbonell, J. & Mitchell T. (eds.) *Machine Learning: An Artificial Intelligence Approach*, Vol. II, 107–148. Morgan Kaufmann: Los Altos, CA.

Van Horn, K. S. & Martinez, T. R. (1993). The BBG Rule Induction Algorithm. In *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence*, 348–355. Melbourne, Australia.

Vapnik, V. N. & Chervonenkis, Y. A. (1971). On the Uniform Convergence of Relative Frequencies to Their Probabilities. *Theory of Probability and Its Applications* **16**: 264–280.

Vapnik, V. N. & Chervonenkis, Y. A. (1981). Necessary and Sufficient Conditions for the Uniform Convergence of Means to Their Expectations. *Theory of Probability and Its Applications* **26**: 532–553.

Venturini, G. (1993). SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts. In Brazdil P (ed.), *Proceedings of the 6th European Conference on Machine Learning (ECML-93)*, Vol. 667 of *Lecture Notes in Artificial Intelligence*, 280–296. Springer-Verlag: Vienna, Austria.

Wallace, C. S. & Boulton, D. M. (1968). An Information Measure for Classification. *Computer Journal* **11**: 185–194.

Watanabe, L. & Rendell, L. (1991). Learning Structural Decision Trees from Examples. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, 770–776.

Webb, G. I. (1992). Learning Disjunctive Class Descriptions by Least Generalisation. Tech. rep. TR C92/9, Deakin University, School of Computing & Mathematics, Geelong, Australia.

Webb, G. I. (1993). Systematic Search for Categorical Attribute-Value Data-Driven Machine Learning. In Rowles, C., Liu, H. & Foo, N. (eds.) *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence (AI '93)*, 342–347. World Scientific: Melbourne.

Webb, G. I. (1994). Recent Progress in Learning Decision Lists by Prepending Inferred Rules. In *Proceedings of the 2nd Singapore International Conference on Intelligent Systems*, B280–B285.

Webb, G. I. (1995). OPUS: An Efficient Admissible Algorithm for Unordered Search. *Journal of Artificial Intelligence Research* **5**: 431–465.

Webb, G. I. & Agar, J. W. M. (1992). Inducing Diagnostic Rules for Glomerular Disease with the DLG Machine Learning Algorithm. *Artificial Intelligence in Medicine* **4**: 419–430.

Webb, G. I. & Brkič, N. (1993). Learning Decision Lists by Prepending Inferred Rules. In *Proceedings of the AI '93 Workshop on Machine Learning and Hybrid Systems*. Melbourne, Australia.

Weiss, S. M. & Indurkhya, N. (1991). Reduced Complexity Rule Induction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, 678–684.

Weiss, S. M. & Indurkhya, N. (1993a). Optimized Rule Induction. *IEEE Expert* **8**(6): 61–69.

Weiss, S. M. & Indurkhya, N. (1993b). Rule-Based Regression. In Bajcsy, R. (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1072–1078.

Weiss, S. M. & Indurkhya, N. (1995). Rule-Based Machine Learning Methods for Functional Prediction. *Journal of Artificial Intelligence Research* **3**: 383–403.

Widmer, G. (1993). Combining Knowledge-Based and Instance-Based Learning to Exploit Quantitative Knowledge. *Informatica* **17**: 371–385. Special Issue on Multistrategy Learning.

Wiese, M. (1996). A Bidirectional ILP Algorithm. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD), 61–72.*

Wnek, J. & Michalski, R. S. (1994). Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning* **14**(2): 139–168. Special Issue on Evaluating and Changing Representation.

Wolpert, D. H. (1993). On Overfitting Avoidance as Bias. Tech. rep. SFI TR 92-03-5001. The Santa Fe Institute, Santa Fe, NM.

Zelle, J. M., Mooney, R. J. & Konvisser, J. B. (1994). Combining Top-Down and Bottom-Up Techniques in Inductive Logic Programming. In Cohen, W. & Hirsh, H. (eds.) *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, 343–351. Morgan Kaufmann: New Brunswick, NJ.