An Exact Dynamic Programming Algorithm for Regression and Classification Trees

Bart Jan van Os, Elise Dusseldorp and Jacqueline J. Meulman [*][†]

Abstract: Greedy algorithms for fitting regression and classification trees use recursive partitioning methods, and employ one of a variety of local criteria that optimizes each separate split, given the previous splits. In contrast, a method will be described that aims at optimizing the complete tree structure with respect to a *global* objective function. An Exact Dynamic Programming algorithm, called ExactTree, is proposed that is currently able to find moderately sized trees for medium sized datasets. An application to the well-known Boston Housing data (with 509 observational units) shows that, compared to a greedy tree fitting algorithm, optimizing the tree structure on the training set decreases the prediction error estimated from the test set. The Exact-Tree algorithm is combined with several ways to digitize continuous predictors, which not only reduces the search space of possible splits, but may also decrease the prediction error.

Keywords: CART, Classification Trees, Exact Algorithm, Exact-Tree, Dynamic Programming, Global Optimization, Regression Trees

1 Introduction

Although trees were introduced for the first time by Morgan and Sonquist (1963) in a method called Automatic Interaction Detection, the methodology

[*]Jacqueline Meulman is Professor of Applied Statistics, Department of Mathematics, Leiden University, The Netherlands (e-mail: jmeulman@math.leidenuniv.nl), Elise Dusseldorp is Statistician, TNO Quality of Life Research, Leiden, The Netherlands (e-mail: elise.dusseldorp@tno.nl) and Bart Jan van Os is Statistician, 4os Consultancy, The Netherlands (e-mail: advies@4os.nl).

[†]This paper is based on work first presented in the PhD thesis Dynamic Programming for partitioning in multivariate data analysis by Bart Jan van Os, Leiden 2001, supervised by Jacqueline J. Meulman and Lawrence J. Hubert

was only fully developed in Breiman, Friedman, Olshen, and Stone (1984). Their CART algorithm complemented regression trees, which deal with the prediction of a continuous outcome variable, with classification trees that deal with a categorical (nominal) outcome variable. Since then a number of different methods and software packages have become available (for an overview, see Hand (1997)), especially after trees obtained renewed interest by research in data mining methods. The methodology has become popular for several reasons: trees automatically select predictors and find complicated interaction effects in the presence of noise variables, they deal with missing data without having to resort to imputation methods, and they can handle a large number of predictors of different measurement level (numeric, ordinal, binary and categorical variables are all handled with equal ease). Trees are immune to the effects of extreme outliers among the predictor variables, and are invariant to the scaling of predictors or any monotone transformation of them. Finally, a fitted tree mimics the way a hierarchically nested set of decision rules can be represented, and therefore has a clear interpretation (although the interpretability of large trees is questioned; for example, see Ripley (1996)).

Despite their success, trees are also known to have a major drawback: their limited prediction accuracy in applications, due to their instability. Instability means that a small change in the data can result in a very different tree, and the latter makes the interpretability unreliable. Because of the hierarchical nature of the process, the effect of an error in the top split is propagated down to all of the splits below it, and this constitutes an inherent instability (Hastie, Tibshirani, & Friedman, 2001), p. 274). Therefore various approaches have tried to extend greedy algorithms, with successful advancements obtained by combining multiple tree structures in ensembles, obtaining a stronger predictor or classifier. Examples are Boosting (Freund & Schapire, 1996; Drucker & Cortes, 1996; Friedman, 2001), Bagging (Breiman, 1996), and Random Forests (Breiman, 2001), which are competitive with other predictive learning methods such as Support Vector Machines (Vapnik, 1996), Kernel methods and local regression (see Loader, 1999), and Neural Networks (see Ripley, 1996).

Unfortunately, the result of an ensemble method is deprived of one of the most important advantages of a single tree, i.e., its high interpretability. At the expense of higher prediction accuracy, we lose the representation of the prediction rules in a simple tree structure. Because prediction using ensemble methods works more or less as a black box, considerable efforts have been invested to obtain secondary tools for interpretation, such as the importance of a variable by looking at partial dependence (for example, Friedman 2001). However, with respect to interpretability, the overview by looking at many partial dependence plots cannot compete with looking at a single tree. Consequently, adapting greedy tree methods remains an active domain of research, also because of their ability to handle large datasets that arise in data mining. Research in this area includes the introduction of more complex splits that combine several predictors through a discriminant analysis in each node (Loh & Vanichesetakul, 1988), finding more advanced split rules to overcome variable selection bias (Loh, 2002), and combining the tree model with standard regression models (Alexander & Grimshaw, 1996; Dusseldorp & Meulman, 2001, 2004; Dusseldorp, Conversano, & Van Os, 2010).

The present paper aims at strengthening tree methods at their heart, i.e., at the way trees are constructed (fitted). A major way to remove the inherent instability, and still to obtain a single tree representation, is to use an algorithm that finds a *globally optimal* tree. However, from the early days on, simultaneous optimization over all possible splits in the complete tree structure has been considered computationally infeasible. Nowadays, the problem is believed to be most likely intractable, because parts of the problem and the related problem of constructing minimum size decision trees from decision tables are proven to be NP-complete (Hyafil & Rivest, 1976; Naumov, 1994). Therefore, the much more computationally feasible *heuristic* approximation strategy of growing a tree and pruning it to its best size is still the most popular way of obtaining a *simple* tree. Unfortunately, attempts to use a look-ahead strategy, where each split is optimized with respect to several splits ahead, did not proof to be much more effective over a greedy heuristic (Murthy & Salzberg, 1995; Murthy, 1997). Luckily, in contrast, a linear programming formulation for simultaneously optimizing all splits in a fixed tree structure combined with an Extreme Point Tabu Search heuristic was shown to outperform greedy trees (Bennett & Blue, 1996).

The current paper proposes an algorithm, called ExactTree, that uses dynamic programming (DP) to find a globally optimal tree. The problem is indeed computationally hard, but due to the particular implementation, the algorithm can find (guaranteed) optimal trees for sample sizes of about 500 observations and 15 predictors, with a maximum size of the tree, the depth, of seven. Note that these numbers applied at the time the current algorithm was implemented, which was some time ago. Although new chip technology speeds up computers amazingly, it remains true that increasing tree sizes require exponentially increasing computing resources.

The effort of finding a globally optimal tree immediately raises a crucial question: is optimizing the performance of prediction rules on the training data actually worth the effort, or does it induce over-fitting so that the expected prediction error (obtained for the test data) actually increases? We believe that the latter is not the case, because the algorithm aims at improving the accuracy without increasing the complexity, or possibly even decreases the complexity. The example in Section 8 will show that indeed prediction accuracy can be improved.

It will be clear that ExactTree is not targeted at replacing the complete search process of suitable predictors when the number of predictors is very large, since it does not have the main advantage of recursive partitioning methods, which is the ability to quickly search for the best splits. ExactTree rather aims at improving the prediction accuracy of the tree model by optimizing the overall tree structure with respect to the partitioning criterion at hand. As such, ExactTree is useful (a) for fitting moderately sized trees to moderately sized datasets, (b) as a tool for studying the possible profits of globally optimizing tree structures, (c) as a benchmark for heuristic methods, (d) as an optimized ingredient in ensembles (combined tree models), for example in boosted trees, where linear combinations of many small trees are used, and e) as a secondary step to find a globally optimal stable tree as soon as the most important variables have been determined by (an ensemble method based on) a greedy algorithm. In Section 2 we will give a quick review of (optimal) classification and regression trees (sometimes jointly called decision trees), in Sections 3-7 we will show how to obtain globally optimal trees, and in Section 8 we will demonstrate our ExactTree algorithm in an application, and compare its results, with respect to both the obtained trees and the prediction error, to a greedy algorithm.

2 Decision Trees

In regression and classification problems, one has a system consisting of a random "outcome", "response", or "dependent" variable Y and a set of random "explanatory" ,"predictor", or "independent" variables $X = \{X_1, \cdots, X_J\}$, where J denotes the number of predictors. The problem defines a "training" sample, $\{y_i, \mathbf{x}_i\}_1^N$ of known values for Y and $\mathbf{X}$, where $(y_i, \mathbf{x}_i)$ links the predictor variables of the ith object with the ith value of the outcome variable, and where $i = 1, \ldots, N$. The space spanned by the predictor variables, denoted by $\mathbf{X}$, is called the *measurement space*. An example of the measurement space for two discrete variables X_1 and X_2 having respectively eight and six different values (categories) is given in Figure 1(a). Because regression and classification trees only differ by the character of the outcome variable (continuous or discrete), we will use the term decision tree problem when indicating their common optimization task. The decision tree problem consists of finding a set of rules that partition the measurement space into regions (see Figure 1(b)), such that within each region R_k we are able

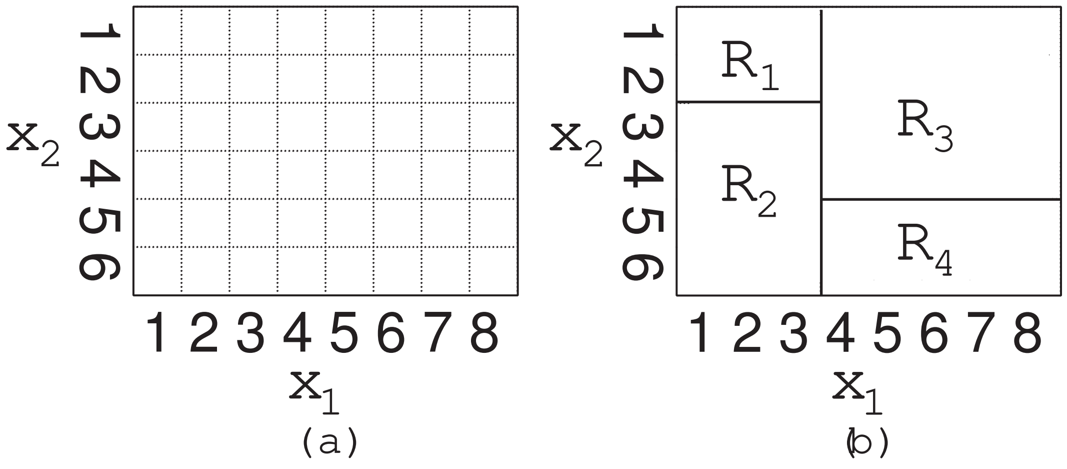


Figure 1: (a) Measurement space for two categorical predictors. (b) Measurement space representation of a tree predictor.

to predict a measurement $\hat{y}_i$ as close as possible to y_i for all objects that are included into that region (for an example, see Hastie et al. (2001), p. 268). The measurement space representation 1(b) has the advantage that it depicts the unique partition corresponding to the final result, whereas tree representations are sometimes not unique, because the same partition may be induced by different trees. Moreover, the measurement space representation provides a concise description of the decision tree problem as a partitioning task, and immediately suggests relations to other problems that were solved globally, such as hierarchical clustering (Hubert, Arabie, & Meulman, 2001; Van Os, 2001; Van Os & Meulman, submitted). Consequently, the measurement space representation will be used to present the global optimization approach and the proposed algorithm.

2.1 Formal Definition of the Optimal Tree Problem

In the sequel we will show how decision trees can be formulated as a combinatorial optimization problem. Without loss of generality we will restrict ourselves to binary trees.

Each tree results in a partition $\mathcal{P}$ of the measurement space $\mathbf{X}$ into a number of $|\mathcal{P}|$ regions $R_k \in \mathcal{P}$ that correspond to the induced terminal nodes. Let $\Omega_d(\mathbf{X})$ denote the set of all admissible partitions of the measurement space $\mathbf{X}$ with binary trees of at most K regions (terminal nodes) and depth d, where d denotes the maximum number of layers from the top to any terminal node in the tree. Note that the definition of $\Omega_d(\mathbf{X})$ may include additional admissibility restrictions that are widely applied in tree methods, such as the required minimum number of observations in a terminal node, or a minimum impurity criterion that stops further splitting if the value of the loss function (that measures the discrepancy between the training data and the representation) is close (or equal) to zero.

We now can formulate decision trees as special cases of the mathematical problem:

$$\begin{cases} f(\mathbf{y}, \mathbf{X}) = \underset{\mathcal{P}}{\text{opt}} \sum_{R_k \in \mathcal{P}} h(\mathbf{y}, \mathbf{X}, R_k) \\ \text{subject to} \\ \mathcal{P} \in \Omega_d(\mathbf{X}), \\ d \leq D, \\ |\mathcal{P}| \leq K. \end{cases} \tag{1}$$

The *impurity* function $h(\cdot)$ can be defined in many ways without violating the Dynamic Programming principles explained below. For regression trees that minimize the sum of squared error by estimating a constant a_k in each region R_k, $h(\cdot)$ is defined by

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i (y_i - a_k)^2 I(\mathbf{x}_i \in R_k), \tag{2}$$

but $h(\cdot)$ also includes other loss functions such as absolute deviations

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i |y_i - a_k| I(\mathbf{x}_i \in R_k), \tag{3}$$

or even so-called Treed Regression (Alexander & Grimshaw, 1996) that estimates a *simple* linear regression model in each region by defining impurity as

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i (y_i - (a_k + \mathbf{b}_k' \mathbf{x}))^2 I(\mathbf{x}_i \in R_k), \tag{4}$$

where $\mathbf{b}_k$ only has one non-zero element corresponding to the best linear predictor in that region, excluding predictors that are used for splits up to the particular terminal node. Note that in all these cases the required additional parameters can be estimated *independently* from other regions.

For classification trees the impurity function can be defined by a simple misclassification rate

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i (I(y_i = \tau(R_k)) I(\mathbf{x}_i \in R_k), \tag{5}$$

where $\tau(R_k)$ denotes the class assigned to region (terminal node) R_k. More commonly, the prior probabilities $\pi_u, u = 1, \ldots, c$ for each class u, and a loss function $L(u, v)$ specifying the loss for incorrectly classifying an object into class u instead of class v, are taken into account, by minimizing the overall risk

$$h(\mathbf{y}, \mathbf{X}, R_k) = P(R_k)\mathcal{R}(R_k). \tag{6}$$

Here, $P(R_k)$ is the probability for future observations to belong to R_k, and $\mathcal{R}(R_k)$ is defined by

$$\mathcal{R}(R_k) = \sum_u p(u|R_k)L(u, y_i). \tag{7}$$

(Note that the probabilities are usually estimated from the training sample.) More elaborate definitions of impurity functions that are encompassed by the current mathematical programming formulation would include the estimation of loglinear or poisson regression models in each terminal node (e.g., Chaudhuri, Lo, Loh, & Yang, 1995).

The mathematical problem formulation also explicitly includes two types of size constraints. In general, the larger the tree, the lower the overall impurity for the training set, and therefore an explicit size constraint is necessary to avoid finding a huge tree, being the largest possible tree given the other admissibility restrictions, for example the minimum number of observations in a terminal node. The first type of constraint ($d \leq D$) imposes the maximum depth D on the tree, and corresponds to the maximum order of interaction between predictors (Friedman, 2001). Let an *np*-split (a new predictor split) be a split that is defined on a predictor that has not been used in previous splits, if existing. The maximum order of interaction is then equal to the largest number of np-splits from the top to any terminal node. The second type of constraint ($|\mathcal{P}| \leq K$) imposes a maximum K on the number of terminal nodes ($|\mathcal{P}|$ in the tree. These two restrictions are obviously related by $K \leq 2^D$, and only one of the two may be in effect for a given problem. Both constraints define the admissible *maximum* size rather than the required maximum size, because additional admissibility conditions, such as the minimum number of observations in a terminal node, may actually prevent larger trees in some cases. Because in general the best prediction accuracy obtained by the least complexity of the tree is preferred, the smallest tree that optimizes (1) under the size constraints is accepted.

Note that in our approach these size constraints do not necessarily have to be chosen a priori. The mathematical problem (1) is solved for a series of values for K and D, and fortunately, the algorithm introduced below will provide such a series of solutions with relatively little additional effort. After a series of solutions has been obtained, the optimal tree size is chosen by selecting the tree with the lowest prediction error in the test set, or by using a more complicated cross-validation procedure. Sometimes, however, D can be specified a priori because of the maximum order of interaction between predictors that may be of interest in a particular application.

The mathematical problem (1) can be generalized without affecting the applicability of the algorithm presented below. In fact, the objective function is not restricted to the sum of a function $h(\cdot)$ for each region, but can be replaced by any *monotone* function of each $h(\cdot)$ for each region. For example, optimizing a tree with respect to

$$f(\mathbf{y}, \mathbf{X}) = \operatorname*{opt}_{\mathcal{P}} \max_{R_k \in \mathcal{P}} h(\mathbf{y}, \mathbf{X}, R_k), \tag{8}$$

or

$$f(\mathbf{y}, \mathbf{X}) = \operatorname*{opt}_{\mathcal{P}} \min_{R_k \in \mathcal{P}} h(\mathbf{y}, \mathbf{X}, R_k) \tag{9}$$

could be pursued. Such criteria for optimal splitting have been considered by Buja and Lee (2001) in a greedy tree framework. For example, for finding each best separate split in region R_k they propose to minimize the variance

$$h(\mathbf{y}, \mathbf{X}, R_k) = \min_{R_k^* = R_k^l, R_k^r} \frac{1}{|R_k^*|} \sum_i (y_i - a_k)^2 I(\mathbf{x}_i \in R_k^*), \tag{10}$$

or to maximize the mean

$$h(\mathbf{y}, \mathbf{X}, R_k) = \max_{R_k^* = R_k^l, R_k^r} \frac{1}{|R_k^*|} \sum_i y_i I(\mathbf{x}_i \in R_k^*), \tag{11}$$

where R_k^l and R_k^r are the left and right node regions after splitting region R_k. Effectively, this results in optimizing only one of the two nodes at each split. The authors admit that these criteria are much more greedy, but claim that they sometimes lead to more easily interpretable trees. Combining these ideas into an overall objective function by combining (9) with (10) or (8) with (11) may be considered even more greedy. Exploiting a minimax principle that minimizes the worst impurity may also be of interest, for example by *minimizing* (8) with respect to (10) or (2). Although such objectives functions are within the scope of the presented algorithm, they will not be studied in any detail in the current paper.

Another possible generalization is to extend trees into non-binary trees. From a combinatorial optimization perspective, binary trees can be easily reformulated into non-binary trees, where at each node multiple splits may occur for one predictor. This can be done by joining consecutive splits if they are performed on the same predictor. The overall objective function value will be the same, as well as the number of terminal nodes. The principal difference is the enforcement of the maximum depth constraint that has a different interpretation for binary trees than for non-binary trees. When non-binary trees are explicitly sought, the current formulation for optimal binary trees can be used by increasing the maximum depth constraint sufficiently to accommodate the number of times consecutive splits are induced by the same predictor. Alternatively, the algorithm can be changed such that the depth in any terminal node is defined by the number of *np*- splits from the top to that terminal node.

3 Improving greedy algorithms

The decision trees problems that are the focus of this paper are assumed to be intractable, because related problems such as building a minimal size decision tree from decision tables is known to be NP-Hard (Hyafil & Rivest, 1976). In practice this means that within the current computing paradigms an *efficient* algorithm, i.e. one where computing resources scale with a polynomial function to the problem size, is not expected to ever exist.

A simple look at the combinatorial explosion of possible trees for even the smallest problems, indicates why an heuristic approach that constructs trees by a growing process is by far the most often exploited approach. This growing methodology evaluates at each existing node all possible ways of splitting that node into two new nodes, extending the tree by taking the best split, and repeating the process for all new nodes. Because such a procedure does not take into account the effects of possible new splits in the next step, it does not actually *optimize* or solve (1) in the strict sense. This deficiency has been recognized from the start and a limited look-ahead-strategy was included in AID (Sonquist & Morgan, 1964); Breiman et al. (1984) also considered the possibility of improving the procedure by using a look-ahead-strategy. Look-ahead-heuristics try to improve the current split by considering the effects of splits several levels beyond the current node. Originally, such look-ahead-strategies were found to run quickly out of bounds of computing power. Moreover, later research suggested that, in general, look-ahead-heuristics are only little more accurate than greedy algorithms (Murthy, 1997).

In contrast, many criteria have been proposed for optimizing splits locally that aim at improving the overall tree while being not too optimistic over the effectiveness of the current split (Breiman et al., 1984, e.g., the GINI index). To illustrate local optimality, consider Figure 2 that illustrates an application of a regression tree to predict treatment success of panic disorders using six predictors (taken from Bakker, 2000). The figure depicts the total sum of squared error of a number of optimal trees for a particular, *given* first split

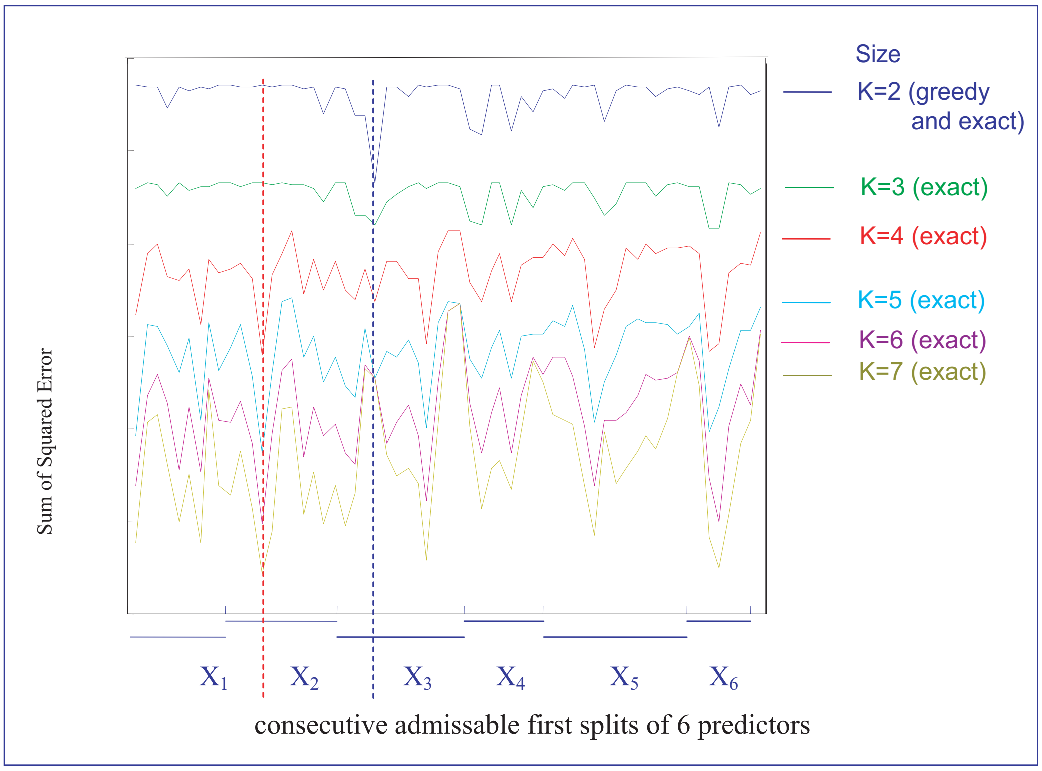


Figure 2: Location of the best first split (the lowest point in each curve) for a greedy regression tree ($K = 2$) and six exact regression trees, with sizes $K = 2, \ldots, 7$.

that depends on the admissible maximum number of terminal nodes K. The horizontal axis contains the sequence of all possible splits for each of the six predictors, the vertical axis gives the overall squared error of the final tree for the given size, as obtained by the ExactTree algorithm. Note that by definition the best split for an optimal tree of size $K = 2$ is equivalent to the best split found by a greedy algorithm with an equivalent local criterion. Not surprisingly, the overall error decreases when the size of the tree increases. Most significant, however, is the fact that the location of the best split vastly differs depending on the maximum size of the tree, K. The location of the best first split (on X_3) for size (K=2) is not the location of the best split (on X_2) for all sizes $K > 3$. The overall similarity in the location patterns for sizes $K > 3$, however, suggests that for the current example a look-ahead-search of depth 3 may be sufficient.

Another way of improving upon greedy methods is a two-stage procedure, where a greedy method is used to find the first part of the tree, while the remaining part of the tree is truly optimized. Several algorithms have been used, including ones that use Dynamic Programming (Meisel & Michalopoulos, 1973; Moret, 1982). Also, exact formulations have been proposed for globally optimizing a fixed tree structure, using a Linear Programming approach combined with modern heuristic algorithms, and using a branch-and-bound strategy with linear programming at the nodes (Bennett, 1994; Bennett & Blue, 1996). Such algorithms, however, are limited to the minimization of the sum of squared error loss, whereas the current proposal can handle a much larger class of loss functions.

4 The Dynamic Programming recursion

To formulate our Dynamic Programming algorithm some more definitions are needed that take into account the hierarchical nesting of the splits. Let v be any node (including the root node) of a decision tree $T(R)$ of the complete measurement space R, and let R' and $T(R')$ denote the region ($R' \subseteq R$) and the (sub)tree associated with v; let $l(R', s, j)$ and $r(R', s, j)$ be a pair of left and right functions that split the region R' into a left and right measurement space using split $s \in \mathcal{S}_j(R')$, where $\mathcal{S}_j(R')$ denotes the set of admissible splits of R' by predictor X_j. Furthermore, let $h(R')$ be a shorthand notation for $h(\mathbf{y}, \mathbf{X}, R')$, and let $t(R')$ denote the value of the objective function for obtaining a (sub)tree $T(R')$ for region R', such that

$$t(R') = \sum_{R_k \in \mathcal{P}(R')} h(R_k). \tag{12}$$

Finally, let $T^*(R')$ be an *optimal* (sub)tree and $t_d(R')$ the associated (optimal) value of the objective function. We have the following *principle of optimality* (Bellman, 1957):

Proposition 1 *Any subtree $T(R')$ starting at some node v of an optimal tree $T^*(R)$ is itself an optimal tree of the measurement space R' associated with its root node v.*

PROOF. The objective function can be split into the sum of two parts

$$t^*(R) = \sum_{R_k \in \mathcal{P}(R)} h(R_k) = \sum_{R_{k_1} \in \mathcal{P}(R')} h(R_{k_1}) + \sum_{R_{k_2} \in \mathcal{P}(R-R')} h(R_{k_2}), \tag{13}$$

where the first part contains the regions that belong to the subtree $T(R')$, and the other part contains all other regions. Hence, we can always replace any subtree of an optimal tree with a better subtree if that would exist, but this would contradict the optimality of the original tree T.

This particular splitting of the objective function is often referred to as the *separability* of the objective function. In this case, it stems from the fact that we can evaluate the part of the objective function for the subtree independently from the way in which the tree partitions the rest of the measurement space.

This principle quite naturally leads to a recursion for optimal decision trees. In fact, the transition from a recursion to a tree and vice versa is so transparent that trees are often used to explain DP algorithms. Moreover, these type of recursions for trees are well-known in computer science, where the classical example is the use of DP for the the minimal size "Optimal Binary Search Tree", a problem that is not NP-hard, and were DP is very efficient. In contrast, DP for NP-hard problems is often not considered because of the explosion of computer resources needed for increasing problem sizes, with the major exception of Hubert et al. (2001), who present DP solutions (and software) for finding exact solutions to a number of combinatorial data analysis problems.

In formulating the recursion, however, we have to take into account the size restrictions of the mathematical problem (1) that are implicit in this principle of optimality. In other words, the principle of optimality only holds if the size constraints for the subtree conform to the overall size constraints of the complete tree. To define practical recursions that can readily be converted into algorithms, it is more convenient to explicitly include the size constraints. For maximum-depth trees, where only the depth-constraint is imposed, we have the following recursion:

$$\begin{cases} t^*_d(R') = & \operatorname*{opt}_{s,j} \left[t^*_{d-1}(l(R',s,j)) + t^*_{d-1}(r(R',s,j)) \right] \\ & \text{subject to } s \in \mathcal{S}_j(R'); j = 1,\dots,J; \\ t^*_d(R') = & h(R') \\ & \text{if } d = 1 \text{ or other admissibility restrictions prevent further splitting of } R'. \end{cases} \tag{14}$$

In words, the process of finding an optimal tree can be described by

i. search through all possible splits at the current node

ii. for each split, find independently an optimal subtree at the left and the right node

iii. a. if a node is allowed to be split further, the search process is recursively started for this new node

iv. b. if a node is an end node, we simply use the objective function value for this node

v. the objective function value for the current split at the current node is the sum of the objective function values of the optimal subtrees

vi. the split that results in the best combination of subtrees gives the optimal (sub)tree at the current node and is returned

vii. if the current node is the root node, we have found the complete optimal tree; otherwise we return to the previous level in the search.

When we also impose the overall size constraint $|\mathcal{P}| \leq K$, the recursion becomes more elaborate, because we generally do not know in advance the size of the left and the right subtree for any split:

$$\begin{cases} t^*_{dk}(R') = & \underset{s,j,k_l}{\text{opt}} \left[t^*_{d-1,k_l}(l(R',s,j)) + t^*_{d-1,k-k_l}(r(R',s,j)\right] \\ & \text{subject to } s \in \mathcal{S}_j(R'); j = 1,\ldots,J; k_l = 1,\ldots,k-1; \\ t^*_{dk}(R') = & h(R') \\ & \text{ if } d = 1 \text{ or } k = 1 \text{ or} \\ & \text{other admissibility restrictions prevent further splitting of } R'. \end{cases} \tag{15}$$

The search process is now extended to not only include an optimal choice of the split s and the predictor j, but also the optimal split of the size constraint k at the node into two size constraints k_l and $k_r, k = k_l + k_r$ for the left and right subtrees. Consequently, imposing the size constraint $|\mathcal{P}| \leq K$ additionally to imposing the depth constraint increases the workload of completely searching the recursion. For an algorithm that finds optimal trees for a range of size constraints, however, the recursion can be reformulated such that the number of possible nodes visited for finding a range of depth-constrained optimal trees is of the same order of magnitude as for finding a range of both depth and size constrained trees.

As indicated above, these recursions do not only hold for the current objective function that takes the sum of a function $h(\cdot)$ for each region, but hold for any *monotone* function of each $h(\cdot)$ for each region. Special cases are the objective functions (8) and (9), where the principle of optimality does not strictly hold. Here, the DP recursion will generate *one* of the possible optimal trees, because multiple trees with the same objective function value may exist. However, restricting the optimal solution to fulfill the principle of optimality for all subtrees might lead to a more interesting overall tree, because in such a tree the overall structure will more likely be unique.

Algorithm 5.1 $h \leftarrow$OptimalTree(R', d)
Find the minimal objective function value h of the best tree, springing from a node defined by region R', with depth d (recursive)

```
1: if $\exists(R', h)$ then
2:     retrieve $(R', h)$
3: else if $(d = 1) \vee (\mathcal{S}_j(R') = \emptyset, \forall j) \vee (R'$ is homogeneous enough$)$ then
4:     $h \leftarrow h(R')$
5: else
6:     $h \leftarrow \inf$
7:     for $j = 1$ to $J$ do
8:         for $s \in \mathcal{S}_j(R')$ do
9:             $h^{(l)} \leftarrow$ OptimalTree$(l(R', s, j), d - 1)$
10:            $h^{(r)} \leftarrow$ OptimalTree$(r(R', s, j), d - 1)$
11:            $h \leftarrow \min(h, h^{(l)} + h^{(r)})$
12:        end for
13:    end for
14:    store $(R', h)$
15: end if
Ensure: $\exists h$
```

5 Algorithm Implementation

Given the recursions (14) and (15), we can immediately formulate recursive algorithms. For example, Algorithm 5.1 gives a DP algorithm for finding an optimal maximum-depth tree. This algorithm explicitly uses a heap to store and retrieve previously searched nodes, and only gives the (optimal) objective function value of an optimal tree. To retrieve the optimal tree itself, the tuple $(R', (s, j))$ that corresponds to the optimal split must be stored in the heap as well. When the algorithm is finished, a simple recursive routine can restore the optimal tree by retrieving all optimal $(R', (s, j))$ from the heap, starting at the root node for R', applying the split (s, j) giving $R'^{(l)}$ and $R'^{(r)}$, retrieving the optimal splits for those measurement spaces, and so forth.

The current algorithm constrains the tree size by maximum depth. For finding optimal trees with a maximum number of terminal nodes K, we have to extend the algorithm. Taken the above recursion, a straightforward implementation is to induce an additional loop that circles over all possible sizes of (sub)trees before stepping into a lower level. However, a more efficient approach is given by Algorithm 5.2 that extends the search by simultaneously searching at each node for all optimal trees of size one up to $K-1$ rather than

Algorithm 5.2 $H \leftarrow$OptimalSCTrees(R', d, K)
Find the minimal objective function values $H = \{h_1, \dots, h_K\}$ of all Size Constrained trees with number of terminal nodes not exceeding $2, \dots, K$, springing from a node defined by region R', with depth d (recursive)

```
1: if $\exists(R', H)$ then
2:    retrieve $(R', H)$
3: else if $(d = 1) \vee (\mathcal{S}_j(R') = \emptyset, \forall j) \vee (R'$ is homogeneous enough$)$ then
4:    $h_k \leftarrow h(R'), k = 1, \dots, K$
5: else
6:    $H \leftarrow$ inf
7:    $h_1 \leftarrow h(R')$
8:    for $j = 1$ to $J$ do
9:       for $s \in \mathcal{S}_j(R')$ do
10:         $H^{(l)} \leftarrow$ OptimalSCTrees$(l(R', s, j), d - 1, K - 1)$
11:         $H^{(r)} \leftarrow$ OptimalSCTrees$(r(R', s, j), d - 1, K - 1)$
12:         $h_k \leftarrow \min(h_k, h^{(l)}_{k'} + h^{(r)}_{k-k'}), k = 2, \dots, K,$ and $k' = 1, \dots, k$
13:      end for
14:   end for
15:   $h_k \leftarrow \min(h_{k'}), k = 2, \dots, K,$ and $k' = 1, \dots, k$
16:   store $(R', H)$
17: end if
Ensure: $\exists H$
```

the optimal tree. To do so, we have to search for all optimal combinations of subtrees such that $2 \leq |P(R'^{(l)})| + |P(R'^{(r)})| \leq K$, and store the optimal combinations for all sizes $1, \dots, K$. (The 'optimal combination' of size one is no split).

The practical details as well as the limits for using a (partial heap) are discussed in the next paragraph. Effective implementations have so far mostly not implemented a heap, but instead simply fully exploit the recursive algorithm.

6 Complexity

Formulating the theoretical (worst case) complexity of the algorithms in closed form is quite cumbersome. This complexity may be more easily calculated using a recursive function, but this formula will give little insight in the complexity. Key aspects of the complexity, however, can be readily discerned by characterizing the number of times nodes are visited by the algorithm at a particular depth. Let a *np*-node be a node that is defined by *np*-splits only (all splits are defined on different predictors) and let's assume that any conceivable node actually exists (i.e. the sample is infinitely large, all combinations of predictor splits have observations and no additional admissibility restrictions are in effect). Then the following holds:

Proposition 2 *Any admissible* np-*node at depth d in the tree is visited exactly $d!$ times by the DP algorithm.*

The proof follows from the fact that an *np*-node at depth d is defined by a combination of d rules on d different predictors. For example, any observation i might belong to some region R_k at depth $d = 2$ if $(X_3 < s_1) \wedge (X_2 < s_1) \wedge (X_6 \in \{1, 4, 5\})$. However, while the ordering of the predictors in the combined rule is irrelevant, the DP algorithm will visit nodes for all possible orderings of the predictors, hence any particular node is visited $d!$ times.
This brings up the question whether it is actually necessary to visit each node $d!$ times at depth d. The answer is that any restriction on the ordering of the predictors during the search will violate the independence of the left and right subtree at a higher level. Although the ordering of the predictors does not affect the definition of any particular node, it does affect which set of other nodes might co-exist with this particular node in the overall tree structure.

Visiting a node $d!$ times will also start the search for an optimal subtree $d!$ times in the recursion (for $d < D$). Consequently, the introduction of a heap at depth d that stores the optimal subtree for any node will approximately reduce the workload at that depth by a factor $d!$, provided storing and retrieving the results (R', h) that characterize the optimal subtree starting at that node is much less work than searching for the optimal subtree (which is generally the case). Recall that the set of possible splits for predictor $X_j, j = 1, \dots, J$ is defined by S_j and the number of possible splits for predictor X_j by $|S_j|$. Because we have both a left and a right split result, we need to refer to $2|S_j|$ split results for each predictor. Let $\kappa\,(k, J, d)$ denote the k'th combination of d indices out of the full set $1, \dots, J$. The (maximum) number of *np*-nodes at depth d is defined by

$$\sum_{k}^{\binom{J}{d}} \prod_{j \in \kappa(k,J,d)} 2\,|S_j|. \qquad (16)$$

In addition, also non *np*-nodes will be visited. In these nodes splits will be induced by two or more splits on the same predictor, leading to at most

Table 1: Complexity of Finding Optimal Trees

Splits	Theoretical #		Heap	Empirical #	
per Predictor	Trees	Nodes	Nodes	Nodes	Nodes
	D=5	D=5	D=5	D=5	D=6
2,2	1241	241	60	231	231
4,4	27657093	6945	465	4749	14529
2,2,2	19628029	3389	325	1600	3179
4,4,4	3.0e+012	78901	3471	32131	193802
2,2,2,2	3.9e+010	18273	1227	9083	36233
4,4,4,4	3.2e+015	372801	15305	158853	1649657

$(|S_j| + 1)^2$ split results (irrespective of the actual number of splits induced on the same predictor). In the simplest case, at depth $d = 2$, we have only nodes with multiple splits for single predictors and no other predictors involved, such that at depth $d = 2$ the number of additional nodes is

$$\sum_j^J (|S_j| + 1)^2. \tag{17}$$

Using these building blocks we can systematically define the number of nodes and required heap size for $d > 2$.

At this point we can envision that the number of nodes and the number of times nodes are visited explodes for increasing depth of the tree. At the same time the number of predictors as well as the number of splits for each predictor influence the heap and workload. In practice, only partial heaps at depth $d = 2$ can be implemented for moderately sized problems, and partial heaps at depth $d = 3$ for the smallest problems. It is not surprising that previously a DP-algorithm has only been considered for optimizing reduced tree problems where both the tree structure and the predictors at specific nodes are given (Meisel & Michalopoulos, 1973), rather than trying to optimize the complete tree structure, as is done in this paper.

However, the maximum heap size does not reflect the actual number of nodes visited. Table 1 gives some examples of relatively small problems, with the number of trees that a complete enumerative process would have to search for in column 1, compared in columns 2-5 to the number of nodes that is used by a DP alternative, both for a finite sample (theoretical) and for the dataset used in the first application (empirical, with $N = 506$). The table clearly demonstrates that the number of nodes to be visited very quickly explodes, especially if the maximum depth constraint is increased, although

DP introduces a major improvement over complete enumeration. The number of nodes searched by DP fitting a tree to an empirical dataset shows that the practical complexity is significantly lower. Two factors strongly decrease the practical complexity of these tree problems. First, the actual sample size strongly limits the possible number of splits. In other words, the measurement space is only sparsely filled with observations, thereby strongly limiting the number of admissible partitionings of this space, because regions that are empty are not admissible in a partitioning. Secondly, the possible correlation between predictors decreases the actual number of possible splits even further.

Therefore, the algorithm can deal with nontrivially sized problems of say a dozen of predictors, fitting trees with $NC \leq 20$ and $D \leq 6$ for a sample size of $N = 500$. The actual maximum problem sizes are strongly data dependent, but as a rule of thumb, a maximum depth of 5 or 6 is a hard limit, while complexity is relatively less affected by the number of predictors and the number of splits within each predictor. For problems where the number of possible splits is fixed (i.e. discrete predictors), the sample size N has only a small effect, because running time is likely to be related to a function of $log(N)$ rather than N.

Given the highly exponential increase in running time for increasing tree problem sizes, the algorithm is not intended as a complete replacement for recursive partitioning methods that grow trees. Especially the arrival of a large number of predictors and the need to grow large trees may often lead to exorbitant running times. In those cases, greedy methods may be used to do an initial search for relevant predictors and possible tree structures. Given these results, the exact DP algorithm can be used to fit optimal trees on a subset of the predictors, with suitable size restrictions. By doing so, one would obtain optimal solutions to approximate (restricted) tree problems, which may improve prediction accuracy.

7 Reducing the search space of continuous predictors

In the case of ordered predictors that are continuous rather than discrete, the possible number of splits can easily become as large as the number of observations. In general this poses a problem to CART because any imbalance in the possible number of splits between predictors tends to lead to variable selection bias: predictor variables with more distinct observed values tend to be selected more often than predictors with less distinct values. For the present exact algorithm, continuous predictors strongly increase the running time for large samples.

Several strategies are used for reducing or limiting the number of possible splits for continuous predictors. These strategies digitize a predictor variable, giving a limited number of categories (binning), or — in other words — groups the original observations in a limited number of ordered classes. This will not only limit the search space, but may lead to increased stability of the trees found, and possibly reduce variable selection bias. We will consider three different methods. The first method is based on dividing the original scale of a predictor variable into M intervals (corresponding to the M new ordered classes) assuming a theoretical distribution of the variable. For uniformly distributed variables, fixed width intervals can be used. For normally distributed variable (Max, 1960) proposes optimal z-values. The second and third method are based on the empirical distribution of a variable. The second one digitizes a predictor variable into four categories at the sample quartiles (GUIDE, (Loh, 2002)). This discretization is not performed once, but repeated at each node of a tree, giving the data available at that node. The third method consists of simply rounding the observed values of a continuous predictor variable.

In combination with ExactTree, we developed a new method, which we call 'optimal digitizing' or 'optimal binning'. It optimally divides the observations in an ordered predictor variable j in a sequence of M ordered classes $C_m, m = 1, \ldots, M$, such that the squared error between each original observation x_{ij} and its class mean $\overline{x}_{mj}$ is minimal, by minimizing

$$\min_{C_1,\ldots,C_M} \sum_{m}^{M} \sum_{i \in C_m} (x_{ij} - \overline{x}_{mj})^2. \tag{18}$$

In contrast to Max, this method optimizes the discretization with respect to the *empirical* distribution. A polynomial time Dynamic Programming algorithm that was originally proposed by Fisher (1958) and Rao (1971) can be used here to provide a so-called 'optimal binning'.

8 Application to Boston Housing data

The proposed exact algorithm will be applied to a classic dataset of Harrison and Rubinfeld (1978) that has been analyzed extensively in the regression literature (Breiman et al., 1984). Harrison and Rubinfeld studied the possible effect of air pollution concentration (NOX) on the median values of owner-occupied homes. The data consist of 13 predictors that were used to predict

Table 2: Overview of the predictor variables in the Boston Housing Dataset

Label	Description
CRIME	crime rate
ZN	percent land zoned for lots
INDUS	percent nonretail business
CHAS	1 if on Charles River, 0 otherwise
NOX	nitrogen oxide concentration, pphm
RM	average number of rooms
AGE	percent built before 1940
DIS	weighted distance to employment centers
RAD	accessibility to radial highways
TAX	tax rate
P/T	pupil teacher ratio
B	percent black
LSTAT	percent lower-status population

these median values of houses in 506 census tracts in the Boston Area. The predictors are given in Table 2.

The present application will concentrate on comparing the prediction accuracy of optimal trees estimated by ExactTree with greedy trees as produced by RPART, available in R (R Development Core Team, 2008) or Splus (Thernau & Atkinson, 2002). Comparing the apparent error rates of optimal trees with greedy trees is not very useful, because given the same search space and size constraints, the exact algorithm will always find trees with the same or lower apparent error. The latter result indicates how suboptimal greedy trees are.

The question is whether the expected prediction error is decreased when the tree structure is optimized. To see whether this is true, both RPART and ExactTree were used in a tenfold cross-validation procedure to estimate the prediction accuracy. Based on the substantive research question, and the trees selected by Breiman et al., the search space was reduced by selecting five predictors and setting the maximum depth to 4. Next, the number of splits of the five predictors was reduced, first by rounding and then by optimal digitizing, giving the following predictors and associated number of bins/categories: CRIME (35), NOX (35), RM (45), DIS (74) and LSTAT (35). The number of categories resulted naturally from the rounding procedure, and thus the same number of categories was chosen for the optimal digitizing procedure. The comparison between RPART and ExactTree was complemented by using RPART to fit regression trees for the five predictors without reducing the number of data values. The division of the observations into the 10 sets needed for the tenfold cross-validation was kept fixed in all the

Table 3: Prediction accuracy of differently sized trees for the Boston Housing data. The predictors used are crime rate, nitrogen oxide concentration, average number of rooms, weighted distance to employment centers, and percent lower-status population.

	Rounding				Optimal Digitization				No Digitization	
Size	RPart		DPTree		RPart		DPTree		RPart	
# Nodes	xerror	σ	xerror	σ	xerror	σ	xerror	σ	xerror	σ
1	1.003	.083	1.003	.083	1.003	.083	1.003	.083	1.003	.083
2	.562	.054	.562	.054	.601	.058	.601	.058	.600	.058
3	.397	.044	.428	.045	.414	.048	.423	.044	.421	.048
4	.333	.046	.291	.039	.358	.045	.299	.039	.375	.050
5	.333	.046	.282	.039	.281	.030	.283	.037	.306	.037
6	.318	.047	.282	.045	.257	.030	.215	.020	.283	.037
7	.278	.045	.294	.048	.244	.030	.235	.040	.269	.037
8	.260	.042	.310	.049	.221	.028	.222	.029	.234	.034
9	.263	.042	.275	.044	.216	.028	.207	.026	.251	.039
10	.277	.047	.268	.046	.204	.025	.199	.025	.245	.041
11	.276	.047	.278	.049	.243	.044	.189	.022	.247	.041
12	.276	.047	.235	.040	.241	.044	.150	.016	.246	.041
13	.273	.047	.201	.034	.234	.044	.138	.013	.246	.042
14	.277	.048	.203	.038	.240	.045	.142	.014	.253	.043
15	.281	.049	.157	.024	.239	.045	.141	.014	.251	.043
16			.150	.022			.130	.012		

xerror denotes the prediction error by tenfold cross-validation, and σ is its standard error.

analyses to avoid differences between results to be based purely on chance. The differently sized trees in RPART were obtained by the usual pruning approach to reduce larger trees, and are necessarily nested. The differently sized trees produced by ExactTree can be nested, but are not necessarily so because each tree is optimized for a particular size, and this is independent from optimizing trees for other sizes. However, the search for an exact tree of a larger size will include many intermediate results that are also used in the search for an exact tree of a smaller size. Therefore, to save computing time, all differently sized exact trees were obtained simultaneously by one search process, while still obtaining independent optimal results (with possibly not necessarily nested trees). The results for the comparison are given in Table 3.

To select the best sized tree, we use the one-standard error rule (Breiman et al., 1984) that selects the smallest tree with cross-validated prediction error within one standard deviation above the minimum cross-validated error. In Table 3, the first two columns give the prediction error for RPART and ExactTree, where the number of distinct values has been reduced by rounding. RPART selects a tree with 7 terminal nodes that has an estimated prediction error of .278. The ExactTree algorithm selects a tree with 15 terminal nodes with an estimated prediction error of .157. The selected tree by ExactTree is much larger, indicating that a large number of splits could be estimated accurately. When the number of distinct values is reduced by the optimal digitizing method (with results given in the columns 3 and 4 in Table 3), ExactTree selects a tree with 13 terminal nodes with prediction error .138, compared to a tree with 8 terminal nodes and prediction error .221 for RPART. For all trees having more than 5 terminal nodes, the prediction error obtained by optimal digitizing is smaller than the estimates obtained for digitizing by rounding, and this is true both for RPART and ExactTree. The best trees selected by ExactTree in the columns 2 and 4 have smaller prediction error than the best result for RPART without digitizing the data, which given in column 5. When we compare the results for RPART in the columns 3 (optimally digitized predictors) en columns 5 (no digitization, thus using the original predictors), it is striking that reducing the number of categories optimally results in lower prediction error. For the results with optimal digitizing in columns 3 and 4, the Figures 3 and 4 show the best trees that were selected by RPART and ExactTree, respectively. The labels at the split points are the original values of the predictor variables. Figure 3 strongly resembles the regression tree displayed in (Breiman et al., 1984), p. 219, which was obtained using the original 13 predictors. When the optimal tree in Figure 4 is compared to the greedy tree in Figure 3, the most striking difference is that the variable air pollution (NOX) is selected in the optimal tree and not in the greedy tree. So, where the optimal tree shows that the median value of owner-occupied homes is the lowest (bottom right: 9.5) in the districts with a high concentration of nitrogen oxide (NOX ≥ 0.6), a high crime rate (CRIME ≥ 12.8), and a high percentage of lower-status population (LSTAT ≥ 14.5), the greedy tree does not indicate that NOX is important for the prediction of the median values of houses in Boston.

In subsequent analyses, we studied the relationship between the number of categories of the five predictors and the prediction error, estimated by RPART and ExactTree. The number of categories was reduced by optimal digitizing to either 35, 25, or 15, with the same number of categories for all five predictors. The results are given in Table 4.

The estimated prediction errors of the selected best sized tree by RPART and by DP are .324 and .235 for the 35 categories situation, .293 and .124 for the 25 categories, and .338 and .260 for the 15 categories situation. Comparing the results within RPART and ExactTree first (columns 3 and 4 in Table 4), the smallest prediction error is achieved when the data are digitized into 25 categories. When we compare the results between RPART and ExactTree, the optimal tree performs better in all three conditions. To con-

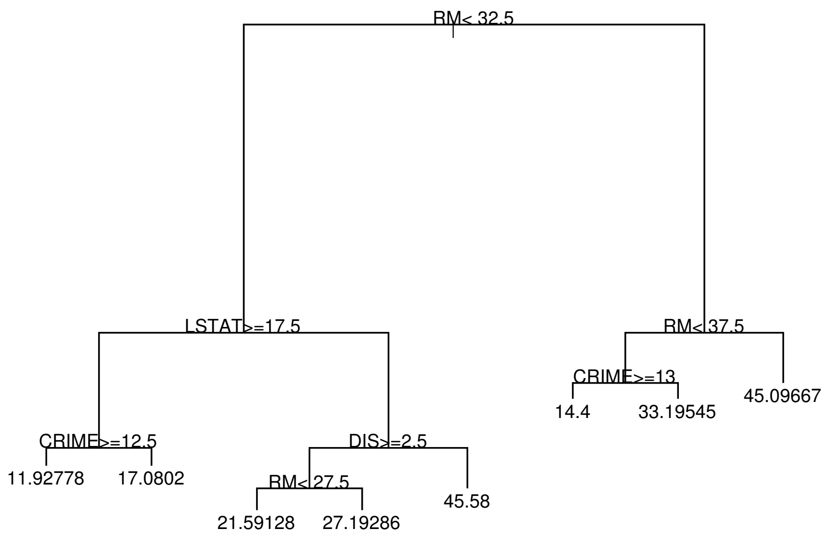
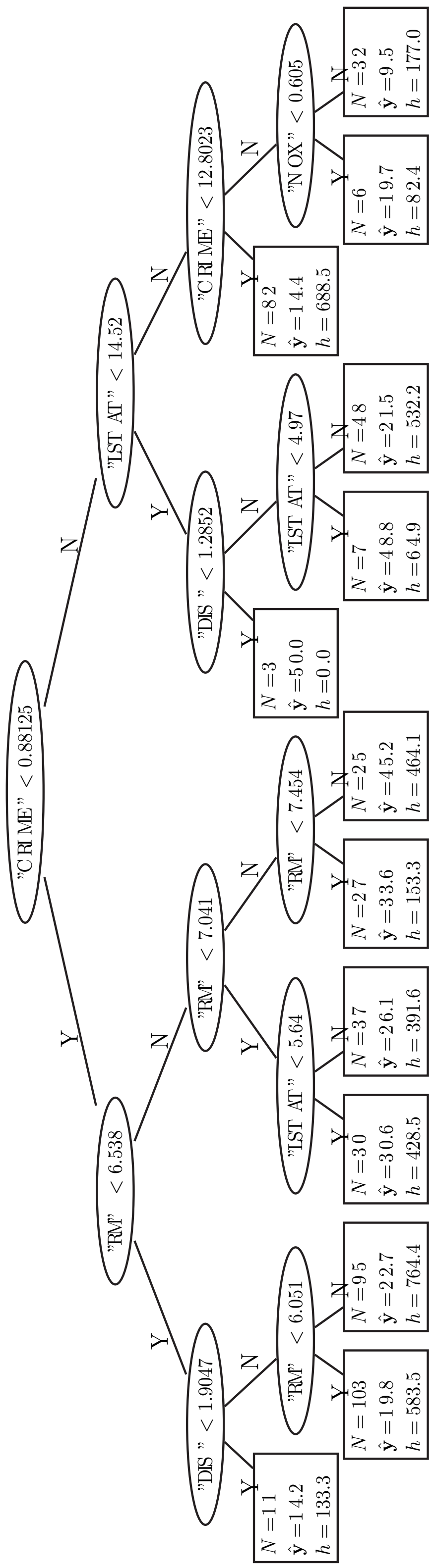


Figure 3: The greedy tree for the Boston Housing data.


"CRIME" < 0.88125
Y
N
"RM" < 6.538
"LSTAT" < 14.52
"DIS" < 1.9047
"RM" < 7.041
"DIS" < 1.2852
"CRIME" < 12.8023
"RM" < 6.051
"LSTAT" < 5.64
"RM" < 7.454
"LSTAT" < 4.97
"NOX" < 0.605
N = 11, ŷ = 14.2, h = 133.3
N = 103, ŷ = 19.8, h = 583.5
N = 95, ŷ = 22.7, h = 764.4
N = 30, ŷ = 30.6, h = 428.5
N = 37, ŷ = 26.1, h = 391.6
N = 27, ŷ = 33.6, h = 153.3
N = 25, ŷ = 45.2, h = 464.1
N = 3, ŷ = 50.0, h = 0.0
N = 7, ŷ = 48.8, h = 64.9
N = 48, ŷ = 21.5, h = 532.2
N = 82, ŷ = 14.4, h = 688.5
N = 6, ŷ = 19.7, h = 82.4
N = 32, ŷ = 9.5, h = 177.0

1

Figure 4: The ExactTree for the Boston Housing data.

Table 4: Prediction accuracy of differently sized trees, varying the number of categories of the predictors. Optimal digitizing is used to reduce the number of distinct values.

	Digitizing into 35 cat.				Digitizing into 25 cat.				Digitizing into 15 cat.			
Size	RPart		DPTree		RPart		DPTree		RPart		DPTree	
# Nodes	xerror	σ	xerror	σ	xerror	σ	xerror	σ	xerror	σ	xerror	σ
1	1.003	.083	1.003	.083	1.003	.083	1.003	.083	1.003	.083	1.003	.083
2	.600	.057	.600	.058	.598	.057	.598	.057	.552	.053	.552	.053
3	.414	.048	.423	.044	.426	.048	.414	.041	.396	.043	.385	.039
4	.382	.049	.354	.051	.385	.049	.396	.049	.338	.041	.366	.045
5	.395	.052	.324	.041	.385	.049	.340	.040	.318	.043	.383	.051
6	.360	.048	.256	.035	.331	.043	.247	.021	.308	.042	.350	.049
7	.344	.048	.279	.049	.326	.043	.260	.041	.327	.044	.319	.046
8	.344	.048	.257	.037	.319	.043	.271	.042	.320	.044	.311	.047
9	.324	.048	.260	.037	.293	.042	.224	.022	.354	.049	.260	.041
10	.305	.045	.247	.039	.311	.046	.220	.024	.342	.048	.280	.046
11	.297	.047	.235	.036	.290	.041	.213	.024	.345	.048	.262	.045
12	.296	.047	.239	.038	.270	.039	.204	.023	.343	.048	.252	.044
13	.287	.047	.236	.038	.269	.039	.160	.016	.339	.047	.256	.045
14	.293	.048	.218	.037	.271	.040	.124	.012	.332	.047	.264	.045
15	.291	.048	.210	.037	.273	.041	.132	.012	.338	.049	.262	.045
16			.229	.042			.155	.020			.262	.045

xerror denotes the prediction error by tenfold cross-validation, and σ is its standard error.

clude, our results suggest that at least for this particular dataset, optimal trees have lower prediction error, smaller standard deviations, and are able to accurately fit larger trees compared to greedy trees. In other words, the ExactTree appears to produce more accurate and stable estimates by fitting an optimal tree structure.

9 Discussion

The present paper proposes an exact Dynamic Programming algorithm for finding optimal decision trees, where optimality is defined as the quality of the predictions at the leaves. DP is often dismissed a priori as a candidate for such hard problems, because of the enormous computational resources involved. On the one hand, it is true that ExactTree is limited by the size of the dataset set to be analyzed, and also by the size of the trees it can solve for. On the other hand, a *depth* of seven, which is feasible with ExactTree, is usually more than sufficient. So ExactTree can find sufficiently large optimal trees for moderately sized datasets, and thus it can deal with nontrivial statistical applications in a lot of domains. Moreover, ExactTree can do this while using a plethora of optimality criteria that are beyond the reach of alternative strategies such as linear programming and branch and bound; it uses a simple interface and does not require special expertise of the user (a Matlab version of the software is available from the third Author).

The application to the Boston Housing data shows that using the Exact algorithm indeed improves upon the prediction of greedy methods; compared to the greedy algorithm, Exact Tree both finds smaller trees with similar prediction error, and larger trees with smaller prediction error (while having a smaller standard error of the prediction error). These results might indicate that using an exact algorithm within the bounds of size constraints on the tree structure, indeed leads to a more stable and accurate estimation of the tree structure without over-fitting on the training data. Moreover, restricting the search space of splits for each variable by optimally digitizing the predictors into an moderate number of categories appears to improve the stability of the prediction (and this seems to be also true for greedy methods). Further study to confirm these results is obviously needed.

At the same time, practical limitations are also imminent. ExactTree cannot deal with the size of datasets that are common in large-scale data mining, nor can it be used to replace automatic variable selection in the presence of a large number of variables. In the latter case, however, it is worthwhile to use ExactTree to estimate an optimal tree for given predictors that were selected by another method. Another application of ExactTree is as a benchmark tool, for truly evaluating the performance of more elaborate heuristic approaches as compared to greedy methods. Finally, an obvious application of ExactTree would be as a part of a so-called Boosting Algorithm, such as MART, where the ensemble is made up by many *small* trees, to obtain a really strong predictor.

References

Alexander, W. P., & Grimshaw, S. D. (1996). Treed regression. *Journal of Computational and Graphical Statistics.*, *5*, 156–175.

Bellman, R. E. (1957). *Dynamic programming.* Princeton, NJ: Princeton University Press.

Bennett, K. P. (1994). Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, *26*, 156–160.

Bennett, K. P., & Blue, J. (1996). *Optimal decision trees* (Tech. Rep. No. 214). R.P.I. Math Report.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *26*, 123–140.

Breiman, L. (2001). Random forrest. *Machine Learning*, *45*, 5–32.

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

Buja, A., & Lee, Y.-S. (2001). Data mining criteria for tree-based regression and classification. In *7^{th} international conference on knowledge discovery in databases* (pp. 27–36). New York: ACM Press.

Chaudhuri, P., Lo, W. D., Loh, W. Y., & Yang, C. C. (1995). Generalized regression trees. *Statistica Sinica*, *5*, 641–666.

Drucker, H., & Cortes, C. (1996). Boosting decision trees. In D. Touretzky, M. Mozer, & M. Hasselmo (Eds.), *Advances in neural information processing systems 8* (pp. 479–485). MIT Press.

Dusseldorp, E., Conversano, C., & Van Os, B. J. (2010). Combining an additive and tree-based regression model simultaneously: Stima. *Journal of Computational and Graphical and Statistics*, *19*, 514–530.

Dusseldorp, E., & Meulman, J. J. (2001). Prediction in medicine by integrating regression trees into regression analysis with optimal scaling. *Methods of Information in Medicine*, *5*, 403–409.

Dusseldorp, E., & Meulman, J. J. (2004). The regression trunk approach to discover treatment covariate interaction. *Psychometrika*, *69*, 355–374.

Fisher, W. D. (1958). On grouping for maximum heterogeneity. *Journal of the American Statistical Association*, *59*, 789–798.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proc. 13th international conference on machine learning* (pp. 148–156). Morgan Kaufman.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Hand, D. J. (1997). *Construction and assessment of classification rules.* Chichester: Wiley.

Harrison, D., & Rubinfeld, D. L. (1978). Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management.*, *5*, 81–102.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction.* New York: Springer.

Hubert, L. J., Arabie, P., & Meulman, J. J. (2001). *Combinatorial data analysis: Optimization by dynamic programming.* Philadelphia: SIAM.

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, *5*, 15–17.

Loader, C. (1999). *Local regression and likelyhood.* New York: Springer-Verlag.

Loh, W.-Y. (2002). Regression trees with and unbiased variable selection and interaction detection. *Statistica Sinica*, *12*, 361–386.

Loh, W.-Y., & Vanichesetakul, N. (1988). Tree structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, *83*, 715–725.

Max, J. (1960). Quantizing for minimum distortion. *IRE Transactions on Information Theory*, *6*, 7–12.

Meisel, W. S., & Michalopoulos, D. A. (1973). A partitioning algorithm with application in pattern classification and the optimization ofdecision trees. *IEEE Transactions on Computers*, *C-22*, 93–103.

Moret, B. M. E. (1982). Decision trees and diagrams. *Computing Surveys*, *14*, 593–623.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data,and a proposal. *Journal of the American Statistical Association.*, *58*, 415–434.

Murthy, S. K. (1997). *On growing better decision trees from data.* Unpublished doctoral dissertation.

Murthy, S. K., & Salzberg, S. (1995). Decision tree induction: How effective is the greedy heuristic? In *Knowledge discovery and data mining* (p. 222-227).

Naumov, G. E. (1994). Np-completeness of problems of construction of optimal decision trees. *Soviet Physics, Doklady*, *36*, 270–271.

R Development Core Team. (2008). *R: A language and environment for statistical computing.* Vienna, Austria. (ISBN 3-900051-07-0)

Rao, M. R. (1971). Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, *66*, 622–626.

Ripley, B. D. (1996). *Pattern recognition and neural networks.* Cambridge: Cambridge University Press.

Sonquist, J. A., & Morgan, J. N. (1964). *The detection of interaction effects.* (Tech. Rep.). Ann Arbor: Institute for Social Research, University of Michigan.

Thernau, T. M., & Atkinson, E. J. (2002). *Recursive partitioning.*

Van Os, B. J. (2001). *Dynamic programming for partitioning in multivariate data analysis.* Unpublished doctoral dissertation, Leiden University.

Van Os, B. J., & Meulman, J. J. (submitted). *Dynamic programming for globally optimal hierarchical cluster analysis.*

Vapnik, V. (1996). *The nature of statistical learning theory.* New York: Springer-Verlag.