

Simplifying decision trees: A survey

LEONARD A. BRESLOW and DAVID W. AHA

Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375, USA. Email: {breslow,aha}@aic.nrl.navy.mil

Abstract

Induced decision trees are an extensively-researched solution to classification tasks. For many practical tasks, the trees produced by tree-generation algorithms are not comprehensible to users due to their size and complexity. Although many tree induction algorithms have been shown to produce simpler, more comprehensible trees (or data structures derived from trees) with good classification accuracy, tree simplification has usually been of secondary concern relative to accuracy, and no attempt has been made to survey the literature from the perspective of simplification. We present a framework that organizes the approaches to tree simplification and summarize and critique the approaches within this framework. The purpose of this survey is to provide researchers and practitioners with a concise overview of tree-simplification approaches and insight into their relative capabilities. In our final discussion, we briefly describe some empirical findings and discuss the application of tree induction algorithms to case retrieval in case-based reasoning systems.

1 Context and motivation

The area of machine learning concerned with inducing classifiers from data focuses primarily on predictive *accuracy*, as measured by the classifiers' performance on unseen test cases. However, for many practical applications, it is desirable that the classifier "provide insight and understanding into the predictive structure of the data" (Breiman et al., 1984), as well as *explanations* of its individual predictions (Michie, 1990). Decision tree induction has been extensively studied in the machine learning and statistics communities as a solution to classification tasks (Breiman et al., 1984; Quinlan, 1986, 1993a). Many tree-simplification algorithms have been shown to yield simpler or smaller trees. The assumption is made that simpler, smaller trees are easier for humans to comprehend. Although this assumption has not been tested empirically, it will serve as a working assumption in what follows. While considerable evidence exists that trees can be simplified, simplification is generally of secondary concern relative to predictive accuracy.

No effort has been made to review the decision tree induction literature from the perspective of simplification. This paper is intended to fill this gap.

A key problem in summarizing the literature on tree simplification is the diversity of approaches that have been introduced. To manage this diversity, we offer a framework for categorizing the approaches, consisting of five categories. Some of these categories are inspired by the view of tree induction as a heuristic *state-space search* in the space of possible trees. Approaches in the first category directly control tree size, either by terminating search early (pre-pruning) or by editing the tree produced by the search (post-pruning), engaging in a second search of edited versions of the tree. The second category's techniques modify the space of states (i.e., trees) searched. The third category's algorithms modify the search algorithm itself. Those in the fourth category restrict the database, either by removing cases or by eliminating certain case features from consideration by the search process. Finally, methods in the fifth category simplify decision trees by translating them into another data structure, such as a decision graph or a set of rules.

Our own interest in decision tree simplification stems from our interest in developing a practical case-based reasoning (CBR) tool. Our effort follows that of others who use decision trees to retrieve stored cases (e.g., REMIND (Barletta, 1994)). We will discuss the applicability of the decision tree literature we review to the development of tree-based case retrieval in section 4.2.

In section 2, we review decision tree induction and discuss issues concerning tree simplification. Section 3 introduces our framework for categorizing tree-simplification approaches, and then surveys and critiques algorithms in each category. In section 4, we review the results of the survey, summarize an empirical comparison of selected algorithms, and discuss the applicability of these tree induction approaches to case retrieval. Finally, we conclude and discuss future research goals in section 5.

2 Decision tree induction

This section first introduces tree-based classification and decision tree induction. It then discusses issues pertaining to tree simplification.

2.1 Classification and induction

Decision tree induction algorithms have long been popular in machine learning, statistics, and other disciplines for solving classification and related tasks¹ (Morgan & Sondquist, 1963; Hunt et al., 1966; Friedman, 1977; Breiman et al., 1984; Quinlan, 1986, 1993a). A decision tree can be used to classify a query (or test) case as follows. Given a query q to classify, a tree is traversed along a path from its root to a leaf node, whose class label is assigned to q . Each internal node contains a *test* that determines which of its subtrees is traversed for q . A test typically evaluates a *feature* used to describe cases, or a (e.g., Boolean or linear) combination of features. For example, the test in the root node of the tree shown in Figure 1 assesses the value of the feature *location*, and each of the children of that node represents cases with a different value of *location*. The output of a logical test (e.g., $location = city$ or $f_1 = v_1 \wedge f_2 = v_2$) is typically a Boolean value.

Figure 1 displays a simplified decision tree that categorizes homes in terms of their discretized

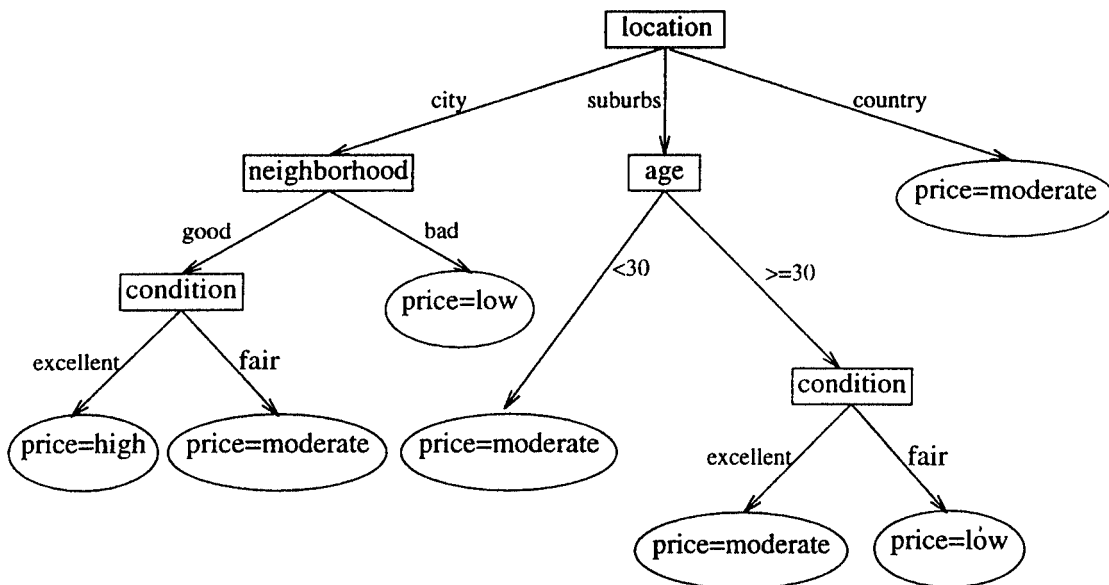


Figure 1 A decision tree. Rectangles represent internal node; ovals represent leaf nodes

¹Decision trees are also frequently used to model continuous functions (Breiman et al., 1984; Quinlan, 1993b), but we limit this discussion to classification tasks.

price. Each path from the root to a leaf represents a *rule* for inferring class membership. For instance, from Figure 1 we can derive:

If $location = city \wedge neighbourhood = good \wedge condition = excellent$, then $price = high$

The conjunction of tests on the path is the rule's premise (left-hand side or antecedent) and the class label of the leaf is its conclusion (right-hand side or consequent). A rule provides an *explanation* for a query's classification.

A generic decision tree induction algorithm is shown in Figure 2. The algorithm has four inputs:

1. C , a set *training cases*, each defined by a set of features and their respective values, and a class label.
2. I , a set of candidate *tests* that *partition* (or *split*) a set of training cases into subsets.
3. $eval()$, a heuristic *evaluation function* that assesses the quality of a given test and resulting partition.
4. $stop()$, a *stopping criterion* function that defines when to terminate tree expansion.

The algorithm outputs a decision tree T whose leaves are classifiers, each typically bearing a single class label. Decision trees are usually *induced* from the root downwards using a recursive divide-and-conquer algorithm. `make_tree` induces T and performs some post-processing operations on it, which might return T unchanged, prune it, or transform it into another data structure (e.g., a set of rules).

`induce_tree` outputs a single tree T . It inputs a subset C' of cases in C , a set of tests I , a test evaluation function $eval()$, and a stopping criterion function $stop()$. `induce_tree` performs a hill-climbing search without backtracking, where states correspond to decision trees. The initial state is a tree with a single node, the root, containing all cases C . State traversal corresponds to tree expansion, represented by a recursive call to `induce_tree`. Each call to `induce_tree` creates a node N to contain the input cases C' . The “best” test used to partition the cases C' is determined by `best_test_and_partition`, which uses function $eval()$ to evaluate alternative tests and

Key: C : Set of labelled training cases

I : Set of tests, where each test is a function from cases to values

$eval(test() \in I, C' \subseteq C)$: Yields quality of a test for splitting C'

$stop(P)$: Function determining whether to terminate tree expansion

T : The induced tree

$best()$: The best test at a node as judged by $eval()$

P : Vector of case subsets returned by $best(C')$

V : Vector of values returned by $best(C')$ (one per subset in P)

N : A node in T

```

make_tree( $C, I, eval(), stop()$ ) =
   $T := induce\_tree(C, I, eval(), stop())$ 
  RETURN  $post\_process(T, C, I)$ 

induce_tree( $C', I, eval(), stop()$ ) =
  { $best(), P, V$ } :=  $best\_test\_and\_partition(C', I, eval())$ 
  IF ( $stop(P) = TRUE$ )
    THEN  $N := create\_leaf\_node(C')$ 
    ELSE  $N := create\_internal\_node(C', best())$ 
    FOR  $j := 1$  TO  $|V|$  DO
       $subtree(N, V_j) := induce\_tree(P_j, I, eval(), stop())$ 
  RETURN  $N$ 

```

Figure 2 A generic algorithm for inducing decision trees

resultant partitions, returning the best test $\text{best}()$, the partition $P_{\text{best}()}$ imposes on C' , and the corresponding list of values $V_{\text{best}()}$ outputs (i.e., one for each subset of C' in P). If the stopping criterion is not satisfied, the tree is expanded.

The set of goal states is defined by $\text{stop}()$, which terminates tree expansion when it returns *true*. For example, when defined as the *homogeneity criterion*, function $\text{stop}()$ yields *true* if all cases in C' have the same class label. The homogeneity criterion may be regarded as the default stopping criterion, since it serves as a component in virtually every proposed stopping criterion. The $\text{stopping_criterion}$ function determines whether to define N as a leaf node or as an internal node. If the latter, then the tree is expanded: node N is made into an internal node and a subtree is induced for each subset P_j of P by a recursive call to induce_tree and linked to N by an edge labeled with the test output value V_j from V . If the former, then the node N is made into a leaf node, whose class label is determined by the cases C' it contains; usually, the class label selected is the label shared by the majority of cases C' .

Tree induction algorithms, such as our generic algorithm, must employ computationally efficient heuristics because constructing an optimal decision tree is an NP-complete task (Hyafil & Rivest, 1976); search complexity grows exponentially with tree depth (i.e., path length from root to lowest leaf). Thus, our algorithm greedily selects a test at each node that maximizes $\text{eval}()$. This limits make_tree 's complexity to $O(|C|f^2)$, where f is the number of features (Utgoff, 1988a). While efficient, this greedy search is subject to *horizon effects* (i.e., it selects tests based on local rather than global measures), and so risks being trapped by local minima. In this survey, we discuss several proposed solutions for reducing this risk.

The generic algorithm in Figure 2 summarizes how trees can be induced without detailing "standard" function definitions and other design decisions. Some of these are worth mentioning. First, while most tree generation algorithms induce *univariate* trees, where each test assesses the value of a single feature, several researchers have recently proposed methods for inducing *multivariate* trees (e.g., Brodley & Utgoff, 1995), where tests can be functions on multiple case features. Second, a typically-adopted constraint is that a test cannot be reused along a tree path. Finally, $\text{eval}()$ is usually a measure of a given partition's homogeneity; the most preferred partitions are those in which a large majority of the instances in each subset P_j belong to the same class. This is frequently accomplished by defining $\text{eval}()$ using an information-theoretic (Quinlan, 1986) or statistical measure (Breiman et al., 1984).

In most research on classification, the database is divided into a *training set* and a *test set*. The training set is used to generate the decision tree, while the test set is used to assess its predictive accuracy. Following Breiman et al. (1984) we will refer to a classification tree's performance on the training set as *resubstitution error* and *resubstitution accuracy*². The terms *error* and *accuracy* alone refer to performance on the test set; we will also sometimes refer to them as *predictive* or *generalization error/accuracy*. Whereas it is relatively easy to generate a tree with zero resubstitution error (in the extreme case, by expanding the tree until each leaf contains a single case), it is harder to generate trees that are highly accurate when applied to previously unseen test cases, as explained in section 2.2.

2.2 Tree simplification motivations

Induced decision trees can often be opaque, preventing them from concisely explaining classification behavior and, therefore, from satisfying the needs of domain expert and novice users. Therefore, many researchers have proposed methods for simplifying these trees, which we survey in section 3. Decision tree complexity is usually measured as the number of nodes in the tree. One exception to this is multivariate trees, whose internal nodes are more complex than those of univariate trees such as the example shown in Figure 1; multivariate trees will be discussed in section 3.2. This section briefly examines causes for complex decision trees and their implications.

²Synonymous terms include *apparent error* and *apparent accuracy*.

Decision trees can become cumbersome large for several reasons. One cause is a *mismatch of representational biases* (Schaffer, 1992a,b). Some tree representations are unable to concisely model some target concepts; the targets have high model complexity when expressed using these representations. In contrast, other representations can often greatly reduce model complexity. Thus, this problem provides strong motivation for including a range of representations.

Another cause for large trees is *noise*. When cases have a large amount of feature noise (i.e., mislabeled feature values) or class noise (i.e., mislabeled class values), the induction algorithm may expand the tree too far based on irrelevant case distinctions (Quinlan, 1986). Noise can cause some irrelevant features to be included among the selected tests. This in turn causes *overfitting* (Cohen & Jensen, 1997), where trees model both the target concept and the inherent noise. This is a pervasive problem given that cases recorded from most applications will contain some degree of noise. As we will show in section 3.1, several methods now exist that prune trees in an attempt to prevent the fitting of noise, but there is no one pruning method that works best for all tasks (Schaffer, 1993), and comparisons with other tree simplification approaches are both rare and noncomprehensive.

Overly large trees can become *fragmented*, having many leaves with only a few cases per leaf. Such leaf nodes are more error-prone classifiers than leaf nodes containing many cases, and so are more likely to be influenced by noise. These leaf nodes (or more precisely, the tree paths corresponding to them) are sometimes referred to as *small disjuncts* (Holte et al., 1989; Quinlan, 1991; Ting, 1994), regions of the problem space with low frequencies of occurrence. Thus, another reason for simplifying trees is to eliminate small disjuncts by pruning leaves having few cases. This yields nonhomogeneous leaves; some function of the cases' class labels (e.g., *majority vote*) is used to determine a leaf's class label.

Overly large trees also often exhibit a structural problem called *subtree replication*, which occurs in trees representing disjunctive concepts such as the one shown in Figure 3. To obtain perfect accuracy using a univariate decision tree, the subtree representing the second disjunct must be duplicated in the subtrees corresponding to the negations of *A* and *B*. Subtree replication is one cause of tree fragmentation, and suggests a mismatch of representational biases, as explained above.

Whatever the cause, a large, fragmented tree is clearly difficult for users to understand. An example (from Quinlan, 1986) is the tree in Figure 4, based on the small database shown in Table 1. In contrast, the tree in Figure 5 is a simpler tree derived from the same database. In addition to being difficult to understand, a complex tree often performs more poorly in classification tasks because

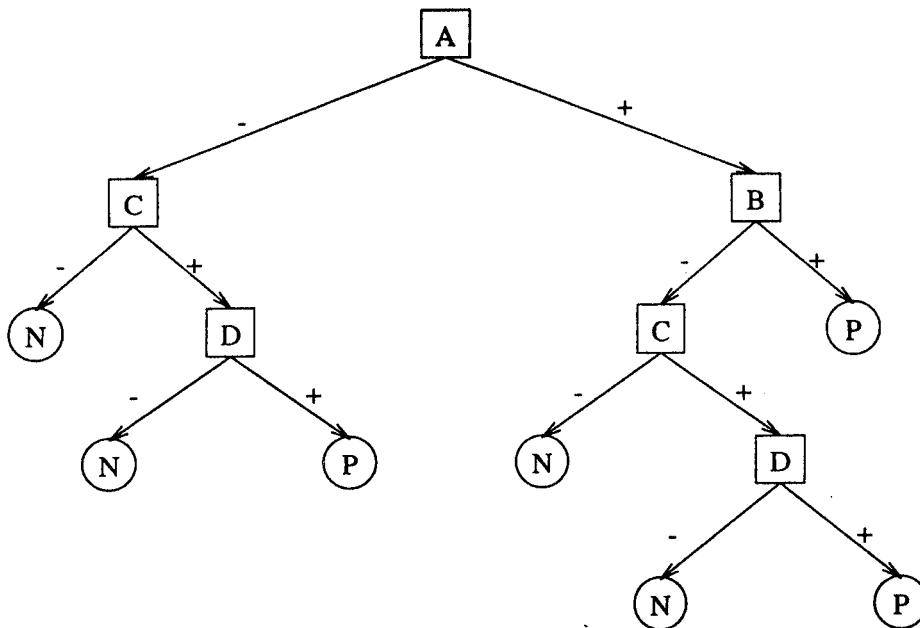


Figure 3 A decision tree representing $(A \wedge B) \vee (C \wedge D)$

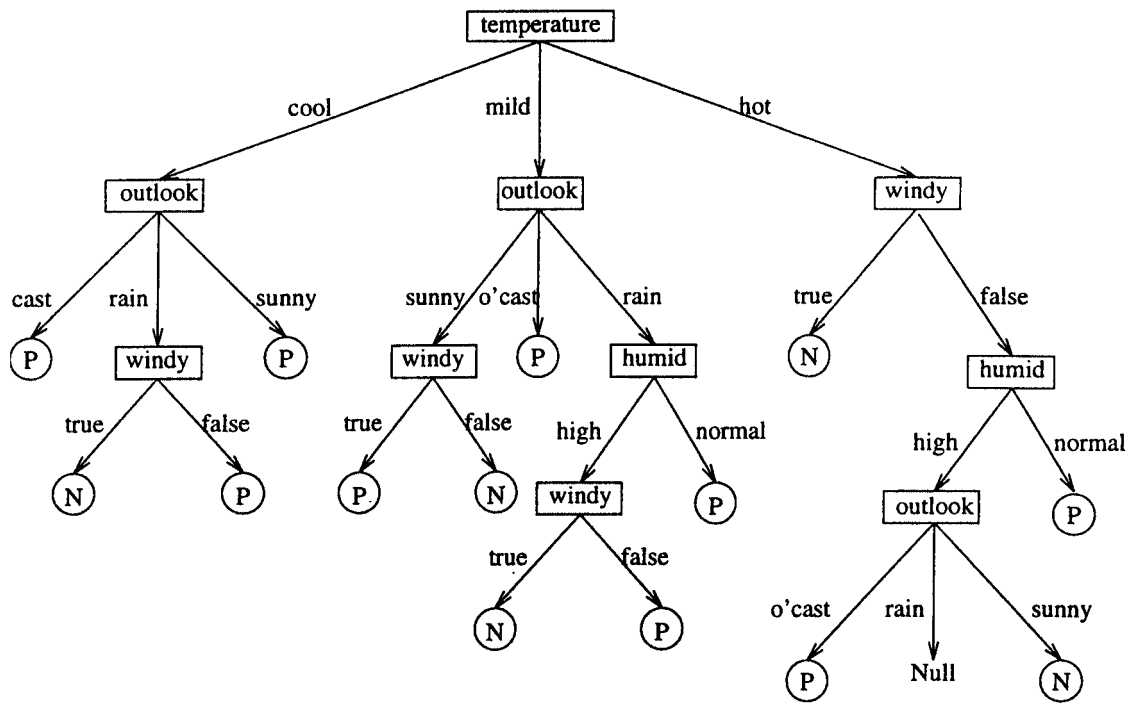


Figure 4 A complex decision tree (from Quinlan, 1986) induced from Table 1

Table 1 A small database (from Quinlan, 1986)

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

small disjuncts are more likely to be error-prone than large disjuncts (Quinlan, 1991). However, it is difficult to determine whether the distinctions induced from the training cases are spurious, because pruning the tree normally *reduces* classification accuracy on the training cases, regardless of its effect on performance with unseen test cases.

2.3 Relation between simplicity and accuracy

The methods we review are intended to improve the comprehensibility of decision trees, while maintaining or improving accuracy. These dual goals are often correlated; an algorithm may yield

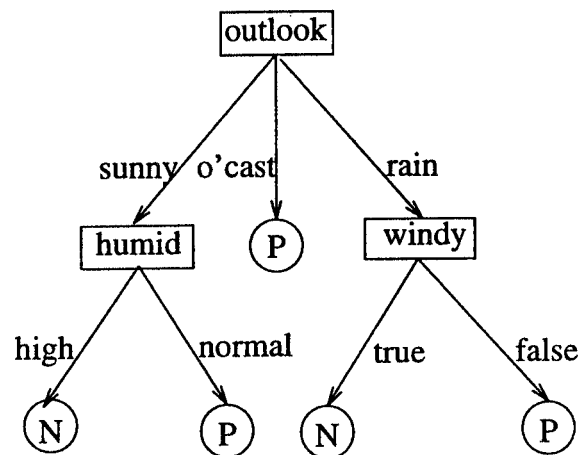


Figure 5 A simpler decision tree (from Quinlan, 1986), also induced from Table 1

improvements in both comprehensibility and accuracy. Indeed, tree simplification methods were originally introduced to tolerate noise in training cases and were found to improve accuracy for many noisy data sets (Breiman et al., 1984; Quinlan, 1986, 1987a).

The amount of simplification to apply has been the subject of much debate. The sacrifice in accuracy resulting from drastic reductions in tree size is often surprisingly small (e.g., Holte, 1993; Murphy & Pazzani, 1994; Auer et al., 1995). However, more conservative reductions in size can sometimes produce significant, if small, improvements in accuracy (Elomaa, 1994) that can be crucial in some practical applications (Danyluk & Provost, 1993). Thus, many authors describe a *trade-off* between the accuracy and simplicity of decision trees (e.g., Bohanec & Bratko, 1994), and some have proposed tree construction methods that favour one or the other criterion. Some of these methods sacrifice comprehensibility for the sake of accuracy (e.g., methods that create a *set* of decision trees (e.g., Oliver & Hand, 1995) or methods that yield small gains in accuracy at the price of large increases in tree size (e.g., Webb, 1996)), while others fine-tune the balance between accuracy and comprehensibility (e.g., Breiman et al., 1984; Bohanec and Bratko, 1994; Holder, 1995).

Schaffer (1992a,b, 1993) argues that the exact trade-off between accuracy and simplicity is indeterminable on the basis of any randomly selected training set, because it is impossible to determine from the training cases alone how much of the tree's complexity is due to noise and how much is due to the underlying model's "true" complexity relative to the representation used for learning. Rather, additional domain-specific knowledge, external to the training set, must be employed to estimate the noise level in the training cases, the underlying model's complexity, and, consequently, the degree of tree simplification to apply. Knowledge-poor tree induction algorithms do not exploit such information. Therefore, they each encode an implicit *bias* (i.e., assumption) concerning the model's complexity and level of training noise, influencing the amount of tree simplification performed. These algorithms also differ in their representational biases. For example, many algorithms assume that the model is expressible as a disjunctive normal form expression in which the tests are univariate functions on case features, while other algorithms assume that tests can be linear combinations of feature values (e.g., Brodley & Utgoff, 1995). Naturally, some representations are more appropriate for some tasks than others. If no inferences can be made concerning what biases are best for a given database, then all possible algorithms must be compared to determine which performs best for it (Schaffer, 1993). Fortunately, some authors have shown that a database's characteristics can be profitably used to heuristically select inductive biases (e.g., Brodley, 1993, 1995a; Ting, 1994).

In summary, the problem of identifying the biases enabling simplification procedures to reduce complexity without sacrificing accuracy is a subject of much debate and research. Although our

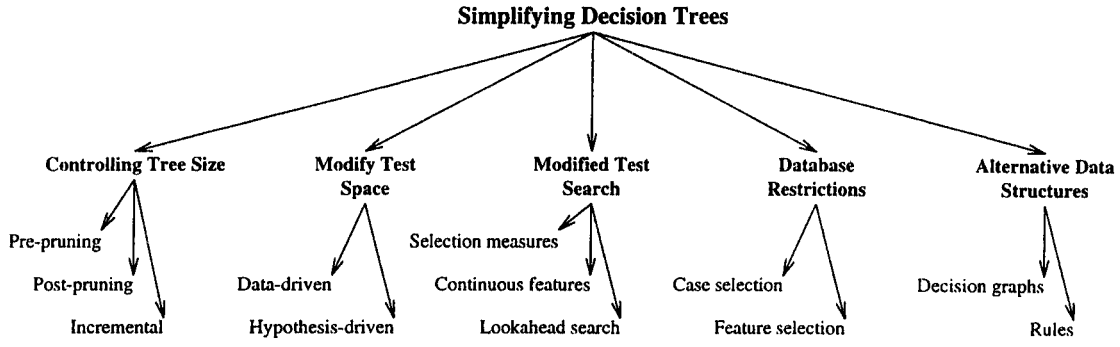


Figure 6 A framework of tree-simplification procedures

main focus will be on tree simplification, we cannot neglect the question of accuracy. Simplification procedures that significantly reduce classification accuracy are unlikely to be useful.

3 Tree-simplification approaches

In this section, we introduce a framework that categorizes tree-simplification procedures. We then summarize methods that fall into each of these categories.

Figure 6 displays our framework. Briefly, it consists of five³ top-level categories of tree-simplification approaches. The first and most commonly used set of procedures directly *control tree size* either by *pre-pruning* (i.e., imposing a non-trivial stopping criterion on tree expansion), *post-pruning* (i.e., deleting subtrees after inducing the tree), or by *incrementally resizing* the tree. Procedures in the second category *expand the set of tests* considered to include *multivariate tests* based on constructed feature combinations, subdivided according to whether feature construction is *data-driven* or *hypothesis-driven* (i.e., using a previously-induced tree to suggest constructed features).⁴ These methods effectively expand the set of trees that are examined during search by modifying the representation of trees. The third category, *modifying the search for tests*, includes methods that modify this search by using alternative test evaluation functions, by modifying the representation and selection of continuous features, or by modifying the search algorithm itself. Procedures in the fourth category, *database restrictions*, simplify trees by reducing the size of the databases or the set of descriptive features characterizing the cases. The fifth and final set of procedures transform trees into *alternative data structures* (e.g., rules, graphs). The following five subsections each detail one of these categories.

We selected this particular framework because, in addition to abstracting a small number of meaningful categories, it clarifies how different types of (optional) simplification procedures can be combined in a decision tree induction algorithm. Methods from different categories can usually be combined in the same algorithm, perhaps adding to each other's strengths. For example, choice of method to control tree size can often be made independently from that of the method for searching tests, or of the space of tests to search.

The ordering of these categories partly reflects the order in which they received research attention. We chose methods for controlling tree size first due to their enormous popularity. Multivariate methods appear next because most are simple representational extensions of the generic algorithm and integrate easily with post-pruning methods. Methods for modifying the search for tests apply to both preceding categories, and so seem well suited for discussing the third. The fourth category is

³One of our goals here is to convince readers that post-pruning methods, while popular, are not the only approaches for simplifying decision trees.

⁴In addition to the research on multivariate tests, other relevant research on alternate tests includes studies assessing whether binary- or multi-valued tests produce smaller trees (see Fayyad & Irani, 1992; Kononenko, 1995). However, we do not feel that enough research currently exists on this question to justify an additional category.

more general, not limited to decision trees, and likewise is applicable to the preceding categories. The final category comprises an even more radical approach for simplifying decision trees: replace them with alternative data structures.

3.1 Controlling tree size

The most popular methods for simplifying decision trees explicitly control the size of the induced tree during construction. These include pre-pruning (section 3.1.1) and post-pruning (section 3.1.2) methods. We also summarize incremental tree-sizing procedures in section 3.1.3 and briefly mention alternative post-processing techniques in section 3.1.4 and hybrids of decision trees and other classifiers in section 3.1.5. Readers already familiar with pruning methods may want to skim sections 3.1.1 and 3.1.2.

The potential benefits of pruning appear to depend on characteristics of the database. Fisher and Schlimmer (1988) found that the benefits of pre-pruning are increased when the training sample is large and when there is high statistical independence between class membership and the defining features. Similarly, Kim and Koehler (1994) showed that the utility of post-pruning is increased when training samples are large and have highly skewed class distributions. Thus, pruning biases should be adjusted based on these data characteristics, and possibly on others.

3.1.1 Pre-pruning

Pre-pruning methods simplify decision trees by preventing the induction of a “complete” tree with the default homogeneity stopping criterion, where each leaf is homogeneous with regard to class membership and the tree’s resubstitution error rate is zero. To do this, they supplement the default with a non-trivial stopping criterion in the function `stop()` in `make_tree` (Figure 2). A simple form of pre-pruning that arbitrarily limits trees to a depth of two performs surprisingly well (Holte, 1993; Auer et al., 1995); this method will be discussed further in section 3.3.2. Usually, however, the stopping criterion estimates the performance gain expected from further tree expansion and terminates expansion when no (or not enough) gain is expected. In comparison, the `post_process` function is trivial, returning the input tree unchanged. Pre-pruning methods are generally more efficient than post-pruning methods because they terminate tree generation earlier and do not include a post-processing step.

The simplest pre-pruning algorithm imposes a minimum threshold on the test selection measure (Gleser & Collen, 1972; Rounds, 1980; Quinlan, 1986). After selecting the best test `best()` for partitioning the case cluster C' as determined by `eval()`, but before expanding the tree, `stop()` assesses the quality of the partition created by `best(C')` and terminates tree expansion if this value fails to exceed a prespecified threshold, even if homogeneity has not been attained.

Quinlan (1986) found that a threshold stopping criterion tends to filter *both* relevant and irrelevant tests. In addition, using the same measure for both test selection and pre-pruning is problematic when it is local to a node because the absolute value of local measures typically vary with sample size (Gleser & Collen, 1972; Rounds, 1980; Quinlan, 1986). Rather than using a local test selection measure that is insensitive to the number of cases in a node (Li & Dubes, 1986; Zhou & Dillon, 1991; Kalkanis, 1993) or adopting a global measure (Sethi & Sarvarayudu, 1982), researchers solved this problem by using stopping criteria that use a *different* measure than that used for selecting tests. For example, Quinlan (1986) used the χ^2 test for stochastic independence as the stopping criterion in ID3, while using an information-theoretic measure to evaluate and select tests in `eval()`.

Pre-pruning methods yield inconsistent performance because they suffer from a *horizon effect* (Breiman et al., 1984; Quinlan, 1993a), which can cause them to terminate tree construction prematurely. That is, the stopping criterion might terminate tree expansion even when further expansion might produce a tree whose expansion would not be prohibited by this same criterion. Consequently, these approaches have been abandoned in favour of post-pruning methods. However, pre-pruning may merit reconsideration for large-scale practical applications because

they are more efficient. Research on lookahead and feature-construction algorithms (sections 3.3.3 and 3.2.2, respectively) might yield solutions to the horizon effects problem, either by projecting forward to determine whether the current tree corresponds to a local minimum or by testing whether an expanded set of candidate tests yields better performance.

3.1.2 Post-pruning

Post-pruning is the most frequently discussed tree simplification method. Post-pruning (or simply *pruning*) algorithms input an unpruned tree T and output a pruned tree T' , formed by removing one or more subtrees from T . They do not usually search all possible T' , but instead use heuristics to guide their search. Pruning replaces a subtree with a leaf, transforming an internal node into a leaf node. In contrast to pre-pruning, post-pruning methods employ a nontrivial `post_process` function, but use the default homogeneity stopping criterion in generating the tree. Figure 7 shows a pruned version of the tree in Figure 4. Ratios under nonhomogeneous leaves indicate the proportion of their cases that are in the majority class.

If a decision tree is grown using the homogeneity stopping criterion, it will contain no resubstitution errors; thus, pruning can only impair its resubstitution accuracy on the training cases. However, when the tree overfits the training cases (i.e., by modelling its noise), pruning can often improve its accuracy on unseen *test* cases. For example, given a set of m training cases, and assuming a leaf node containing n training cases is labeled by its majority class in $n' \leq n$ cases, then it contributes $(n - n')/m$ to the tree's resubstitution error. Not surprisingly, low-level internal nodes subsuming few cases and having similar class distributions to their children tend to be pruned first because they contribute least to resubstitution accuracy. Post-pruning methods employ different functions to estimate whether pruning a node will impair or improve accuracy on test cases.

Several researchers offer evidence that pruning increases classification accuracy (Breiman et al., 1984; Quinlan, 1987a; Mingers, 1989a; Weiss & Indurkha, 1994a,b). Many argue that pruning is most effective when cases have high noise levels. However, Schaffer (1993) provides evidence that this is not always the case (see section 2.3).

Mingers (1989a) and Esposito et al. (1993, 1995a) review and compare many of the existing post-pruning methods. Table 2 summarizes these methods and Table 3 compares their capabilities.

Some post-pruning methods divide the training data into two subsets, one for growing the tree and one for subsequent pruning (see Table 2). Following Esposito et al. (1993), we refer to these as the *growing* and *pruning* sets, respectively. The growing set is typically used to generate a tree and then to generate a (sub)set S of its pruned variants, while the pruning set is used to select the best tree

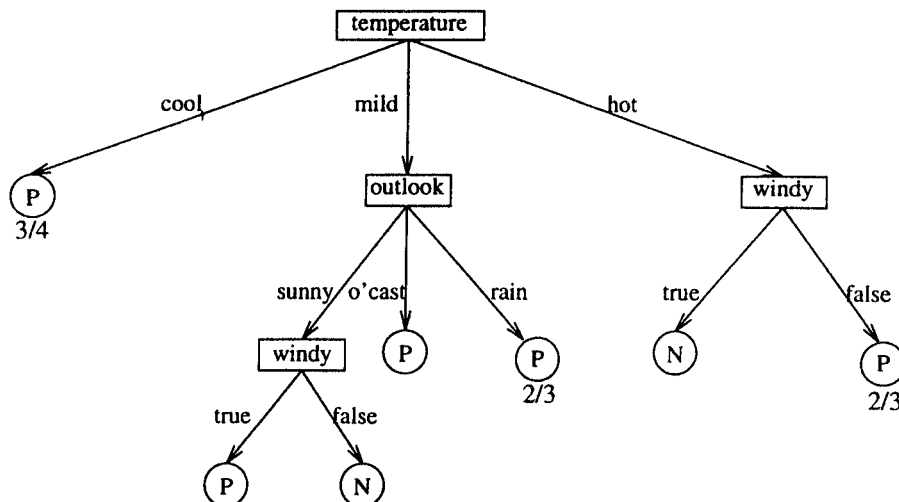


Figure 7 A tree produced by post-pruning the tree shown in Figure 4

Table 2 Summary of post-pruning algorithms (from Esposito et al., 1995)

<i>Method</i>	<i>Pruning set</i>	<i>Strategy</i>	<i>Characteristics</i>
Minimal Cost-Complexity (MCCP) (0-SE, 1-SE, CV) (Breiman et al., 1984)	0-SE: yes 1-SE: yes CV: no	bottom-up (any internal node)	Can use cross-validation or standard error to estimate accuracy.
Reduced Error Pruning (REP) (Quinlan, 1987a)	yes	bottom-up	Finds the optimally pruned tree wrt the pruning set.
Minimum Error Pruning (MEP-2) (Cestnik and Bratko, 1991)	yes	bottom-up	Based on the m-probability estimate of the error rate.
Critical Value Pruning (CVP) (Mingers, 1989a)	yes	bottom-up	Uses a critical value threshold on the test selection measure.
Pessimistic Error Pruning (PEP) (Quinlan, 1987a)	no	top-down	Uses the continuity correction.
Error-Based Pruning (EBP) (Quinlan, 1993a)	no	bottom-up	Estimates confidence intervals on the training set.

Table 3 A comparative summary of post-pruning algorithms. ‘M’ refers to Mingers (1989a), ‘E’ to Esposito et al. (1995)

<i>Method</i>	<i>Benefits</i>	<i>Limitations</i>
MCCP (M, E)	Accurate (M, E: 0-SE) ‘Good’ tree size with 0-SE (E)	Inaccurate with 1-SE (E) Unstable accuracy (CV) (E) Overprunes with 1-SE (E)
REP (M, E)	Accurate (M, but not E) ‘Good’ tree size (E) Optimal wrt pruning set (E)	
MEP-1 (M)		Underprunes Sensitive to number of classes Unstable under noise Low accuracy Assumes uniform class distribution
MEP-2 (E)		Underprunes High complexity
CVP (M, E)		Low accuracy (M, E) Underprunes (E)
PEP (M, E)	Efficient (E) Very Accurate (E, but not M)	Underprunes (M but not E) Unstable under noise (M) Top-down search causes horizon effect (E)
EBP (E)	Very Accurate	Underprunes

in S . An exception is *reduced error pruning* (REP), which uses the pruning set to generate the pruned trees as well as to select among them.

An advantage of pruning set methods, according to some authors (e.g., Mingers, 1989a), is that they produce a *set of trees*, rather than a single tree. This may be especially desirable when an expert is available who can compare and select among the trees, because the expert might disagree with (and override) the selection made by the algorithm. Similarly, Oliver and Hand (1995) have shown that a majority vote among multiple induced trees can often increase predictive accuracy.

One observed disadvantage of dividing the training set into two subsets is that it decreases the number of training cases involved in tree induction, which is not always advisable with small training sets. In such cases, *cross-validation* (CV) can be used instead. CV methods divide the training cases into N equal-sized blocks. In each of N iterations, a different block is used as the

pruning set, with the remaining cases serving as the growing set. In this way, all training cases contribute to the growing set on most iterations. The best of the N trees is chosen by criteria such as accuracy and simplicity. The drawback of CV is that N trees must be produced rather than one; thus, CV is less efficient than using a single growing set and pruning set.

Typically, the pruning set is smaller than the growing set. This design decision likely embodies a bias; small growing sets tend to yield overly-small decision trees because their lower-level subtrees will be pruned (i.e., they will not contain enough instances to pass significance tests used by many pruning methods). However, while this is a good reason to maximize the size of the growing set, reducing the size of the pruning set reduces the certainty on estimates of predictive accuracy used in pruning.

While Mingers (1989a) found the methods that employ a separate pruning set to be superior to those that do not, Esposito et al. (1993, 1995) disputed this claim on both methodological and empirical grounds. They claimed that Mingers' empirical methodology gave the separate pruning set methods more training cases than other methods. They found that, when both types of methods were trained on the same amount of data, no overall performance difference was found. However, Mingers' findings concerning the relative merits of pruning methods *within* each of these two categories are useful. Also, Mingers' results are valuable because he empirically compared five pruning methods together with four test selection methods; he found no interaction between the pruning and test selection methods. This suggests that it may be possible to study these two factors separately.

Table 2 characterizes post-pruning algorithms by their search strategy. The *bottom-up* pruning strategy starts with the lowest internal nodes in the tree (i.e., those whose children are all leaves) and prunes those that meet the algorithm's pruning criterion. This process iterates with the new lowest internal nodes, etc., stopping when the pruning criterion is not met. The bottom-up order of evaluation prevents a node from being pruned if any of the nodes in its subtree should not be pruned by the same criterion. In contrast, the top-down strategy proceeds from the root downward, considering nodes for pruning, and stopping with nodes that should be pruned. This strategy risks pruning a node even when one of its descendants should not be pruned by the same criterion.

Mingers (1989a) compared five methods on six data sets, while Esposito et al. (1995a) compared six pruning methods on 11 data sets. We summarize the findings of both studies in Table 3. Esposito et al. (1995) compared the sizes of trees produced by the various post-pruning methods against an "ideal-sized" tree, produced by REP pruning (discussed below) using the test set as the pruning set. This evaluation is based on the assumption that REP produces optimally-pruned trees relative to the pruning set. Pruning methods that typically produce trees larger than this standard *underprune*, those that produce trees smaller than the standard *overprune*, and the rest are said to produce "good" tree sizes. Trees are compared to the standard by measuring overall tree size (i.e., number of nodes) rather than tree structure. As a result, correct tree size is not perfectly correlated with classification accuracy. We now summarize the main features, strengths, and limitations of each of these algorithms.

1. **Minimal Cost-Complexity Pruning (MCCP).** The first post-pruning method, MCCP,⁵ was developed by Breiman et al. (1984) as part of their well-known CART system. They defined *cost complexity* for a tree T as the sum of T 's resubstitution error (cost) and the product of the number of T 's leaves (complexity) and a user-set parameter which determines the trade-off between cost and complexity. The method consists of two steps. First, a sequence of increasingly smaller trees, beginning with the induced tree and ending with the root tree, is generated by successively pruning the subtree yielding minimal cost complexity. The strategy is bottom-up; thus, the trees in the sequence are nested and all match on the root node. However, in contrast to other bottom-up strategies, each iteration considers *all* (i.e., not only the lowest) internal nodes at the same time for pruning. In the second step, the "best" tree from this sequence is selected based on its relative

⁵This is sometimes referred to as *error-complexity pruning* or simply *cost-complexity pruning*.

size and, depending on the MCCP variant, either (1) pruning set accuracy or (2) cross-validation (CV) accuracy on the training set. MCCP outputs the smallest tree whose error on the pruning set is not more than one standard error (1-SE) greater than the lowest error observed among these trees.

Esposito et al. (1995a) introduced a 0-SE variant of MCCP in which the 1-SE restriction was not imposed (i.e., selection is based only on accuracy). This yields four variants: using a pruning set *versus* using CV and using 0-SE *versus* 1-SE. They compared these four MCCP variants to some of the methods in Table 2. They found that the 1-SE variants overpruned, which is understandable given their conservative criterion for subtree retention, and were less accurate, especially the 1-SE CV variant. In contrast, the 0-SE CV variant performed as well as most other methods in both size and accuracy, while the 0-SE pruning set variant outperformed most other methods in both respects. Crawford (1989) found that the CV variants yielded trees with highly variable accuracy, especially for small training samples.

2. **Reduced Error Pruning (REP).** REP, developed as part of the ID3 system (Quinlan, 1987a, 1993a), also uses a pruning set. However, while MCCP uses the growing set for both induction and pruning, REP uses it only for induction and uses the pruning set to both generate and evaluate pruned trees. REP prunes a tree from the bottom up, pruning all nodes it finds that, when pruned, do not decrease the tree's pruning set accuracy. According to Esposito et al.'s analysis, REP is guaranteed to produce the smallest pruned tree with the lowest error rate with respect to the *pruning set*. These authors found that REP performed as well as most of the other methods in terms of test accuracy and better than most in terms of tree size.
3. **Minimum Error Pruning (MEP).** MEP was introduced by Niblett and Bratko (1986; MEP-1) and improved by Cestnik and Bratko (1991; MEP-2). We focus on the later version. This method computes probabilistic estimates of the error rates of internal nodes based on the growing set. A parameter m determines the contribution of *a priori* class probabilities to the error rate computation. Examining internal nodes from the bottom up, MEP prunes an internal node if its subtree's estimated error is greater than it would be if the node were replaced by a leaf. A set of pruned trees is generated by repeating this procedure for various values of m , and a domain expert selects the best tree. Esposito et al. (1995a) automate this selection process to select the smallest tree with lowest pruning set error. They report that MEP-2 tends to underprune, but its accuracy is comparable to that of most other methods.
4. **Critical Value Pruning (CVP).** Mingers' (1987) CVP method, which can optionally be used with a pruning set, consists of two steps. First, a threshold parameter cv (*critical value*) is applied to the test selection function (e.g., `eval()` in Figure 2). During bottom-up pruning, nodes with training cases C and test `test()` are pruned if $\text{eval}(\text{test}(), C) \leq cv$. Repeating this procedure with different critical values yields a set of candidate pruned trees. Second, a pruned tree is selected based on (1) its accuracy (i.e., a probabilistic measure based on either the growing set or pruning set error) and (2) its significance, as determined by a contingency-table G statistic, with leaf nodes as rows and classes as columns. However, whereas G might be appropriate to determine the statistical significance of a table, Esposito et al. (1995a) argue that the table's G values cannot be used to compare the "relative significance" of different trees. They also argue that the probabilistic measure is not valid, and therefore tested CVP using a pruning set. They found that CVP has a strong tendency to underprune and selects trees with comparatively low predictive accuracy.
5. **Pessimistic Error Pruning (PEP).** PEP (Quinlan, 1987a) was developed in the context of ID3 and, like MCCP's CV variants, does not use a separate pruning set. PEP attempts to compensate for the overly *optimistic* estimates based on resubstitution error; these estimates are overly optimistic since error rates on training data are typically lower than on test data. PEP attempts to get a more accurate error estimate by imposing a continuity correction for the binomial distribution by adding 0.5 to the number of errors associated with each node. Also, subtrees are not pruned only if their corrected error estimate is at least one standard error less than the estimated error of their root node.

Although Esposito et al. (1995a) found PEP to be one of the most accurate algorithms they tested, it has certain limitations. PEP is the only method they examined that uses a top-down pruning strategy, which encounters the same problem as pre-pruning (section 3.1.1): the tree might be pruned at a node even when the node's descendants would not have been pruned by the same criterion. Esposito et al. (1993) describe examples of this. Second, PEP can sometimes fail to prune trees even when generated from random data, as Esposito et al. (1995a) demonstrate. Third, as Quinlan (1993a) himself notes, it is improper to treat the data sample used to create a tree as a random sample representing the population for statistical purposes. Despite these objections, PEP did record high accuracies, and its top-down pruning strategy would likely make it more efficient than the other methods.

6. **Error-Based Pruning (EBP).** EBP is a more pessimistic descendant of PEP that is used in C4.5 (Quinlan, 1993a), a descendant of ID3. Given the set C of training cases at a node n , its majority class, and the number of cases not in that class, EBP interprets C as a binomially distributed sample, with well defined confidence limits, and estimates n 's error rate as the upper limit on its posterior probability distribution. In addition, EBP adds a novel pruning operation, called *grafting* by Esposito et al. (1995b), that allows a node to be replaced by one of its subtrees. This expands the search space, permitting a bottom-up procedure to prune the parent of a good node without pruning the good node, thus reducing EBP's vulnerability to horizon effects. Although EBP was designed to be more biased towards pruning (i.e., pessimistic) than PEP, Esposito et al. (1995a) found that it tends to underprune, but still attains higher accuracies than the other methods they tested.

In summary, PEP and EBP produced trees with the highest accuracy among the methods compared. Also, they were the only methods that do not require either a pruning set or expensive cross validation. However, they tended to underprune. In contrast, REP and the 0-SE variants of MCCP produced trees closest to the "correct" size (i.e., with the least underpruning or overpruning) and with good accuracies. These findings on post-pruning algorithms are preliminary; further investigation is needed before we can draw more definitive conclusions concerning their relative merits.

Several other post-pruning methods have been introduced, but their merits relative to those described above have not been systematically examined and space constraints prevent their more detailed discussion here. Crawford (1989) introduced a .632 bootstrap (i.e., an alternative evaluation strategy to, for example, cross validation) extension of MCCP in CART, and demonstrated gains in accuracy. Gelfand et al. (1991) described an efficient tree growing and pruning algorithm in an improved version of CART that is guaranteed to converge. It divides the training sample in half, and then iteratively grows the tree using one half while pruning using the other, exchanging these roles with each iteration. They reported evidence that their algorithm was faster and more accurate than CART, but produced *larger* trees. Forsyth et al. (1994) introduced a pruning method based on analysing the decision tree as a description of the training cases. Bohanec and Bratko (1994) used a dynamic programming technique to prune trees optimally and efficiently in the absence of noise. They speculated on how to extend this technique for noisy cases. Cockett and Herrera (1990) described a method for reducing binary decision trees to an irreducible form using principles from discrete decision theory. Vadera and Nechab (1994) propose using a cost-sensitive post-pruning approach that seems more appropriate for applications where some types of errors have more serious consequences than others. Finally, Quinlan and Rivest (1989) and Wallace and Patrick (1993) use minimum description length for both tree construction and pruning. We summarize this research in section 3.3.1.

Very recently, Cohen and Jensen (1997) proposed a powerful theory for explaining the causes of overfitting effects for classification tasks. Their *multiple testing theory* (MT) states that overfitting occurs when algorithms overestimate the gains in predictive accuracy from adding complexity (e.g., subtrees) to concept descriptions. In turn, overestimation has three primary causes:

1. *Number of models examined*: The probability that a decision tree inducer overfits its data is positively correlated with the number of models it examines (e.g., “multiple testing”).
2. *Variance of the accuracy estimates*: Small training sets are less likely to represent the underlying distribution and so are more likely to contribute to overfitting and inaccurate decision trees. Larger training sets tend to have less variance and thus diminished susceptibility to overfitting.
3. *Selecting a “best” tree*: Selecting the tree that maximizes some evaluation function also maximizes the probability of overfitting.

They tested their theory using a post-pruning algorithm that is consistent with MT theory: it uses a Bonferroni adjustment to test statistical significance when evaluating candidate tests (Feelders & Verkooijen, 1995). In tests on simple synthesized data sets, their algorithm produced the least overfitting and simplest decision trees among the four algorithms tested. It also had the highest accuracies for tasks where overfitting could occur. The comparison algorithms used either an unadjusted test of significance, EBP pruning, or a minimum description length (MDL) approach (see section 3.3.1). In summary, MT theory shows promise for suggesting designs of post-pruning algorithms.

3.1.3 Incremental tree sizing

Decision trees provide an abstract summary of the database that frees one from having to store the database and yields fast predictions. However, if users are willing to incur the storage costs of retaining the training cases and subsequently encountered test cases, they can adopt an algorithm that incrementally induces and refines decision trees (Schlimmer & Fisher, 1986; Utgoff, 1988a, 1989a, 1994). These algorithms progressively expand and contract trees. Utgoff’s (1994) ITI system permits the user to shift between an incremental mode (for accuracy) and a delayed update mode (for efficiency), and is relatively efficient in incremental mode. Iterative post-pruning extensions of the algorithm, discussed by Utgoff (1994), could prevent trees from being unduly influenced by random variations in the incoming stream of test cases. Since pruning is more beneficial with larger training samples (Fisher & Schlimmer, 1988; Kim & Koehler, 1994), it makes sense to revisit earlier pruning decisions after accumulating additional cases.

A less purely incremental approach is *dynamic pruning* (Langley, 1996, p. 232) or *virtual pruning* (Utgoff, 1994). In this approach, the decision tree structure can be created statically at the outset. However, it is *used* in a dynamic manner: the entire tree need not be used for classification. Instead, a fringe of nodes at the bottom of the tree is treated as virtual, and the boundary between real and virtual nodes changes dynamically as the system accumulates experience. Leaf nodes can be pruned, without being irrevocably lost, by being made virtual. They may later be recovered on the basis of subsequent experience. Fisher (1989) describes a similar approach for unsupervised learning tasks. Virtual pruning can be used in conjunction with incremental tree induction, as recommended by Utgoff (1994).

Thus, incremental induction and pruning algorithms allow a decision tree to be continually right-sized based on its experience, while retaining the other advantages offered by decision trees (e.g., a comprehensible knowledge structure and fast prediction). Although insufficient evidence is available, we hypothesize that incremental induction can produce more optimally-sized, accurate trees than algorithms where induction is limited to an initial training set. For example, incremental updates when combined with case selection (section 3.4.1) can reduce tree size and increase accuracy (Utgoff, 1989a, 1994). The price for these advantages is, of course, the requirement to store the database. However, methods to be discussed in section 3.4 suggest ways to prune the database itself.

3.1.4 Other post-processing procedures

Approaches for post-processing decision trees, other than post-pruning, have also been proposed. For example, Jordan (1994) discusses using post-pruning methods to initialize probabilistic decision trees, and using ridge regression to “shrink” the regression parameters to zero, but did not detail any advantages of this approach. Buntine (1992) discussed using smoothing and averaging techniques in

place of pruning, but did not compare tree complexity results. Murthy and Salzberg (1995) introduced a post-processing technique, decision tree *balancing*, that can be applied after pruning to reduce the tree's depth (i.e., the length of the longest path, and thus of the longest explanation) without affecting the tree's accuracy or overall complexity. This is similar to techniques used for search trees that reduce depth by applying left and right rotation operators (e.g., Nakamura et al., 1993).

3.1.5 Hybrid procedures

Some evidence suggests that hybrid classifiers combining decision trees and another classifier might allow for greater tree pruning with adequate accuracy. These approaches retain the training cases and use the decision tree to retrieve the cases stored at the leaf node corresponding to the current test case. The cases retrieved are then subjected to a further classifier, such as k -nearest neighbour (Barletta, 1994) or kernel density estimators (Smyth et al., 1995). Both approaches substantially improve accuracy over decision trees alone. This suggests the possibility of *over-pruning* a decision tree within a hybrid algorithm without sacrificing accuracy. The reduction in accuracy that would normally result from over-pruning may be offset by the contribution of the second classifier. The price paid for this tree simplification is, again, that all of the training cases must be stored.

3.2 Modifying the test space

Although procedures that directly control tree size have been most frequently used, alternatives exist that indirectly simplify decision trees. This section describes procedures that simplify decision trees by using a modified set of tests. Fayyad and Irani (1992) report evidence that binary tests often yield smaller trees than n -ary tests (i.e., which yield one subtree for each of n feature values), although exceptions to this trend exist (Kononenko, 1995). However, most of the procedures in this category modify the set of tests via *feature construction*: they construct new multivariate tests by applying mathematical or logical combination operators to individual case features. Some of these procedures retain feature combinations for possible reuse in tests; this is an example of *constructive induction* (Michalski, 1983; Rendell, 1985). The use of multivariate tests to simplify decision trees will be the focus of the remainder of this section.

When nodes are defined by multivariate tests, tree complexity cannot be measured as the number of nodes in the tree, since multivariate tests are more complex than univariate tests and display varying amounts of complexity. Instead, complexity is typically measured by the sum of the number of features included in all (internal) node tests and the number of leaf nodes.

Creating multivariate tests has both potential benefits and tradeoffs. Whereas pruning algorithms search the space of trees while maintaining a fixed set of tests, feature construction enlarges the space of hypotheses (i.e., possible trees) by increasing the expressiveness of the language defining tests. (Of course, these approaches can be combined.) For some situations, this allows tree induction algorithms to build more accurate and concise decision trees. However, searching this larger space increases computational complexity. Also, any reduction in the number of nodes might be offset by the increased complexity of the tree's tests.

Two problems that feature construction procedures target are *feature interaction* and subtree replication (discussed in section 2.2). Feature interaction occurs when a combination of two or more features is ranked highly on the test selection measure, but the component features individually have low rankings.⁶ Feature interaction prevents a univariate tree induction algorithm from accurately representing certain concepts. For example, if the conjunction $A \wedge B$ has high discriminative value, but the individual features A and B have no value, a univariate tree induction algorithm cannot create a tree to represent the conjunctive concept.

⁶Alternately, one might refer to these as *strong* feature interactions. The existence of *weaker* feature interactions is a presupposition of the decision tree approach; features providing useful tests lower in the tree would often not be selected were it not for the case partitioning provided by features higher in the tree.

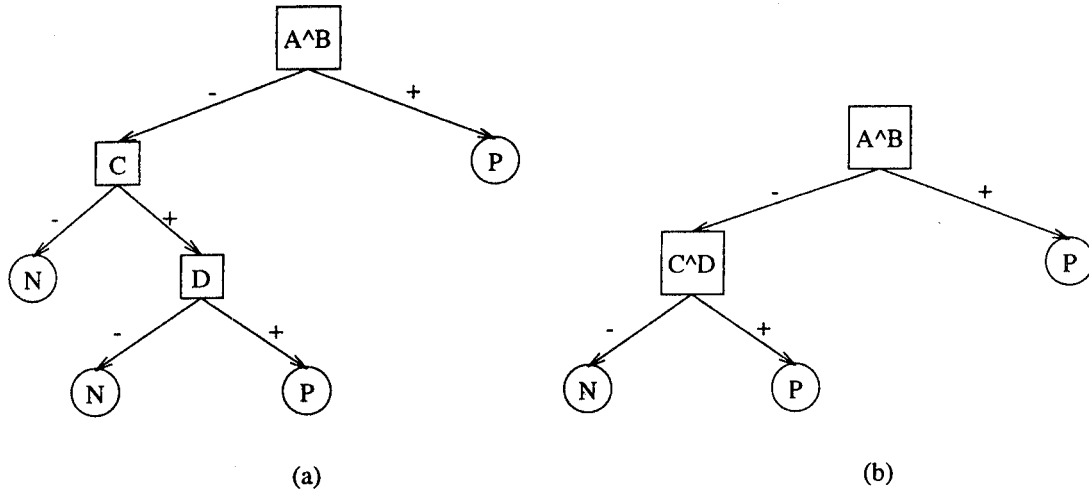


Figure 8 Multivariate decision trees representing $(A \wedge B) \vee (C \wedge D)$

Even when feature interaction is low, a univariate tree induction method might induce overly complex trees due in part to subtree replication. Again, the construction of multivariate tests provides a remedy. Figure 8(a) illustrates how a conjunctive feature $(A \wedge B)$ composed of two primitive features can help solve the replication problem and thereby reduce tree size (i.e., compared to Figure 3). Figure 8(b) shows how a multivariate test $(C \wedge D)$ can further reduce tree size even when there is no subtree replication.

We distinguish feature construction procedures by how they construct features, either locally based on the cases (“data”) at a node or iteratively using feedback from an induced tree “hypothesis.” Sections 3.2.1 and 3.2.2 describe examples of these respective approaches.

3.2.1 Data-driven test construction

This section reviews tree simplification algorithms that construct complex tests by applying construction operators to primitive features and evaluating the constructed features’ usefulness for a given subset of cases. Thus, these procedures modify the `best_test_and_partition` function (Figure 2) to construct and evaluate tests in addition to those based on primitive features. We categorize these algorithms into those that use numeric and those that use logical construction operators.

Numeric construction operators. Breiman et al. (1984, p. 132) were perhaps the first to recommend using linear combinations of numeric features to reduce tree size when the case data is best partitioned by hyperplanes not aligned with the features’ axes. They describe an extension of CART that, at each node, hill-climbs in the space of linear combinations of continuous features to search for a “best” test. Because few features usually contribute significantly to the test at any given node, they use a variant of backward sequential elimination (Devijver & Kittler, 1982) to discard features from each node’s test. To prevent CART from overfitting training data, internal nodes are required to have an above-threshold number of cases. Although this extension is inefficient, they demonstrated that it can reduce tree sizes for a synthetic waveform classification task from 21 to five nodes. However, its multivariate tests are more difficult to interpret than the larger tree’s univariate tests.

Utgoff (1988b, 1989b) introduced an incremental variant of this approach that induces *perceptron trees*, which are n -ary decision trees (i.e., with arity n) whose internal nodes are univariate and whose leaves are linear threshold units (LTUs) (Minsky & Papert, 1969). Utgoff’s algorithm, which is guaranteed to find a *consistent* decision tree (i.e., for each leaf, the cases are all in the same class), if one exists, incrementally updates the weights in an LTV so as to ensure it linearly separates its cases. It splits a leaf node when it (heuristically) determines that its set of cases is not linearly separable. Finally, it incrementally grows branches for its internal nodes as needed, and a new leaf’s initial

LTU will include only those features not already used along the path to it. His algorithm reduced the number of nodes for a simple task from 17 to five. Unlike Breiman et al.'s extension to CART, the perceptron tree algorithm is limited to binary classification tasks.

Utgoff and Brodley (1990) argued that multivariate trees have the potential for greatly reducing tree size. They introduced PT2, which extends perceptron trees by incrementally finding multivariate tests at all nodes. They encode all features as numeric, incorporate CART's method for pruning features from a node's test, and use Gallant's (1986) pocket algorithm to find optimal weight vectors for the LTUs. PT2 greatly reduced tree size (i.e., number of nodes) for four tasks compared with a univariate tree-generating algorithm, but, in doing so, sometimes yielded unintelligible tests. Also, it is limited to binary classification tasks and is computationally expensive (i.e., it may train $O(n^2)$ LTUs at a node containing n cases). More recently, Brodley and Utgoff (1995) greatly extended their empirical investigations. In one experiment, they compare methods for learning coefficients of LTUs and report that one based on recursive least squares clearly produced the simplest trees, although it also required the most CPU time.

Heath et al. (1993) introduced SADT, which uses a randomized procedure (i.e., a variant of simulated annealing) to search for combinations of numeric tests. They show that finding even one *optimal* hyperplane test is an NP-complete task, and use this as motivation to find approximately optimal tests. In their empirical evaluation, SADT is run 100 times and the smallest tree it generates is retained. Using this approach, SADT consistently generated smaller trees than ID3 (Quinlan, 1986), while usually increasing accuracy.

Murthy et al. (1993, 1994) introduced OC1, which extends previous work on CART and SADT by combining both deterministic and non-deterministic strategies for finding multivariate tests. In particular, it first uses a deterministic strategy that individually perturbs the coefficients of a (numeric) multivariate test. To combat local minima, it then repeats this process and/or randomly perturbs these coefficients. OC1 is relatively efficient; its asymptotic complexity is $O(fn^2 \log n)$ for f features and n cases at a node, which is more costly than similar univariate algorithms ($O(fn^2)$), but less costly than some other multivariate algorithms (e.g., SADT). OC1 produced slightly smaller and more accurate trees than SADT in their experiments, but in much less time, and its performance improved with the amount of randomization search used. In comparison with an extension of the perceptron tree algorithm, LMDT (Utgoff & Brodley, 1991), OC1 showed advantages in terms of accuracy and tree size when the cases were not linearly separable.

Most of these approaches use post-pruning techniques similar to those described in section 3.1.2. An alternative post-processing method designed specifically for multivariate trees is *hyperplane merging* (Kubat & Flotzinger, 1995), in which a node and its parent are combined by merging their hyperplanes. Tested on three artificial and four real-world data sets, hyperplane merging produced smaller trees that were at least as accurate as post-pruning. With noise levels exceeding 10%, hyperplane merging demonstrated superior accuracy and greater reductions in tree size relative to post-pruning.

Logical construction operators. Several algorithms search for tests containing logical combinations of symbolic features, constructed using simple (e.g., conjunction) or more complex (e.g., parity) logical operators.⁷ Exploring all possible logical combinations can be computationally expensive. For example, given m feature combination operators that can each apply to up to (one subset of) f primitive Boolean features, the space of feature combinations is $O(m2^f)$, and choosing which subset of these constructed features to apply yields a space of size $O(2^{m2^f})$ (Thornton, 1990). The complexity of iterative constructive induction algorithms is even greater (see section 3.2.2). Fortunately, the number of feature combinations to consider can be reduced by applying only some operators to only some feature subsets. Test construction algorithms constrain this space by heuristically selecting which features to combine. This section focuses on data-driven algorithms; we discuss hypothesis-driven methods that use logical construction operators in section 3.2.2.

⁷LFC (Ragavan et al., 1993) constructs new features using logical *conjunction*, but will be described in section 3.3.3, on lookahead search methods, because it incorporates *both* lookahead and feature construction.

Breiman et al. (1984, p. 136) were perhaps the first to argue for using logical combinations of tests to reduce tree size. They note that for speech recognition, medical diagnosis and other classification tasks, experts define features that are logical combinations of primitive features (i.e., disjuncts, conjuncts and negations). They describe a sequential method for constructing these tests and use it to classify mass spectra readings according to whether they show traces of bromine. Breiman et al. do not compare the relative sizes of univariate and multivariate trees for this application.⁸

Seshu (1989) examined using *parity* operators in tests; these operators are applied to a set of Boolean features and return TRUE if an odd number of the features have the value 1. Parity concepts represent an extreme form of feature interaction. After tentatively selecting a test, Seshu's algorithm randomly selects R features and applies the exclusive-or operator to all 2^R subsets of R . According to Seshu, values of three or four for R often produce significant improvements in accuracy on parity problems, yet the cost of this additional computation does not increase with problem size. However, the utility of this approach seems restricted to tasks with characteristics similar to parity problems.

Another class of feature interaction tasks are *m-of-n* concepts (i.e., *Boolean threshold functions*), which are true if at least m of a particular subset of n feature-value pairs are true. M-of-n concepts subsume logical conjunction, because a conjunction is equivalent to an n-of-n concept. In general, univariate decision trees for representing m-of-n concepts are quite large, and determining whether there is an m-of-n concept consistent with a set of cases is NP-hard (Pitt & Valiant, 1988).

Murphy and Pazzani (1991) proposed the GS algorithm in their ID2-OF-3 system as a greedy solution to the m-of-n problem. When selecting a test at a node, GS first finds the best feature with regard to information gain and converts it to a simple 1-of-1 test. GS then greedily makes small modifications to the test (i.e., adding a feature-value to n , either with or without increasing m) so long as they produce discrimination gains. The number of candidate tests considered at a node is $O(kf^2)$, where each feature has at most k values and f is the number of primitive features. Generating one m-of-n concept takes time $O(ekf^3)$, where e is the number of training cases. Murphy and Pazzani compared their algorithm with a non-pruning variant of ID3 on eight artificial and four benchmark data sets. ID2-OF-3 significantly increased accuracy relative to ID3 on all but one data set, and also significantly decreased the number of tree nodes. However, it is unclear whether overall tree complexity (i.e., the number of nodes *and* test complexity) also decreased.

Zheng (1995) studied tests represented by the *x-of-n* operator, a very general logical combination operator. Unlike the other logical operators reviewed here, x-of-n operators output integer, rather than Boolean, values. As in m-of-n tests, n represents the cardinality of a set of feature-value pairs, but the value x of an x-of-n test is the *number* of these pairs that are true. Thus, x-of-n tests can represent m-of-n ($x \geq m$), parity ($x \bmod 2 = 1$), and conjunctive ($x = n$) tests. The search space of x-of-n tests is as large as that of conjunction and smaller than that of m-of-n,⁹ despite their greater expressiveness. However, these tests have higher arity than Boolean-valued tests and thus might partition a set of cases into small subsets, exacerbating the fragmentation problem (see section 2.1).

Zheng's XOFN algorithm, which extends C4.5 (Quinlan, 1993a), is similar to GS; it starts at each node with a simple test, and then iteratively and greedily modifies it in small steps (i.e., by adding or deleting one feature-value pair to n). A newly constructed test is retained if, relative to the current test, it increases information gain without increasing coding cost or if it reduces coding cost without affecting information gain.¹⁰ Search terminates whenever a maximum allowable test complexity is reached or when a maximum number of search steps occurs without improvement. The selected test is added to the set of possible features to use at subsequent nodes; thus, this algorithm performs constructive induction. Zheng reported that XOFN tends to yield higher accuracies than a

⁸However, they report that CART's post-pruning algorithm reduced the multivariate tree's size from 50 to 25 nodes while simultaneously increasing predictive accuracy.

⁹Search for an m-of-n concept includes first a search for the subset n and then for the value of m , while search for an x-of-n concept requires searching only for n .

¹⁰The coding cost of a test is computed in a manner similar to Quinlan and Rivest's (1989) minimum description length (see section 3.3.1).

conjunctive feature construction algorithm (Zheng, 1992), ID2-of-3 and C4.5, both for some artificial and benchmark data sets, and frequently also reduced tree size compared with the latter two algorithms. However, XOFN is much slower than C4.5 due to its increased search effort. It remains to be seen how the level and type of feature interaction in a task impact XOFN's relative performance.

In summary, several automated tree induction variants can apply logical and/or numeric operators to reduce tree size without sacrificing (and sometimes increasing) accuracy, at least for some classification tasks. There is insufficient evidence on whether decreases in the number of nodes are offset by increased test complexity. The relative abilities of these methods with respect to specific data characteristics have not been fully explored. However, Brodley (1995a) describes a promising algorithm that automatically determines whether to use univariate or multivariate tests (or k -nearest neighbour) at each node based on the node's data characteristics.

3.2.2 Hypothesis-driven test construction

The algorithms described in section 3.2.1 select tests based on how they partition the set of cases at a node. This section describes algorithms that construct new tests based on feedback from the induction process itself. These algorithms are intended to solve the subtree replication problem (section 2.2). Unlike most of those in the preceding section (i.e., except for XOFN and LFC), these algorithms perform constructive induction; constructed feature combinations are cached for reuse.

These methods construct new features in binary trees by iteratively applying simple logical operators (e.g., conjunction, disjunction, negation) to combine primitive features. They iteratively create a tree and then induce new features. Considering all available features for constructive induction would produce an enormous space of feature combinations (Thornton, 1990). The analysis in section 3.2.1 showed that a single iteration explores $O(m2^f)$ feature combinations for f primitive Boolean features and m feature combination operators. Thus, the number C_i of possible feature combinations after i iterations is

$$\begin{aligned} C_1 &= O(m2^f) \\ C_i &= O(m2^{C_{i-1}}) \end{aligned}$$

Clearly, heuristics are needed to limit this search. The algorithms described here prune the search space by using the previously-induced tree to strongly constrain the selection of features to combine;

Key: C : Set of cases
 I : Set of tests
 $\text{eval}(\text{test}) \in I, C' \subseteq C$: Yields quality of a test for splitting C'
 $\text{stop}()$: Indicates whether to terminate tree expansion
 T : The induced tree
 I_{new} : A set of new features
IMAX: maximum number of tests (user-specified)

```
create_tree( $C, I, \text{eval}(), \text{stop}()$ ) =
   $I_{\text{new}} := \phi$ 
  REPEAT
     $T := \text{make\_tree}(C, I, \text{eval}(), \text{stop}())$  // From Figure 2
     $I_{\text{new}} := \text{create\_features}(T, C, I, \text{eval}())$ 
     $I := I + I_{\text{new}}$ 
  UNTIL  $I_{\text{new}} = \phi$  OR  $|I| \geq \text{IMAX}$ 
  RETURN  $T$ 
```

Figure 9 A decision tree induction algorithm with iterative constructive induction

new feature combinations are generated using logical combinations of tests in the tree. This limits the otherwise expensive search and is intended to guide it in productive directions.

Figure 9 shows a generic tree creation algorithm using iterative constructive induction. On each iteration, it calls the `make_tree` function, described in Figure 2, followed by a call to `create_features`, corresponding to one of the algorithms described in this section.

CITRE (Matheus & Rendell, 1989; Matheus, 1990) and FRINGE (Pagallo, 1989, 1990) are examples of algorithms in this category. Constructed features are DNF (disjunctive normal form) formulae. These algorithms are limited to binary classification problems, where leaf nodes are labeled either positive or negative. CITRE induces trees using a variant of ID3 and constructs new features by conjoining or negating tests along positive paths in the tree (i.e., those paths terminating in positively-labelled leaves). This induction-construction cycle is repeated until no new features can be constructed. We will describe CITRE’s behaviour using an example from *tic-tac-toe*, where $\text{pos}_{n,m}$ refers to the square in row n and column m and possible values are X , O and **blank**. If the tests $\text{pos}_{1,1} = X$ and $\text{pos}_{1,2} = X$ are on the same positive path of the most recently induced tree, then they can be conjoined to form the feature $(\text{pos}_{1,1} = X \wedge \text{pos}_{1,2} = X)$. CITRE generalizes its constructed features by changing constants to variables. For example, the previously named conjunct becomes $(\text{pos}_{1,1} = v \wedge \text{pos}_{1,2} = v)$, where v represents either X , O or **blank**. CITRE constrains the number of constructed features in two ways. First, it uses domain knowledge to filter new features thought likely to be irrelevant. For example, in *tic-tac-toe*, domain knowledge might limit construction to features representing adjacent board positions (e.g., $\text{pos}_{1,1}$ and $\text{pos}_{1,2}$). Second, a prespecified limit is imposed on the size of the feature set; constructed features are ranked by their information gain on the entire database and are selected according to this ranking.

When tested on a *tic-tac-toe* endgame classification task in ablation studies, CITRE’s constructive induction component was shown to improve trees’ accuracy (Matheus, 1990). Variants of CITRE including either generalization or domain knowledge demonstrated further gains in accuracy (Matheus & Rendell, 1989). Trees produced by CITRE had fewer nodes than those produced without constructive induction. However, improved comprehensibility due to fewer nodes is offset by the complexity of the nodes’ descriptions; constructive induction tended to *increase* the overall complexity of the trees, measured by the total number of primitives used to define nodes. Domain knowledge filtering decreased tree complexity, while generalization increased it. Without domain-knowledge filtering, the number of new features created was very large. In sum, despite its ability to solve the replication problem and reduce the tree’s number of nodes, CITRE’s construction of new features may not reduce tree complexity.

FRINGE (Pagallo, 1989, 1990) is similar to CITRE but uses features more selectively during constructive induction; it conjoins only the tests in the parent and grandparent of a positive leaf (i.e., nodes on the fringe of the path). FRINGE uses a tree induction algorithm that is a hybrid of ID3 and CART (Breiman et al., 1984) with reduced error post-pruning (Quinlan, 1987a). The induction-construction cycle iterates until no new features are added or a specified maximum number of features has been reached. FRINGE was tested on nine data sets with irrelevant features and various amounts of noise added. Feature construction was found to consistently reduce tree size and increase predictive accuracy. Matheus (1989) reported that conjoining fringe tests is preferable to conjoining all pairs of tests along a path, as is done in CITRE. Tested on *tic-tac-toe* and other tasks, the fringe heuristic yields comparable accuracy, smaller trees (i.e., fewer nodes), and faster execution speeds than CITRE. While FRINGE has performed well on artificial data sets, it is not clear that it performs as well on other data sets.

Like CITRE, FRINGE was originally limited to constructing features as DNF formulae. Several extensions to FRINGE have enabled it to construct additional kinds of features. Pagallo’s (1990) SYMMETRIC FRINGE constructs conjunctive features using both negative *and* positive paths. Yang et al. (1991a,b) introduced DCFRINGE, which constructs both disjunctive and conjunctive features. This allows DCFRINGE to succinctly represent both CNF (conjunctive normal form) and DNF concepts, while constructing fewer features than SYMMETRIC FRINGE. In comparison studies on 160 randomly generated concepts over 10 features, both SYMMETRIC FRINGE and DCFRINGE

produced smaller trees than FRINGE and recorded higher accuracies. However, no evidence was given that the FRINGE variants can reduce overall tree complexity.

The set of tasks for which iterative constructive induction reduces tree size and complexity has not yet been determined. These algorithms are not well suited to handle feature interactions because they apply feature construction only to features that appear in the previous decision tree. In tasks with highly interactive features, a multivariate feature may be highly ranked on the test selection measure even when its component features are ranked too low individually to be included in the tree. Some of the algorithms in the preceding section and the lookahead methods discussed in section 3.3.3 appear to be more relevant for solving feature interaction problems. However, the algorithms in this section can address the subtree replication problem because it involves features already included in the (initially univariate) tree. Thus, one promising approach is to combine algorithms from these complementary categories.

3.3 *Modifying the search for tests*

While the previous section focussed on changing the search space from which tests are selected, this section focuses on changing how that search is performed. The greedy search which is typically used to induce decision trees can be modified in several ways. First, alternative test selection measures (section 3.3.1) modify the search's state transitions by influencing the tests selected for tree expansion. Second, the representation of continuous features can be modified to reduce search bias for, and selection of, tests on continuous attributes (section 3.3.2). Finally, more extensive search methods can be employed, including forms of lookahead (section 3.3.3). All of these methods modify the `best_test_and_partition` function in the generic decision tree induction algorithm displayed (Figure 2).

3.3.1 *Alternative test selection measures*

Minimum Description Length. Many test selection measures have been proposed, including information gain (Quinlan, 1986) and statistical measures of purity (Breiman et al., 1984). In a comparative study, Mingers (1989b) found no significant differences in accuracy for several popular test selection measures.¹¹ However, some selection measures might be superior in producing smaller trees.

One example is a selection measure based on the minimum description length (MDL) principle (Quinlan & Rivest, 1989), which combines considerations of tree size and accuracy. The MDL principle states that the best theory to infer from a set of data is the one that minimizes the sum of

1. the length of the theory, and
2. the length of the data when encoded using the theory as a predictor for the data,

where both lengths are measured in bits. For decision trees, the first quantity is the number of bits needed to encode the tree (theory) and the second is the number of bits needed to encode the cases misclassified by the tree (i.e., the *exceptions* to the theory). Minimizing the tree code length corresponds to tree simplification. Minimizing the exceptions encoding corresponds to accuracy improvement.

MDL has been used both as a test selection measure during tree construction and as a criterion for post-pruning. During tree construction, the test that produces the smallest increase in total (i.e., tree + exceptions) description length (TDL) is chosen as the basis for tree expansion. Expansion continues, even if TDL increases, until no further tree expansion is possible. Then the tree is pruned from the bottom up as long as pruning decreases TDL. Wallace and Patrick (1993) corrected an error in the coding technique used by Quinlan and Rivest, but confirmed their findings: the trees created using the MDL principle were consistently smaller and usually more accurate than trees

¹¹In his study, Mingers also concluded that a measure which selects tests *randomly* performs as well as others, but Buntine and Niblett (1992) present contrary evidence.

generated by C4 (a precursor of C4.5) with pessimistic error pruning (Quinlan, 1987a). This seems a promising approach for reducing the size of decision trees with an appealing theoretical foundation.

Weak Theory Learning Measure. Dietterich et al. (1996) describe theoretical motivation for an improved information gain test selection measure, $G(p) = 2\sqrt{p(I-p)}$ (where p is the percentage of positive cases in a node), based on the *weak learning* or *boosting* model (Kearns & Mansour, 1996). They prove that the number of splits required by C4.5 to yield a low error ε changes from a superpolynomial to a polynomial in ε when using G rather than using the usual information-theoretic function to compute information gain ratio. They further argue that G selects better tests than the latter because, when plotted in two dimensions, it has higher concavity and thus yields larger information gain differences for comparable values of p ; i.e., it is a more discriminating function.

They present evidence that, C4.5 modified to use the G test selection measure yields smaller and more accurate unpruned trees on average than when using one of its own measures of information gain. However, both of these improvements were quite small, their algorithm is currently restricted to binary classification tasks, and C4.5's usual *gain ratio* test selection measure was not included for comparison. Nonetheless, showing any performance improvements obtained using methods with theoretically provable advantages is quite welcome, and should clearly direct further research on similar test selection measures.

Kolmogorov–Smirnov Measure. Utgoff and Clouse (1996) extended earlier work by Friedman (1977) on using the Kolmogorov–Smirnov distance (KS) equation for selecting tests. Given a continuous feature f , KS yields the value of f corresponding to the maximum distance between the cumulative class distributions. This use of KS yields the Bayes-optimal cut-off points for continuous variables, assuming that the class-conditional probability density functions are known, the prior class probabilities are identical, and the class misclassification costs are identical. Utgoff and Clouse extend KS for discrete variables and missing values. The KS test is currently restricted to binary classification tasks.

Using the ITI (Utgoff, 1994) incremental decision tree induction algorithm, Utgoff and Clouse compared using KS versus gain ratio for selecting tests in experiments with 27 data sets. While the classification accuracies did not differ significantly, KS yielded significantly smaller trees for these data sets. KS also required significantly fewer tests when classifying test cases.

Class Separation. Fayyad and Irani (1992) note that popular test selection methods measure the *impurity* of the resulting partition. That is, they assess the homogeneity of class membership in each subset of the partition; different subsets are related to one another only indirectly when their impurity scores are averaged.

In contrast, they introduce a selection measure that directly evaluates candidate partitions based on their ability to differentiate classes (i.e., to *separate* cases of different classes and group together cases of the same class). This property should favour partitions that have lower branching factors and thus produce less data fragmentation. They propose the C-SEP (from *Class-SEParation*) family of test selection measures and encode one in their O-BTREE system, which induces binary trees. This measure assesses the orthogonality of the class vectors of the case subsets yielded by a partition; the O-BTREE algorithm selects the test producing the partition with greatest orthogonality. Compared to binary and multi-valued versions of ID3 on one synthetic and four benchmark data sets, O-BTREE produced trees that were much smaller and also more accurate.

Hybrid Test Selection Measures. Lubinsky (1995) and Brodley (1995b) both studied resubstitution accuracy as a test selection measure. Because Pazzani et al. (1994) found that using it as the selection measure reduces predictive accuracy, these authors integrated it with a “traditional” test selection measure; Lubinsky used the Gini criterion (Breiman et al., 1984) and Brodley used information gain (Quinlan, 1986).

Both authors report small gains in predictive accuracy when the traditional measure is used for nodes close to the root and an accuracy measure (Brodley) or a hybrid accuracy/traditional measure

(Lubinsky) is used for nodes closer to the leaves. Brodley reported that tree sizes were *greater* with this mixed method. However, Lubinsky found that using the hybrid measure *throughout* the tree dramatically reduced tree size (often by greater than 50%), at the cost of small losses in accuracy (usually less than one percentage point).

3.3.2 Discretizing continuous features

The methods described here modify the representation of continuous features. Decision trees primarily target symbolic features; continuous features are discretized. However, Quinlan (1996) argues that the typical discretization process (e.g., in C4.5) gives them an unfair advantage during test selection compared with symbolic features. For each continuous feature, this procedure orders all of its distinct values among the stored cases and evaluates the midpoint between every successive pair as a possible threshold for binary discretization. These candidate tests then compete as virtual features against symbolic features, each of which has only one representative. Thus, the selection method is biased against symbolic features (or continuous features with few distinct values); random variation in the data can cause one of the many continuous features' thresholds to be chosen over a more relevant symbolic feature. Also, discretization typically produces only binary splits with continuous features.

Dougherty et al. (1995) adapted an entropy-based discretization procedure introduced by Catlett (1991) and modified by Fayyad and Irani (1993). They applied discretization *globally* to continuous features *prior* to induction, rather than locally based on the subset of cases at a node (e.g., as done in C4.5). They argue that the local procedure is based on more fragmented data and so is more likely influenced by random variation. Their procedure recursively performs binary partitions on continuous features based on class information entropy to produce a discretized feature with multiple values, and uses an MDL-based stopping criterion. They found that global discretization did not degrade the accuracy of C4.5 and occasionally improved its accuracy. Their algorithm, like its predecessors, reduced tree sizes compared with C4.5 (Quinlan, 1993).¹²

Auer et al. (1995) reported a local method for discretizing continuous features. Like the global method just discussed, their T2 procedure produces *n*-ary (multiple), rather than binary, partitions of the features. However, rather than making recursive binary splits, T2 conducts a more thorough search to find the set of up to *m* intervals that minimizes error on the training cases. The default value of *m* is $C + 1$ where *C* is the number of classes. With this default, T2's complexity is proportional to $C^6 f^2$, where *f* is the number of features. To reduce complexity, they restrict decision trees to two levels of internal nodes, with only the second level nodes using non-binary splits of continuous features. Their algorithm produces much smaller trees than C4.5 (Quinlan, 1996), yet with comparable accuracy for several tasks.

Quinlan (1996) introduced a local, MDL-based method to penalize continuous features having many values in release 8 (R8) of C4.5. In experiments with 20 data sets, he found that R8 generally increases classification accuracy and reduces tree size relative to release 7 (R7). Quinlan also compared C4.5 R8 with Dougherty et al.'s (1995) global discretization method and with T2 (Auer et al., 1995). On average, C4.5 R8 produced more accurate, but larger, trees than these methods. However, on small data sets, global discretization was superior to R8 in both accuracy and tree size. On data sets with less than five classes, T2 remained less accurate but produced trees less than half the size than R8. When inducing a classifier for five or more classes, T2 was found to be extremely slow.

Examining the data set characteristics in Quinlan's experiments suggests that C4.5 R8's relative benefits over R7 depends in part on the percentage of case features that are continuous. R8 increases accuracy and reduces tree size on 69% of the data sets where the number of continuous features is greater or equal to the number of symbolic features. This value drops to 43% when the majority of

¹²Quinlan found that global discretization produced smaller trees than C4.5 R8 (release 8), which, in a separate experiment, he found yields smaller trees than C4.5 R7 (release 7). Their conclusion, then, follows from transitive inference over these two experiments.

features are symbolic. This is expected; R8 was designed to enhance C4.5's performance for continuous features.

To summarize, C4.5 R8's MDL method seems preferable for discretizing continuous features for large data sets and global discretization seems preferable for small data sets. T2 is best used with data sets containing few classes, and when the user is willing to incur some sacrifice in accuracy for radical tree simplification. N-ary (rather than binary) splits on continuous features show promise for yielding more comprehensible trees. These techniques are especially useful when a majority of the features are continuous.

3.3.3 Lookahead search

This section describes a more extensive search mechanism for selecting tests which can be applied to most selection measures. The `best_test_and_partition` in Figure 2 is modified to include a prespecified amount of lookahead search during test selection. Lookahead search addresses the horizon effect by evaluating the quality of tests based on the quality of the subtrees they create to a certain depth. It is particularly useful for tasks with feature interactions.

Unfortunately, lookahead can be computationally expensive. Feature selection at a node has complexity $O(fn)$ (Quinlan, 1986) for f features and n cases. In contrast, one-level lookahead's complexity is $O(f^2n^2)$ (Murthy & Salzberg, 1995), or more generally $O(f^dn^d)$ for d levels of lookahead. Thus, lookahead is typically constrained, either by limiting the set of features involved or lookahead depth.

In early work, Norton (1989) showed that exhaustive lookahead applied to ID3 reduced tree sizes on average and produced small gains in accuracy, but could be expensive. More recently, Ragavan and Rendell (1993; Rendell & Ragavan, 1993; Ragavan et al., 1993) introduced LFC (lookahead feature construction), which performs both lookahead and constructive induction (section 3.2) (i.e., it caches the results of lookahead in constructed features). They show that both are needed for LFC to perform well on tasks involving feature interactions.

LFC reduces computational costs by controlling lookahead using a constrained branch-and-bound search. User-specified parameters include the beam size α , a maximum search depth, and a window width λ . Included in the beam are the current best α features, ranked by information gain (Quinlan, 1986). Feature construction conjoins one of these features (f_i) with another (or its negation) outside the beam but within window λ of f_i . Thus, unlike the iterative constructive induction methods, LFC is not limited to selecting features with high enough individual merit for inclusion in the tree.

Ragavan and Rendell (1993; Ragavan et al., 1993) empirically compared LFC with several algorithms, including decision tree algorithms distinguished by whether they perform feature construction. They reported that the trees produced by LFC were more compact and informative to experts. LFC (Ragavan & Rendell, 1993) also had the highest, or tied for the highest, accuracies on the selected data sets, and its accuracy degraded much more gracefully with increasing feature interactions than did the other algorithms. When feature interaction was high, LFC's accuracy was about 30% higher than ID3 and 10% higher than any of the other algorithms. In both studies, they reported that its speed was comparable to the other systems.

While lookahead may be beneficial on tasks with high feature interaction, it does not always aid tree induction. In fact, Murthy and Salzberg (1995) found one-level lookahead to yield larger, less accurate trees on many tasks. Further, they found that post-pruning was superior to lookahead in terms of the accuracy and simplicity of the induced trees. Quinlan and Cameron-Jones (1995) reported similar findings for decision list induction and hypothesize that lookahead can yield "fluke theories" that fit the training data but have poor predictive accuracy. Thus, it is unclear whether lookahead is beneficial on tasks not characterized by feature interaction.

3.4 Database restrictions

Another category of approaches simplifies decision trees by simplifying the database from which

Key: C : Set of cases
 I : Set of tests
 $\text{eval}(\text{test}() \in I, C' \subseteq C)$: Yields quality of a test for splitting C'
 T : The induced tree

```

create_tree( $C, I, \text{eval}()$ ) =
  REPEAT
     $T := \text{make\_tree}(C, I, \text{eval}())$  // From Figure 2
     $E := \text{eliminate\_cases}(T, C)$ 
     $C := C - E$ 
  UNTIL  $E = \phi$ 
  RETURN  $T$ 

```

Figure 10 ROBUST-C4.5: an iterative decision tree induction algorithm with case selection

they are induced. These methods can be distinguished by whether they perform *case selection* or *feature selection*, which respectively eliminate a subset of cases or features from the database before creating the tree. The following two sections describe examples of these approaches.

3.4.1 Case selection

An early example of automatic case selection for tree induction was Quinlan's (1986) *windowing* strategy. This strategy starts with a randomly-selected training sample and then iteratively (1) induces a decision using this sample, (2) tests the tree on a separate sample of cases, and (3) adds the misclassified cases to the original training sample, repeating this cycle until no training cases are misclassified. Windowing did not produce significantly smaller trees (Wirth and Catlett, 1988). However, Utgoff's (1989a, 1994) sampling strategy, which randomly selects only a *single* case both during induction and when extending the training sample, was more successful in reducing tree size. Rather than inducing trees from scratch, his algorithm updates trees *incrementally* as each new case is added (see section 3.1.3).

John's (1995) ROBUST-C4.5 system (Figure 10) operates in the reverse sequence; instead of starting from a single case and gradually adding misclassified cases, it iteratively induces a tree based on all available cases, post-prunes it, and then removes all misclassified training cases. Thus, each iteration works with only those cases correctly classified by the previously generated tree, and this cycle terminates when all retained cases are correctly classified. This process assumes the induced classifier is a better source of information than are misclassified cases, in contrast to windowing methods, which assume that previously unseen misclassified cases represent deficiencies in the tree.¹³ Whereas post-pruning algorithms only prevent misclassified cases from influencing branching *below* the pruned node (i.e., by removing the subtree beneath it), John's algorithm prevents these cases from influencing tree induction *above* that node, as well (i.e., in the next iteration of tree induction).

In comparisons with C4.5 on 21 data sets, ROBUST-C4.5 slightly increased accuracy, decreased variability in accuracy, and generated much smaller trees (i.e., 29% smaller). John notes that ROBUST-C4.5 could be modified to update decision trees incrementally, without having to completely re-induce the tree as noisy cases are detected and removed (see section 3.1.3). Indeed, incrementality together with case selection has been shown to reduce tree size and increase accuracy (Utgoff, 1989a, 1994).

3.4.2 Feature selection

Since decision tree induction can be misled by the presence of irrelevant features, several researchers

¹³This is similar to the distinction between edited nearest neighbor algorithms that delete misclassified cases (Hart, 1968) versus those that delete correctly classified cases (Wilson, 1972).

have investigated algorithms for removing such features *prior* to decision tree induction (Almuallim & Dietterich, 1991; Doak, 1992; Kira & Rendell, 1992a,b). Recently, some have proposed *hypothesis-driven* approaches to feature selection, whereby induction iteratively guides feature selection, much as it iteratively guides constructive induction in the hypothesis-driven methods discussed in section 3.2.2. John et al. (1994) call these approaches *wrapper* models. They propose greedy feature selection procedures that search the space of feature subsets using either *forward stepwise selection* (FSS), which begins with no features, or *backward stepwise elimination* (BSE), which begins with all of the features. FSS iteratively adds the “best” feature, stopping when no extension of the existing feature subset improves accuracy. Similarly, BSE iterates until it is unable to delete a feature without decreasing accuracy. Feature subsets are evaluated by the average accuracy of decision trees induced using them in an n -fold cross validation. Afterwards, the induction algorithm (in this case C4.5) is applied to the entire training set using only the relevant features.

John et al. (1994) compared the wrapper approach to C4.5 on six data sets selected for demonstrating large differences between algorithms. The FSS variant of the wrapper algorithm produced markedly smaller trees using substantially fewer features, and with comparable accuracy, in comparison to C4.5 both with and without pruning. The BSE variant of the algorithm was not as effective.

Cherkauer and Shavlik (1996) described a different wrapper algorithm, SET-GEN, for feature selection. SET-GEN uses a genetic algorithm to search the space of feature subsets. Whereas feature subsets are evaluated by John et al. in terms of tree accuracy only, SET-GEN uses functions of average subset size, tree size, and accuracy, assessed using 10-fold cross validation. Compared with C4.5 (with post-pruning) on ten data sets, SET-GEN produced trees that were uniformly at least as accurate, significantly smaller, and based on fewer features for all but one data set (on which they did not differ significantly). Tree sizes often differed by a factor of two or more and the differences in number of features used were also usually of that order.

In sum, both case selection and feature selection methods can yield substantial reductions in tree sizes while preserving classification accuracy.

3.5 Alternative data structures

The fifth and final category of tree-simplification approaches translates an induced tree into another data structure, such as a graphs or a set of rules. This conversion takes place in the `post_process` function in the generic tree-induction algorithm (Figure 2).

3.5.1 Decision graphs

This section describes approaches that reduce data structure size by translating the tree into a *decision graph* (i.e., a directed acyclic graph, or DAG). The decision graphs typically used in classification research are related to decision trees in at least three ways. First, they are *rooted*, having only one node, the root, with indegree zero. Second, nodes with outdegree zero are leaves that categorize cases. Third, the graph’s edges are oriented in a consistent direction from the root to the leaves, as in a tree. Thus, these DAGs have parent and children nodes, but a child may have multiple parents. We show an example of a decision graph in Figure 11, which we discuss below.

Despite these similarities, decision graphs have two advantages over decision trees. First, leaves with the same class label can be merged, leaving exactly one leaf per class. This alleviates the fragmentation problem, discussed in section 2.2, that besets decision trees. Second, decision graphs address the replication problem, in which subtrees are replicated to represent DNF concepts. In a decision graph, identical subtrees can be merged. For example, Figure 11 represents the concept $(A \wedge B) \vee (C \wedge D)$ with a decision graph containing four internal nodes and two leaves. In contrast, the decision tree representation has six internal nodes and seven leaf nodes (see Figure 3). Both leaf and subtree merging reduce DAG size and so are likely to enhance the comprehensibility of the induced data structure.

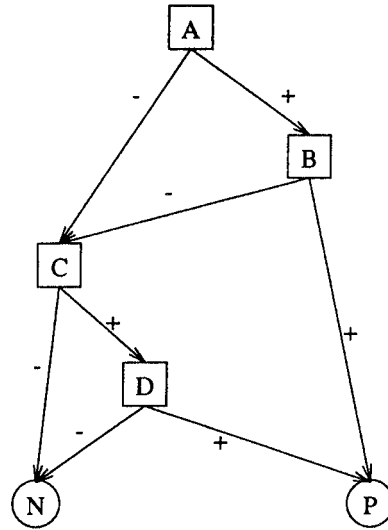


Figure 11 A decision graph representing $(A \wedge B) \vee (C \wedge D)$

Oliver (1993) studied decision graphs based on the minimum message length (MML) principle, which is similar to the minimum description length principle (Rissanen, 1983; Quinlan & Rivest, 1989) discussed in section 3.3.1. He introduced a node-merging operation that transforms a tree into a graph, the new node having both parents of the original pair. His algorithm, like Quinlan and Rivest's, uses MML as a criterion for test selection and post-pruning. In addition, MML also provides a common measure for two operations during tree generation: tree expansion via test selection and node merging. The operation that produces the greatest MML savings is selected on each iteration.

Oliveira and Sangiovanni-Vincentelli (1995) report that Oliver's algorithm failed when applied to complex tasks due to premature node merging which overreduced the graphs. They introduce *reduced ordered decision graphs* (RODGs). An ordered decision graph is a DAG whose tests appear in the same order on every path, though possibly skipped on some paths. A *reduced* decision graph is one in which no two nodes have identical branching and there are no *constant nodes*, whose outgoing edges all terminate at the same node. Their SMOG algorithm begins by generating three data structures: (1) a decision tree produced by a standard induction algorithm (Quinlan, 1986) without pruning; (2) a tree generated using constructive induction (see section 3.2.2) (Oliveira & Vincentelli, 1993); and (3) a RODG. SMOG then selects the structure with minimal description length. If needed, the selected structure is converted into a RODG. Then, the structure is compressed by greedily merging nodes with identical branching and by pruning constant nodes. Node merging requires $O(s^2m)$ operations, where s is the number of nodes in the current RODG and m is the training set size. Pruning requires $O(s^3m)$ operations. Finally, the features' ordering in the RODG is revised to further reduce the total description length. This potentially expensive procedure can be optionally applied once to each feature or iterated to convergence.

In a comparison with C4.5 on 80 classification tasks, SMOG's accuracy was significantly higher on 48, comparable on 27, and significantly lower on only five tasks. However, the authors report that SMOG is much slower than C4.5, though they provide no details. Unfortunately, size comparisons between RODGs and decision trees were not reported. Also, SMOG handles only binary tests; it converts n -ary symbolic tests into sets of binary tests during a preprocessing phase.

Unlike RODGs, *oblivious read-once decision graphs/trees* (OODG/Ts) (Kohavi, 1994a,b; Kohavi & Li, 1995) are not limited to binary tests. An *oblivious* decision graph (tree) is a *levelled* decision graph (tree) in which all nodes at a given level test the same feature, where a levelled decision graph (tree) is one whose nodes are divided into a series of pairwise disjoint sets, or levels, with the

outgoing edges from each level all terminating at the next level. Each test in a *read-once* decision graph (tree) occurs at most once per path.¹⁴

Kohavi and Li's (1995) *entropy-based oblivious decision graph* (EODG) algorithm first induces an OODT, then prunes its levels, merges subtrees, and prunes constant nodes. OODTs are created top-down using a *mutual information* test selection criterion. All training cases are used to select tests at each level because a single test is chosen for each level. In contrast, only a subset of cases are used to select a decision tree's test at any non-root node. Next, levels are pruned in a bottom-up order, guided by maximizing cross-validation accuracy. Third, EODG merges compatible subtrees in a top-down manner, thereby transforming the OODT to an OODG. Compatible subtrees are either leaves with the same class labels or internal nodes with the same test and whose corresponding child subtrees are compatible. An OODT is more likely to contain compatible subtrees than a standard decision tree because its nodes at the same levels use the same tests. To tolerate noise, subtrees are merged even when they are only *approximately compatible*, as defined by confidence intervals. Finally, constant nodes are again pruned.

Merging subtrees dominates EODG's computational complexity. At each level with k nodes, $O(k^2)$ compatibility tests are conducted. However, Kohavi (1994a,b) proves that there is a limit on the number of nodes at lower levels and argues that many node merges succeed at higher levels, which have fewer nodes. These merges reduce the number of nodes at lower levels.

Kohavi and Li (1995) reported that EODG attained higher or equivalent classification accuracy than C4.5 on 15 of 22 tasks, and yielded smaller data structures on 16 of these tasks. The reduction in size was sometimes dramatic.

EODG might benefit from methods introduced by Langley and Sage (1994) for oblivious decision trees (ODT). Their OBLIVION algorithm first generates an ODT using all the cases' features. It then prunes levels using a variant of BSE, iteratively eliminating levels so as to maximize the ODT's accuracy, and terminating when further pruning can only decrease accuracy. This *backward elimination pruning* might be preferable to the post-pruning method used by Kohavi and Li for handling feature interaction, because removing one of a set of interdependent features from an oblivious tree would detectably decrease the tree's accuracy.

Differing in structure from the decision graphs discussed thus far are *exception directed acyclic graphs* (EDAGs), introduced by Gaines (1996) as a common data structure for simplifying decision trees, production rules, or decision lists. An example is shown in Figure 12. Any node in an EDAG

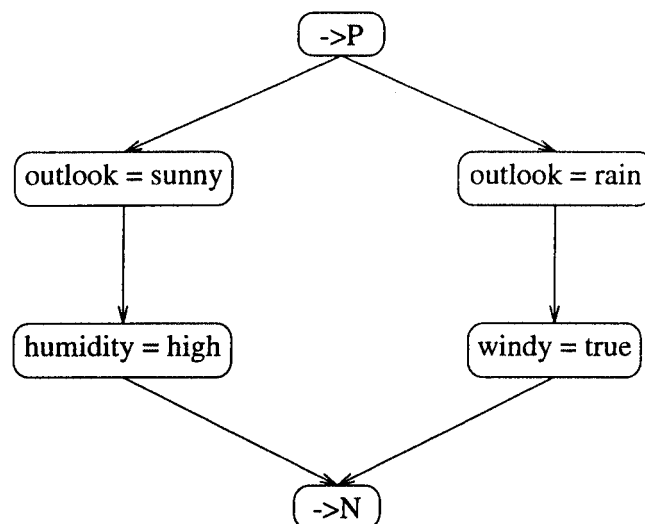


Figure 12 An exception directed acyclic graph (EDAG)

¹⁴However, continuous-valued features can be tested more than once provided that the thresholds differ.

can contain *rule conditions* (i.e., *feature = value* expressions or disjunctions of these expressions), *conclusions* (i.e., class assignments, indicated by $\rightarrow$ in the figure), or both. A rule condition corresponds to a feature-value test. Because nodes contain conditions rather than simply tests (as in decision trees), sibling nodes can be defined by tests containing *different* features. Thus, a *node's cases need not be partitioned into exhaustive or disjoint subsets*. The edges are inheritance links with disjunctive multiple inheritance. The graph can be disjoint, with more than one root.

Given a query, the algorithm traces *all* possible paths from all roots as far as possible, rather than a single path as in a decision tree. Because the partitioning is not disjoint, it is possible to draw multiple compatible conclusions for a query. An EDAG is *exception-directed* in that the first conclusion reached on a path is the *tentative* conclusion; subsequently reached conclusions (*exceptions*) replace it. Because the partitioning is not exhaustive, query traversal can terminate at an internal node. If that internal node contains no conclusion, then the current tentative conclusion is asserted. In either case, the premise of the conclusion is the conjunction of (a) the conditions on the traversed path and (b) the *negation* of the disjunction of the node's *children*.

To clarify, the six-node EDAG shown in Figure 12 is representationally equivalent to the eight-node decision tree shown in Figure 5, although it is not obvious which is more comprehensible. Both structures represent the following five decision rules:

1. IF (outlook = sunny) and (humidity = high) THEN (class = N).
2. IF (outlook = sunny) and (humidity = normal) THEN (class = P).
3. IF (outlook = overcast) THEN (class = P).
4. IF (outlook = rain) and (windy = true) THEN (class = N).
5. IF (outlook = rain) and (windy = false) THEN (class = P).

A decision tree represents each rule by a distinct path from the root to a leaf. In contrast, an EDAG represents rules by *partial paths* from the root. Although the rules that conclude *class = N* are expressed explicitly by the conditions on the two paths to the leaf node, the rules that conclude *class = P* instead have some premise conditions obtained by negating the disjunction of the terminal node's children on a partial path.

For example, given a query q satisfying the conditions of rule 2, EDAG traversal begins at the root, tentatively adopting the conclusion *class = P*. It then traces parallel paths to the root's children whose conditions are satisfied by q . Only the left child's condition, *outlook = sunny*, is satisfied and so is added to the explanation's premise. Traversal proceeds to this child node. The condition of its only child is not satisfied by q . Thus, the traversal ends. The conclusion *class $\rightarrow$ P* is asserted and the negation of the child's condition, $\neg$ (*humidity = high*), which a *knowledge representation server* replaces with the equivalent (*humidity = normal*), is added to the supporting conditions.

Decision trees and rule sets can be converted to EDAGs by an algorithm that factors common sub-premises from the trees' rules (corresponding to its paths). Alternately, EDAGs can be induced directly using INDUCT (Gaines, 1989). Gaines proposed a common complexity measure for trees, rule sets, and EDAGs that accounts for the number of nodes, edges and conditions. Based on this measure, Gaines found EDAGs to be less complex than decision trees (including ID3 and C4.5) and rule sets induced by PRISM (Cendrowska, 1987) and by C4.5RULES (section 3.5.2) when compared on three tasks (Gaines, 1995, 1996).

At present, EDAGs have been tested on only three small noise-free tasks; it is unknown how they perform on larger, noisy tasks.

In sum, decision graphs can improve the accuracy and comprehensibility of concept descriptions in many domains. However, their biases have not been investigated; we do not yet know which problems are most amenable to these approaches.

3.5.2 Rule sets

Trees can also be simplified by translating them to rules, which are then pruned. Two approaches have been proposed. The first induces a single decision tree and then prunes the rule set derived from

it (Quinlan, 1987a,b). The second iteratively generates decision trees, selecting the best rule from each iteration (Fayyad & Irani, 1993). We examine each approach in turn.

Quinlan's (1993a) C4.5 system includes an option for translating the induced tree to a set of rules. The C4.5RULES option consists of the following steps:

1. **Tree creation.** A tree T is created, without pruning, using C4.5.
2. **Translation.** T is then converted into a set R of mutually exclusive and exhaustive rules. Each root-to-leaf path constitutes a rule; its antecedent (premise) is the conjunction of the path's tests and its consequent (conclusion) is the leaf's class label.
3. **Initial Pruning.** Each condition in R 's antecedents is evaluated, using an algorithm similar to EBP (section 3.1.2), to estimate whether pruning impairs R 's accuracy. C4.5RULES greedily searches the space of rule sets, iteratively pruning the condition that minimizes R 's error estimate, stopping when further pruning cannot reduce this estimate. Then, duplicate rules and those with unacceptably high error rates are deleted.
4. **Form Class Rule Subsets.** Because the resulting rules are no longer mutually exclusive, some form of conflict resolution is needed. Thus, rules are grouped by class and pruned using an MDL (section 3.3.1) criterion. Then, the rule subsets are ranked by decreasing number of false positive errors. Given a case, the first matching rule subset's class is predicted. Since the rule set is no longer exhaustive, a default rule is needed for queries that match no rule in R ; the default rule predicts the class with the highest false negative rate.
5. **Final Pruning.** Rules whose omission would reduce resubstitution error are deleted.

Rule sets offer several advantages over trees. For example, they form the basis for most expert systems because they are highly intelligible to human experts. Although we expect they can reduce the complexity of data structures, we have not found studies comparing the total complexity of rule sets derived from trees with pruned decision trees. Rule pruning might produce more accurate classifiers than tree pruning algorithms (see section 3.1.2) because they are more discriminating; pruning a rule's condition is equivalent to pruning only one child of a tree node, an operation that current post-pruning algorithms do not support. Indeed, Pérez and Rendell (1995) report that C4.5RULES displays higher accuracy than C4.5 with post-pruning on 18 of 19 data sets.

Fayyad and Irani (1993) described another approach for translating decision trees to rules. Their RULER system iteratively induces trees from different sets of training samples. On each iteration, it prunes conditions in individual rules using a nonparametric significance test (Fisher's exact test) and then uses this same test to select the "best" remaining rule. Finally, a minimal subset of the rules that covers the entire training sample is selected using a greedy covering algorithm. RULER can use any tree induction algorithm. In their evaluation, the authors used O-BTREE (Fayyad & Irani, 1992; see section 3.3) and an improved version of GID3 (Cheng et al., 1988) named GID3*. Tested on a large astronomy data set, both RULER variants produced more accurate rule sets with lower complexity than trees produced by O-BTREE, GID3*, or ID3. We did not find studies that directly compare C4.5RULES with RULER.

Unfortunately, translating trees to rules can be computationally expensive. C4.5RULES is slow for some databases (Daelemans et al., 1997); its complexity is $O(n^3)$, where n is the number of cases (Cohen, 1995). One alternative might be to generate rules directly, without initial tree creation. On many tasks, rule learning systems are more accurate than decision tree learners (Pagallo & Haussler, 1990; Weiss & Indurkha, 1991). They often produce simpler data structures (Weiss & Indurkha, 1991), although Gaines (1996) found that they increase complexity for two small data sets without noise.

While early rule induction algorithms were computationally expensive for large data sets (e.g., $O(n^4)$ (Cohen, 1995)), fast rule induction algorithms have recently been developed that scale much better than C4.5RULES (Furnkranz & Widmer, 1994; Cohen, 1995). One of these (Cohen, 1995) attains accuracies comparable to C4.5RULES, which is in turn often more accurate than C4.5, and scales nearly linearly with the number of cases.

Table 4 Some promising tree-simplification algorithms

<i>Category</i>	<i>Subcategory</i>	<i>Algorithm</i>	<i>Accuracy</i>
Control tree size	Post-pruning	REP	
		MCCP, 0-SE	
		MT	+
Modified test space	Data-driven	OC1	+
		ID2-OF-3	+
		XOFN	+
		FRINGE	+
Modified test search	Hypothesis-driven	MDL	+
		Kolmogorov-Smirnoff	
		C-SEP	+
	Selection measures	Lubinsky's (1995) hybrid	—
		Global Discretization	+
		T2	—
	Discretization	LFC	+
		Lookahead	
		Case selection	
Database restrictions	Feature selection	ROBUST-C4.5	
		SET-GEN	
Alternative data structure	Decision graphs	EODG	
		EDAG	
	Rules	C4.5RULES	+
		RULER	+

4 Discussion and future directions

This section summarizes our survey and describes some of our future research objectives. In particular, section 4.1 discusses promising tree-simplification strategies, both as suggested from our survey and from our recent empirical comparison (Breslow & Aha, 1997), while section 4.2 describes our plans for enhancing a case-based reasoning shell.

4.1 Discussion

Based on our literature survey, Table 4 lists those algorithms that performed particularly well with regard to simplifying trees. Algorithms that produced good levels of accuracy but did not simplify the trees, or whose simplification capabilities were not reported, are not included. Some of the included algorithms were noteworthy for either enhancing or diminishing accuracy; we indicate this by a + or —, respectively, in the *Accuracy* column in the table.

We empirically compared some of the more promising tree simplification procedures; the details are reported in Breslow and Aha (1997). Using available software,¹⁵ we compared C4.5 release 7 (Quinlan, 1993) (with and without EBP pruning, inducing trees or rules (C4.5RULES)), C4.5 release 8 (Quinlan, 1996) (tree induction with post-pruning), ITI (Utgoff, 1994) (with the gain ratio and with the KS (Utgoff & Clouse, 1996) test selection measures), ROBUST-C4.5 (John, 1995), T2 (Auer et al., 1995), LMDT (Brodley & Utgoff, 1995), XOFN (Zheng, 1995), and SET-GEN (Cherkauer & Shavlik, 1996) on eight data sets.

Below we summarize highlights from our survey and comparison study according to each of the five top-level categories in our framework.

Among the techniques that explicitly control tree size, post-pruning methods have shown the greatest promise to date. Several researchers have reported that REP and MCCP (0-SE) reduce tree sizes while maintaining accuracy. In our study, the post-pruning methods used in recent releases of C4.5 (EBP) and in ITI (MDL) compared favourably to other methods for reducing tree size, although the C4.5 variants were somewhat superior to ITI in maintaining accuracy. The recently

¹⁵Some of the best algorithms listed in Table 4 were not tested because the software was not available.

introduced post-pruning method based on MT theory (Cohen & Jensen, 1997) also appears quite promising. In contrast to post-pruning, pre-pruning methods have been largely abandoned in recent years. One exception is T2, which limits tree size to a depth of two. We found that T2 output the smallest-size trees, but scored low accuracies on several data sets. Incremental tree sizing has not been well-researched as a method to reduce tree size, but deserves attention for practical applications involving dynamically-changing data sets.

The second category, focusing primarily on algorithms that generate multivariate tests, is more problematic. These algorithms do not always reduce tree complexity, since reductions in the number of nodes are often offset by increased complexity of the node tests. In addition, some of the algorithms run very slowly, as we found for LMDT. We also found the accuracy of LMDT to be highly variable. Similarly, XOFN was slow. However, it produced accurate, though relatively large, trees. Promising results for hypothesis-driven constructive induction algorithms (e.g., FRINGE) have not yet been reported for non-artificial data sets. In general, methods in this category are most appropriate where there is a high level of feature interaction or where the data are best partitioned by oblique splits of the feature space.

The third category, methods that modify the search for tests, was more impressive. Promising new test selection measures include MDL, KS and, especially, C-SEP. Also, Lubinsky's hybrid measure drastically reduced tree sizes at the price of small sacrifices in accuracy. Methods for discretizing continuous features, such as Dougherty et al.'s (1995) global and T2's local discretization (Auer et al., 1995) show great promise for reducing tree size for data sets containing many continuous variables. Finally, LFC's lookahead search method produced dramatic improvements in accuracy, as well as reductions in size. However, lookahead methods sometimes yield larger, less accurate trees (Murthy & Salzberg, 1995), suggesting that it is not always appropriate.

The few methods in the fourth category restrict the database through either case selection or feature selection. We tested one method from each subcategory: ROBUST-C4.5 and SET-GEN. Both displayed somewhat variable accuracy (especially SET-GEN), but SET-GEN yielded greater reductions in tree size. When tested on a proprietary data set containing almost 200 features, many of which are irrelevant, SET-GEN excelled in accuracy relative to the other algorithms. This is expected, since SET-GEN was designed to handle irrelevant features. However, SET-GEN is very slow, while ROBUST-C4.5 is fast. *Racing* algorithms (Moore & Lee, 1994) and other fast methods for feature selection might be more appropriate than slower methods when applied to large data sets.

The last category, converting trees to other data structure, shows great promise. Converting trees to decision graphs, whether OODGs or EDAGs, can often greatly reduce data structure size without sacrificing accuracy. However, EDAGs have been tested on only a few data sets. Conversion to rule sets by RULER yields improvements in both size and accuracy. Similarly, C4.5RULES was the best algorithm we tested in terms of overall accuracy and size. While some have reported that it performs slowly (Daelemans et al., 1997), we found it to be quite fast.

Many of the methods from different categories can be combined. For example, several of the algorithms are wrappers that can operate on any tree induction algorithm; these include the hypothesis-driven feature construction algorithms, the database restriction algorithms, and the algorithms that convert a tree into another data structure. In addition, almost any method that directly controls tree size may be used in conjunction with any method that modifies the search for tests. Future research will assess the advantages of combining tree simplification methods. For instance, combined-method algorithms may be less biased than single-method algorithms, performing well in a wider range of tasks.

4.2 Case-based reasoning

This survey summarizes various methods available for simplifying decision trees, while our empirical comparison (Breslow & Aha, 1997) evaluates their relative merits. Our ultimate objective for this research is to determine which approaches might prove useful in the design of commercial case-based reasoning (CBR) systems.

CBR is a problem solving paradigm that stresses the reuse of stored solutions for new problems, where cases can be viewed as $\langle \text{problem}, \text{solution} \rangle$ pairs. The input to a CBR process is a problem p , a library L of cases, and domain-specific knowledge to guide the problem solving process. The output is a solution for the problem. According to Aamodt and Plaza (1994), the CBR problem solving cycle has four steps:

1. *Retrieve*: Fetch cases from L whose problems are most similar to p .
2. *Reuse*: Form a solution s to p from the retrieved cases' solutions, either by selecting one or by combining them.
3. *Revise*: If s fails to solve p , then revise it until it solves p .
4. *Retain*: Store the pair $\langle p, s \rangle$ in L .

Domain-specific knowledge guides these behaviours, such as by defining how problems are matched during the retrieval step and by determining how solutions are revised. Crucial design decisions for CBR systems include, among others, selecting a vocabulary of features for defining cases, an indexing method for the cases, and a memory organization for the case library. These selections are inherently problem dependent.

Approximately 15 organizations are currently marketing commercial CBR software. Several CBR shells use decision trees to index cases, including REMIND (Cognitive Systems Inc.), KATE (AcknoSoft), THE EASY REASONER (The Haley Enterprise), and KNOWLEDGE BUILDER (ServiceSoft). Some CBR research software also uses decision trees (e.g., INRECA (Auriol et al., 1995)). One motivation for simplifying decision trees in this context is to provide understandable *explanations* of CBR solutions. CBR systems typically generate *precedent* explanations, derived from the retrieved cases and the similarity measure. In addition, some systems provide more general *characteristic* explanations. When decision trees are used for case retrieval, characteristic explanations describe the tree path traversed during retrieval; this is similar to the antecedent (premise) of a rule derived from a tree.

Clear explanations can be crucial to the practical success of a CBR system. One of our sponsors told us that explanation capability is necessary for their task, and that the CBR shell that they are using yields huge, unintelligible trees for their application. Unfortunately, the shell doesn't incorporate tree simplification procedures. In general, the tree simplification procedures (if any) in CBR shells are usually limited and were not selected according to a careful study of available approaches.

While we would like to suggest which tree simplification procedures should prove useful in CBR shells, there is one problem: these procedures were designed for classification tasks, while CBR shells focus on case retrieval tasks. Extending tree-based classification algorithms to perform case retrieval is straightforward; simply store the (training) cases at the tree's leaves and retrieve the cases located in a leaf rather than a class label. However, these procedures have been tested only for their predictive accuracy, whereas retrieval methods are usually evaluated for their *precision* (i.e., percentage among those retrieved that are wanted) and *recall* (i.e., percentage among those wanted that are retrieved). It is unclear how performance on classification tasks will relate to performance on retrieval tasks, especially on typical CBR tasks where the set of features used to describe each case varies tremendously.

In our future research, we will compare tree simplification procedures in the case retrieval component of a CBR shell. We envision that procedures from several of our framework's categories will prove useful and that some will be complementary. This suggests a design in which multiple tree simplification procedures will be made available as options in a CBR shell. We intend to investigate which methods are most useful for given sets of conditions (e.g., number of features, number of features defined per case, number of cases), implement them in a CBR shell, and then perform rigorous user studies to determine which methods are worthwhile. We expect additional feedback from Navy and other DoD applications of our CBR shell.

5 Conclusion

This article introduces a framework for procedures that simplify decision trees and summarizes existing approaches. We describe the five top-level categories in this framework (i.e., explicitly controlling tree size, searching a different space of tests, changing the search for tests, database reduction, and alternative data structures) and their subcategories. We discuss example procedures from each subcategory, focusing on their ability to simplify trees while maintaining or enhancing predictive accuracy. In our discussion section, we summarize the promising approaches, briefly discuss our own initial empirical comparison of these procedures, and discuss how this research relates to the design of case-based reasoning systems.

Our future objectives include integrating the most promising tree simplification procedures in a state-of-the-art case-based reasoning shell. Thus, we plan to modify these classification procedures to store cases at their leaves, and then evaluate them on case retrieval tasks. We anticipate that different procedures will be most useful in different conditions and that some procedures will complement each other's strengths when combined.

Acknowledgements

We thank Patrick Harrison and others at NCARAI for their support of this research. We also thank Carla Brodley, Kirt Pulaski, Ian Watson, and our anonymous reviewers for their excellent feedback on an earlier version of this paper. This work was sponsored by the Office of Naval Research and by the Office of Research and Development.

References

- Almuallim, H and Dietterich, TG, 1991. "Learning with many irrelevant features" *Proc. Ninth National Conference on Artificial Intelligence* 547–552, AAAI Press.
- Auer, P, Holte, RC and Maass, W, 1995. "Theory and applications of agnostic PAC-learning with small decision trees" *Proc. Twelfth International Conference on Machine Learning* 21–29, Morgan Kaufmann.
- Auriol, E, Wess, S, Manago, M, Althoff, K-D and Traphöner, R, 1995. "INRECA: a seamlessly integrated system based on inductive inference and case-based reasoning" *Proc. First International Conference on Case-Based Reasoning*, Sesimbra, Portugal.
- Barletta, R, 1994. "A hybrid indexing and retrieval strategy for advisory CBR systems built with REMIND" *Proc. Second European Workshop on Case-Based Reasoning* 49–58, Chantilly, France.
- Bohanec, M and Bratko, I, 1994. "Trading accuracy for simplicity in decision trees" *Machine Learning* **15** 223–250.
- Breiman, L, Friedman, JH, Olshen, RA and Stone, CJ, 1984. *Classification and Regression Trees*, Wadsworth International.
- Breslow, L and Aha, DW, 1997 "Comparing tree-simplification procedures" *Proc. Sixth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL.
- Brodley, CE, 1993. "Addressing the selective superiority problem: automatic algorithm/model class section" *Proc. Tenth International Conference on Machine Learning* 17–24, Morgan Kaufmann.
- Brodley, CE, 1995a. "Recursive automatic bias selection for classifier construction" *Machine Learning* **20** 63–94.
- Brodley, CE, 1995b. "Automatic selection of split criterion during tree growing based on node location" *Proc. Twelfth International Conference on Machine Learning* 73–80, Morgan Kaufmann.
- Brodley, CE and Utgoff, PE, 1995. "Multivariate decision trees" *Machine Learning* **19** 45–77.
- Buntine, W, 1992. "Learning classification trees" *Statistics and Computing* **2**, 63–73.
- Buntine, W and Niblett, T, 1992. "A further comparison of splitting rules for decision-tree induction" *Machine Learning* **8** 75–86.
- Catlett, J, 1991. "On changing continuous attributes into ordered discrete attributes" *Proc. European Working Session on Learning* 164–178, Springer-Verlag.
- Cendrowska, J, 1987. "PRISM: An algorithm for inducing modular rules" *International Journal of Man-Machine Studies* **27** 349–370.
- Cestnik, B and Bratko, I, 1991. "On estimating probabilities in tree pruning" *Proc. Fifth European Working Session on Learning* 138–150, Springer-Verlag.

- Cheng, J, Fayyad, UM, Irani, KB and Qian, Z, 1988. "Improved decision trees: a generalized version of ID3" *Proc. Fifth International Conference on Machine Learning* 100–106, Morgan Kaufmann.
- Cherkauer, KJ and Shavlik, JW, 1996. "Growing simpler decision trees to facilitate knowledge discovery" *Proc. Knowledge Discovery and Data Mining Conference* AAAI Press.
- Cockett, JRB and Herrera, JA, 1990. "Decision tree reduction" *Journal of the ACM* **37** (4) 815–842.
- Cohen, PR and Jensen, D, 1997. "Overfitting explained" *Proc. Sixth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL.
- Cohen, WW, 1995. "Fast effective rule induction" *Proc. Twelfth International Conference on Machine Learning* 115–123, Morgan Kaufmann.
- Crawford, SL, 1989. "Extensions to the CART algorithm" *International Journal of Man-Machine Studies* **31** (2) 197–217.
- Daelemans, W, van den Bosch, A and Weijters, G, 1997. "IGTree: using trees for compression and classification in lazy learning algorithms" *Artificial Intelligence Review* (to appear).
- Danyluk, AP and Provost, FJ, 1993. "Small disjuncts in action: learning to diagnose errors in the local loop of the telephone network" *Proc. Tenth International Conference on Machine Learning* 81–88, Morgan Kaufmann.
- Devijver, PA and Kittler, J, 1982. *Pattern Recognition: a statistical approach*, Prentice-Hall.
- Dietterich, T, Kearns, M and Mansour, Y, 1996. "Applying the weak learning framework to understand and improve C4.5" *Proc. Thirteenth International Conference on Machine Learning* Morgan Kaufmann (to appear).
- Doak, J, 1992. *An evaluation of feature selection methods and their application to computer security* Technical Report CSE-92-18, University of California, Department of Computer Science.
- Dougherty, J, Kohavi, R and Sahami, M, 1995. "Supervised and unsupervised discretization of continuous features" *Proc. Twelfth International Conference On Machine Learning* 194–202, Morgan Kaufmann.
- Elomaa, T, 1994. "In defense of C4.5: notes on learning one-level decision trees" *Proc. Eleventh International Conference on Machine Learning* 62–69, Morgan Kaufmann.
- Esposito, F, Malerba, D and Semeraro, G, 1993. "Decision tree pruning as a search in the state space" *Proc. European Conference on Machine Learning* 165–184, Springer-Verlag.
- Esposito, F, Malerba, D and Semeraro, G, 1995a. "A further study of pruning methods in decision tree induction" *Proc. Fifth International Workshop on Artificial Intelligence and Statistics* 211–218, Ft. Lauderdale, FL.
- Esposito, F, Malerba, D and Semeraro, G, 1995b. "Simplifying decision trees by pruning and grafting: new results" *Proc. European Conference on Machine Learning* 287–290, Springer-Verlag.
- Fayyad, UM and Irani, KB, 1992. "The attribute selection problem in decision tree generation" *Proc. Tenth National Conference on Artificial Intelligence* 104–110, AAAI Press.
- Fayyad, UM and Irani, KB, 1993. "Multi-interval discretization of continuous-valued attributes for classification learning" *Proc. Thirteenth International Joint Conference on Artificial Intelligence* 1022–1029, Morgan Kaufmann.
- Feelders, A and Verkooijen, W, 1995. "Which method learns most from the data? Methodological issues in the analysis of comparative studies" *Proc. Fifth International Workshop on Artificial Intelligence and Statistics* 219–225, Ft. Lauderdale, FL.
- Fisher, DH, 1989. "Knowledge acquisition via incremental conceptual clustering" *Machine Learning* **2** 139–172.
- Fisher, DH and Schlimmer, JC, 1988. "Concept simplification and prediction accuracy" *Proc. Fifth International Conference on Machine Learning* 22–28, Morgan Kaufmann.
- Forsyth, RS, Clarke, DD and Wright, RL, 1994. "Overfitting revisited: an information-theoretic approach to simplifying discrimination trees" *Journal of Experimental Artificial Intelligence* **6** 289–302.
- Friedman, JH, 1977. "A recursive partitioning decision rule for nonparametric classification" *IEEE Transactions on Computers* **26** 404–408.
- Furnkranz, J and Widmer, G, 1994. "Incremental reduced error pruning" *Proc. Eleventh Conference on Machine Learning* 70–77, Morgan Kaufmann.
- Gaines, BR, 1989. "An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction" *Proc. Sixth International Workshop on Machine Learning* 156–159, Morgan Kaufmann.
- Gaines, BR, 1995. "Structured and unstructured induction with EDAGs" *Proc. First International Conference on Knowledge Discovery and Data Mining* AAAI Press.
- Gaines, BR, 1996. "Transforming rules and trees into comprehensible knowledge structures" In: UM Fayyad, G Piatetsky-Shapiro, P Smyth and R Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, MIT Press.
- Gallant, SI, 1986. "Optimal linear discriminants" *Proc. International Conference on Pattern Recognition* 849–852, IEEE Press.

- Gelfand, SB, Ravishankar, CS and Delp, EJ, 1991. "An iterative growing and pruning algorithm for classification tree design" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13** (2) 163–174.
- Gleser, MA and Collen, MF, 1972. "Towards automated medical decisions" *Computers and Biomedical Research* **5** (2) 180–189.
- Hart, PE, 1968. "The condensed nearest neighbour rule" *IEEE Transactions on Information Theory* **14** 515–516.
- Heath, D, Kasif, S and Salzberg, S, 1993. "Induction of oblique decision trees" *Proc. Thirteenth International Joint Conference on Artificial Intelligence* 1002–1007, Morgan Kaufmann.
- Holder, LB, 1995. "Intermediate decision trees" *Proc. Fourteenth International Joint Conference on Artificial Intelligence* 1056–1063, Morgan Kaufmann.
- Holte, RC, 1993. "Very simple classification rules perform well on most commonly used datasets" *Machine Learning* **11** 63–91.
- Holte, RC, Acker, LE and Porter, BW, 1989. "Concept learning and the problem of small disjuncts" *Proc. Eleventh International Joint Conference on Artificial Intelligence* 813–818, Morgan Kaufmann.
- Hunt, EB, Marin, J and Stone, PJ, 1996. *Experiments in Induction*, Academic Press.
- Hyafil, L and Rivest, R, 1976. "Constructing optimal binary decision trees is NP-complete" *Information Processing Letters* **5** (1) 15–17.
- John, G, 1995. "Robust decision trees: removing outliers in databases" *Proc. First International Conference on Knowledge Discovery and Data Mining AAAI Press*.
- John, G, Kohavi, R, and Pfleger, K, 1994. "Irrelevant features and the subset selection problem" *Proc. Eleventh International Machine Learning Conference* 121–129, Morgan Kaufmann.
- Jordan, MI, 1994. "A statistical approach to decision tree modelling" *Proc. Eleventh Conference on Machine Learning* 363–370, Morgan Kaufmann.
- Kalkanis, G, 1993. "The application of confidence interval error analysis to the design of decision tree classifiers" *Pattern Recognition Letters* **14** 355–361.
- Kearns, M and Mansour, Y, 1996. "On the boosting ability of top-down decision tree learning algorithms" *Proc. 28th ACM Symposium on the Theory of Computing* ACM Press.
- Kim, H and Koehler, GJ, 1994. "An investigation on the conditions of pruning an induced decision tree" *European Journal of Operational Research* **77** (1) 82–95.
- Kira, K and Rendell, LA, 1992a. "The feature selection problem: traditional methods and a new algorithm" *Proc. Tenth National Conference on Artificial Intelligence* 129–134, AAAI Press.
- Kira, K and Rendell, LA, 1992b. "A practical approach to feature selection" *Proc. Ninth International Conference on Machine Learning* 249–256, Morgan Kaufmann.
- Kohavi, R, 1994a. "Bottom-up induction of oblivious, read-once decision graphs" *Proc. European Conference on Machine Learning* 154–169, Springer-Verlag.
- Kohavi, R, 1994b. "Bottom-up induction of oblivious, read-once decision graphs: strengths and limitations" *Proc. Twelfth National Conference on Artificial Intelligence* 613–618, AAAI Press.
- Kohavi, R and Li, C-H, 1995. "Oblivious decision trees, graphs, and top-down pruning" *Proc. Fourteenth International Joint Conference on Artificial Intelligence* 1071–1077, Morgan Kaufmann.
- Kononenko, I, 1995. "A counter example of the stronger version of the binary tree hypothesis" *ECML-95 Workshop on Statistics and Machine Learning in KDD*, Crete.
- Kubat, M and Flotzinger, D, 1995. "Pruning multivariate decision trees by hyperplane merging" *Proc. European Conference on Machine Learning* 190–199, Springer-Verlag.
- Langley, P, 1996. *Elements of Machine Learning*, Morgan Kaufmann.
- Langley, P and Sage, S, 1994. "Oblivious decision trees and abstract cases" In: DW Aha (ed.), *Case-Based Reasoning: Papers from the 1994 Workshop* Technical Report WS-94-01, AAAI Press.
- Li, X and Dubes, RC, 1986. "Tree classifier design with a permutation statistic" *Pattern Recognition* **19** 229–235.
- Lubinsky, D, 1995. "Increasing the performance and consistency of classification trees by using the accuracy criterion at the leaves" *Proc. Twelfth International Conference on Machine Learning* 371–377, Morgan Kaufmann.
- Matheus, CJ, 1990. "Adding domain knowledge to SBL through feature construction" *Proc. Eighth National Conference on Artificial Intelligence* 803–808, AAAI Press.
- Matheus, CJ and Rendell, LA, 1989. "Constructive induction on decision trees" *Proc. Eleventh International Joint Conference on Artificial Intelligence* 645–650, Morgan Kaufmann.
- Michalski, RS, 1983. "A theory and methodology of inductive learning" In: RS Michalski, JG Carbonell and TM Mitchell (eds.), *Machine learning: an artificial intelligence approach*, Morgan Kaufmann.
- Michie, D, 1990. "Inducing knowledge from data: first principles". Unpublished manuscript for a talk given at the Seventh International Conference on Machine Learning, Austin, Texas.
- Mingers, J, 1987. "Expert systems-rule induction with statistical data" *Journal of the Operational Research Society* **38** 39–47.

- Mingers, J, 1989a. "An empirical comparison of pruning methods for decision tree induction" *Machine Learning* **4** 227–243.
- Mingers, J, 1989b. "An empirical comparison of selection measures for decision-tree induction" *Machine Learning* **3** 319–342.
- Minsky, M and Papert, S, 1969. *Perceptrons: an introduction to computational geometry*, MIT Press.
- Moore, AW and Lee, MS, 1994. "Efficient algorithms for minimizing cross validation error" *Proc. Eleventh International Conference on Machine Learning* 190–198, Morgan Kaufmann.
- Morgan, J and Sonquist, JA, 1963. "Problems in the analysis of survey data, and a proposal" *Journal of the American Statistical Association* **58** 415–435.
- Murphy, PM and Pazzani, MJ, 1991. "Constructive induction of M-of-N terms" *Proc. Eighth International Workshop on Machine Learning* 183–187, Morgan Kaufmann.
- Murphy, PM and Pazzani, MJ, 1994. "Exploring the decision forest: an empirical investigation of Occam's razor in decision tree induction" *Journal of Artificial Intelligence Research* **1** 257–275.
- Murthy, S, Kasif, S and Salzberg, S, 1994. "A system for induction of oblique decision trees" *Journal of Artificial Intelligence Research* **2** 1–32.
- Murthy, S, Kasif, S, Salzberg, S and Beigel, R, 1993. "OC1: randomized induction of oblique decision trees" *Proc. Eleventh National Conference on Artificial Intelligence* 322–327, AAAI Press.
- Murthy, SK and Salzberg, S, 1995. "Lookahead and pathology in decision tree induction" *Proc. Fourteenth International Joint Conference on Artificial Intelligence* 1025–1031, Morgan Kaufmann.
- Nakamura, Y, Abe, S, Ohsawa, Y and Sakauchi, M, 1993. "A balanced hierarchical data structure for multidimensional data with highly efficient dynamic characteristics" *IEEE Transactions on Knowledge and Data Engineering* **5** (4) 682–694.
- Niblett, T and Bratko, I, 1986. "Learning decision rules in noisy domains" *Proc. Expert Systems 1986* 25–34, Cambridge University Press.
- Norton, SW, 1989. "Generating better decision trees" *Proc. Eleventh International Joint Conference on Artificial Intelligence* 800–805, Morgan Kaufmann.
- Oliveira, AL and Sangiovanni-Vincentelli, A, 1995. "Inferring reduced ordered decision graphs of minimum description length" *Twelfth International Conference on Machine Learning* 421–429.
- Oliveira, AL and Vincentelli, AS, 1993. "Learning complex boolean functions: algorithms and applications" In: SJ Hanson *et al.* (eds.), *Advances in Neural Information Processing Systems* **5**, Morgan Kaufmann.
- Oliver, JJ, 1993. "Decision graphs—an extension of decision trees" *Fourth International Workshop on Artificial Intelligence and Statistics* 343–350, Ft. Lauderdale, FL. Unpublished.
- Oliver, JJ and Hand, DJ, 1995. "On pruning and averaging decision trees" *Proc. Twelfth International Machine Learning Conference* 430–437, Morgan Kaufmann.
- Pagallo, G, 1989. "Learning DNF by decision trees" *Proc. Eleventh International Joint Conference on Artificial Intelligence* 639–644, Morgan Kaufmann.
- Pagallo, G, 1990. *Adaptive decision tree algorithms for learning from examples*. Doctoral dissertation, Department of Computer Science, University of California at Santa Cruz, CA.
- Pagallo, G and Haussler, D, 1990. "Boolean feature discovery in empirical learning" *Machine Learning* **5** 71–100.
- Pazzani, M, Merz, C, Murphy, P, Ali, K, Hume, T and Brunk, C, 1994. "Reducing misclassification costs: knowledge-intensive approaches to learning from noisy data" *Proc. Eleventh International Conference on Machine Learning* 217–225, Morgan Kaufmann.
- Pérez, E and Rendell, LA, 1995. "Using multidimensional projection to find relations" *Proc. Twelfth International Conference on Machine Learning* 447–455, Morgan Kaufmann.
- Pitt, L and Valiant, LG, 1988. "Computational limitations on learning from examples" *Journal of the ACM* **35** 965–984.
- Quinlan, JR, 1986. "Induction of decision trees" *Machine Learning* **1** 81–106.
- Quinlan, JR, 1987a. "Simplifying decision trees" *International Journal of Man-Machine Studies* **27** 221–234.
- Quinlan, JR, 1987b. "Generating production rules from decision trees" *Proc. Tenth International Joint Conference on Artificial Intelligence* 304–307, Morgan Kaufmann.
- Quinlan, JR, 1991. "Improved estimates for the accuracy of small disjuncts" *Machine Learning* **6** (1) 93–98.
- Quinlan, JR, 1993a. *C4.5: programs for machine learning*, Morgan Kaufmann.
- Quinlan, JR, 1993b. "Combining instance-based learning and model-based learning" *Proc. Tenth International Conference on Machine Learning* 236–243, Morgan Kaufmann.
- Quinlan, JR, 1996. "Improved use of continuous attributes in C4.5" *Journal of Artificial Intelligence Research* **4** 77–90.
- Quinlan, JR and Cameron-Jones, RM, 1995. "Oversearching and layered search in empirical learning" *Proc. Fourteenth International Joint Conference on Artificial Intelligence* 1019–1024, Morgan Kaufmann.
- Quinlan, JR and Rivest, RL, 1989. "Inferring decision trees using the minimum description length principle" *Information and Computation* **80** 227–248.

- Ragavan, H and Rendell, L, 1993. "Lookahead feature construction for learning hard concepts" *Proc. Tenth International Conference on Machine Learning* 252–259, Morgan Kaufmann.
- Ragavan, H, Rendell, L, Shaw, M and Tessmer, A, 1993. "Complex concept acquisition through directed search and feature caching" *Proc. Thirteenth International Joint Conference on Artificial Intelligence* 946–951, Morgan Kaufmann.
- Rendell, L, 1985. "Substantial constructive induction using layered information compression: tractable feature formation in search" *Proc. Ninth International Joint Conference on Artificial Intelligence* 650–658, Morgan Kaufmann.
- Rendell, L and Ragavan, H, 1993. "Improving the design of induction methods by analyzing algorithm functionality and data-based concept complexity" *Proc. Thirteenth International Joint Conference on Artificial Intelligence* 952–958, Morgan Kaufmann.
- Rissanen, J, 1983. "A universal prior for integers and estimation by minimum description length" *Annals of Statistics* **11** 416–431.
- Rounds (1980). "A combined non-parametric approach to feature selection and binary decision tree design" *Pattern Recognition* **12** 313–317.
- Schaffer, C, 1992a. "Deconstructing the digit recognition problem" *Proc. Ninth International Conference on Machine Learning* 394–399, Morgan Kaufmann.
- Schaffer, C, 1992b. "Sparse data and the effect of overfitting avoidance in decision tree induction" *Proc. Tenth National Conference on Artificial Intelligence* 147–152, AAAI Press.
- Schaffer, C, 1993. "Overfitting avoidance as bias" *Machine Learning* **10** 113–152.
- Schlimmer, JC and Fisher, D, 1986. "A case study of incremental concept induction" *Proc. Fifth National Conference on Artificial Intelligence* 496–501, Morgan Kaufmann.
- Seshu, R, 1989. "Solving the parity problem" *Proc. Fourth European Working Session on Learning* 263–271, Morgan Kaufmann.
- Sethi, IK and Sarvarayudu, GPR, 1982. "Hierarchical classifier design using mutual information" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4** 441–445.
- Smyth, P, Gray, A and Fayyad, UM, 1995. "Retrofitting decision tree classifiers using kernel density estimation" *Proc. Twelfth International Conference on Machine Learning* 506–514, Morgan Kaufmann.
- Thornton, CJ, 1990. *The complexity of constructive induction*, Technical Report 463. Edinburgh, Scotland: University of Edinburgh, Department of Artificial Intelligence.
- Ting, KM, 1994. "The problem of small disjuncts: its remedy in decision trees" *Proc. Tenth Canadian Conference on Artificial Intelligence* 91–97.
- Utgoff, PE, 1988a. "ID5: an incremental ID3" *Proc. Fifth International Conference on Machine Learning* 107–120, Morgan Kaufmann.
- Utgoff, PE, 1988b. "Perceptron trees: a case study in hybrid concept representations" *Proc. Seventh National Conference on Artificial Intelligence* 601–606, Morgan Kaufmann.
- Utgoff, PE, 1989a. "Improved training via incremental learning" *Proc. Sixth International Workshop on Machine Learning* 362–365, Morgan Kaufmann.
- Utgoff, PE, 1989b. "Perceptron trees: a case study in hybrid concept representations" *Connection Science* **1** 377–391.
- Utgoff, PE, 1994. "An improved algorithm for incremental induction of decision trees" *Proc. Eleventh International Conference on Machine Learning* 318–325, Morgan Kaufmann.
- Utgoff, PE and Brodley, CE, 1990. "An incremental method for finding multivariate splits for decision trees" *Proc. Seventh International Conference on Machine Learning* 58–65, Morgan Kaufmann.
- Utgoff, PE and Brodley, CE, 1991. *Linear machine decision trees*, Technical Report 10. Amherst, MA: University of Massachusetts, Department of Computer Science.
- Utgoff, PE and Clouse, JA, 1996. *A Kolmogorov–Smirnov metric for decision tree induction*, Technical Report 96-3. Amherst, Massachusetts: University of Massachusetts, Department of Computer Science.
- Vadera, S and Nechab, S, 1994. "ID3, its children and their safety" *BCS Specialist Group on Expert Systems Newsletter* **31** 11–21.
- Wallace, CS and Patrick, JD, 1993. "Coding decision trees" *Machine Learning* **11** 7–22.
- Webb, GI, 1996. "Further experimental evidence against the utility of Occam's razor" *JAIR* **4** 397–417.
- Weiss, S and Indurkha, N, 1994a. "Small sample decision tree pruning" *Proc. Eleventh International Machine Learning Conference* 335–342, Morgan Kaufmann.
- Weiss, SH and Indurkha, N, 1994b. "Decision tree pruning: biased or optimal?" *Proc. Twelfth National Conference on Artificial Intelligence* 626–632, AAAI Press.
- Weiss, SM and Indurkha, N, 1991. "Reduced complexity rule induction" *Proc. Twelfth International Joint Conference on Artificial Intelligence* 678–684, Morgan Kaufmann.
- Wilson, D, 1972. "Asymptotic properties of nearest neighbour rules using edited data" *IEEE Transactions on Systems, Man and Cybernetics* **2** 408–421.

- Wirth, J and Catlett, J, 1988. "Experiments on the costs and benefits of windowing in ID3" *Proc. Fifth International Conference on Machine Learning* 87–99, Morgan Kaufmann.
- Yang, D, Blix, G and Rendell, LA, 1991a. "The replication problem: A constructive induction approach" *Proc. European Working Session on Learning* 44–61, Springer-Verlag.
- Yang, D, Blix, G and Rendell, LA, 1991b. "A scheme for feature construction and a comparison of empirical methods" *Proc. Twelfth International Joint Conference on Artificial Intelligence* 699–704, Morgan Kaufmann.
- Zheng, Z, 1992. "Constructing conjunctive tests for decision trees" *Proc. Fifth Australian Joint Conference on Artificial Intelligence* 355–360, World Scientific.
- Zheng, Z, 1995. "Constructing nominal X-of-N attributes" *Proc. Fourteenth International Joint Conference on Artificial Intelligence* 1064–1070, Morgan Kaufmann.
- Zhou, XJ and Dillon, TS, 1991. "A statistical-heuristic feature selection criterion for decision tree induction" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13** 834–841.