Machine Learning, 15, 223–250 (1994)

Trading Accuracy for Simplicity in Decision Trees

MARKO BOHANEC MARKO.BOHANEC@IJS.SI
"Jožef Stefan" Institute, Jamova 39, SI-61111 Ljubljana, Slovenia

IVAN BRATKO IVAN.BRATKO@NINURTA.FER.UNI-LJ.SI
University of Ljubljana, Faculty of Electrical and Computer Engineering, Tržaška 25, SI-61000 Ljubljana, Slovenia

Editor: Steve Minton

Abstract. When communicating concepts, it is often convenient or even necessary to define a concept approximately. A simple, although only approximately accurate concept definition may be more useful than a completely accurate definition which involves a lot of detail. This paper addresses the problem: given a completely accurate, but complex, definition of a concept, simplify the definition, possibly at the expense of accuracy, so that the simplified definition still corresponds to the concept "sufficiently" well. Concepts are represented by decision trees, and the method of simplification is tree pruning. Given a decision tree that accurately specifies a concept, the problem is to find a smallest pruned tree that still represents the concept within some specified accuracy. A pruning algorithm is presented that finds an optimal solution by generating a *dense* sequence of pruned trees, decreasing in size, such that each tree has the highest accuracy among all the possible pruned trees of the same size. An efficient implementation of the algorithm, based on dynamic programming, is presented and empirically compared with three progressive pruning algorithms using both artificial and real-world data. An interesting empirical finding is that the real-world data generally allow significantly greater simplification at equal loss of accuracy.

Keywords: decision trees, knowledge representation, pruning, dynamic programming

1. Introduction

1.1. Motivation

When communicating concepts, it is often convenient or even necessary to define a concept approximately. Some general idea that is simple and only approximately accurate is often more useful than a completely accurate concept definition which involves a lot of detail and exceptions. In general, the problem here is that of trading the *accuracy* for *simplicity* of a concept description. In this respect, a problem of interest is: given a completely accurate, but complex, definition of a concept, simplify the definition, possibly at the expense of accuracy, so that the simplified definition still corresponds to the concept well in general, but may be inaccurate in some details. This paper is concerned with the problem of how to do such simplifications when concepts are represented by decision trees.

It should be emphasized that our motivation for simplifying decision trees is somewhat different from, although not completely unrelated to, the typical motivation for pruning decision trees when learning from noisy data (Mingers, 1989). In learning from noisy data the assumption is that the initially induced large decision tree is inaccurate and appropriate pruning would improve its accuracy. That is, we hope that after simplification the decision tree better approximates the target concept. In our case, however, we assume that a decision tree is given which *correctly* represents the concept in question. The problem then is not

the tree's inaccuracy, but its *size*, which makes it impractical to be communicated to and understood by the user. The difficulty is that the overall main idea of the concept is hard to discern in a forest of detail. Relevant to this point are Donald Knuth's thoughts which we here quote from Michie (1989):

> The programming theorist Donald Knuth once said that to communicate a concept to a student you first have to tell him or her a lie. By subsequently working through the qualifications and exceptions, the original over-generalization is successively refined. Finally you confess: "What I first said was not strictly true. But it got us started!"

A more direct practical motivation for our work came from multi-attribute decision making. The users there actually requested that the formalized decision knowledge be presented at various levels of detail, trading accuracy for simplicity (Rajkovič & Bohanec, 1991). The higher, grosser levels are useful for quick justification of decisions. In the majority of cases the low level detail does not affect the decision at all. This application is discussed in Section 5.3 where many real-world decision knowledge bases were used for experimenting with the algorithm developed in this paper.

1.2. A detailed example

As an example consider the chess endgame with three pieces: White king, White rook and Black king (KRK ending for short). Let the concept of interest be the *legality* of a White-to-move chess position with these three pieces on the board. Assume that our description language comprises relations like: two given pieces on the same file, two pieces next to each other, etc. The concept of position legality is then to be defined in terms of these relations. This problem has been used by several authors in Machine Learning experiments (Michie, et al., 1989; Muggleton & Feng, 1990; Quinlan, 1990; Bain & Muggleton, 1991; Lavrač & Džeroski, 1991; Pazzani & Brunk, 1991; Džeroski & Bratko, 1992). There the problem was to induce a definition of the concept of legality by a learning program from examples of legal and illegal positions.

Figure 1 shows four such examples of KRK positions. Positions (a) and (d) are legal and the remaining two are illegal. Basically, according to the rules of chess a KRK position with White to move is illegal if the Black king is under attack. In position (b), the White king attacks the Black king, and in position (c) the White rook attacks the Black king. A rook attacks along a file or a rank, so an attempt to define illegality due to White rook attack on Black king could be:-

> *if White rook and Black king are on the same file or rank*
> *then the position is illegal.*

However, position (d) is an exception to this rule because White king interposes the rook's line of attack.

Let WK, WR and BK denote the position of White king, White rook and Black king, respectively. Let WR_r denote White rook rank, and WR_f White rook file, and similarly BK_r and BK_f for Black king. So, for example, the position WR of White rook is a pair:

$$WR = (WR_r, WR_f), \qquad 1 \leq WR_f \leq 8, \quad 1 \leq WR_r \leq 8$$

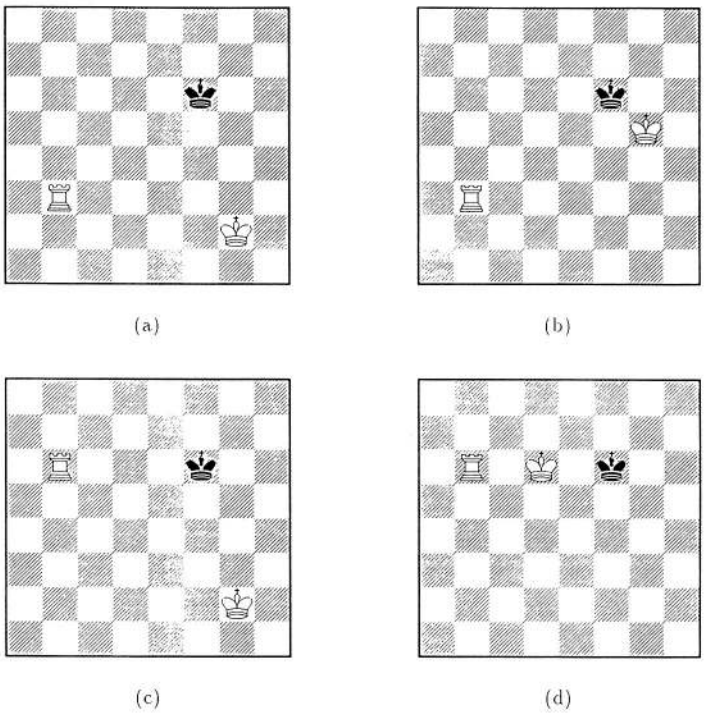

Figure 1. Examples of KRK positions, White to move. Positions (a) and (d) are legal, (b) and (c) are illegal.

Figure 2 shows a definition of the legality concept in the form of a decision tree. This tree decides between the legality and illegality depending on the positions WK, WR and BK. The meaning of the tests that appear in the internal nodes of the tree is:
$WK \approx BK$: White king's position is approximately equal to Black king's position, that is:

$$|WK_f - BK_f| \leq 1 \wedge |WK_r - BK_r| \leq 1$$

In such a situation the kings attack each other or they are on the same square.
$WR_f = BK_f$: White rook attacks Black king along a file.
$WR_r = BK_r$: White rook attacks Black king along a rank.
The conjunctive condition:

$$(WR_f = BK_f) \wedge (WR_f = WK_f) \wedge (WR_r < WK_r < BK_r)$$

describes positions in which the White king interposes the rook's attack on Black king.

Figure 3 shows the same decision tree pruned to various degrees as indicated by the dotted cross-lines. The numbers attached to these lines show the accuracy (in percent) of the remaining part of the tree after the tree has been pruned below the line and the thus

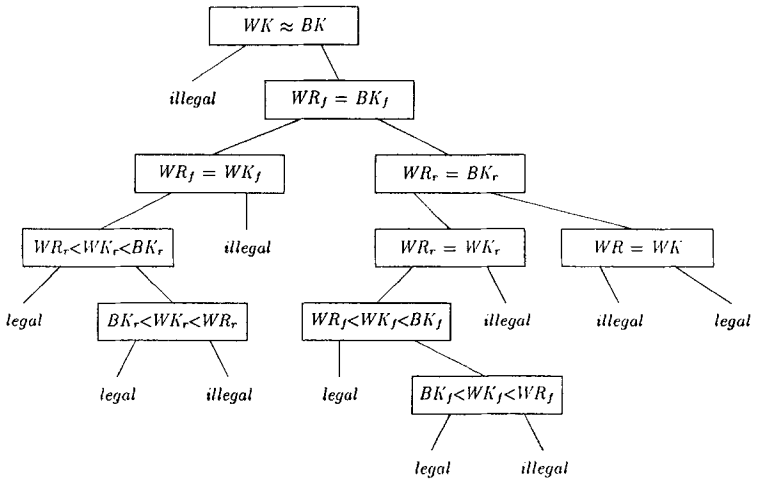

Figure 2. A decision tree that defines the concept of KRK-legality. The left branches correspond to positive outcomes of the tests.

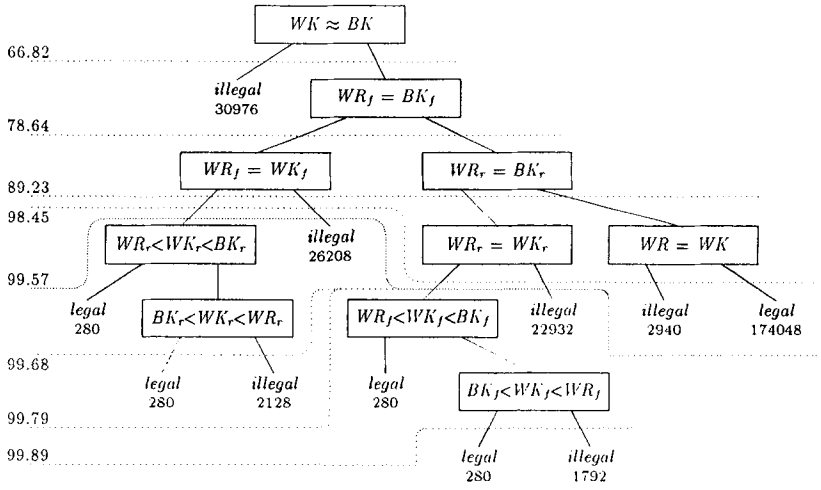

Figure 3. Pruning the KRK-legality tree to various sizes and corresponding degrees of accuracy. At each leaf, the number of corresponding positions is shown.

resulting new leaf-nodes have been assigned the majority decision (*legal* or *illegal*). Notice that the simplified tree with just four leaves (or seven nodes in total) still correctly classifies 98.45% of all the positions (in total, there are $64^3 = 262144$ positions). The tree with five leaves is 99.57% correct.

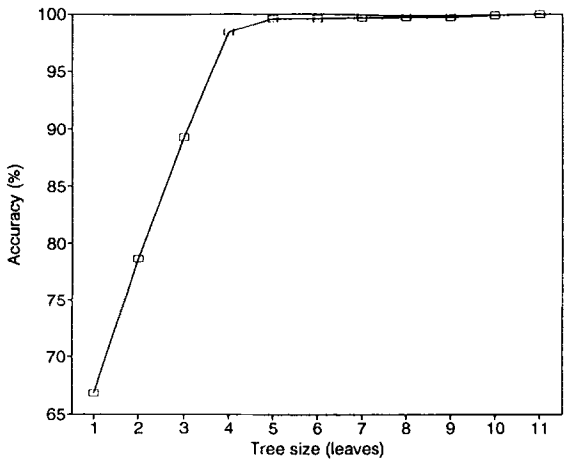

Figure 4. Accuracy vs. size in representing the KRK-legality with pruned decision trees.

The series of pruned trees indicated in Figure 3 are optimal in the sense that they optimize the accuracy within given size. For example, the tree with five leaves (nine nodes, 99.57% accuracy) indicated in Figure 3 with the emphasized pruning line is the most accurate tree among all possible pruned trees of the same size. Figure 4 relates the size of the optimal pruned tree and its accuracy.

The point illustrated by this example is that nine nodes (including leaves) are sufficient to represent the concept almost completely, and the remaining 12 nodes to the full-sized tree only account for less than 0.5% of the total accuracy. Therefore, to communicate the concept of KRK legality in an economic and still rather accurate manner, Figures 2 and 3 suggest that only the upper seven or nine nodes be used.

1.3. Related work

The idea of simplifying a concept description to improve its comprehensibility is related to Michalski's two-tiered representation (Michalski, 1990) where a complete and consistent concept description, in the form of rules, is simplified by a special procedure to maximize a description quality measure at the possible expense of accuracy. Accuracy is restored by the second tier when "flexible matching" takes place. A related idea is also discussed by Michie (1989) when the "base case" of concept specification represents a *point of view*. It is typically very approximate and is refined by adding exceptions (approach known as "exception programming"). Ripple-down rules (Compton & Jansen, 1988; Catlett, 1992) are based on a similar idea.

In this paper, the representation used is decision trees and the simplification mechanism is tree pruning. The description quality measure to be optimized is the accuracy of the simplified tree. In this sense, our work is related to the important family of learning

systems that perform induction of decision trees (Quinlan, 1986). These systems, such as ID3 (Quinlan, 1979, 1983), C4 (Quinlan, 1987), CART (Breiman, et al., 1984) and ASSISTANT (Cestnik, et al., 1987), develop decision trees from sets of examples. An approach perhaps closest to ours is Breiman, et al.'s (1984) error-complexity pruning. They weigh the relative importance of error and of complexity, and minimize, by pruning, a combined cost measure (weighed sum of error and size of decision tree). Our formulation is somewhat different: given error threshold, minimize the size. Similarly to Breiman, et al., error in our work can be viewed as their "resubstitution error".

It is well-known that *pruning* of decision trees is an effective method for dealing with noisy data (Breiman, et al., 1984; Niblett & Bratko, 1986; Niblett, 1987; Bratko, 1989; Mingers, 1989). Initially, a full-sized tree covering all the learning examples, including errors, is created. Unreliable branches of the tree are then removed.

In this paper, in contrast with learning from noisy data, the initial, unpruned decision trees will be assumed completely correct. No assumption is made regarding the way the unpruned tree is obtained. It may result from induction from examples, but this is not necessary; the tree may be obtained in ways other than induction. If induction *is* involved, then the unpruned tree is as if induced from *reliable* data defining the relation between attributes and classes completely. The representation of the concept is detailed and totally *accurate*.

The goal of pruning here is not to eliminate noise, but to simplify the description to make it easier to understand, communicate or explain. Accordingly, our goal could also be described as *summarizing* data with decision trees. When the tree is pruned, some details are eliminated, resulting in a smaller but less accurate description. A sequence of smaller and smaller trees can be generated, representing knowledge at *varying levels of detail and accuracy*. Such representations can be particularly useful in the acquisition and refinement of knowledge for expert systems (Rajkovič, et al., 1988), or in presenting ("explaining") knowledge to people (Bohanec, et al., 1988; Bohanec & Rajkovič 1988; Bohanec 1990). One can expect that the pruning of decision trees improves the comprehensibility particularly in problem domains where a small decrease of accuracy is accompanied by a large reduction of size.

This paper develops and analyses algorithms for obtaining the smallest pruned decision trees that represent concepts within some chosen accuracy. In the following section, basic concepts of decision trees and pruning are introduced. Section 3 formally defines the problem of optimal pruning and presents an algorithm for its solution. The algorithm is compared with three progressive pruning algorithms that are described in Section 4. The results of empirical measurements are presented in Section 5 and summarized in Section 7. Section 6 is a note about pruning with the algorithm in noisy domains.

2. Formalizing the problem in terms of tree pruning

We will here formalize our approach in the usual terms of attribute-based learning. Let there be a set of examples $\mathcal{E}$. Each example is an attribute-value vector accompanied by a class value. The example set is assumed consistent in the sense that each attribute-value vector in $\mathcal{E}$ corresponds to exactly one class. Mathematically, the set $\mathcal{E}$ can be viewed as a tabular

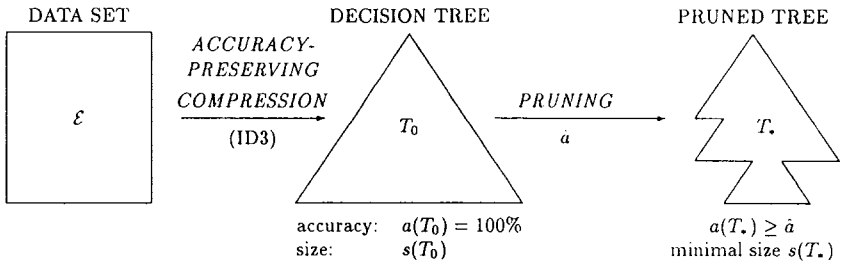

Figure 5. A two-stage approach to represent concepts by pruned trees

representation of a function that maps attribute-value vectors into the set of classes. There is no requirement that $\mathcal{E}$ covers the entire attribute space, so this function specification is partial.

The set $\mathcal{E}$ can be represented by a decision tree T_0. The internal nodes of the tree correspond to attributes, the branches of the tree are labeled by the corresponding attribute values, and the leaves are labeled by classes. All the examples from $\mathcal{E}$ can be reconstructed from T_0 in the sense that T_0 correctly classifies all the examples from $\mathcal{E}$. We therefore say that T_0 has 100% accuracy. T_0 can be constructed from $\mathcal{E}$ by a usual tree induction algorithm without pruning, such as ID3 (Quinlan, 1986). See the compression stage in Figure 5.

For a large set $\mathcal{E}$, T_0 will typically be large, which will affect its readability. For the reason of comprehensibility, we are interested in smaller trees. Pruning T_0, producing T (the pruning stage in Figure 5), will have two effects: (1) the size of the tree is reduced (a desirable effect), and (2) the accuracy will decrease (an undesirable effect). Note that here the accuracy is measured with respect to the data set $\mathcal{E}$ ("resubstitution accuracy" of Breiman, et al., 1984). The question is how to reduce the size as much as possible whereby losing as little accuracy as possible.

More formally, the goal of the pruning stage is to find T_*, a *smallest* pruned tree whose accuracy $a(T_*)$ is not lower than some given level of accuracy $\hat{a}$. T_* is referred to as an *optimal pruned tree.*

An algorithm for finding T_* is developed in Section 3. This section continues with some additional definitions of pruning, pruned trees, involved measures and their properties that are used in the algorithm.

Pruning. The pruning stage begins with tree T_0 and proceeds by pruning T_0 at some chosen nodes. Pruning at node p is done by replacing the subtree rooted at p with a leaf. The newly created leaf is labeled with the class that maximizes the accuracy; this is the majority class of the subset of $\mathcal{E}$ that corresponds to p. A tree that remains after a series of such replacements is referred to as a *pruned tree* of T_0.

Measures. There are three measures on decision trees involved in pruning: size, error frequency and accuracy. They are defined as follows:

- *size*, $s(T)$, is equal to the number of leaves of T,

- *error frequency*, $e(T)$, is the number of examples from $\mathcal{E}$ that are misrepresented by T (we say an example is *misrepresented* if it is incorrectly classified by T), and
- *accuracy*, $a(T)$, is defined as the proportion of examples from $\mathcal{E}$ that are correctly classified by T:

$$a(T) = 1 - \frac{e(T)}{E} \tag{1}$$

where E is the number of examples in $\mathcal{E}$.

Property 1. *The error frequency increases with pruning or remains unchanged.*

Proof: Let x be any node of T. Let $\mathcal{E}_x$ denote the subset of examples corresponding to x, E_x the size of $\mathcal{E}_x$ and m_x the number of examples in $\mathcal{E}_x$ belonging to the majority class.

Consider a subtree rooted at node p. Denote with $\mathcal{L}$ the set of its leaves and recall that the subsets $\mathcal{E}_\ell, \ell \in \mathcal{L}$, form a partition of $\mathcal{E}_p$. Each leaf $\ell \in \mathcal{L}$ correctly represents m_ℓ examples form $\mathcal{E}_\ell$, so the whole subtree correctly represents $\sum_{\ell \in \mathcal{L}} m_\ell$ examples from $\mathcal{E}_p$.

After T is pruned at p, the newly created leaf correctly represents m_p examples from $\mathcal{E}_p$. Consequently, the difference in error frequency after and before the pruning is

$$r_p(T) = (E_p - m_p) - \sum_{\ell \in \mathcal{L}} (E_\ell - m_\ell) = \sum_{\ell \in \mathcal{L}} m_\ell - m_p \tag{2}$$

When some class c is the majority class in $\mathcal{E}_p$ and in all the subsets $\mathcal{E}_\ell$, then $m_p = \sum m_\ell$, so $r_p(T) = 0$. On the other hand, when there is at least one leaf with a majority class other than c, then $m_p < \sum m_\ell$ and $r_p(T) > 0$. ■

Note that Property 1 implies that the accuracy (1) *decreases* with pruning or remains unchanged.

Property 2. *Disjoint subtrees of T are mutually independent with respect to $r_p(T)$. That is, $r_p(T)$ only changes if the subtree rooted at p changes.*

Proof: Notice in (2) that $r_p(T)$ depends only on node p and its descendants, but not on the rest of the tree. Consequently, $r_p(T)$ remains unchanged after pruning a subtree of T that is disjoint to the subtree rooted at p. ■

3. Optimal pruning algorithm

The task of an optimal pruning algorithm is to find an optimal pruned tree T_* with given:

1. initial decision tree T_0 whose accuracy $a(T_0) = 1$, and
2. required minimal accuracy of the pruned tree $\hat{a} \in [0, 1]$.

T_* is a smallest pruned tree of T_0 that satisfies $a(T_*) \geq \hat{a}$. If there are multiple solutions, it will be considered sufficient to find one.

This task may seem straightforward. One can design, or select from the available ones (Mingers, 1989), an algorithm that progressively selects branches of T_0 and prunes them until the overall accuracy becomes lower than $\hat{a}$. However, algorithms of this type fail to find an optimal pruned tree for some values of $\hat{a}$; the resulting tree may not be the smallest one, as illustrated in Section 5. In order to achieve optimal solutions, a specialized algorithm is needed for this specific task. Such an algorithm is gradually developed in this section. We first introduce the concept of optimal pruning sequences as the basis for the algorithm. This is followed by the description of the algorithm's data structures and procedures, and illustrated by an example. Finally, a theoretical analysis of the algorithm's time complexity is carried out.

3.1. Optimal pruning sequence

An approach to optimal pruning was suggested by Breiman, et al. (1984, p. 65). In this approach, a sequence of increasingly pruned trees of T_0 is constructed

$$T_0, T_1, T_2, \ldots, T_n \tag{3}$$

such that

1. $n = s(T_0) - 1$,
2. the trees in the sequence decrease in size by one, i.e., $s(T_i) = s(T_0) - i$ for $i = 0, 1, \ldots, n$ (unless there is no pruned tree of the corresponding size), and
3. each T_i has the highest accuracy among all the pruned trees of T_0 of the same size (therefore sequence (3) is called *optimal pruning sequence*).

In other words, this is a "dense" sequence of optimal pruned trees with respect to the number of their leaves. Note that in the case of a non-binary tree it might be impossible to prune T_0 so that the resulting tree would have, say ℓ leaves. In this case, the corresponding element $T_{s(T_0)-\ell}$ is undefined.

Given $\hat{a}$, the optimal solution T_* can be easily found in (3): simply, take T_k, the smallest tree in the sequence such that $a(T_k) \geq \hat{a}$.

The problem is how to *construct* an optimal pruning sequence *efficiently* in polynomial time with respect to the size of T_0. An extensive search of all pruned trees of T_0 is unfeasible, since their number grows extremely fast with tree size. Even with small trees of, say, 10 internal nodes there might be too many pruned trees to cope with. For example, in a complete binary decision tree whose leaves are all at level h, there are $[b^{2^h}]$ pruned trees, where $b \doteq 1.5028$ and $[x]$ the nearest integer less than x (Breiman, et al., 1984, p. 284). There is an even larger number of distinct sequences of pruning T_0 up to the root.

When suggesting the above approach, Breiman, et al. (1984) stated that an efficient algorithm exists for the construction of optimal pruning sequences. Unfortunately, they did not present the algorithm; they rather adopted another method called *minimal cost-complexity*

pruning (presented here in Section 4). In fact, Leo Breiman (Private Communication, December 1990) did implement such an algorithm for optimal pruning; he was satisfied that it worked, but no further development was done, and the algorithm was not published. Here we present an independently developed optimal pruning algorithm.

3.2. *Data representation*

A sequence of pruned trees of T will be represented by a structure $S = \langle R, P \rangle$. R and P are arrays of integers and sets of nodes, respectively. The elements are denoted $R[i]$ and $P[i]$, where $i = 0, 1, \ldots, n$. The interpretation of S is as follows:

- each element of S, $S[i] = \langle R[i], P[i] \rangle$, represents a pruned tree of size $s(T) - i$;
- $P[i]$ contains the set of nodes at which T was pruned;
- $R[i]$ is equal to the increase of error frequency, $r(T)$, that occurs when the nodes from $P[i]$ are replaced by leaves.

Since the error cannot decrease with pruning (Property 1), $R[i] < 0$ is used denote that T cannot be pruned by i leaves. Note that $R[0]$ is always 0, since there is no additional error when T is left unchanged.

3.3. *Algorithm (OPT)*

The goal of the optimal pruning algorithm (OPT) is to construct an optimal pruning sequence S_0 for decision tree T_0. The construction is *recursive* in that each subtree of T_0 is again a decision tree with its own optimal pruning sequence. The algorithm starts by constructing sequences that correspond to small subtrees near the leaves of T_0. These are then combined together, yielding sequences that correspond to larger and larger subtrees of T_0, until S_0 is finally constructed.

A single leaf is a smallest subtree of T_0. It cannot be pruned at all. Consequently, the corresponding pruning sequence contains only that single leaf.

A somewhat larger tree T consists of a single node and k leaves (Figure 6a). This tree can be pruned only at the root. The pruned tree is of size 1, meaning that the size decreased by $k - 1$. Consequently, the pruned tree is T_{k-1} in sequence (3). When pruned, the error increased by $r_0(T)$ according to (2). Therefore, the sequence S can be constructed such that $P[k-1] = \{0\}$ and $R[k-1] = r_0(T)$. The elements from $R[1]$ to $R[k-2]$ should be assigned some negative value since they represent non-existing pruned trees of T.

When dealing with larger trees, OPT recursively combines optimal sequences of their subtrees. To illustrate one level of such combining, consider a general decision tree T_0 that consists of a root and k subtrees, possibly leaves (Figure 6b). At this stage, optimal pruning sequences $S_1, S_2, \ldots, S_k$ that correspond to the subtrees are already known. They should be combined together to yield S_0, the sequence that corresponds to T_0.

There are two observations that lead to the combining part of the algorithm. First, subtrees $1, 2, \ldots, k$ (Figure 6) are disjoint and, according to Property 2, independent with respect

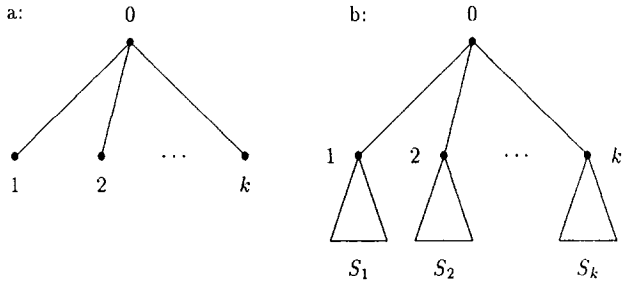

Figure 6. Two decision trees to illustrate optimal pruning.

to pruning. For example, if a part of subtree 1 is pruned, the error changes by the same amount regardless of any previous pruning in, say, subtree 2. Once both parts have been pruned, the total error equals to the sum of the partial errors. Consequently, the sequences S_i are mutually independent; there is no need to change them when combining them into S_0.

The second observation is that a tree can be pruned to a given size by several combinations of the pruning of its subtrees. For example, prune T_0 so that its size is reduced by two. This can be achieved by reducing the size of subtree 1 by two according to its optimal sequence. Another try might be to reduce the size of subtrees 1 and 2 by one. Among all such combinations, the one that minimizes $r(T_0)$ should be chosen and assigned to $S_0[2]$. A similar combinatorial search for the remaining sizes could be used to obtain the elements of S_0.

Such a combinatorial task can be efficiently implemented by *dynamic programming* (see for example Sedgewick, 1983). S_0 can be obtained iteratively by combining two sequences in each iteration. This iterative combination can be justified as follows. Let T_{ij} denote any pruned tree in sequence S_i. To optimally prune T_0 to a given size s, find a combination of its subtrees $T_{1j_1}, T_{2j_2}, \ldots, T_{kj_k}$ that minimizes the total error of these subtrees:

$$\min_{s(T_{1j_1})+s(T_{2j_2})+\cdots+s(T_{kj_k})=s} (e(T_{1j_1}) + e(T_{2j_2}) + \cdots + e(T_{kj_k})) \tag{4}$$

Note that the restriction on the subtrees is that their total size be equal to s.

Expression (4) can be optimized iteratively by combining optimal sequences for subtrees 1 and 2 into an optimal sequence for a fictitious tree composed of these two subtrees, and using this sequence in place of the first two terms in the sums for size and error.

The complete algorithm is shown in Figure 7. Procedure OPT takes T_0 and constructs the corresponding sequence S_0. In step 1, S_0 is initialized by inserting T_0 into the sequence (steps 1.1 and 1.2) and assuming that the remaining elements are empty (step 1.3). Step 2 performs the majority of tasks discussed so far. For each subtree of T_0, the corresponding sequence S_i, is first constructed recursively by OPT (step 2.1). In step 2.2, S_0 is combined with S_i, yielding the optimal sequence with respect to pruning subtrees 1, 2, ..., i. Finally, pruning of T_0 at the root is added to S_0 in step 3.

```
Procedure OPT(T0, S0):   { construct optimal pruning sequence S0 of T0 }

1.      Initialize S0:
1.1.        P0[0] := empty set;
1.2.        R0[0] := 0;
1.3.        for i ∈ [1, n] do R0[i] := −1;
2.      for i ∈ [1, k] do                       { for subtrees of T0 }
            begin
2.1.            OPT(subtree i, Si);             { recursively construct Si }
2.2.            Combine(S0, Si, S0);            { combine S0 and Si into S0 }
            end;
3.      if n > 0 then                           { if T0 is not a single leaf }
            begin                               { add pruning at the root }
3.1.            P0[n] := {root of T0};
3.2.            R0[n] := r0(T0);
            end.

Procedure Combine(S', S'', S):   { combine sequences S' and S'' into S }

1.      Initialize S as in step 1 of OPT;
2.      for i ∈ [0,length(S')] such that R'[i] ≥ 0 do
2.1.        for j ∈ [0,length(S'')] such that R''[i] ≥ 0 do
              begin
2.1.1.          s := i + j;                     { combined reduction of size }
2.1.2.          r := R'[i] + R''[j];            { combined error }
2.1.3.          if R[s] < 0 or r < R[s] then
                  begin
2.1.3.1.            P[s] := P'[i] ⋃ P''[j];
2.1.3.2.            R[s] := r;
                  end;
              end.
```

Figure 7. Algorithm for constructing optimal pruning sequences.

Procedure *Combine* takes two sequences, S' and S'', and combines them into one, S. All existing pairs $S'[i]$ and $S''[j]$ are considered in a double loop. Each pair represents a situation where i and j leaves are optimally pruned from the subtrees that correspond to S' and S'', respectively. When $S'[i]$ and $S''[j]$ are applied together, the size of T_0 decreases by $s = i + j$ and the error increases by $r = R'[i] + R''[j]$. The combination is recorded in S (steps 2.1.3.1 and 2.1.3.2) when it represents a new solution ($R[s] < 0$) or has lower error than any of the previously considered combinations ($r < R[s]$).

Note that it is possible to implement *Combine* more efficiently without introducing a new sequence S, but rather updating S_0 directly (in place) based on S_i and previous values of S_0. This can be achieved by the reorganization of the two loops. However, this results in a somewhat larger and less understandable program. Also, space can be economized in OPT by using the same local array for all S_i variables.

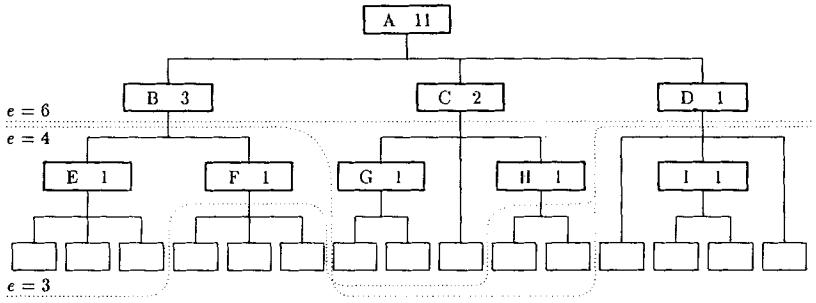

Figure 8. Sample decision tree T_0 with shown increase of error $r_p(T_0)$. Dotted lines indicate optimal pruning to various degrees of error e.

3.4. Example

To illustrate the construction of an optimal pruning sequence, take a decision tree T_0 shown in Figure 8. Only the most relevant information is shown: the nodes are labeled with letters and $r_p(T_0)$ is given for each node. To recall, $r_p(T_0)$ represents the number of examples that become misrepresented when T_0 is pruned at p; it follows from (2). In Figure 8, leaves are presented only for the purpose of determining subtree size.

The construction of the optimal pruning sequence that corresponds to node A, S_{A}, proceeds recursively from the bottom to the top of T_0. Therefore, the subtree rooted at node E is considered first. This subtree can be pruned only at the root. When pruned, three leaves are removed, but one new leaf appears. The size is therefore reduced by two leaves. In addition, the error increases by 1. The corresponding sequence S_{E} therefore consists of only one entry $S_{\mathrm{E}}[2]$ such that $R_{\mathrm{E}}[2] = 1$ (error) and $P_{\mathrm{E}}[2] = \{\mathrm{E}\}$ (place of pruning). This is obtained by steps 1 and 3 of OPT (Figure 7). The result, S_{E}, is denoted as follows:

$$S_{\mathrm{E}} = [2 : (1; \mathrm{E})]$$

The sequences that correspond to the remaining small subtrees, rooted at nodes F to I, are constructed similarly:

$$S_{\mathrm{F}} = [2 : (1; \mathrm{F})]$$
$$S_{\mathrm{G}} = [1 : (1; \mathrm{G})]$$
$$S_{\mathrm{H}} = [1 : (1; \mathrm{H})]$$
$$S_{\mathrm{I}} = [1 : (1; \mathrm{I})]$$

As soon as S_{E} and S_{F} are known, they are combined, giving S_{B}, the sequence that corresponds to the subtree rooted at node B. All the combinations of the elements of S_{E} and S_{F} are considered by the *Combine* procedure (Figure 7). The combinations are the following: pruning at E, pruning at F, and pruning at both E and F. The first two result in the same reduction of size, so the one that minimizes $r_p(T_0)$ should be chosen. Since they are equal in this case, pruning at F appears in the solution (the selection depends on the loops of *Combine*). Finally, OPT adds pruning at the root B (step 3), giving

Table 1. Optimal sequence of pruning the tree in Figure 8.

size s [leaves]	error e	accuracy a [%]	pruned at nodes
14	1	94.44	1
13	1	94.44	F
12	1	94.44	D, H
11	2	88.89	D
10	2	88.89	D, F
9	3	83.33	D, F, H
8	3	83.33	C, D
7	4	77.78	B, D
6	4	77.78	C, D, F
5	6	66.67	B, D, G, H
4	5	72.22	C, D, E, F
3	6	66.67	B, C, D
2	-	-	
1	11	38.89	A

$$S_{\mathrm{B}} = [2 : (1; \mathrm{F})\ 4 : (2; \mathrm{E},\ \mathrm{F})\ 5 : (3; \mathrm{B})]$$

Similarly, S_{C} and S_{D} are obtained:

$$S_{\mathrm{C}} = [1 : (1; \mathrm{H})\ 2 : (2; \mathrm{G},\ \mathrm{H})\ 4 : (2; \mathrm{C})]$$
$$S_{\mathrm{D}} = [1 : (1; \mathrm{I})\ 3 : (1; \mathrm{D})]$$

In order to find the final solution S_{A}, the algorithm first combines S_{B} and S_{C}, giving

$$S_{\mathrm{A}}(\text{intermediate}) =$$
$$[1 : (1; \mathrm{H})\ 2 : (1; \mathrm{F})\ 3 : (2; \mathrm{F},\ \mathrm{H})\ 4 : (2; \mathrm{C})\ 5 : (3; \mathrm{B})$$
$$6 : (3; \mathrm{C},\ \mathrm{F})\ 7 : (5; \mathrm{B},\ \mathrm{G},\ \mathrm{H})\ 8 : (4; \mathrm{C},\ \mathrm{E},\ \mathrm{F})\ 9 : (5; \mathrm{B},\ \mathrm{C})]$$

The final pruning sequence S_{A} is obtained by combining the above intermediate S_{A} with S_{D}. The result is interpreted in Table 1. The sequence is dense with respect to size; pruned trees of all sizes are available, except for the size of two which can not be obtained at all.

The accuracy column in Table 1 is obtained using equation (1), where the total number of examples E equals to 18. In general, the accuracy decreases with pruning, but there is an exception at size $s = 4$. Note that this exception does not contradict Property 1; the tree at $s = 4$ is *not* a pruned tree of the tree at $s = 5$, but rather of the one at $s = 6$. In the latter case, the accuracy did decrease in accordance with Property 1.

The sequence in Table 1 illustrates an interesting property of optimal pruning sequences, which was already observed by Breiman, et al. (1984): the sequence is not necessarily formed by a progressive upward pruning. Some branches that were previously removed may reappear in the sequence. For example, the subtree rooted at node F is pruned at $s = 13$, but it reappears in the sequence at $s = 12$, 11 and 8. Some other examples can be found in Table 1 as well.

3.5. *Time complexity*

To analyze time complexity of OPT with respect to the size of T_0, consider a complete k-ary decision tree of depth $A+1$ (A can be interpreted as the number of attributes in ID3-like decision trees). This means that each node has exactly k-descendants and all the leaves appear at level A (assuming that the root is at level 0). The number of leaves is $s = k^A$. More generally, the number of nodes at level d is $n_d = k^d$ and a subtree rooted at level d has $s_d = k^{A-d}$ leaves.

In the analysis we assume that T_0 has already been built and that the error differences, $r_p(T_0)$, are known for all nodes (they are easily obtainable in the tree development stage). Time complexity is measured by the number of comparisons in the innermost loop of *Combine* (step 2.1.3). Given two sequences of lengths L' and L'', *Combine* makes at most $(L'+1)(L''+1)$ such comparisons.

The number of comparisons needed to construct a sequence that corresponds to *one* node depends on the level of that node in the tree. Consider a node at level d. The corresponding partial sequence S_0 is gradually developed by combining S_0 with sequences S_i whose length is $L = s_{k-1} - 1$. Initially, the length of S_0 is 0, but it increases by L after each step. Consequently, the number of elementary comparisons needed to construct one sequence is at most

$$\begin{aligned}\sum_{i=0}^{k-1}(iL+1)(L+1) &= s_{d-1}\Big(k + L\sum i\Big)\\ &= s_{d-1}\left(k + (s_{d-1}-1)\frac{(k-1)(k-2)}{2}\right)\\ &= O(k^2 s_{d-1}^2)\\ &= O(s_d^2).\end{aligned}$$

Therefore, the time spent for each node is proportional to s_d^2.

In order to construct the final sequence, OPT recursively visits each internal node exactly once. There are n_d nodes on each level d; OPT visits all levels from 0 to $A-1$, but skips the leaves at level A. Consequently, the total number of elementary comparisons is proportional to

$$\begin{aligned}\sum_{d=0}^{A-1} n_d s_d^2 &= \sum k^d k^{2(A-d)} = \sum k^{2A-d}\\ &= k^{A+1}\sum k^d = k^{A+1}\frac{k^A - 1}{k-1}\\ &= O(k^{2A})\\ &= O(s^2).\end{aligned}$$

In summary, the time complexity of OPT is quadratic with respect to the number of leaves of T_0.

4. Progressive pruning algorithms

In this section we compare OPT with other pruning algorithms that generate sequences of progressively pruned trees. We call these algorithms *progressive* pruning algorithms and consider the following three of them:

1. GREEDY; a simple progressive pruning algorithm, defined below,
2. MCC_0: the so-called *minimal cost-complexity* pruning algorithm that was developed by Breiman, et al. (1984) and used in their CART system, and
3. MCC_1: a derivative of MCC_0.

4.1. Basic concepts

The progressive pruning algorithms share several characteristics. They construct a sequence of smaller and smaller pruned trees of T_0:

$$T_0, T_1, T_2, \ldots, T_m \tag{5}$$

Basically, the sequence is similar to OPT's (3). However, sequence (5) is formed by a progressive pruning of T_0, meaning that each subsequent tree in the sequence is obtained from its predecessor by removing one or more subtrees. Put another way, pruned trees in the sequence are nested. As a consequence of this, the accuracy of the trees in the sequence decreases monotonically. Another difference is that the size of trees usually decreases faster than in OPT.

The three algorithms are iterative. First, they take T_0 as a current pruned tree, T, which is then pruned by iterating the following three steps:

1. *select* a node p of T;
2. *prune* T at node p;
3. if T meets *requirements*, then *append* T to sequence (5).

The steps are repeated until T_0 is pruned up to the root or the accuracy of T falls below threshold $\hat{a}$.

The algorithms differ in steps 1 and 3. In step 1, each algorithm applies its own *selection criterion* that determines the node to be pruned next. Similarly, different *requirements* are used in step 3 to decide whether to make T a member of the sequence or not.

4.2. Simple progressive pruning (GREEDY)

GREEDY is probably the simplest and most intuitive algorithm that can be designed for finding small pruned trees of high accuracy. While pruning, it simply follows the rule: keep accuracy high and size low. Therefore, the following criteria are used in GREEDY:

Table 2. A pruning sequence of GREEDY.

size	accuracy	selected node
12	94.44	D
10	88.89	E
8	83.33	F
7	77.78	B
6	72.22	G
3	66.67	C
1	38.89	A

- The *selection* is based on the increase in error with respect to the current tree, $r_p(T)$; the node p that minimizes $r_p(T)$ is selected. If there are several such nodes, the one that is the root of the largest subtree is taken.

- The *requirements* are null, meaning that each pruned tree is added to the sequence.

Example: Table 2 shows the sequence obtained by GREEDY when pruning the tree from Figure 8. In the first iteration, the node D was selected for pruning due to its small error ($r_D(T) = 1$) and large subtree (4 leaves). The process continued by choosing nodes E and F (error 1, size 3). After the subtrees E and F had been pruned off, the error difference of the parent node B with respect to the current tree decreased from 3 to $3 - 2 \times 1 = 1$. Consequently, the node B was chosen for pruning in the next step. The rest of the sequence was obtained in a similar way.

In comparison with the pruning sequence of OPT (Table 1), the GREEDY's one is about half its length, so the density of the generated trees with respect to size is lower. Additionally, there is a pruned tree of size 6 in the sequence (underlined in Table 2) whose accuracy (72.22%) is lower than the accuracy of the optimal tree of the same size (77.78%). The sequence of GREEDY is therefore suboptimal. □

4.3. Minimal cost-complexity pruning (MCC)

Breiman, et al. (1984, pp. 66–71) have developed a more sophisticated method of progressive pruning, called *minimal cost-complexity pruning*. It was made a part of their CART system. In (Mingers, 1989), this method is referred to as *error-complexity* pruning.

MCC takes into account classification error costs that are obtained as follows. The error cost of node p is defined as the proportion of examples from $\mathcal{E}$ that are misclassified at node p. Using the notation of Section 2, the error cost equals to

$$R_p = \frac{r_p(T)}{E} \tag{6}$$

Table 3. Pruning sequences of MCC_0 (marked with *) and MCC_1 (all).

	size	accuracy	selected node
*	12	94.44	D
	10	88.89	E
	8	83.33	F
*	4	72.22	C
*	3	66.67	B
*	1	38.89	A

Let T_p be a subtree rooted at node p of T. Then, the error cost of the subtree, $R(T_p)$, equals to the sum of error costs of its leaves.

When pruning a tree, MCC extends the error measure $R(T_p)$ by adding a *complexity* cost of T_p:

$$R_\alpha(T_p) = R(T_p) + \alpha s_p \tag{7}$$

Here, α is the cost of an additional leaf in the tree, and s_p is the number of leaves of T_p.

The pruning method now works as follows. For each $\alpha > 0$, a pruned tree is found that minimizes the total cost. When α is small, the additional cost for each leaf in the tree is small too, so the pruned tree will be large. On the other hand, with sufficiently large α, the size will prevail over the error and, consequently, the tree will be pruned up to the root.

Although α runs through a continuum of values, only a finite number of solutions exist. They can be found by a progressive pruning algorithm (Breiman, et al., 1984, p. 69 and 294). The algorithm fits well into the general schema presented above. In each iteration, a node p is *selected* that minimizes

$$\alpha_p = \frac{R_p - R(T_p)}{s_p - 1} = \frac{r_p(T)}{E(s_p - 1)} \tag{8}$$

The original version of MCC puts a special *requirement* on whether the current tree should be added to the sequence or not. Namely, α_p generally increases with pruning, but may remain constant in some iterations. A tree is added only when α_p has increased (Breiman, et al., 1984, p. 70). Here, this version of MCC is referred to as MCC_0.

The above requirement makes sense in the context of learning from noisy data which was the purpose of this algorithm. However, for our purpose the requirement is unsuitable as it makes the sequences more sparse than necessary. For this reason, a modified version of the algorithm was tested as well that imposes no restrictions on the generated pruned trees. This version is referred to as MCC_1.

MCC generates pruned trees that have an important property (Breiman, et al., 1984, p. 71): each tree is optimal with respect to size. This means that no other pruned tree of the same size exists whose accuracy would exceed the one obtained. Notice in (8) that when s_p is fixed, both MCC and OPT minimize the same measure.

Example: Take again the tree of Figure 8 and prune it with MCC_0 and MCC_1. The result is shown in Table 3. The whole sequence corresponds to MCC_1. Only six pruned trees

were generated. However, they are all optimal with respect to size (compare with Table 1). The trees generated by MCC_0 are marked with an asterisk in Table 3. Two trees of MCC_1 were skipped, resulting in the sequence of only four trees. □

4.4. *Complexity of progressive pruning algorithms*

The most time-consuming step in progressive pruning algorithms is the selection of the next node to be pruned. In MCC (Breiman, et al., 1984, pp. 293–296), the search proceeds as follows. In each internal node of the tree, the algorithm maintains the current value of the minimal α_p (8) that appears in the subtree rooted at that node. Starting at the root, the algorithm follows the path of minimal such values until it locates the node. A node at level d is found in d steps. After the tree has been pruned, the algorithm spends additional d steps to update the path from the node up to the root. A similar approach can be used to implement the search in GREEDY.

In the worst case, all internal nodes have to be visited in this way. Using the notation from Section 3.5, in a balanced tree the pruning takes time proportional to

$$\sum_{d=1}^{A-1} 2dn_d = \sum 2dk^d = O(Ak^A) = O(s \log s)$$

However, when the tree is totally unbalanced (i.e., linear), this approach may take up to $O(s^2)$ steps. Consider a linear k-ary tree of depth A. It contains A internal nodes and $s = A(k-1)+1$ leaves. Each level from 1 to $A-1$ contains exactly one internal node. Thus, the time needed to prune the tree is at most

$$\sum_{d=1}^{A-1} 2d = (A-1)(A-2) = O(A^2) = O(s^2)$$

Therefore, the worst case time complexity of progressively pruning a tree with s leaves is between $O(s \log s)$ and $O(s^2)$, depending on the degree of balance in the tree.

5. Empirical comparison of the algorithms

The four pruning algorithms (OPT, GREEDY, MCC_0 and MCC_1) were empirically tested on artificial and real-world data. The experimental procedure and obtained results are presented in this section.

5.1. *Experimental procedure*

According to the approach of Figure 5, a decision tree was developed by ID3 from each set of examples. The tree was then independently pruned by the four algorithms. In all cases, the threshold $\hat{a}$ was set to 0, so the complete sequences of pruned trees were constructed. In each sequence, the accuracy of pruned trees was observed with respect to their size.

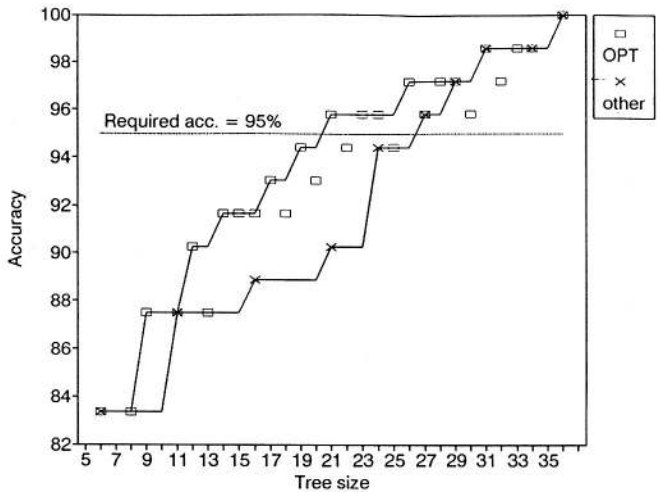

Figure 9. A sample measurement of two pruning sequences.

An example of one such measurement is shown in Figure 9. Two pruning sequences are shown. The first one, consisting of points shown by □, is optimal. Note that the accuracy does not strictly increase with size. The second sequence is suboptimal and contains less trees than the first one.

Two additional adjustments were applied to each sequence in order to:

- make optimal pruning sequence monotone with respect to size, and
- make all sequences comparable with each other regardless of their density.

By the the first adjustment, we remove from the sequence each tree whose accuracy is lower than the accuracy of a smaller tree in the sequence. When all such trees are removed, the sequence becomes monotone. The second adjustment generalizes the accuracy of a sequence over all sizes. When the accuracy is undefined at some size, the accuracy of the first smaller tree is assumed there.

The two adjustments result in accuracy as a monotonically increasing function of size (line plots in Figure 9). The accuracy is defined for each size, so various measurements can be compared or averaged regardless of the density of the initially obtained pruning sequence.

5.2. *Experiments using artificial data*

The objective of the first experiment was to observe and compare the *average* (expected) performance of the four algorithms under well-controlled conditions. Therefore, problem domains of various dimensions were constructed artificially. The dimensions were determined by:

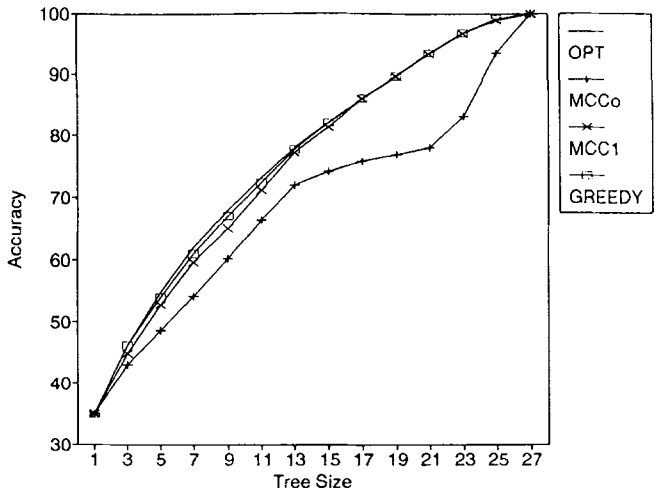

Figure 10. Average accuracy in pruning sequences (artificial domain with $A = 3$, $M = 4$ and $N = 3$).

- A: number of attributes (varied in the range from 2 to 5),
- M: number of classes (2 to 5), and
- N_i: number of values of the i-th attribute (2 to 4); denoted simply by N when all the attributes have the same number of possible values.

In each domain, 100 sets of examples were randomly generated such that

- the examples completely and consistently covered the problem domain, and
- the probability distribution of classes was uniform.

For each domain, 100 corresponding decision trees were developed by ID3 and independently pruned by OPT, GREEDY, MCC_0 and MCC_1, and the accuracy of the obtained pruning sequences was averaged.

Figure 10 shows the accuracy of pruning sequences that were obtained in a domain defined by three three-valued attributes and four classes. In this domain, MCC_0 performed inferior to the remaining three algorithms. Since MCC guarantees to generate trees that are optimal with respect to size, the only reason for such performance is the sparsity of sequences; the number of generated trees was too small. A qualitatively similar behavior of MCC_0 was observed in the remaining experiments. As it was actually designed for a different purpose, and seems unsuitable for the present one, we omit MCC_0 from further comparison.

The algorithms OPT, GREEDY and MCC_1 performed almost equally well in the domain shown in Figure 10. There was a particularly small difference between OPT and GREEDY,

which occurred only in the range from 5 to 15 leaves. The situation was similar in other uniform domains, i.e., domains where all the attributes had the same number of values. All the three algorithms performed similarly with respect to accuracy. In nonuniform domains, however, the differences in accuracy between the three algorithms tended to increase.

In general, the differences between OPT and GREEDY were small in the average of 100 experiments. However, there were specific sequences where GREEDY performed considerably worse than OPT. For example, in the domain where $A = 3$, $M = 4$, $N_1 = 2$ and $N_2 = N_3 = 3$, GREEDY "guessed" exactly the same sequence as OPT (in terms of accuracy) in 67 of 100 tests. In each of the remaining 33 cases, however, there was a range of sizes where a substantial difference between OPT and GREEDY occurred, reaching 11.12%.

There are two explanations of such cases. One reason is that GREEDY, when choosing between equivalent subtrees to be pruned, looks only one step ahead and sometimes makes a wrong decision. The second cause can be explained by the property of optimal pruned trees that was already demonstrated in Section 3.4: a subtree that was cut off in some stage of pruning may reappear later as a part of an optimal solution. Algorithms that work by progressively pruning a decision tree, such as GREEDY, can never discover such sequences. It seems that such cases occur more frequently in nonuniform than in uniform problem domains.

5.3. *Experiments using real-world data*

Real-world sets of examples were taken from knowledge bases that were developed with DECMAK, an expert system shell for multi-attribute decision making (Bohanec, et al., 1983; Bohanec & Rajkovič, 1987). Basically, DECMAK is intended for decision-making problems where an option has to be chosen from a set of possible ones so as to best satisfy the goals of the decision maker. In decision theory (French, 1986), this is called the option with the highest *utility*.

There are two principal stages of solving such problems. First, a knowledge base is developed that consists of

- a hierarchy of attributes that describe the problem domain (similar as in Shapiro's (1987) structured induction);
- the so-called *utility functions*, that define a mapping from lower-level to higher-level attributes in the hierarchy; the functions are defined by *examples*, provided by an expert.

In the second stage, the knowledge base is used to evaluate options and explain their characteristics.

Since 1981, 40 knowledge bases have been developed with DECMAK in various fields, such as personnel management, investment planning and performance evaluation of enterprises. Each knowledge base contains several (tens of) utility functions, so the total number of utility functions obtained so far was 534. They were used in the experiments described here.

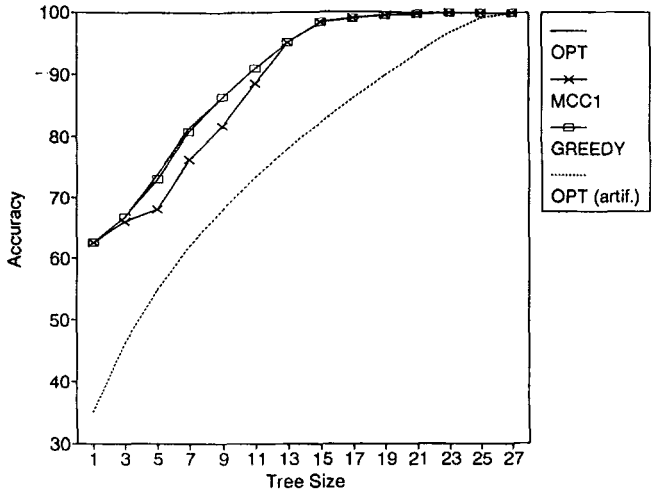

Figure 11. Average accuracy in the pruning of 39 utility functions, compared with OPT in the corresponding artificial domain ($A = 3$, $M = 4$ and $N = 3$).

Each utility function is defined by a set of examples. Each example is described in terms of attributes (i.e., lower-level attributes in the hierarchy) and belongs to a particular class (i.e., a value of the corresponding higher-level attribute). So, a utility function is actually a problem domain in itself. For comparison with typical machine learning applications, these problem domains have some specific characteristics: the number of attributes is relatively small, usually from 2 to 5, and never exceeds 8, attributes are discrete, problem domains are deterministic, and sets of examples are consistent.

Therefore, utility functions of DECMAK meet the requirements and assumptions from Section 2. Moreover, it is highly desirable to have utility functions represented at different levels of detail in various stages of practical decision making (Rajkovič & Bohanec, 1991). So, utility functions provide a useful application of decision trees and the kind of pruning investigated in this paper. In fact, this was the starting point and motivation for the research reported here.

A similar set of experiments as with artificial data was performed on utility functions. To facilitate comparison, utility functions that were defined in problem domains of exactly the same dimensions were grouped together. There were 53 different groups, containing from 1 to 80 utility functions. The results of pruning were averaged within each group.

To illustrate the results, consider the problem domain of the same dimensions as in Figure 10 ($A = 3$, $M = 4$, $N = 3$). There were 39 utility functions available for this domain. The average pruning sequences are presented in Figure 11, showing that the pruning sequences of MCC_1 were inferior to those of OPT and GREEDY. For comparison with artificial data, the sequence obtained by OPT in the corresponding artificially generated domain is presented with a dashed line.

An important difference occurred between artificial and real-world domains. In real-world domains, pruned trees were substantially smaller than the trees of the same accuracy in artificial domains; or, conversely, the accuracy of equally sized trees was substantially higher in the real-world domains. Consider, for example, Figure 11 and set the required accuracy to $\hat{a} = 90\%$. In the case of artificial data, this accuracy is achieved with trees of approximately 20 leaves; only 7 leaves were cut off in average. With real-world data, the same accuracy was obtained by trees of only 10 leaves in average.

Although this difference varies with different values of $\hat{a}$, different problem domains and different examples, and might not be as substantial as in the above case, it clearly indicates that real-world sets of examples are far from being random. There are many cases where a substantial reduction of size can be achieved at a small loss of accuracy. It seems that decision trees and pruning provide an appropriate and useful framework for knowledge representation in such cases.

We complete the comparison of pruning algorithms with an illustration of the actual execution times of our implementations in Pascal. As discussed earlier, OPT's complexity grows with the square of the tree size, whereas MCC and GREEDY have generally lower complexity. For the tree sizes in our experiments, the actual execution times are not at all critical. Average pruning times on a 40 MHz PC 386 for trees in turn of size 20, 40, 60 and 80 leaves, were for OPT 0.02, 0.04, 0.08 and 0.13 CPU seconds, respectively. MCC_1's times for the same trees grew almost linearly between 0.005 and 0.02 CPU seconds.

6. Note on pruning with OPT in noisy domains

Although OPT was not directly motivated by learning in noisy domains, it can be applied to noise-reduction pruning. For such pruning, the resubstitution error is unsuitable because we are interested in the tree's accuracy on *new* data, not on *learning* data. The accuracy on new data can either be computed by using an independent test set of examples or by estimating the probability of incorrect classification, using, for example, the m-estimate (Cestnik, 1990; Cestnik & Bratko, 1991). With exception of some insignificant implementation details, any one of these error measures can be used in OPT. In fact, the dynamic programming principle used in OPT will produce optimal pruning sequences for any error measure that guarantees the independence of the optimal sequences of the subtrees. By independence we mean that the optimal sequence for a tree is constructed by combining the optimal sequences of the tree's subtrees.

For illustration, let us check this condition for the m-estimated error probabilities. Let us consider the optimal pruning of a tree T to size s. Let T have the left subtree L and right subtree R. Let the subtrees' optimal pruning sequences be L_1, L_2, ... and R_1, R_2, ..., respectively. Here L_i denotes an optimal pruned left subtree of size i. Optimal pruned tree T of size s minimizes the expression

$$\min_{i+j=s} e(L_i, R_j)$$

where $e(L_i, R_j)$ is the error of the tree with subtrees L_i and R_j.

Let $p(L)$ and $p(R)$ be the probabilities of a data vector belonging to L and R, respectively. Then

$$e(L_i, R_j) = p(L)e(L_i) + p(R)e(R_j)$$

The error probabilities $e(L_i)$ and $e(R_j)$ are m-estimated on the basis of data subsets belonging to L and R, and the apriori probabilities of classes. Thus the probabilities $e(L_i)$ and $p(L)$ for the left subtree are estimated independently of the right subtree. Therefore the *independently* constructed pruning sequences for L and R can be combined by OPT in constructing the optimal pruning sequence for T.

An interesting question is: How well would OPT perform in noisy domains in comparison with other pruning algorithms motivated specifically by noise? Are there any comparative gains in terms of classification accuracy or size when decision trees are pruned by OPT? When trying to answer this question, one can compare OPT with MCC, which performs well in noisy domains (Mingers, 1989). Both algorithms generate trees of maximal accuracy with respect to size; the only difference is that OPT generates more trees than MCC. Consequently, OPT would perform at least as well as MCC in terms of classification accuracy, which is measured on an independent set of examples. Also, OPT would occasionally outperform MCC. On the other hand, the variation of classification accuracy is usually very small near its maximum (Breiman, et al., 1984; Mingers, 1989). So, if MCC missed a tree near the maximum, but OPT discovered it, no significant gains in classification accuracy can be expected in general. Consequently, we expect that OPT would perform very similarly to MCC in noisy domains.

7. Conclusion

Decision trees and pruning were considered in the context of knowledge representation at different levels of accuracy. There is a tradeoff between the size and representation accuracy of a decision tree: the smaller the tree, the less accurate the representation.

In particular, the problem of finding the smallest pruned tree that still represents knowledge with some chosen accuracy was addressed here. Based on initial work by Breiman, et al. (1984), an algorithm for finding minimal size trees was developed. It generates a dense sequence of pruned trees with respect to their size, obtained by pruning a fully-developed initial decision tree. An optimal tree always appears in the sequence and can be easily located. The algorithm is based on recursive dynamic programming and is efficient; its time complexity is quadratic with respect to the size of the initial tree.

The optimal pruning algorithm, OPT, was empirically compared with three progressive pruning algorithms: a simple progressive algorithm, GREEDY, and two derivatives of Breiman, et al.'s (1984) minimal cost-complexity pruning, MCC_0 and MCC_1. The experiments were performed on randomly generated domains and real-world sets of examples taken from multi-attribute decision making. The conclusions are as follows:

1. MCC_0, the original version of minimal cost-complexity pruning, is unsuitable for this specific problem; it generates too few trees. A modified version, MCC_1, is better, but still attains lower accuracy than OPT.

2. In uniform domains, where the attributes have the same number of possible values, the performance of GREEDY and MCC_1 was almost optimal. Since they are faster than OPT, one can prefer them in such domains. In nonuniform domains, the difference between the algorithms increased, particularly between MCC_1 and OPT. The average difference between OPT and GREEDY was still relatively small, but since severe deficiencies of GREEDY were recorded in some experiments, OPT should be preferred in such cases.

3. When comparing artificially generated and real-world sets of examples, there was a large difference in the achieved level of accuracy at equal size. With real-world data, a relatively high accuracy was achieved with relatively small pruned trees.

Although the relation between the accuracy and size depends on the characteristics of the corresponding set of examples and properties of the problem domain, we believe that pruning of decision trees is a useful technique for obtaining knowledge representations that trade between size and accuracy. One can expect that smaller representations, although less accurate, are more comprehensible.

However, future work will have to assess the validity of this expectation and, in particular, answer the questions: How comprehensible are pruned trees? Are optimal pruned trees more comprehensible than trees obtained by some simpler method of pruning? What is the threshold of accuracy ($\hat{a}$) that still allows people to understand the underlying concepts?

Acknowledgments

The work reported here was supported by the Ministry of Science and Technology of the Republic of Slovenia. The authors wish to thank Steve Minton and anonymous referees for valuable comments and suggestions. Thanks also to Leo Breiman for a clarification regarding his development of a pruning algorithm similar to OPT.

References

Bain, M., & Muggleton, S.H. (1991). Non-monotonic learning. In Hayes, J.E., Michie, D, and Tyugu, E. (Eds.), *Machine Intelligence* 12. Oxford: Clarendon Press.

Bohanec, M., Bratko, I., & Rajkovič, V. (1983). An expert system for decision making. In Sol, H.G. (Ed.), *Processes and Tools for Decision Support.* Amsterdam: North-Holland.

Bohanec, M., & Rajkovič, V. (1987). An expert system approach to multi-attribute decision making. In Hamza, M.H. (Ed.) *Proc. IASTED Conference on Expert Systems.* Anaheim: Acta Press.

Bohanec, M., Rajkovič, V., & Lavrač, N. (1988). Knowledge explanation in expert systems: A decision support system and machine learning view. In Hamza, M.H. (Ed.) *Proc. IASTED Conference on Expert Systems.* Anheim: Acta Press.

Bohanec, M., & Rajkovič, V. (1988). Knowledge acquisition and explanation for multi-attribute decision making. Proc. International Workshop "Expert Systems and Their Applications Avignon 88", Vol. 1, 59–78.

Bohanec, M. (1990). Methods for evaluation of alternatives and knowledge explanation in multi-attribute decision making. Ph.D. Thesis, University of Ljubljana (in Slovenian).

Bratko, I. (1989). Machine learning. In Gilhooly, K.J. (Ed.), *Human and Machine Problem Solving.* Plenum Press.

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984). *Classification and Regression Trees*. Belmont: Wadsworth.

Catlett, J. (1992). Ripple-down-rules as a mediating representation in interactive induction. Proc. Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'92. Japanese Society for Artificial Intelligence, Kobe and Hatoyama, 155–170.

Cestnik, B., Kononenko, I., & Bratko, I. (1987). ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In Bratko, I., and Lavrač, N. (Eds.), *Progress in Machine Learning*. Wilmslow: Sigma Press.

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. Proc. European Conference on Artificial Intelligence ECAI-90. Stockholm.

Cestnik, B., & Bratko, I. (1991). On estimating probabilities in tree pruning. In Kodratoff, Y. (Ed.), *Proceedings of the European Working Session on Learning EWSL-91, Porto, Portugal. March 6–8, 1991, Lecture Notes in Artificial Intelligence*, Vol. 482. Berlin: Springer-Verlag.

Compton, P., & Jansen, R. (1988). Knowledge in context: A strategy for expert system maintenance. In *Proceedings AI'88: 2nd Australian Joint Artificial Intelligence Conference, Adelaide Australia*. Berlin: Springer-Verlag.

Džeroski, S., & Bratko, I. (1992). Handling noise in inductive logic programming. Proc. FGCS-92 International Workshop on Inductive Logic Programming, ICOT TM-1182. Tokyo.

French, S. (1986). *Decision Theory: An Introduction to the Mathematics of Rationality*. New York: Wiley.

Lavrač, N., & Džeroski, S. (1991). Learning nonrecursive definitions of relations with LINUS. In Kodratoff, Y. (Ed.), *Proceedings of the European Working Session on Learning EWSL-91, Porto, Portugal. March 6–8, 1991, Lecture Notes in Artificial Intelligence*, Vol. 482. Berlin: Springer-Verlag.

Michalski, R.S. (1990). Learning flexible concepts: Fundamental ideas and a method based on two-tiered representation. In Kodratoff, Y., and Michalski, R.S. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. 3. San Mateo, CA: Morgan Kaufmann.

Michie, D. (1989). Problems of computer-aided concept formation. In Quinlan, J.R. (Ed.), *Applications of Expert Systems*, Vol. 2. Turing Institute Press in association with Addison-Wesley.

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

Muggleton, S., Bain, M., Hayes-Michie, J., & Michie, D. (1989). An experimental comparison of human and machine learning formalisms. In Spatz, B. (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning, Cornell University, Ithaca, New York: June 26–27, 1989*. San Mateo, CA: Morgan Kaufmann, 1989.

Niblett, T., & Bratko, I. (1986). Learning decision rules in noisy domains. Proc. Expert Systems 86, Brighton. Cambridge: Cambridge University Press.

Niblett, T. (1987). Constructing decision trees in noisy domains. In Bratko, I., and Lavrač, N. (Eds.), *Progress in Machine Learning*. Wilmslow: Sigma Press.

Pazzani, M.J., & Brunk, C.A. (1991). Detecting and correcting errors in rule-based expert systems: An integration of empirical and explanation-based learning. *Knowledge Acquisition* 3(2), 157–173.

Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh: Edinburgh University Press.

Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R.S., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning, An Artificial Intelligence Approach*. Los Altos: Kaufmann.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning* 1, 81–106.

Quinlan, J.R. (1987). Generating production rules from decision trees. Proc. International Conference on Artificial Intelligence. Los Altos: Kaufmann, 304–307.

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

Rajkovič, V., Bohanec, M., & Batagelj, V. (1988). Knowledge Engineering Techniques for Utility Identification. *Acta Psychologica* 68, 271–286.

Rajkovič, V., & Bohanec, M. (1991). Decision support by knowledge explanation. In Sol, H.G., Vecsenyi, J. (Eds.), *Environments for Supporting Decision Processes*. Amsterdam: North-Holland, 47–57.

Sedgewick, R. (1983). *Algorithms*. Reading: Addison-Wesley.

Shapiro, A. (1987). *Structured Induction of Expert Systems*. Reading: Addison-Wesley.

Received January 9, 1992
Final Manuscript June 8, 1993