INFORMS Journal on Computing
Vol. 18, No. 4, Fall 2006, pp. 494–505
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1804 | 0494

DOI 10.1287/ijoc.1050.0133
© 2006 INFORMS

FBP: A Frontier-Based Tree-Pruning Algorithm

Xiaoming Huo, Seoung Bum Kim, Kwok-Leung Tsui, Shuchun Wang
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0205, USA
{xiaoming@isye.gatech.edu, sbkim@isye.gatech.edu, ktsui@isye.gatech.edu, swang1@isye.gatech.edu}

A frontier-based tree-pruning algorithm (FBP) is proposed. The new method has an order of computational complexity comparable to cost-complexity pruning (CCP). Regarding tree pruning, it provides a full spectrum of information: namely, (1) given the value of the penalization parameter λ, it gives the decision tree specified by the complexity-penalization approach; (2) given the size of a decision tree, it provides the range of the penalization parameter λ, within which the complexity-penalization approach renders this tree size; (3) it finds the tree sizes that are *inadmissible*—no matter what the value of the penalty parameter is, the resulting tree based on a complexity-penalization framework will never have these sizes. Simulations on real data sets reveal a "surprise:" in the complexity-penalization approach, most of the tree sizes are inadmissible. FBP facilitates a more faithful implementation of cross validation (CV), which is favored by simulations. Using FBP, a stability analysis of CV is proposed.

Key words: decision trees; tree pruning; data mining; classification
History: Accepted by Amit Basu, Area Editor for Knowledge and Data Management; received May 2002; revised May 2003, September 2003, May 2004, August 2004; accepted October 2004.

1. Introduction

Due to their flexibility and interpretability, tree-based methods have gained enormous popularity in statistical modeling and data mining. Li et al. (2001) gave an excellent survey on tree building (including tree growing and tree pruning). An important technical question in building a statistically optimal tree is to find an optimal strategy to *prune* a large tree.

This paper focuses on the methodology of tree pruning. The concept of *complexity-penalization* is well adopted in this topic. The essence of this approach is to solve

$$\min_T L(T) + \lambda|T| \tag{1}$$

where

- T denotes an *admissible* subtree, which corresponds to a partition of the state space,
- $L(T)$ is the error rate of the corresponding decision tree,
- λ is a penalization parameter, which was mentioned earlier, and
- $|T|$ denotes the size of a tree, normally the number of *leaves* (i.e., terminal nodes) in tree T.

Such an approach was essentially the objective in cost-complexity pruning (CCP) (Breiman et al. 1984). In the tree-pruning framework, there are two interwoven quantities: the value of the penalization parameter λ and the size of the tree. As a necessary clarification, in evaluating the performance of a tree model on testing data, *generalization error* is considered. Generalization error is intentionally left out in the above list, but will be considered in evaluating the obtained trees. In tree pruning, the following four questions are of interest:

1. Given a value of the penalization parameter, applying the criterion of "minimizing the complexity-penalized loss function" (which will be elaborated later), what will be the size of the pruned tree?
2. Given a target tree size, what is the range of the value of the penalization parameter λ, so that when the principle of "minimum complexity-penalized loss function" is applied and the parameter λ is in this range, the size of the pruned tree is equal to this target size?
3. Are there some sizes of trees for which no value of the penalization parameter will render pruned trees that are of these sizes? These tree sizes are called *inadmissible*.
4. Given the unseen data, does the pruning method help improve classification accuracy? In other words, can the generalized error be minimized? This will be addressed by integrating cross validation (CV).

The first three questions are related to the algorithmic parameter. The last one concerns the generalization error.

The key contribution of this paper is a different algorithm for tree pruning. Apart from the local greedy approach adopted in CCP (Breiman et al. 1984), the proposed method, *frontier-based pruning* (FBP), keeps *all* the useful information and propagates bottom-up to the top layer (i.e., the root node). A specific algorithm is designed so that the above can be done efficiently—having the same order of complexity as CCP.

The proposed algorithm can automatically and simultaneously answer the first three of the above four questions. Identification of inadmissible tree sizes has not been considered in any other tree-pruning methods, although it seems to be observed by other researchers—see the justification of a dynamic-programming approach in Li et al. (2001). To our knowledge, this is the first time it is explicitly addressed in a *quantitative* way. The proposed method has a very nice graphical interpretation that has a connection to pareto-optimality, and consequently is highly intuitive.

To prune a single tree, both CCP and FBP solve the same problem, which is to minimize the objective in (1). However, because FBP provides a full spectrum of information on pruning, it facilitates a more faithful realization of CV. In simulations, it is shown that such a difference can lead to improvement in testing errors.

One may ask about the stability of CV—whether the CV partitioning can significantly affect the output of CV. This can be studied by FBP as well. Due to the construction of FBP, nice illustrations can be generated, as shown in later parts of this paper.

The rest of the paper is organized as follows. The key motivation of the algorithmic design is previewed in Section 2. Section 3 reviews the complexity-penalized tree-pruning algorithm. Section 4 describes the proposed FBP algorithm. Section 5 describes how FBP can be integrated with CV, and how the CV in FBP is different from the CV in CCP. A study of the stability of CV will be given in this section too. Section 6 presents simulations. Section 7 describes the architecture of software developed by the authors to carry out the simulations. (The software is available in the Online Supplement to this paper on the journal's website.) Some concluding remarks are made in Section 8.

2. The Main Idea of the Algorithm

An illustration of the key idea of the proposed method is in Figure 1. The value of λ is considered the cost of an additional node in a tree. For a given λ, when the target size of the tree is m, the minimum value of the complexity-penalized loss function (CPLF) is $c_m + m\lambda$, where c_m is a constant (i.e., the intercept). The y-axis is the value of the CPLF. Given λ, the slope m of line $c_m + m\lambda$ is the size of the subtree. Define $f(\lambda) = \min_{m=1,2,\ldots}\{c_m + m\lambda\}$. For a given value λ_0, the slope of $f(\lambda_0)$ is the size of the optimal subtree. It is not hard to see that $f(\cdot)$ is piecewise linear. Given a fixed integer (denoted by m_0), there is an interval of λ within which the slope of $f(\cdot)$ is equal to m_0. To check if a specific tree size m_0 is optimal, one can check whether a segment of line $c_m + m_0\lambda$ composes a part of the curve $f(\cdot)$.

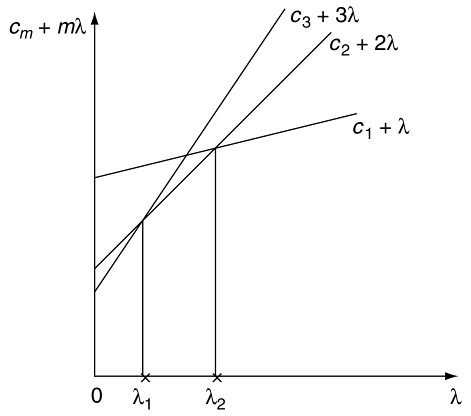

Figure 1 An Illustration of the Frontier-Based Tree-Pruning Algorithm

The lower bound of the lines in the above graph corresponds to sizes of the optimal subtrees; this is analogous to an efficient frontier in investment theory (Luenberger 1998) in which an efficient frontier represents a set of efficient portfolios that maximize expected returns at each level of portfolio risks; see also Markowitz (1952). This interpretation explains why the proposed method is called frontier-based tree pruning.

A key question is that, given two sets of lines, how do we find their common lower bound? More challengingly, how do we achieve this with the lowest possible computational cost? The proposed algorithm in Section 4 answers this question.

3. Tree Pruning

Tree methods in data analysis and modeling were "boosted" by Breiman et al. (1984). Since then, a tremendous literature has been developed. It is known (Li et al. 2001) that in building a tree, the most important step is to prune a redundant tree. Tree methods have been extremely successful in data mining. For example, the most popular data-mining tools—CART and MARS (Salford Systems 2004)—are based on tree methods. Various issues in building an "optimal" tree have been studied (Buja and Lee 2001). In this paper, we assume that a big and redundant tree has been built. The complexity-penalized tree-pruning approach was described in Breiman et al. (1984). The idea is originally rooted in statistical model selection. A significant advantage of adopting this framework is that various optimality results can be derived, e.g., Donoho (1997, 1999), and many more. The general principle of complexity-penalized tree pruning is reviewed in Section 3.1; a bottom-up tree-pruning algorithm is described in Section 3.2.

3.1. The Principle of Minimizing a Complexity-Penalized Loss Function

More details are given here on a previously mentioned idea. The objective of a complexity-penalized tree-pruning approach (from a big redundant tree) is to find a subtree that minimizes the CPLF. The principle of minimizing the CPLF will be described in the following. Let T denote a big unpruned tree. In tree modeling, each tree is associated with a *recursive dyadic partitioning* (RDP) of a state space. Without loss of generality, we focus on binary trees. Binary trees are often used in tree-based algorithms. The size of a tree (or a subtree) is the number of *terminal* nodes, which is the number of regions in an RDP of the state space. If two regions in an RDP are merged, the size of the tree is reduced by one. In tree pruning, a region merging is equivalent to pruning a two-leaf branch. A subtree is a tree that can be obtained by repeating the above procedure. A subtree is denoted by T_b, where b is an index. Since each subtree corresponds to a partition of the state space, one can choose the tree that minimizes a criterion function. In this case, this criterion function is CPLF: let $L(T_b)$ denote the loss function associated with tree T_b; let $|T_b|$ denote the size of the tree T_b; the CPLF is $L(T_b)+\lambda|T_b|$, where λ is a penalization parameter. The principle of minimum CPLF is to choose a subtree T_b that minimizes the CPLF. In other words, it is to solve

$$T_{b_0}=\arg\min_{T_b}\{L(T_b)+\lambda|T_b|\}. \tag{2}$$

3.2. Bottom-Up Tree-Pruning Algorithm

A well-known algorithm—*the bottom-up tree-pruning algorithm*—can efficiently solve the above problem. This algorithm uses the same idea as in other algorithms such as *best basis* (Coifman and Wickerhauser 1992), which originated in the wavelet literature. The same algorithm was described in Breiman et al. (1984). Figure 2 illustrates the idea.

In Figure 2, in a binary tree, consider two terminal nodes N_1 and N_2. Their common parent is N_0. Let $L(N_0)$, $L(N_1)$, and $L(N_2)$ denote the values of the loss function for the nodes N_0, N_1, and N_2, respectively. The CPLF associated with N_1 and N_2 are $L(N_1)+\lambda$ and $L(N_2)+\lambda$, respectively. Recall that in a binary tree, each node is associated with a region in an RDP of the state space. When one goes up to the node N_0, there are two possibilities: splitting the region into

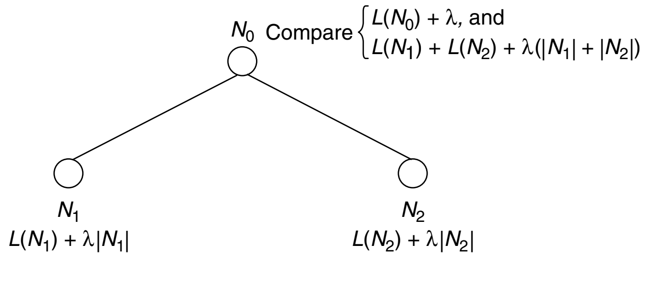
N_0 Compare $\begin{cases} L(N_0)+\lambda, \text{ and} \\ L(N_1)+L(N_2)+\lambda(|N_1|+|N_2|) \end{cases}$
N_1
$L(N_1)+\lambda|N_1|$
N_2
$L(N_2)+\lambda|N_2|$

Figure 2 An Illustration of Bottom-Up Tree Pruning

two regions N_1 and N_2, or no splitting. If the region N_0 is split, then the value of the CPLF should be $L(N_1)+L(N_2)+2\lambda$; otherwise, the CPLF should be $L(N_0)+\lambda$. Which one should be chosen depends on which value of the CPLFs is smaller. In a bottom-up tree-pruning algorithm, one starts at the bottom of a tree (or terminal nodes), and then repeatedly applies the above procedure until the top (root) node is reached. Note that when the nodes N_1 and N_2 are not terminal nodes, the expressions of their penalization functions are different. They should be $\lambda|N_1|$ and $\lambda|N_2|$. In the parent node (N_0), the comparison is between

$$L(N_1)+L(N_2)+\lambda(|N_1|+|N_2|), \quad \text{(splitting)}$$

and

$$L(N_0)+\lambda, \quad \text{(no splitting)}$$

which are also indicated in Figure 2. The bottom-up tree-pruning algorithm finds the subtree that minimizes the CPLF (Breiman et al. 1984).

An advantage of the bottom-up tree-pruning algorithm is that it is computationally efficient. If the unpruned tree has size $|T|$, it takes no more than $O(|T|)$ prunings to find the minimizer of (2).

The bottom-up tree-pruning approach has some intrinsic links with other approaches. For example, in Breiman et al. (1984) and Li et al. (2001), a method based on analyzing the reduction of the total loss function is discussed. This method is called CCP. Apparently, in the bottom-up tree-pruning algorithm, the choice of every step depends on whether the reduction of the loss function (following splitting the parent node, which is equal to $L(N_0)-L(N_1)-L(N_2)$) is larger than the value of the parameter λ (or the reduction of the penalization function $\lambda(|N_1|+|N_2|-1)$ when the nodes N_1 and N_2 are nonterminal). This leads to the discussion of a pruning algorithm in the original CART book (Breiman et al. 1984).

4. Frontier-Based Tree-Pruning Algorithm (FBP)

Section 4.1 describes the FBP algorithm. Section 4.2 interprets the occurrences of inadmissible tree sizes. Section 4.3 provides a fast algorithm in numerically realizing the FBP. Section 4.4 derives the computational complexity. Finally, Section 4.5 explains the connection between the frontier-based method and the dynamic-programming-based method.

4.1. Algorithm

Now we start describing the frontier-based pruning approach. The key point is to create a list of linear functions that have the form $c+m\lambda$ at each node, in which c is the value of a loss function and m is the

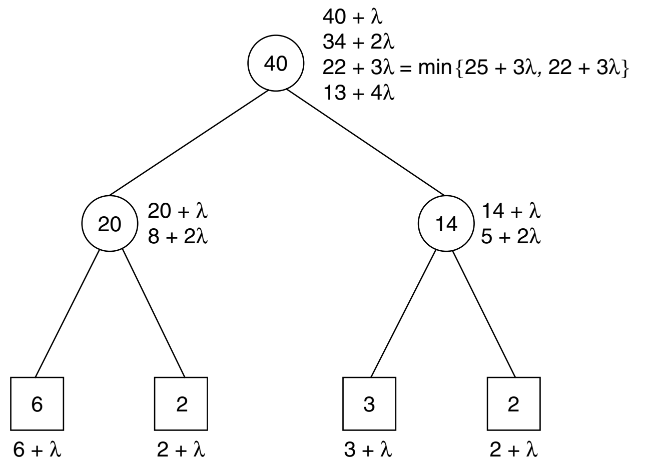

Figure 3 **An Example of the Frontier-Based Tree-Pruning Approach**

size of a subtree. At the root node, the information is summarized.

To explain the FBP algorithm, we use an example in Figure 3 in which a circle indicates an intermediate node and a square indicates a terminal node. The values in the circles and squares are the values of the loss function. Let λ denote the penalizing parameter. At each node, all possible expressions of the CPLF are listed as the linear functions of λ. For example, at all terminal nodes, the CPLFs have the form $c+\lambda$, where c is the value of the loss function. When one goes up in the tree, at an intermediate node, the expressions for the CPLFs have forms $c_m + m\lambda$, $m = 1, 2, \ldots,$ where c_m is the value of the loss function when the size of the tree is m. Each node will have a list of linear functions. At each node, one needs only to determine a sequence of c_m's. The list at the parent node can be derived from the list at the two siblings. For example, for the node where a value 20 is inside a circle, the value of c_1 should be 20, and the value of c_2 should be $6+2=8$. There should be two linear functions at this node. Sometimes the value of the intercept c_m is not uniquely defined. For example, at the root node of the above example, when the size of the tree is 3, the intercept should take the minimum value between $25 = 20+5$ and $22 = 8+14$. We start from the bottom of the tree, and the lists of linear functions are built (following a bottom-up approach) for all nodes.

The list building, described above, is the first step of the FBP algorithm. Consequently, the list at the root node is processed in the following way: in a Cartesian plane, the x-axis is taken to be the value of λ, and the y-axis is the value of the linear functions $c_m + m\lambda$. All the linear functions are plotted on this plane. An illustration of these linear functions is in Figure 1. The lower bound of these functions is considered. This function was defined earlier:

$$f(\lambda) = \min_{m=1,2,\ldots,T} \{c_m + m\lambda\},$$

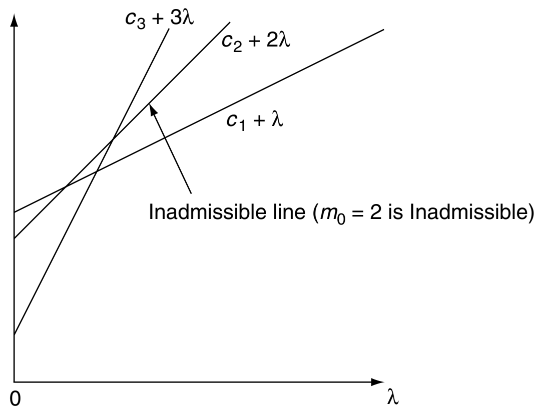

Figure 4 **An Inadmissible Case**

where T is the number of terminal nodes, and $c_m + m\lambda$ are linear functions at the root node. The following observations indicate how to use f:

1. For fixed λ, the value $f(\lambda)$ gives the minimum CPLF.
2. For a fixed size of the tree m_0, let (a_0, b_0) denote the interval of the parameter λ such that when λ takes a value inside this interval, the slope of the function $f(\lambda)$ is m_0. On the other hand, to get a subtree that has m_0 terminal nodes, the value of λ should be chosen in the interval (a_0, b_0).
3. It is possible that for an integer m_0, there is no part of curve $f(\lambda)$ such that its slope is m_0. In this case, the tree size m_0 is inadmissible—no matter what the value of λ is, in the minimum CPLF approach, the final tree size will never be m_0. An illustration of this case is in Figure 4.

4.2. Inadmissibility

It is possible that for a certain tree size, regardless of λ, the minimum CPLF approach will not render a tree that is of this size. This tree size is called *inadmissible*, which is analogous to inadmissible estimators in statistics. Apparently the definition of *inadmissibility* relies on the minimum CPLF criterion. Even if a tree size is inadmissible, it does not necessarily imply that a tree of this size is not optimal with respect to other criteria. We would like to emphasize the dependence of this definition on the CPLF, and warn about the possible confusion between *admissibility* and *optimality*.

An illustration of an inadmissible case is in Figure 4. In the FBP algorithm, inadmissible cases can be quickly identified. A fast numerical algorithm, which identifies admissible tree sizes as well as associated intervals, is presented in Section 4.3.

For the example in Figure 3, the tree sizes 2 and 3 are inadmissible, because

$$34+2\lambda \geq \min(40+\lambda, 13+4\lambda), \quad \text{and}$$
$$22+3\lambda \geq \min(40+\lambda, 13+4\lambda).$$

So the linear functions $34+2\lambda$ and $22+3\lambda$ are dominated by linear functions $40+\lambda$ and $13+4\lambda$.

4.3. Algorithm to Find a Lower Bound in a Bundle of $a\lambda+b$ Lines

An algorithm that finds the lower bound of a bundle of lines having the form $k\lambda+c_k$ is described here. Here, slope k is a positive integer, λ is a variable, and the c_k's are the intercepts. When the number of lines is N, this algorithm finds the lower bound with at most $O(N)$ operations.

Formation. Suppose the lines are $\lambda+c_1, 2\lambda+c_2, 3\lambda+c_3, \ldots, N\lambda+c_N$. The lower bound of them is the function $f(x)=\min_{1\le k\le N}\{k\lambda+c_k\}$. Apparently, $f(x)$ is a piecewise linear function, which is determined by the positions where the slope changes and the constant slope within each interval. Recall that in tree pruning, we have $c_1>c_2>c_3>\cdots>c_N$.

Algorithm. We start with the following table.

Slope:	1
Positions:	0

This indicates that there is one interval: $(0,+\infty)$. Within this interval, the slope is 1.

Now we take into account the line $2\lambda+c_2$. A new table is established,

Slope:	2	1
Positions:	0	$x_{1,2}$

where $x_{1,2}$ is the intersecting position of lines $\lambda+c_1$ and $2\lambda+c_2$. This is done by inserting a slope 2 at the beginning of the *slope* row and an intersecting position $x_{1,2}$ at the end of the *positions* row.

Now consider adding the line $3\lambda+c_3$. Recall that if

$$3x_{1,2}+c_3<2x_{1,2}+c_2, \tag{3}$$

the line $3\lambda+c_3$ is lower than the line $2\lambda+c_2$ at the position $x_{1,2}$. Hence the line $2\lambda+c_2$ is always above the minimum of the line $\lambda+c_1$ and the line $3\lambda+c_3$. In other words, line $2\lambda+c_2$ is dominated; we do not need to consider the line $2\lambda+c_2$. Based on the above, a new table should be

Slope:	3	1
Positions:	0	$x_{1,3}$

where $x_{1,3}$ is the position where $\lambda+c_1$ and $3\lambda+c_3$ intersect. If (3) is false, the line $2\lambda+c_2$ is not dominated. The new table should be

Slope:	3	2	1
Positions:	0	$x_{2,3}$	$x_{1,2}$

which indicates that in intervals $(0,x_{2,3})$, $(x_{2,3},x_{1,2})$, and $(x_{1,2},+\infty)$, the slopes of the lower bound are 3, 2, and 1, respectively.

In general, suppose that after step $n-1$ the table is

Slope:	i_1	i_2	$\cdots$	i_k	1
Positions:	0	x_{i_1,i_2}	$\cdots$	x_{i_{k-1},i_k}	$x_{i_k,1}$

where the k is the number of remaining lines. Consider adding the line $n\lambda+c_n$. It is not hard to verify that the set of points

$$\begin{gathered}(x_{i_1,i_2}, i_1x_{i_1,i_2}+c_{i_1}),\\ (x_{i_2,i_3}, i_2x_{i_2,i_3}+c_{i_2}),\\ \vdots\\ (x_{i_{k-1},i_k}, i_{k-1}x_{i_{k-1},i_k}+c_{i_{k-1}}),\\ (x_{i_k,1}, i_kx_{i_k,1}+c_{i_k}),\end{gathered}$$

is concave. Hence there exists an integer l, such that when $j\le l$, we have

$$n\lambda+c_n\le i_jx_{i_j,i_{j+1}}+c_{i_j}; \tag{4}$$

and when $j>l$, we have

$$n\lambda+c_n> i_jx_{i_j,i_{j+1}}+c_{i_j}. \tag{5}$$

Hence the lines $i_j\lambda+c_{i_j}$, $j=1,2,\ldots,l$ are dominated. Hence the new table should be

Slope:	n	i_{l+1}	i_{l+2}	$\cdots$	i_k	1
Positions:	0	$x_{n,i_{l+1}}$	$x_{i_{l+1},i_{l+2}}$	$\cdots$	x_{i_{k-1},i_k}	$x_{i_k,1}$

The above procedure is repeated until all lines are added. The final table determines the configuration of the lower bound.

Cost of the Algorithm. Obviously, it takes constant numbers of operations both to compute an interesting position and to carry out comparisons like (3), (4), and (5). Each line will be added once, and will be eliminated at most once. Thus, the overall cost of this algorithm is at most $O(N)$.

4.4. Computational Complexity

In a binary tree, suppose that the number of nodes and terminal nodes are equal to N and T, respectively. Then the number of operations required for FBP is no more than $N+T(T-1)$. Moreover, in a binary tree, we should have $N=2T-1$. Therefore, the order of complexity is $N+\frac{1}{4}(N^2-1)$ for the minimum CPLF algorithm with a fixed λ. But this is the price to pay for more information.

Theorem 4.1. *Recall that N denotes the number of nodes in a binary tree. It takes no more than $N+\frac{1}{4}(N^2-1)$ operations to generate the lists of linear functions at all nodes.*

Proof. There are two stages in the proof. First, at all nodes, for terms like $c+\lambda$, it takes N operations to create them. Second, for terms like $c+m\lambda$, where $m\geq 2$, it can be shown that it takes no more than $T(T-1)/2$ operations to calculate all the related linear functions, and it takes no more than $T(T-1)$ operations to generate the lists. The second statement can be proved by induction. Here we explain the idea. Assume that the above statement is true for any tree with number of terminal nodes smaller than T. For a binary tree with T terminal nodes, let m_1 (resp. m_2) denote the number of terminal nodes that have the left (resp. right) child of the root node as an ancestor. Note here we follow the common terminology of a classification and regression tree. We have $T=m_1+m_2$. Based on the assumption, for each binary tree whose root node is one of the immediate children of the root node in the original tree, it takes $m_1(m_1-1)/2$ and $m_2(m_2-1)/2$ operations to compute all the linear functions. For the root node, it takes m_1m_2 operations to compute all the necessary linear functions. Altogether, the number of operations to compute linear functions is

$$\begin{aligned}&m_1(m_1-1)/2+m_2(m_2-1)/2+m_1m_2\\&\quad=(m_1+m_2)(m_1+m_2-1)/2=T(T-1)/2.\end{aligned}$$

Note that to find the minimum values, in some cases, it takes no more than the steps needed to scan through all the sums. Hence it requires no more than $T(T-1)/2$ operations. From all the above, the number of operations to create the lists of linear functions *at all nodes* is no more than $T(T-1)$, which is equivalent to $\frac{1}{4}(N^2-1)$.

The summary of the above two facts leads to the proof of the theorem. □

4.5. Connection with the Dynamic-Programming-Based Approach

In Li et al. (2001), a dynamic-programming-based pruning (DPP) is proposed. The advantage is that the DPP can find optimal trees with any sizes. In our FBP algorithm, by tracing back the origins of the generated linear functions $c_m+m\lambda$, one can extract the optimal size-m subtree. It can be shown that if the optimal subtree is unique and admissible, both approaches will find it.

Based on the previous description of the FBP algorithm, tracing back is straightforward. Figure 3 shows an example. Suppose one wants to find the optimal subtree that has three terminal nodes. The corresponding linear function at the root node is $22+3\lambda$, which is induced by $(8+2\lambda)+(14+\lambda)$. Moreover, the linear function $8+2\lambda$ is made by $(6+\lambda)+(2+\lambda)$. Every time a linear function having the form $c+\lambda$ is reached, a terminal node is reached. Hence in this case, the optimal size-three subtree is made by the first two terminal nodes (from the left) and the intermediate node with misclassification rate being 14. The above procedure determines a subtree with size three. As mentioned in Li et al. (2001), it is possible that the optimal subtree is not unique. In that case, one can design some ad-hoc rules to specify it. The following theorem describes the consistency between the two methods.

Theorem 4.2. *Among the subtrees that have admissible size m, the one with the smallest misclassification rate can be found by both DPP and FBP.*

Proof. We prove only the case when the optimal subtree is unique and the goodness of fit is measured by the number of misclassifications. In DPP, it is known that the algorithm finds the subtree with the smallest number of misclassifications. In FBP, the algorithm also finds the subtree with the fewest misclassifications. This can be proved by showing that if there is a subtree with fewer misclassifications, then at the root node, this subtree will generate a linear function with a smaller intercept. This is contradictory to the definition in the FBP algorithm. Hence DPP and FBP should generate the same subtree. □

5. Integration with Cross Validation

A nice feature of the FBP algorithm is that it can be integrated with CV. We begin with the principle of CV, then describe how the FBP can be used in implementing CV.

Suppose the observations are $\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\ldots,(x_M,y_M)\}$, where M is the number of observations, the x_i's are predictor variables, and the y_i's are responses. Note that the x_i's can be multivariate. Suppose the above set is (equally) partitioned into K nonintersecting subsets $S_1\cup S_2\cup\cdots\cup S_K$. At each step, one leaves out one subset (say S_i) and uses the remaining subsets to grow a tree and then prune a tree; the lower-bound function will be denoted by $f_{-i}(\cdot)$. For each value of the parameter λ, the size of the optimal subtree and the subtree itself can be extracted. The optimal subtree determines a model. This model is then applied to the left-out subset (i.e., S_i). This step is called *testing*. The error rate in testing can be computed and is denoted by $e_{-i}(\cdot)$. Note that functions $f_{-i}(\cdot)$ and $e_{-i}(\cdot)$ are of the same variable. Since $f_{-i}(\cdot)$ is piecewise linear, it is not hard to prove that $e_{-i}(\cdot)$ is piecewise constant (i.e., a step function). The principle of CV is to find the value of the parameter λ such that the average of the e_{-i}'s, $\sum_{i=1}^{K}e_{-i}/K$, is minimized.

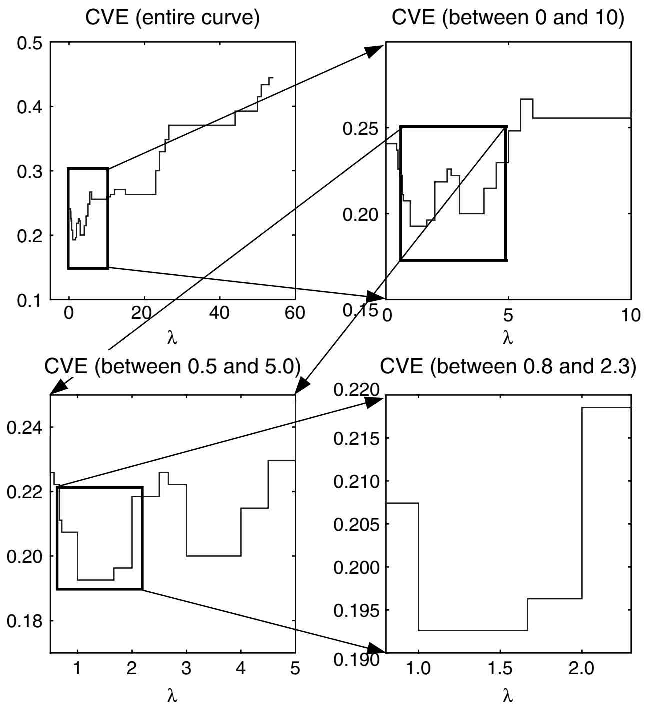

Figure 5 Using FBP in CV
Note. The minimum of the CVE was found by zooming in.

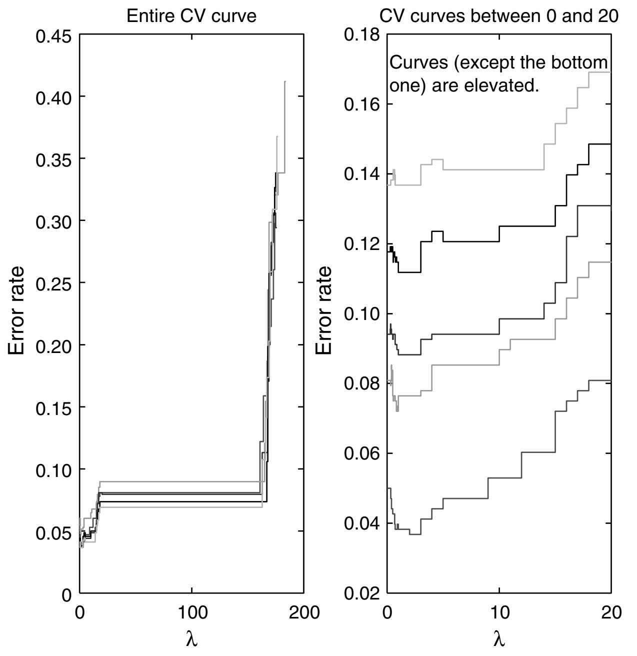

Figure 6 Five Experiments of CV with the Wisconsin Breast-Cancer Data
Notes. The left panel shows the entire CV curves. The right panel focuses on the region within (0, 20), the interval that includes the minima in all five experiments.

In the rest of this paper, the above quantity will be called *cross-validation error* (CVE).

This principle can be easily implemented with FBP. The generation of functions $f_{-i}(\cdot)$ is already in FBP. The generation of functions $e_{-i}(\cdot)$ can be easily implemented. A simulation example for the Cleveland Heart Disease data (Blake and Merz 1998) is presented in Figure 5. Based on this, one can conclude that the model made by using λ between 1 and 1.7 gives the minimum CVE.

5.1. Numerical Analysis of the Stability of the Cross-Validation Method

As mentioned earlier, FBP can be used to study (and graphically illustrate) the stability of CV. Here, a tenfold CV is studied. In each experiment, the data set is randomly and equally partitioned into ten subsets. The CV is to leave out one subset at a time, and uses the remaining nine subsets to train a model, then the omitted one is used to compute the testing error rate. This process is repeated for each subset, and the overall error rate is the average of the ten that are generated. Apparently in a ten-fold CV, the partition of the data set will change the result of CV. If CV is stable, the variation that is introduced by a different partition should be small. Is this always the case? By using FBP, one can develop graphical tools to examine this condition.

We study the stability of the CV approach in the Wisconsin Breast Cancer data (Blake and Merz 1998). In Figure 6, five CV curves are plotted, showing that the minimization intervals for the penalization parameter λ are roughly in the same neighborhood.

In Figure 7, 100 minimization intervals derived based on CV are plotted. Each row gives the location of the interval, inside of which when the parameter λ takes a value, the CVE is minimized—this explains why they are called minimization intervals. The figure shows the locations of these intervals out of 100 simulations. It demonstrates that, even though in many cases the minimization intervals overlap significantly, in some cases a minimization interval can be dramatically different from most of others. Note that in Figure 7, the horizontal axis is in logarithmic scale. More quantitative analysis on this phenomenon would be an interesting future research project.

5.2. Difference Between the CV in CCP and the CV in FBP

A careful comparison between the CV that is described in Section 11.5 of Breiman et al. (1984) and the CV in FBP reveals a difference between the two. We argue that the proposed method (i.e., CV in FBP) more faithfully realizes the principle of CV, while the CV in CCP examines only a subset of potential cases. An illustration of a case when the difference occurs will be provided at the end of this section.

To explain the difference, let us recall the CV procedure in CCP. More details are available in Breiman

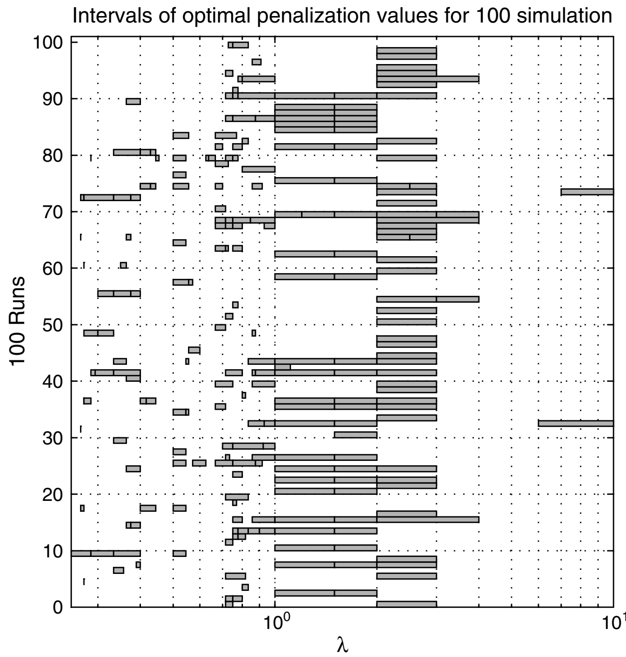

Figure 7 **The Locations of Minimization Intervals**

et al. (1984). In CCP, the real axis for the penalization parameter λ is partitioned into intervals with boundaries:

$$\alpha_0 = 0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots < \alpha_K < +\infty = \alpha_{K+1},$$

where the constant K is equal to the size of an unpruned tree, and the above boundaries are computed based on pruning a tree with the entire data—equivalent to running FBP on the entire data set. In CCP plus CV, a special set of quantities is chosen:

$$\{\alpha_j^*: \alpha_j^* = \sqrt{\alpha_j \alpha_{j+1}}, \ j = 1, 2, \ldots, K-1\}.$$

For a given λ, let CVE(λ) denote the cross validation error that was defined at the beginning of this section. The CV in CCP basically solves $\min_j \text{CVE}(\alpha_j^*)$, where $j \in \{1, 2, \ldots, K-1\}$ plus two cases corresponding to the two boundary intervals.

Apparently, the CV in CCP is computed based on a finite set of possible values of λ. This is not the CV principle described at the beginning of this section. FBP allows us to examine CVE(λ) for all possible values of λ. Based on this, we say that CV in FBP is more "faithful." Simulations will indicate that such a faithfulness can be linked to better performance in applications. Figure 8 illustrates the difference between the two CV procedures and when different optimal results will occur.

CV implementation of FBP is described here. For each leave-out set, one computes the error rate (e_{-i} in the previous description) as a piecewise-constant function *on the entire real line* of λ. In a ten-fold CV, ten functions like this will be generated. The (pointwise) average of them is the CVE(λ), which is another piecewise-constant function. The optimal value of λ is the minimizer of this function. Figure 8 gives a graphical illustration of the difference of the CV procedures between CCP and FBP. The vertical dotted lines indicate where the CV-CCP tries to find the optimal values; the solid piecewise constant curve reveals the function that was minimized. The dotted curve indicates the function that was minimized by the CV-FBP approach. One can see that CV-CCP considers only an incomprehensive subset.

To illustrate further the difference between CV-FBP and CV-CCP, they are applied to the Iris data (Blake and Merz 1998). Ten-fold CV is used. Each iteration provides a set of λ's and corresponding error rates. Note that the error rate is a step function. Ten step functions are then averaged. Table 1 provides the average error rates for each interval of λ.

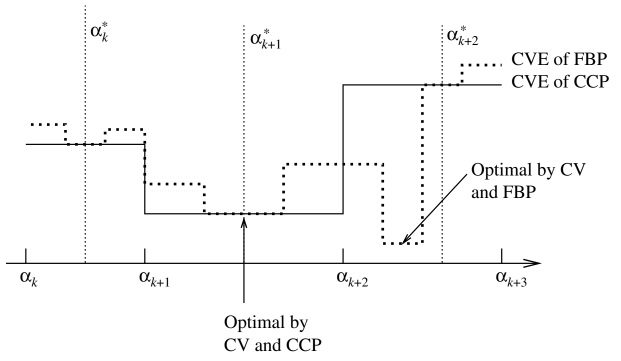

Figure 8 **Difference Between the Two CV Procedures (in CCP and FBP)**

Table 1 **Iris Example: λ's and Error Rates from 10-Fold CV**

λ	Error rate
0	0.060
0.1667	0.0667
0.3333	0.0600
0.3571	0.0667
0.4000	0.0533
0.4167	0.0467
0.4545	0.0533
1.0000	0.0733
2.0000	0.0867
6.3333	0.1200
36.0000	0.1733
37.0000	0.2133
38.0000	0.2933
39.0000	0.3200
40.0000	0.3933
41.0000	0.4133
43.0000	0.5067
44.0000	0.6267
45.0000	0.6600
46.0000	0.7667

Table 2 Iris Example: The Range of λ's from the Entire Training-Data Set and Geometric Means

Tree size	13	7	5	3	2	1
α (range of λ)	0	0.3333	0.5000	1.5000	44.0000	50.0000
α^* (geometric means)	0	0.4802	0.8660	8.1240	46.9042	50.0000

The difference between CV-CCP and CV-FBP can be explained:

• CV-FBP considers all ranges of λ's (in Table 1) and finds the one that has the minimum CV error. In our example, the minimum CV error is 0.0467 and the optimal λ is between 0.4167 and 0.4545.

• CV-CCP first uses the entire training data to find the partition of the λ-axis. Then the geometric means of pairs of adjacent λ's are computed. Table 2 shows the range of λ's and their geometric means. CV-CCP then chooses the optimal λ among the α^*'s. In our example, the minimum CV error in CCP is 0.0533 and the optimal λ is 0.4802. FBP produces smaller CV error than does CCP (0.0467 vs. 0.0533). In general, CV-FBP should always generate a smaller CV error than does CV-CCP.

In this example, both CV-FBP and CV-CCP lead to an identical tree because the optimal λ from both methods are in the same interval $(0.3333, 0.5000]$. This leads to the same testing error rates for both methods. However, in many of our simulations, CV-FBP and CV-CCP lead to different testing errors. This can be seen in Tables 5 and 6 in the next section.

We have not addressed the problem of whether smaller CV errors lead to smaller testing errors. We leave it for future research.

6. Simulations

This section contains the following:

• Section 6.1 gives the simulation results that are related to the *inadmissible* tree sizes.

• Section 6.2 compares the CVE errors, which serve as a sanity check.

• Section 6.3 compares CCP and FBP for testing errors.

• Section 6.4 compares tree sizes.

• Section 6.5 illustrates the overall comparison.

The CCP algorithm is available from Salford Systems (2004) and has also been implemented by the authors in Matlab (see the Online Supplement to this paper on the journal's website). The following experimental setups are used throughout:

1. The Gini index is used as the impurity measure for tree growing.

2. Twelve data sets available on the UCI database (Blake and Merz 1998) are used. A detailed description of these data sets can be found in the appendix of Li et al. (2001).

Table 3 Comparison of the Effective Tree Sizes with the Sizes of the Largest Possible Trees

Data set	Admissible	Largest tree size
Australian credit approval	21	481
Cleveland heart disease	11	100
Congressional voting records	8	44
Wisconsin breast cancer	11	45
Iris plants	6	13
BUPA liver disorder	13	132
PIMA Indian diabetes	20	260
Image segmentation	30	242
German credit	26	344
Vehicle silhouette	30	269
Waveform	20	367
Satellite image	46	745

3. Ten-fold CV is performed. Each data set is split into two parts—a training set and a testing set—for ten times. Each time, a ten-fold CV is applied to the training set; the CVE, testing error, and tree size of the trained model are recorded. The reported values in the following tables are the averages of these ten simulations.

6.1. Inadmissibility

In all cases, the number of admissible tree sizes is significantly smaller than the total number of possible tree sizes. Here the tree sizes are measured by the number of terminal nodes, i.e., the number of regions in an RDP. The simulation results are presented in Table 3. The *admissible* column gives the number of admissible tree sizes. The last column contains the number of terminal nodes in the unpruned trees. In general, the number of admissible tree sizes is roughly 10% of all the possible sizes of the subtrees. As mentioned in Section 1, we have not seen a quantitative illustration of this phenomenon.

6.2. Cross-Validation Errors

Based on the explanation in Section 5.2, between CCP and FBP, FBP *always* gives smaller CV errors. The simulations verify this, in Table 4. Statistical analysis was

Table 4 Comparison of the CV Error Rates Between CCP and FBP

Data set	CCP	FBP	Winner
Australian credit approval	14.13	14.01	FBP
Cleveland heart disease	21.15	20.89	FBP
Congressional voting records	4.16	4.12	FBP
Wisconsin breast cancer	4.56	4.47	FBP
Iris plants	5.20	5.07	FBP
BUPA liver disorder	31.27	31.03	FBP
PIMA Indian diabetes	24.37	23.97	FBP
Image segmentation	3.84	3.83	FBP
German credit	24.61	24.48	FBP
Vehicle silhouette	28.02	27.90	FBP
Waveform	22.86	22.83	FBP
Satellite image	12.67	12.65	FBP

Table 5 **Comparison of the Testing Error Between CCP and FBP**

Data set	CCP	FBP	Winner
Australian credit approval	14.84	14.58	FBP
Cleveland heart disease	26.82	27.19	CCP
Congressional voting records	5.50	5.60	CCP
Wisconsin breast cancer	5.09	4.94	FBP
Iris plants	7.73	7.33	FBP
BUPA liver disorder	35.20	34.74	FBP
PIMA Indian diabetes	25.34	25.26	FBP
Image segmentation	5.80	5.81	CCP
German credit	27.60	27.06	FBP
Vehicle silhouette	30.25	30.27	CCP
Waveform	27.47	27.47	FBP, CCP
Satellite image	12.85	12.85	FBP, CCP

implemented as well. Comparing CCP with FBP, the p-value of the paired t-test is roughly 0.001. A 95% confidence interval for the mean difference of the CV error between CCP and FBP is $(0.0770, 0.2196)$. As mentioned earlier, we treat this as a "sanity check."

6.3. Comparison of Testing Errors

In simulation studies, testing errors are important. Table 5 gives the *average* testing errors for ten simulations of CCP and FBP (see more explanation at the beginning of this section). FBP tends to give smaller testing errors: six cases of smaller than CCP, and two ties, out of 12 simulations. However, the difference is not dramatic. The paired t-test reveals that the p-value is 0.152. A 95% confidence interval for the mean difference of the testing errors (between CCP and FBP) is $(-0.0496, 0.2813)$. Note that zero is inside this interval.

6.4. Tree Sizes

In tree models, the size of a tree indicates the complexity of the model. In Table 6, the average tree sizes are given for 12 data sets; each has ten repetitions, as explained earlier. We observe that FBP gives smaller tree sizes than does CCP. Statistical analysis for the difference between CCP and FBP is performed. The p-value for the paired t-test is 0.006. A 95% confidence interval for the mean difference of tree size between CCP and FBP is $(0.617, 2.950)$. Based on this, it appears that a combination of CV and FBP generates smaller trees. This is just an empirical result; theoretical understanding will be needed.

Table 6 **Comparison of the Tree Sizes (Number of All Nodes) Between CCP and FBP**

Data set	CCP	FBP	Winner
Australian credit approval	10.60	8.00	FBP
Cleveland heart disease	8.20	6.80	FBP
Congressional voting records	7.00	5.80	FBP
Wisconsin breast cancer	14.20	10.60	FBP
Iris plants	7.20	6.60	FBP
BUPA liver disorder	16.60	13.80	FBP
PIMA Indian diabetes	15.20	9.00	FBP
Image segmentation	89.40	87.20	FBP
German credit	29.20	29.20	FBP, CCP
Vehicle silhouette	63.40	62.60	FBP
Waveform	69.00	69.00	FBP, CCP
Satellite image	249.00	249.0	FBP, CCP

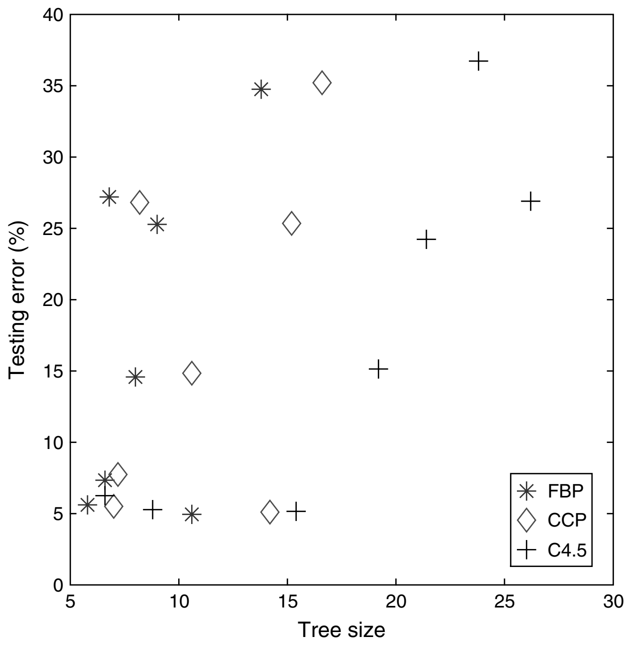

Figure 9 **Testing Errors vs. Tree Sizes**
Note. Lower left is optimal.

6.5. Overall Comparison

Figure 9 compares three methods (C4.5, CCP, and FBP), based on the testing errors and tree sizes. Here, we include C4.5, which is another major tree-building method (Hamilton 2004). Note that it is not appropriate to compare C4.5 directly with CCP or FBP. Because C4.5 uses a different tree-growing algorithm (i.e., ID3 from Mitchell 1997) and generates multi-way splits: it allows nodes to have more than two child nodes, while CCP and FBP are binary trees. However, it is still interesting to see the difference of their performance on some universal measures: smaller testing errors and smaller tree sizes, for ideal classifiers. Based on this, being close to the lower-left corner in Figure 9 is desirable. Comparing the tree methods, we can see that FBP is relatively better than the other two methods. Only the first seven data sets are plotted in the figure, for legibility (the same trends were observed for all data sets).

7. Reproducible Research and Software

The principle of *reproducible research* is followed by many researchers, e.g., Claerbout and Schwab (2004) and Donoho et al. (2004). Buckheit and Donoho (1995) stated: "An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures." Following this principle, our software is made available in the Online Supplement to this paper on the journal's website. The algorithms are implemented in MATLAB. Care is given to ensure the efficiency of each implemented algorithm. There are seven basic components:

1. *Tree growing.* Grow a big and potentially redundant tree from data.
2. *Tree pruning.* Use FBP to generate lists of linear functions at each node.
3. *Find lower bound.* Based on the list of linear functions at the root node, find the lower-bound function $f(\cdot)$.
4. *Identify the best subtree.* Given a value of parameter λ, identify the size of the best subtree, together with the subtree itself.
5. *Testing.* Apply a tree model generated above to test data and report the results.
6. *Application in CV.* Use FBP to realize CV.

Figure 10 provides an overview of the software.

8. Conclusion

A FBP algorithm is proposed, which provides a graphical way to implement the task of minimizing CPLF. FBP has the same objective as does CCP; however it is more advantageous because it provides a full spectrum of information in tree pruning. It can be used to realize the principle of CV more "faithfully." A combination of FBP and CV renders better classifiers in simulations, compared to other existing methods. Simulation results on real data sets render several other interesting findings; for example, the number of admissible tree sizes is always a small proportion (roughly 10%) of the number of all possible tree sizes. Computational complexity analysis is provided. This method can be used in other tree-related problems, e.g., studying the stability of CV in tree models.

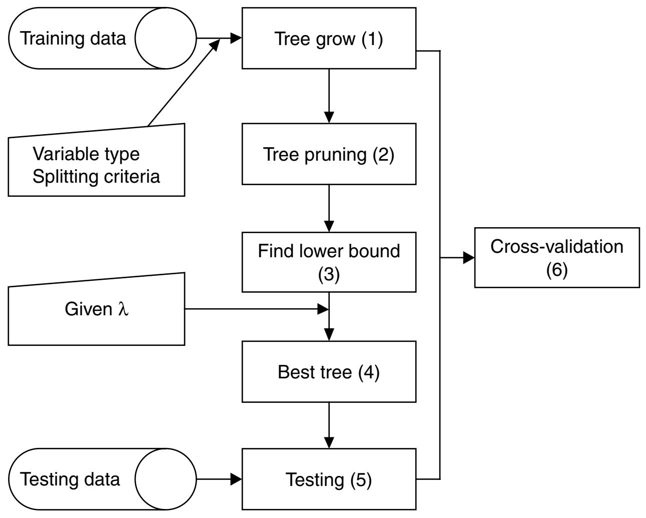

Figure 10 **Relation Between the Functions**
Note. Numbers in parentheses are steps.

Table A.1. **Summary of Data Sets**

Data set	Classes	Attributes	Sample size training (testing)
Wisconsin breast cancer	2	9 numeric	683
Cleveland heart disease	2	7 numeric, 6 categorical	270
PIMA Indian diabetes	2	7 numeric	768
BUPA liver disorder	2	6 numeric	345
Congressional voting records	2	16 categorical	435
Australian credit approval	2	6 numeric, 8 categorical	690
German credit	2	7 numeric, 13 categorical	1,000
Iris plants	3	4 numeric	150
Satellite image	6	36 numeric	4,435 (2,000)
Image segmentation	7	19 numeric	2,310
Vehicle silhouette	4	18 numeric	846
Waveform	3	21 numeric	600 (3,000)

Appendix. Data Sets

The 12 data sets are from the UCI repository (Blake and Merz 1998). Table A.1. provides a summary of these data sets.

Acknowledgments

The authors thank the editor, area editor, associate editor, and two anonymous referees, whose comments helped greatly in improving the presentation of this paper. They also thank Ms. Judith Norback for her useful comments. This work was partially supported by the National Science Foundation under Grants 0140698, 0140587, and 0346307. The first author thanks the Institute of Pure & Applied Mathematics for its hospitality during Fall 2004.

References

Blake, C. L., C. J. Merz. 1998. UCI Repository of machine learning databases. Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Breiman, L., J. Friedman, R. Olshen, C. Stone. 1984. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.

Buckheit, J., D. L. Donoho. 1995. WaveLab and reproducible research. http://www-stat.stanford.edu/~donoho/Reports/1995/wavelab.pdf.

Buja, A., Y.-S. Lee. 2001. Data mining criteria for tree-based regression and classification. *Proc. Knowledge Discovery and Data Mining-2001*. ACM Press, New York, 27–36.

Coifman, R. R., M. V. Wickerhauser. 1992. Entropy-based algorithms for best basis selection. *IEEE Trans. Inform. Theory* **38** 713–718.

Claerbout, J., M. Schwab. 2004. http://sep.stanford.edu/research/redoc/.

Donoho, D. L. 1997. CART and best-ortho-basis: A connection. *Ann. Statist.* **25** 1870–1911.

Donoho, D. L. 1999. Wedgelets: Nearly minimax estimation of edges. *Ann. Statist.* **27** 859–897.

Donoho, D., M. R. Duncun, X. Huo, O. Levi. 2004. WaveLab. http://www-stat.stanford.edu/~wavelab.

Hamilton, H. J. 2004. C4.5. http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html.

Li, X.-B., J. Sweigart, J. Teng, J. Donohue, L. Thombs. 2001. A dynamic programming based pruning method for decision trees. *INFORMS J. Comput.* **13** 332–344.

Luenberger, D. G. 1998. *Investment Theory*. Oxford University Press, New York.

Markowitz, H. 1952. Portfolio selection. *J. Finance* **7** 77–91.

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill, New York.

Salford Systems. 2004. CART and MARS. http://www.salford-systems.com/.