SurTree: constructing optimal survival trees with MurTree

Tim Huisman[1]

Supervisor(s): Emir Demirović[1]**, Jacobus G. M. van der Linden**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 20, 2023

Name of the student: Tim Huisman
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Jacobus G. M. van der Linden, Burcu Özkan

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Survival analysis revolves around studying and predicting the time it takes for a particular event to occur. In clinical trials on terminal illnesses, this is usually the time from the diagnosis of a patient until their death. Estimating the odds of survival of a new patient can be done by analyzing survival data from past patients in similar conditions. To cluster similar patients based on a set of features, survival trees may be employed, which act as decision trees that assign a survival distribution to each cluster. Many algorithms exist for creating useful survival trees, but not for creating *optimal* survival trees. In this paper, research on finding optimal classification trees is applied to survival analysis, by adapting the MurTree algorithm to construct survival trees. We present *SurTree*, an algorithm that applies many of MurTree's techniques to create globally optimal survival trees. Furthermore, we compare the output quality and runtime performance of SurTree to a state-of-the-art method for constructing survival trees, showing its optimality and its fast computation times on smaller datasets.

1 Introduction

In medical research, particularly in studies on terminal illnesses, one of the most informative and studied outcomes of an experiment is the time between the diagnosis of a disease and the death of the patient, may it occur during the observation period. By comparing these interval lengths between groups that have received different treatments, conclusions can be drawn about the effectiveness of said treatments. This field of research is called *survival analysis*, as it focuses on analyzing survival rates. The field can, however, be generalized to other situations as well, as long as there is a clearly defined point in time from which the experiment and timing start, and a concrete and non-repeatable *event of interest* to observe, like the termination of a contract or the malfunctioning of a machine.

Besides treatment trials, another common usage of survival analysis is prediction. Estimates on the survival chances of a patient after a certain amount of time can be made by observing how many others have survived for at least as long. Based on certain attributes of a patient, like age, BMI, or previous diagnoses, these estimates can be made more accurate, by comparing the patient to others who were similar to them.

A *decision tree*, also called a *survival tree* in this context, is a model that can classify patients into one of a set of predetermined buckets by answering a small number of questions about them (see Fig. 1). These questions are often binary predicates (*yes/no*-questions), such as "Is the patient at least 50 years old?" or "Does the patient have a BMI of at most 25?" Due to their easily interpretable structure, decision trees are a popular model. This is not just because humans can use them to classify patients manually, but also because the construction of a good decision tree can reveal important correlations between patient data and their survival rates.

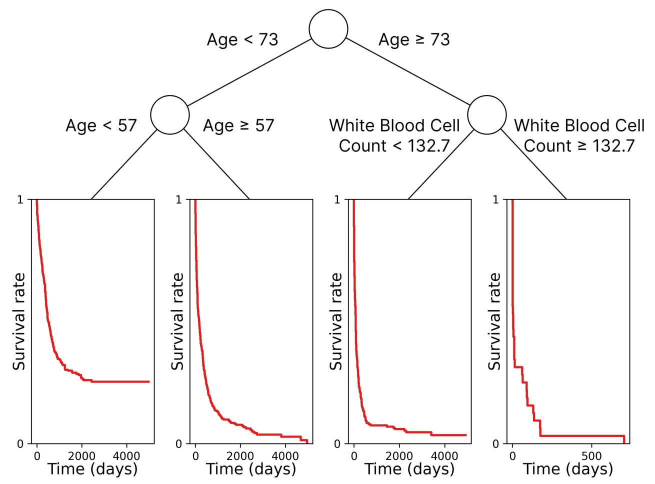

Figure 1: An example of a survival tree. In each leaf node, the survival rate is calculated using the Kaplan-Meier estimator (Kaplan & Meier, 1958).

While the traversal of decision trees is easy, the construction of such a decision tree can be quite complicated, as it is not clear at first glance what attributes matter the most and how exactly they affect the chances of survival. In fact, it has been proven that when faced with certain size constraints, constructing *globally optimal* decision trees, for which the misclassification on a certain dataset is as small as it can possibly be, is an NP-hard problem (Hyafil & Rivest, 1976).

Because of the difficulty of the problem, efforts have been made to construct non-optimal but still effective decision trees. The most popular method is the Classification And Regression Trees (CART) algorithm (Breiman, 1984), which uses greedy heuristic techniques to recursively partition a dataset to minimize the classification error or the mean squared error, depending on the type of tree. This algorithm has become quite popular, but cannot be directly translated to survival analysis, due to its inability to support censored data, which often occurs in survival analysis. This obstacle was later tackled by Davis and Anderson (1989), who provided an algorithm that applies recursive splitting of censored survival data with an interface similar to CART. Further algorithms based on CART would be developed by LeBlanc and Crowley (1993), Su and Fan (2004), Bertsimas et al. (2022), and others, each applying different splitting techniques. Despite these efforts, research on *globally optimal* survival trees has not yet become popular.

However, with hardware and algorithms having improved significantly in the last decades, research on globally optimal *classification* and *regression* trees, which optimize an error function to its absolute minimum, has flourished. Instead of heuristic methods, these algorithms tend to rely on exhaustive search. This search can be optimized using techniques such as dynamic programming (Aglin et al., 2020; Demirović et al., 2022; Nijssen & Fromont, 2007), mixed-integer optimization (Bertsimas & Dunn, 2017; Zhu et al., 2020) or SAT (Narodytska et al., 2018).

The recent developments in optimal decision trees motivate the study of whether its techniques can be translated to the field of survival analysis. The aim is to extend an algorithm to find *globally optimal survival trees* with regard to a certain metric. The mixed-integer optimization and SAT-techniques might prove difficult support survival analysis, due to many formulas and floating-point numbers involved. The focus of this research therefore lies on MurTree (Demirović et al., 2022), a state-of-the-art dynamic programming algorithm, which implements several optimization techniques to improve its scalability. These techniques might prove to be useful in the search for the best survival tree for a given dataset.

Main contributions. In this work, we aim to show that MurTree can be adapted to construct optimal survival trees. Pointwise, the main contributions offered in this paper are:

- A new algorithm called *SurTree*[1], which adapts and optimizes MurTree for survival analysis.
- A numerical analysis on the objective score of the trees generated by SurTree, comparing them to the trees generated by a state-of-the-art heuristic method and thereby demonstrating the optimality of SurTree.
- A numerical analysis showing how an improvement of the objective score correlates with an improvement of Harrell's C-index (Harrell et al., 1982), another metric often used in survival analysis.
- A numerical analysis describing how the runtime of SurTree scales with the number of instances, the maximum depth of the tree, and other attributes, while comparing it to a heuristic method's runtime.

2 Preliminaries

To properly communicate the aim of this paper and the process that was followed, it needs to be clear what terms and symbols will be used to describe the concepts and which concrete objective function needs to be optimized. In Section 2.1, the terminology that will be used in this paper will be introduced and explained. Following that in Section 2.2, we will derive the objective function that will be minimized by our algorithm.

2.1 Terminology

A *feature* f is a piece of information that describes a certain attribute of an entity. Similarly to MurTree (Demirović et al., 2022), we will consider all features to be *binary predicates*, i.e. statements about the entity which are either true or false. $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ is the set of all relevant features that could apply to an entity. A *feature vector* $\mathbf{fv}$ is a list of features that are considered true for a particular entity and can be used to describe the entity as a whole. If a feature f_j is true, we write $f_j \in \mathbf{fv}$ or say that the feature is *present*. If it is false, we write $\overline{f_j} \in \mathbf{fv}$ or say that the feature is *absent*.

An *event of interest* is the non-repeatable event for which the time until occurrence is measured within a certain trial. In future text, this will also be referred to as *event* or *death*. The *time-to-event* is the amount of time between the start of the observation of a particular entity and the occurrence of the event of interest.

In trials, it might happen that the exact time-to-event is unknown, possibly due to a patient leaving the trial before the event of interest could be observed. This phenomenon in which information after a certain timestamp is lacking is called *right-censoring*, from now on referred to as just *censoring*. Despite the ambiguity of censored data, it still provides useful information about an entity's survival up until the time of the last observation. Censored data and non-censored data are therefore both used for training.

An *instance* $(t_i, \delta_i, \mathbf{fv}_i)$ is an observation of one particular entity within a trial. It is described by a feature vector $\mathbf{fv}_i$, a binary *censoring indicator* δ_i stating whether the event of interest was observed, and a strictly positive *time* t_i. If the observation is not censored ($\delta_i = 1$), t_i denotes the time-to-event, otherwise ($\delta_i = 0$) it denotes the latest time of observation. A *dataset* $\mathcal{D}$ is a list of instances from a certain trial.

A *decision tree* is a model that partitions instances based on their features. In this setting, we consider decision trees as *full binary trees*, where each node is either a *decision node* with two children, or a *leaf node* with no children. Each decision node is assigned a certain feature to split on, and each leaf node is assigned a certain label. An instance that needs to be classified is first passed to the root. Whenever it is at a decision node, it is passed down to its left or right child, depending on whether the feature that the node checks for is present within the instance. Eventually, the instance will reach a leaf node, at which point it will be classified with the label of that node.

A *survival tree* (see Fig. 1) is a special type of decision tree that can be used in survival analysis. A classified instance in a survival tree is assigned not a numeric label, but a survival distribution that describes their odds of survival after a certain amount of time.

If a dataset $\mathcal{D}$ is split on feature f_j, the subset of instances for which f_j is present is written as $\mathcal{D}(f_j)$ and the subset for which f_j is absent is written as $\mathcal{D}(\overline{f_j})$. As more splits are made, more features are added in between the parentheses. For example $\mathcal{D}(f_1, \overline{f_3}, f_4)$ denotes the set of all instances in $\mathcal{D}$ for which f_1 and f_4 are present, but f_3 is absent.

2.2 Objective function

In order to construct optimal survival trees, a definition of *optimality* must be stated. Usually, a tree is considered optimal if it minimizes or maximizes a certain objective function when classifying a given dataset. Several different types of metrics can be considered as an objective function, such as Harrell's C-index (Harrell et al., 1982) or the Integrated Brier score (Graf et al., 1999). In this paper, we will follow the example of the *Optimal Survival Trees* (Bertsimas et al., 2022) algorithm, from now on referred to as *OST*, and make use of the objective function defined by LeBlanc and Crowley (1992).

The objective function assumes that for each leaf, the curve indicating the survival rate over time is described by Eq. (1). It is important to note that this formula only has to be used to quantify the error of a split; once a tree has been created,

[1]Publicly available at: https://github.com/TimHuisman1703/surtree/tree/master

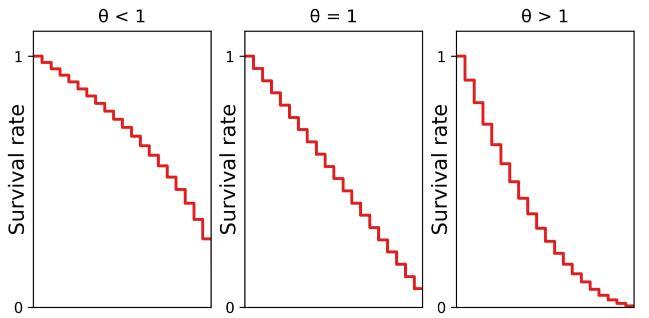

Figure 2: A visualization of how θ affects a survival distribution $S(t)$. Every plot uses the same $\hat{\Lambda}(t)$, but use $\theta = 0.5$, $\theta = 1$ and $\theta = 2$ respectively.

the actual survival curve can instead be estimated using the Kaplan-Meier estimator (Kaplan & Meier, 1958).

$$S(t) = P(T \geq t) = e^{-\theta\Lambda(t)} \tag{1}$$

In this equation, $\Lambda(t)$, represents the baseline cumulative hazard function, which can be interpreted as the cumulative probability of dying at a given time, according to the dataset. θ is a coefficient that can differ per class or instance. A higher θ indicates a higher risk of dying sooner, an example being shown in Fig. 2.

The true cumulative hazard function cannot be deduced from samples alone, but it can be estimated using the Nelson-Aalen estimator (Aalen, 1978; Nelson, 1972):

$$\hat{\Lambda}(t) = \sum_{i:t_i \leq t} \frac{\delta_i}{\sum_{j:t_i \leq t_j} 1} \tag{2}$$

While deciding the value of θ for a set of observations, we are aiming to maximize the likelihood of these observations. The formula used for the likelihood of an observation depends on whether it is censored or not. For non-censored observations, we know the exact time t at which the event took place, and can therefore calculate the likelihood using $s(t) = \frac{d}{dx}(1 - S(x))|_{x=t}$. Here, $1 - S(t)$ represents the probability of the event happening *before* the time t. For censored observations, however, we only know the event takes place later than a given time and we should therefore use $S(t)$.

The likelihood for a given observation can thus be written as follows:

$$L(t, \delta) = \begin{cases} e^{-\theta\hat{\Lambda}(t)} & \delta = 0 \\ (\theta \frac{d}{dt}\hat{\Lambda}(t))e^{-\theta\hat{\Lambda}(t)} & \delta = 1 \end{cases} \tag{3}$$

The likelihood for all observations in $\mathcal{D}$ within a certain leaf can be generalized to:

$$L = \prod_{(t_i, \delta_i, \mathbf{fv}) \in \mathcal{D}} (\theta \frac{d}{dt}\hat{\Lambda}(t_i))^{\delta_i} e^{-\theta\hat{\Lambda}(t_i)} \tag{4}$$

By maximizing the likelihood, the following formula for any coefficient for a leaf node is obtained:

$$\theta = \frac{\sum_{(t_i, \delta_i, \mathbf{fv}) \in \mathcal{D}} \delta_i}{\sum_{(t_i, \delta_i, \mathbf{fv}) \in \mathcal{D}} \hat{\Lambda}(t_i)} \tag{5}$$

If $\mathcal{D}$ were to contain only one instance $(t_i, \delta_i, \mathbf{fv}_i)$, Eq. (5) could be rewritten as Eq. (6). This formula gives the saturated coefficient of the instance, i.e. the coefficient that perfectly maximizes the likelihood for that one instance alone.

$$\theta_i^{sat} = \frac{\delta_i}{\hat{\Lambda}(t_i)} \tag{6}$$

The error that a particular instance contributes is defined as the difference between the log-likelihood of the chosen θ for its leaf node and the log-likelihood of θ_i^{sat}. This function translates to Eq. (7). It is worth pointing out that if a leaf node only classifies *one* instance from the dataset, then it holds that $\theta = \theta_i^{sat}$. Consequently, the error for that instance, and therefore for the entire leaf node, will be 0.

$$\text{Error}(\mathcal{D}, \theta) = \sum_{(t_i, \delta_i, \mathbf{fv}) \in \mathcal{D}} (-\delta_i \log \hat{\Lambda}(t_i) - \delta_i \log \theta - \delta_i + \hat{\Lambda}(t_i)\theta) \tag{7}$$

The error of the entire survival tree is calculated by summing the error at each of the leaves.

$$\text{Error}(\mathcal{D}, T) = \sum_{n \in \text{Leaves}(T)} \text{Error}(\mathcal{D}_n, \theta_n) \tag{8}$$

This is the error function that will need to be minimized during the construction of the survival tree T. OST uses this error function as well, but it uses gradient descent to find local optima for it. For that reason, it does not always manage to minimize this error as much as possible.

3 Review of MurTree

In this section, a short overview will be given on the original MurTree algorithm. Since the algorithm is used for classification instead of survival analysis, there are a few differences with regard to the problem description:

- Instead of a time t and a status δ, each instance is assigned a binary label $l \in \{0, 1\}$. A dataset $\mathcal{D}$ can then be partitioned in $\mathcal{D}_0$ and $\mathcal{D}_1$ containing the instances with a label of 0 and 1 respectively. It holds that $\mathcal{D}_0 \cup \mathcal{D}_1 = \mathcal{D}$ and $\mathcal{D}_0 \cap \mathcal{D}_1 = \emptyset$.
- Instead of being assigned a survival distribution, each leaf node is assigned either the label 0 or 1, which will be used to classify the instances that reach it.
- Instead of a particular survival analysis metric, the error that has to be minimized is the number of instances that are misclassified.

In Section 3.1 the general recursive structure of MurTree will be explained. Following that, Section 3.2 will elaborate on the terminal tree solver used to speed up base cases. Finally, Section 3.3 describes the computation of lower bounds based on similar datasets.

3.1 Recursive definition

MurTree is a dynamic programming algorithm that constructs optimal classification trees. applies an exhaustive search over all possible decision trees by recursively splitting over every possible feature at any node in the tree. Given a dataset $\mathcal{D}$, a maximum depth d, and a maximum number of decision nodes n, the baseline recursive function for finding the error of the best decision tree is described in Eq. (9) (Demirović et al., 2022).

$$T(\mathcal{D}, d, n) = \begin{cases} T(\mathcal{D}, d, 2^d - 1) & n > 2^d - 1 \\ T(\mathcal{D}, n, n) & d > n \\ \min\{|\mathcal{D}_0|, |\mathcal{D}_1|\} & n = 0 \lor d = 0 \\ \min\{T(\mathcal{D}(\overline{f}), d-1, n-i-1) & \\ \quad + T(\mathcal{D}(f), d-1, i) & \\ \quad : f \in \mathcal{F}, i \in [0, n-1]\} & n \neq 0 \land d \neq 0 \end{cases} \tag{9}$$

The first and second cases in this equation enforce the depth d and the number of nodes n to be consistent, since there exist no trees for which $n > 2^d - 1$ or $d > n$. The third case is a base case, reached when the maximum depth or the maximum number of nodes does not allow for more recursions. At this point, a label needs to be assigned to the new leaf node. The best strategy to minimize the error is to misclassify the smallest class among $\mathcal{D}_0$ and $\mathcal{D}_1$, resulting in an error of $\min\{|\mathcal{D}_0|, |\mathcal{D}_1|\}$. Finally, the fourth equation iterates over every possible feature to split on and every distribution among the two subtrees of the number of nodes that can still be used. This ensures that every possible tree is observed, guaranteeing optimality.

To prevent redundant calculations, dynamic programming is employed, memoizing $T(\mathcal{D}, d, n)$ for later use. $\mathcal{D}$ can be summarized by either hashing each instance (*dataset-caching*) or by hashing the splitting features used to obtain $\mathcal{D}$ (*branch-caching*).

3.2 Terminal solver

Instead of the base case where $d = 0$, shown in Eq. (9), MurTree makes use of a specialized terminal solver that can efficiently calculate trees with a maximum depth of 2. The idea is to iterate over each label $l \in \{0, 1\}$ and each pair of features $(f_i, f_j) \in \mathcal{F}^2$, and calculate the frequency count $FQ_l(f_i, f_j)$, which is the number of instances in $\mathcal{D}_l$ for which f_i and f_j are both present. $FQ_l(f_i, f_j)$ can be interpreted as the obtained error if the branch with $\mathcal{D}(f_i, f_j)$ is *not* assigned the label l. $FQ_l(f_i, f_i)$ will be abbreviated as $FQ_l(f_i)$.

Once these frequency counts are computed, similar counts of other branches can be deduced using the following formulas:

$$FQ_l(\overline{f_i}) = |\mathcal{D}_l| - FQ_l(f_i) \tag{10}$$

$$FQ_l(f_i, \overline{f_j}) = FQ_l(f_i) - FQ_l(f_i, f_j) \tag{11}$$

$$FQ_l(\overline{f_i}, f_j) = FQ_l(f_j) - FQ_l(f_i, f_j) \tag{12}$$

$$FQ_l(\overline{f_i}, \overline{f_j}) = |\mathcal{D}_l| - FQ_l(f_i) - FQ_l(f_j) + FQ_l(f_i, f_j) \tag{13}$$

The error for a branch can be calculated if the FQ_0- and FQ_1-value for that branch are known, by taking the minimum of these two values. By iterating over each (f_i, f_j)-pair, the tree with the smallest error can be found. This increases performance drastically, as the algorithm spends a lot of time constructing trees of this size and this method does not require any more actual splits to be made or recursions to be done. A drawback of this method is its heaviness in terms of space complexity, as storing these frequency counts requires a matrix of $\mathcal{O}(|\mathcal{F}|^2)$-size. However, if the number of features stays below for example 10^3, no problems will occur.

3.3 Similarity lower bounds

If a lower bound is known for the error of a certain tree, that lower bound can be used to determine whether it is still possible to improve on the smallest error known so far and prune the search if it is not. MurTree determines lower bounds for $T(\mathcal{D}_{new}, d, n)$ by comparing it to a previously calculated error $T(\mathcal{D}_{old}, d, n)$. Each instance can contribute at most 1 to the total error, making it possible to construct a lower bound for the error yet to be calculated, using the following formula (Demirović et al., 2022):

$$T(\mathcal{D}_{old}, d, n) - 1 \cdot |\mathcal{D}_{old} \setminus \mathcal{D}_{new}| \leq T(\mathcal{D}_{new}, d, n) \tag{14}$$

4 SurTree

In the coming sections, we will discuss the changes that are needed to develop a variant of MurTree for survival analysis, from now on referred to as *SurTree*. This is considered to be the main contribution of this research. Section 4.1 indicates how instances are represented to support the new task. In Section 4.2, the chosen objective function is analyzed, and technical conclusions about the algorithm are drawn from it. Section 4.3 describes how to adapt the terminal solver to fit the new objective function. In the end, Section 4.4 explains why the similarity lower bound does not translate to SurTree.

4.1 Instance representation

In MurTree, instances are defined as a combination of a feature vector and a binary label. In code, this label is not explicitly assigned as an integer stored together with the vector; instead, the vectors are bucketed per class, with a list per class containing all instances that have been assigned their label.

In SurTree, the binary label is replaced with a time and a censoring indicator. The instances are no longer bucketed and instead are stored unordered in one list.

Another important matter is the cumulative hazard function $\hat{\Lambda}(t)$. Once this function is estimated using Eq. (2), the output of this function for each instance is precomputed and stored within the instance itself. This precomputation is done to prevent redundant method calls and calculations.

4.2 Objective function

As stated in Section 2.2, the error of a tree T can be calculated using Eq. (5), Eq. (7) and Eq. (8). A major technical difference between MurTree's misclassification and SurTree's error is that the error is not defined as a natural number. This means that the data type of any variable related to the error of a tree must be converted from integers to floating-point numbers.

Since θ appears in a logarithm in Eq. (7), it is required that θ is strictly positive for any leaf node. Otherwise, the error would be an undefined number or negative infinity, neither option being applicable in calculations. When observing Eq. (5), it becomes clear that for θ to be positive, the following condition must hold:

$$0 < \sum_{(t_i,\delta_i,\mathbf{fv})\in\mathcal{D}} \delta_i$$

This means there must be at least one non-censored observation in each leaf node. For this reason, before a split is made, the algorithm first verifies whether both sides receive at least one instance for which $\delta_i = 1$. If that is not the case, no recursion takes place.

To encourage the algorithm to return less complex trees, a *complexity parameter* α can be used to provide a bias towards trees with fewer nodes. The objective function from Eq. (8) may then be extended by including a function Complexity(T), defined as the number of decision nodes in T, which is multiplied by α. This turns the objective function into Eq. (15).

$$\text{Objective}(\mathcal{D}, T) = \text{Error}(\mathcal{D}, T) + \alpha \cdot \text{Complexity}(T) \quad (15)$$

There are two important properties of this error function that allow it to be adapted to MurTree. Firstly, the error for each leaf node can be independently calculated, given only the instances that reach the node and nothing else; the error function is *separable*, which is one of the assumptions that MurTree relies on (Van der Linden et al., 2023). Secondly, the error of a tree is the sum of the errors of its children; this allows the errors to be easily merged, similarly to MurTree's misclassification score. These two properties indicate that this objective function is fit to be utilized for SurTree as well (Van der Linden et al., 2023).

4.3 SurTree's terminal solver

The terminal solver from MurTree provides an enormous increase in performance, and should therefore make a return in SurTree. Van der Linden et al. (2023) provides a format to generalize this solver, which is applied in this section.

Due to the complexity of the error function and the fact that the error depends on the calculation of θ, more informative values than the frequency count need to be precomputed in order to obtain usable data. We can determine which values to precompute by considering the formula for θ in Eq. (5) and the error function in Eq. (7).

First, we split Eq. (7) into several summations:

$$\begin{aligned}\text{Error} &= \sum_i -\delta_i \log \hat{\Lambda}(t_i) + \sum_i -\delta_i \log\theta \\ &\quad + \sum_i -\delta_i + \sum_i \hat{\Lambda}(t_i)\theta \\ &= \sum_i -\delta_i \log \hat{\Lambda}(t_i) - \log\theta \sum_i \delta_i - \sum_i \delta_i + \theta \sum_i \hat{\Lambda}(t_i)\end{aligned}$$

Then, by substituting Eq. (5) into the above formula, we get the following:

$$\begin{aligned}\text{Error} &= \sum_i -\delta_i \log \hat{\Lambda}(t_i) - \log(\frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)}) \sum_i \delta_i \\ &\quad - \sum_i \delta_i + \frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)} \sum_i \hat{\Lambda}(t_i) \\ &= \sum_i -\delta_i \log \hat{\Lambda}(t_i) - \log(\frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)}) \sum_i \delta_i - \sum_i \delta_i + \sum_i \delta_i \\ &= \sum_i -\delta_i \log \hat{\Lambda}(t_i) - \log(\frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)}) \sum_i \delta_i \qquad (16)\end{aligned}$$

This redefinition of the error function is expressed as a function of a few sums, which can be precomputed in the same way the frequency counts FQ are in MurTree. There are three values that need to be calculated, which can be referred to as the *event sum* ES, the *hazard sum* HS, and the *negative log hazard sum* $NLHS$.

$$ES(f_i, f_j) = \sum_{(t_k,\delta_k,\mathbf{fv})\in\mathcal{D}(f_i,f_j)} \delta_k \qquad (17)$$

$$HS(f_i, f_j) = \sum_{(t_k,\delta_k,\mathbf{fv})\in\mathcal{D}(f_i,f_j)} \hat{\Lambda}(t_k) \qquad (18)$$

$$NLHS(f_i, f_j) = \sum_{(t_k,\delta_k,\mathbf{fv})\in\mathcal{D}(f_i,f_j)} -\delta_k \log \hat{\Lambda}(t_k) \qquad (19)$$

To calculate similar values for other branches, Eq. (10), Eq. (11), Eq. (12) and Eq. (13) may be used.

The definitions of the three values in Eq. (17), Eq. (18), and Eq. (19) can be substituted in Eq. (16), which leaves us with the following equation, used by SurTree to calculate the error:

$$\text{Error} = NLHS - ES \log(\frac{ES}{HS}) \qquad (20)$$

After this calculation, the same process from MurTree is applied to find the optimal decision trees with a maximum depth of 2.

4.4 Removal of the similarity lower bound

The similarity lower bound as it is implemented in MurTree relies on the assumption that removing one instance can only decrease the error at a leaf node by at most 1. For SurTree's objective function, however, the maximum error is not limited to 1. Since the log-likelihood of a parameter is not bound between two values, the difference between two log-likelihoods is not bound below a certain number. Therefore, the similarity lower bound from MurTree cannot be transferred without giving up the guarantee of optimality. Consequently, this feature has been removed from SurTree.

5 Computational Evaluation

To test SurTree on a variety of aspects, experiments have been conducted on a collection of datasets. Section 5.1 describes how the experiments were prepared. Section 5.2 shows how SurTree improves on OST in optimizing the objective function. In Section 5.3, both algorithms are compared in runtime, showing both the strengths and weaknesses of SurTree. Finally, in Section 5.4, a new tree quality metric called *Harrell's C-index* is introduced, to demonstrate how optimizing the objective can also help in optimizing other metrics.

A summary of the experiment results can be found in Appendix B.

5.1 Setup

In order to test the performance of SurTree, datasets are needed. Even though many datasets that are based on real trials are publicly available, the decision has been made to test the algorithm on *synthetically generated* datasets instead. This is to better test the scalability of SurTree with regard to certain adjustable parameters. These parameters include the number of instances n, the number of binary features f, and the fraction of censored instances c.

To generate the dataset, the following procedure was followed:

1. Generate n feature vectors of size f such that each feature has a 50% probability of being 1, regardless of other features.
2. Randomly generate a survival tree T of depth 5 with 2^5 leaf nodes that splits on random features, and assign a random distribution to each leaf node (see Appendix A for a list of used distributions).
3. For each of the n instances, classify the instance using the tree and assign it a random time-to-event t_i by sampling from the corresponding leaf distribution. After that, assign the instance a random value u_i, uniformly distributed between 0 and 1.
4. Choose the lowest value for k such that for at most $c \cdot 100\%$ of the observations it holds that $k(1 - u_i^2) < t_i$.
5. For each observation for which $k(1 - u_i^2) < t_i$, set $t_i = k(1 - u_i^2)$ and $\delta_i = 0$. For every other observation, leave t_i and set $\delta_i = 1$.

Five datasets were generated for each combination of the parameters $n \in \{100, 500, 1000, 5000, 10000\}$, $f \in \{10, 50, 100\}$ and $c \in \{0.1, 0.5, 0.8\}$. The generated datasets were run on both SurTree and OST for different maximum depths $d \in \{2, 3, 4, 5\}$, without any limit being posed on the number of nodes specifically, implicitly allowing $2^d - 1$ decision nodes. This makes for a total of 900 experiments. The code used to generate the datasets can be found in SurTree's repository[2].

Both algorithms have been tested on the same personal computer. The complexity parameter α has been set to 0, as this parameter seems to be interpreted differently by MurTree and OST, making a fair comparison difficult.

[2]See footnote 1.

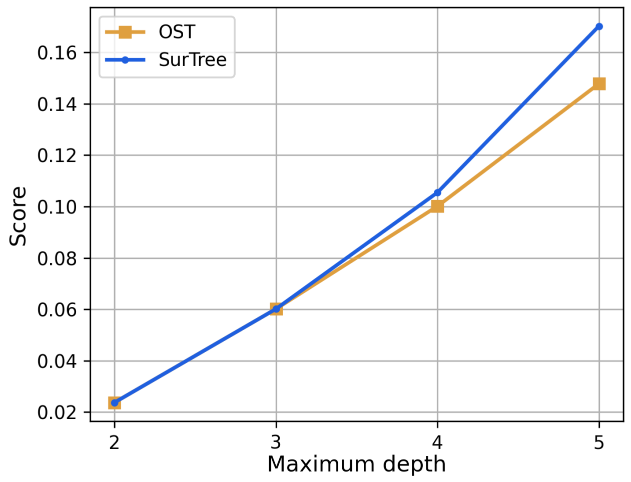

Figure 3: The objective score of SurTree and OST over the maximum tree depth for one particular dataset. n is set to 5000, f to 50 and c to 0.5.

For SurTree, a time limit of 10 minutes was enforced for each run. To save on time, whenever a test on a particular dataset timed out on a certain depth, all tests that on paper are at least as hard to compute were skipped. Formally, if a test with settings (n_i, f_i, c_i, d_i) times out, then any later test with settings (n_j, f_j, c_j, d_j) such that $n_i \leq n_j \wedge f_i \leq f_j \wedge c_i = c_j \wedge d_i \leq d_j$ was skipped and assumed to time out as well.

For OST, a total of 100 starting positions were used for each test. As all parameters were already defined from the start, no parameter tuning has been done during runtime.

5.2 Objective score

The objective function as defined in Eq. (7) can be any number from 0 up to infinity. To normalize this, we will compare the error of a tree T to the error of a tree without any decision nodes, referred to as T_0. T can then be assigned a score using the formula in Eq. (21). Under the assumption that adding new splits to T_0 will never increase the error, this score is limited between 0 and 1.

$$\text{Score}(\mathcal{D}, T) = 1 - \frac{\text{Error}(\mathcal{D}, T)}{\text{Error}(\mathcal{D}, T_0)} \tag{21}$$

Since SurTree explicitly searches for global optima of $\text{Error}(\mathcal{D}, T)$, whereas OST searches for local optima, the expectation is that SurTree's score is never less than OST's score. The experiments that were run back up this claim, as SurTree always seems to equal or outperform OST with regard to the objective function. In Fig. 3 an example of the difference between the scores is visualized. For lower maximum depths, generally up to 2 or 3, the algorithms tend to find the same tree with naturally also the same score. As the maximum depth increases, however, SurTree starts surpassing OST in score, as OST cannot easily find the best tree any longer.

5.3 Runtime

The difference in approach between SurTree and OST becomes even more apparent once their runtime is observed. As

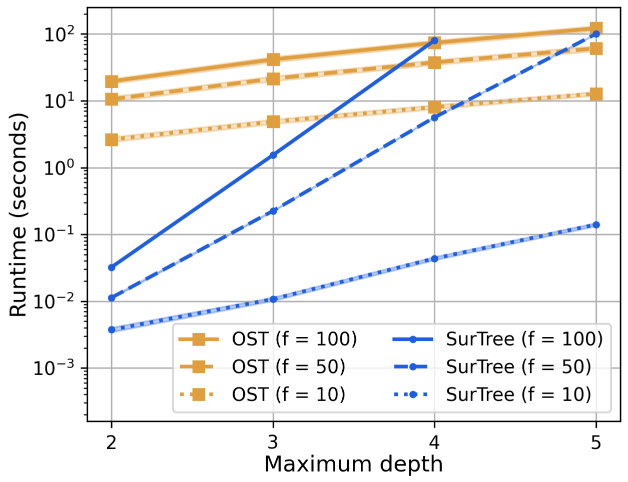

Figure 4: The average runtime of SurTree and OST over the maximum tree depth. n is set to 5000, the average over each value for c is taken and f varies per curve. One data point is missing, indicating a time-out. The 95% confidence intervals, though thin, are included.

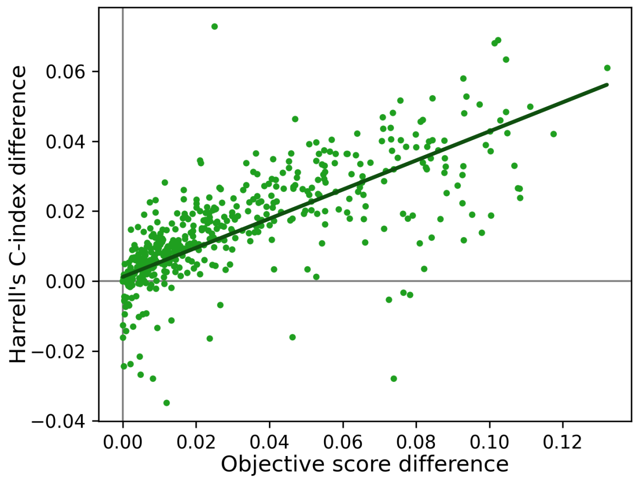

Figure 5: SurTree's difference in Harrell's C-index in comparison to OST, plotted over its improvement on the objective score.

seen in Fig. 4, SurTree is faster for lower maximum depths. As the maximum depth increases, SurTree's runtime seemingly grows exponentially, whereas OST's runtime appears to grow in a more linear fashion. As more features are added, OST starts to perform better than SurTree. From the 900 runs that were executed, SurTree completed 838 of them within the time limit of ten minutes.

With regards to the number of instances n, SurTree and OST both scale almost linearly. However, SurTree's runtime seems to gradually *flatten*, whereas OST's runtime grows *steeper* over n.

It needs to be mentioned that a problem was encountered in the runtime analysis of OST. In some occasions, a delay of 10 seconds would occur before the algorithm would start training. This problem was identified by sudden peaks in the runtime data and the fact that the progress bar, which usually appears immediately on screen, would sometimes only appear after 10 seconds. OST is closed-source, which made it impossible for us to remove the delay or exclude it from the timing.

Instead, this problem was fixed by trying to identify when a delay happened in a test. Since the length of the delay was very consistent in each occurrence, we could detect a delay by reading the console output 3 seconds after initiating the training process. If no progress bar was printed after 3 seconds, the delayed test would be rerun. Anyone trying to replicate the runtime results should be aware of this problem and our solution.

5.4 Harrell's C-index

Harrell's C-index (Harrell et al., 1982) is another metric that can be used to evaluate survival trees. The principle behind this statistic is that if an instance i is known to die earlier than an instance j (meaning $t_i < t_j$), then it should also be classified with a higher risk ($\theta_i > \theta_j$). If this is the case for a certain pair of instances, this pair is called *concordant*. If instead $\theta_i < \theta_j$, then the pair is called *discordant*. In the case that $\theta_i = \theta_j$, the pair is said to have a *tied risk*.

Since some observations are censored, and therefore the time-to-event might not be known, we can only make statements about pairs for which the instance with an *earlier time* is not censored. This way, no matter whether the later instance is censored or not, its event cannot have occurred earlier than the other event.

The number of concordant, discordant, and tied-risk pairs can be calculated using the following formulas respectively:

$$CC = \sum_{i,j} \mathbb{1}(t_i < t_j)\mathbb{1}(\theta_i > \theta_j)\delta_i \tag{22}$$

$$DC = \sum_{i,j} \mathbb{1}(t_i < t_j)\mathbb{1}(\theta_i < \theta_j)\delta_i \tag{23}$$

$$TR = \sum_{i,j} \mathbb{1}(t_i < t_j)\mathbb{1}(\theta_i = \theta_j)\delta_i \tag{24}$$

Harrell's C-index is computed using the following formula:

$$H_C = \frac{CC + 0.5 \cdot TR}{CC + TR + DC} \tag{25}$$

This value can range between 0 and 1, with higher values being preferred. Without any splits, the C-index will equal 0.5.

For each of the 838 runs that SurTree completed, the C-index was calculated. As can be seen in Fig. 5, an improvement from SurTree on OST in the objective score correlates positively with an improvement in the C-index. Such an improvement is not guaranteed, as SurTree produced 49 trees with a worse C-index than the trees produced by OST for the same settings. However, in the other 426 cases where the SurTree's objective score was higher, so was its C-index. The average difference in Harrell's C-index between SurTree and OST was 0.0087.

6 Responsible Research

To ensure that this research is conducted and published responsibly, we need to take a moment to reflect on some important matters. These matters include ethical implications of this research, reproducibility of the experiment results, and possible biases in the data.

6.1 Ethical implications

The output of this research is a method to compute globally optimal survival trees. The objective function is a numerical metric, which is the only thing the algorithm tries to optimize. The algorithm is therefore unbiased with regard to how to split instances. It only chooses a split if it improves the objective score and for no other reason.

We acknowledge that a particular feature that has historically been discriminated against, such as gender or race, might be picked by the algorithm as a significant splitting feature. A person using the algorithm could draw conclusions from the output that reinforce this discrimination. However, the opposite might also happen; it could bring certain types of discrimination to light, allowing users to take action against it. The actual effect of the algorithm on this matter is therefore mostly dependent on the end user.

6.2 Reproducibility

The source code for SurTree is publicly available as a branch in the MurTree repository. The exact code used to generate the test datasets is included in this repository, allowing anyone to perform the experiments in Section 5 themselves.

6.3 Possible biases

It is important to acknowledge that biases with regard to runtime may have occurred during the evaluation process. For example, due to the difference in parameters that can be set for both algorithms, it is possible that one or both algorithms are not running optimally, affecting runtime. This can skew the comparison towards one side unjustly. Alternatively, since the evaluation has been done on a personal computer instead of a higher-end computer, the runtime performance might differ when the experiment is rerun by other researchers.

Furthermore, the datasets that were generated for the experiment provide some variety in a handful of parameters, but still have some commonalities. For example, each feature always has a probability of 50% to be true, and there is no dependence between features. This means that on average, every decision node splits the dataset in half. This is an oversimplification of real-life data, which could have biases and dependence in its features.

7 Discussion

The experimental results show that the objective function value of SurTree's trees is always at least equal to that of OST's trees. This supports the claim that SurTree finds the absolutely optimal trees, with regard to the chosen objective. Another metric, Harrell's C-index, also tends to improve as the objective is improved. This improvement is, however, not guaranteed, as an increase in the objective score can lead to a decrease in the C-index. SurTree's approach, therefore, does not necessarily lead to optimality on all fronts.

Particularly for lower depths, the runtime of SurTree seems to be well under that of OST. If the maximum depth is set to at most 3, SurTree can finish in one-tenth of the time OST does, despite the fact that it optimizes the objective even more. This significant difference in performance can be attributed to a few properties. For example, while SurTree runs its algorithm only once, OST is by default set to run the same algorithm 100 times, causing many calculations to be done even if the best tree is already found. Another reason might be the fact that SurTree is implemented in C++ whereas OST is implemented in Julia, as code written in Julia tends to be a bit slower than code written in C++.

As the number of features or the maximum depth increases, SurTree starts to take significantly more time, whereas the time required by OST increases less. At 10 features, SurTree can still outperform OST for every depth up to 5. At 50 features, SurTree is generally faster than OST at a depth of 4, but at a depth of 5, it tends to be slower. At 100 features, either algorithm could be faster at a depth of 4 depending on the number of instances, but SurTree will almost certainly take longer than 10 minutes at a depth of 5 when the dataset includes more than 200 instances.

8 Conclusions and Future Work

In this research, we present SurTree, an extension of the MurTree-algorithm that constructs globally optimal survival trees, i.e. decision trees that split up survival data such that the error function as defined by LeBlanc and Crowley (1992) is minimized. Some notable changes include a new objective function, a new formula to be used in MurTree's terminal solver, and the removal of MurTree's similarity lower bound.

Through experiments, it has been verified that SurTree is successful in minimizing the objective function at least as much as Optimal Survival Trees (OST), a state-of-the-art method for creating survival trees. For cases with a low maximum depth or a small number of features, SurTree outperforms OST by a significant factor in terms of runtime. Since SurTree solves an NP-hard problem, runtime becomes a problem when the maximum depth increases. However, for reasonably-sized cases, SurTree can compete with the state-of-the-art.

In the future, further research can be done on adapting MurTree to optimize other metrics often seen in survival analysis. Furthermore, it could be beneficial to analyze how SurTree could be rewritten to work with non-binary features, such as categorical or continuous features.

References

Aalen, O. O. (1978). Nonparametric Inference for a Family of Counting Processes. *Annals of Statistics*, *6*(4).

Aglin, G., Nijssen, S., & Schaus, P. (2020). Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. *Proceedings of the ... AAAI Conference on Artificial Intelligence*, *34*(04), 3146–3153.

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning*, *106*(7), 1039–1082.

Bertsimas, D., Dunn, J., Gibson, E., & Orfanoudaki, A. (2022). Optimal survival trees. *Machine Learning*, *111*(8), 2951–3023.

Breiman, L. (1984). Classification and regression trees. *Biometrics*, *40*(3), 874.

Davis, R. J., & Anderson, J. M. (1989). Exponential survival trees. *Statistics in Medicine*, *8*(8), 947–961.

Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., & Stuckey, P. J. (2022). MurTree: Optimal Decision Trees via Dynamic Programming and Search. *HAL (Le Centre pour la Communication Scientifique Directe)*.

Graf, E., Schmoor, C., Sauerbrei, W., & Schumacher, M. (1999). Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, *18*(17-18), 2529–2545.

Harrell, F. E., Califf, R. M., Pryor, D. B., Lee, K. L., & Rosati, R. A. (1982). Evaluating the yield of medical tests. *JAMA*, *247*(18), 2543.

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, *5*(1), 15–17.

Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, *53*(282), 457–481.

LeBlanc, M., & Crowley, J. (1992). Relative Risk Trees for Censored Survival Data. *Biometrics*, *48*(2), 411.

LeBlanc, M., & Crowley, J. (1993). Survival trees by goodness of split. *Journal of the American Statistical Association*, *88*(422), 457–467.

Narodytska, N., Ignatiev, A., Pereira, F., & Marques-Silva, J. (2018). Learning optimal decision trees with sat.

Nelson, W. (1972). Theory and Applications of Hazard Plotting for Censored Failure Data. *Technometrics*, *14*(4), 945–966.

Nijssen, S., & Fromont, E. (2007). Mining optimal decision trees from itemset lattices. *HAL (Le Centre pour la Communication Scientifique Directe)*.

Su, X., & Fan, J. (2004). Multivariate survival trees: A maximum likelihood approach based on frailty models. *Biometrics*, *60*(1), 93–99.

Van der Linden, J. G. M., de Weerdt, M. M., & Demirović, E. (2023). Optimal decision trees for separable objectives: Pushing the limits of dynamic programming.

Zhu, H., Murali, P., Phan, D. T., Nguyen, L. H., & Kalagnanam, J. R. (2020). A scalable mip-based method for learning optimal multivariate decision trees. *33*, 1771–1781.

A Distributions in Ground Truth Trees

To determine the time-to-event for each instance during the dataset generation, a ground truth tree is generated for each dataset. Each leaf node is assigned a random distribution from the list below. Each option has an equal probability of being assigned.

- Exponential(λ), with $\lambda \in \{0.3, 0.4, 0.6, 0.8, 0.9, 1.15, 1.5, 1.8\}$
- Weibull(k, λ), with $(k, \lambda) \in \{(0.8, 0.4), (0.9, 0.5), (0.9, 0.7), (0.9, 1.1), (0.9, 1.5), (1.0, 1.1), (1.0, 1.9), (1.3, 0.5)\}$
- Lognormal(μ, σ^2), with $(\mu, \sigma^2) \in \{(0.1, 1), (0.2, 0.75), (0.3, 0.3), (0.3, 0.5), (0.3, 0.8), (0.4, 0.32), (0.5, 0.3), (0.5, 0.7)\}$
- Gamma(k, θ), with $(k, \theta) \in \{(0.2, 0.75), (0.3, 1.3), (0.3, 2.0), (0.5, 1.5), (0.8, 1.0), (0.9, 1.3), (1.3, 0.9), (1.5, 0.7)\}$

B Experiment Results

A summary is given in Table 1 describing the results of the experiment that was conducted on the synthetically generated datasets.

n is the number of instances in the dataset, f is the number of features per instance, c is the fraction of censored data, and d is the maximum depth of the generated tree. Five datasets were generated for each setting. For both SurTree and OST, the median runtime is given in seconds, together with the first and third quantiles within the parentheses. The averages are given for both the objective score and Harrell's C-index.

Note that if a cell reads "$>$ 600", this means that *at least one* of the five runs timed out, not necessarily *all* five runs.

Table 1: A summary of the results obtained from the experiment.

				Runtime				Score		Harrell's C	
n	f	c	d	OST		SurTree		OST	SurTree	OST	SurTree
100	10	0.1	2	.1060	(.0965, .1080)	.0001	(.0001, .0010)	.1206	.1206	.6213	.6213
			3	.1530	(.1510, .1665)	.0001	(.0001, .0005)	.2555	.2561	.6862	.6847
			4	.2410	(.2320, .2420)	.0030	(.0030, .0035)	.4689	.5043	.7642	.7831
			5	.3450	(.3320, .3625)	.0120	(.0115, .0130)	.7107	.7831	.8470	.8741
		0.5	2	.0920	(.0865, .0955)	.0001	(.0001, .0005)	.1086	.1086	.6613	.6565
			3	.1420	(.1365, .1445)	.0001	(.0001, .0010)	.2453	.2457	.7451	.7403
			4	.2090	(.2080, .2180)	.0030	(.0025, .0030)	.4440	.4839	.8139	.8350
			5	.2870	(.2755, .3020)	.0130	(.0130, .0140)	.5734	.6387	.8668	.8910
		0.8	2	.0840	(.0760, .0915)	.0001	(.0001, .0005)	.1320	.1320	.7209	.7209
			3	.1200	(.1165, .1280)	.0001	(.0001, .0010)	.2447	.2463	.7799	.7808
			4	.1580	(.1525, .1605)	.0020	(.0020, .0030)	.3332	.3690	.8068	.8123
			5	.1990	(.1915, .2060)	.0120	(.0100, .0120)	.4287	.4828	.8272	.8440
	50	0.1	2	.2670	(.2665, .2755)	.0010	(.0010, .0010)	.1667	.1667	.6460	.6460
			3	.4980	(.4960, .5200)	.0200	(.0195, .0205)	.3663	.3799	.7314	.7400
			4	.8100	(.8075, .8535)	.6270	(.6165, .6410)	.6274	.7086	.8217	.8504
			5	1.239	(1.187, 1.246)	13.87	(13.12, 14.18)	.8418	.9283	.9042	.9413
		0.5	2	.2390	(.2380, .2650)	.0010	(.0010, .0010)	.1685	.1685	.6779	.6779
			3	.4150	(.4030, .4495)	.0200	(.0185, .0205)	.3595	.3776	.7795	.7790
			4	.6830	(.6535, .7190)	.6380	(.6285, .6540)	.5589	.6472	.8631	.8888
			5	.9440	(.9235, .9680)	16.88	(16.56, 17.41)	.6855	.7814	.9080	.9501
		0.8	2	.2040	(.1955, .2160)	.0010	(.0001, .0010)	.2236	.2236	.7572	.7572
			3	.3390	(.3335, .3490)	.0180	(.0165, .0190)	.3748	.3926	.8508	.8474
			4	.5000	(.4830, .5245)	.5850	(.5785, .6390)	.5061	.5798	.8710	.8941
			5	.5770	(.5740, .5890)	11.94	(11.71, 13.37)	.5593	.6415	.9010	.9122
	100	0.1	2	.5290	(.4955, .5345)	.0020	(.0020, .0025)	.2204	.2204	.6662	.6662
			3	.9120	(.8810, .9605)	.1310	(.1300, .1405)	.4108	.4351	.7532	.7518
			4	1.546	(1.512, 1.554)	8.863	(8.733, 10.05)	.6550	.7532	.8262	.8589
			5	2.446	(2.305, 2.462)	418.6	(405.1, 434.9)	.8728	.9374	.9167	.9494
		0.5	2	.4310	(.3945, .4350)	.0020	(.0020, .0030)	.1898	.1915	.6810	.6856
			3	.7380	(.7265, .7585)	.1300	(.1280, .1400)	.3953	.4222	.7972	.8012
			4	1.252	(1.202, 1.257)	9.585	(9.313, 10.08)	.6141	.7076	.8709	.9063
			5	1.816	(1.776, 1.882)	$>$ *600*		.7394	-	.9230	-

Table 1: A summary of the results obtained from the experiment.

				Runtime		Score		Harrell's C	
n	f	c	d	OST	SurTree	OST	SurTree	OST	SurTree
		0.8	2	.3450 (.3270, .3815)	.0020 (.0020, .0030)	.2709	.2709	.7819	.7819
			3	.6180 (.5980, .6240)	.1340 (.1300, .1470)	.4263	.4593	.8406	.8705
			4	.9130 (.8620, .9255)	9.501 (9.149, 9.619)	.5651	.6399	.8956	.9127
			5	1.066 (1.036, 1.075)	389.2 (377.3, 404.0)	.6076	.6796	.8988	.9308
500	10	0.1	2	.3510 (.3345, .3685)	.0010 (.0001, .0010)	.0408	.0408	.5755	.5755
			3	.5940 (.5825, .6275)	.0010 (.0010, .0020)	.0877	.0901	.6165	.6153
			4	.9650 (.9495, .9910)	.0060 (.0055, .0065)	.1574	.1659	.6566	.6650
			5	1.440 (1.423, 1.537)	.0230 (.0225, .0260)	.2604	.2963	.6997	.7144
		0.5	2	.2860 (.2740, .3140)	.0010 (.0005, .0010)	.0487	.0487	.6123	.6123
			3	.5130 (.4965, .5225)	.0020 (.0010, .0020)	.1068	.1082	.6692	.6702
			4	.8450 (.8210, .8760)	.0060 (.0050, .0065)	.1836	.1981	.7230	.7341
			5	1.243 (1.228, 1.306)	.0210 (.0205, .0225)	.2747	.3194	.7630	.7832
		0.8	2	.2660 (.2530, .3995)	.0001 (.0001, .0010)	.0536	.0536	.6367	.6335
			3	.4180 (.4085, .4305)	.0020 (.0010, .0020)	.1088	.1088	.7033	.7033
			4	.6650 (.6560, .7095)	.0050 (.0050, .0065)	.1775	.1945	.7582	.7666
			5	1.010 (1.006, 1.024)	.0190 (.0185, .0195)	.2472	.2831	.7977	.8163
	50	0.1	2	1.585 (1.208, 1.605)	.0020 (.0015, .0020)	.0411	.0411	.5714	.5714
			3	2.844 (2.244, 3.097)	.0360 (.0355, .0375)	.1043	.1078	.6187	.6206
			4	4.132 (3.775, 4.492)	1.105 (1.098, 1.133)	.1812	.2299	.6619	.6873
			5	6.652 (5.843, 8.280)	26.45 (26.22, 27.36)	.3125	.4048	.7144	.7583
		0.5	2	1.414 (1.357, 1.771)	.0020 (.0010, .0020)	.0477	.0477	.6091	.6091
			3	2.508 (2.296, 2.986)	.0350 (.0340, .0355)	.0976	.1034	.6547	.6608
			4	4.334 (4.030, 4.849)	1.115 (1.095, 1.116)	.1710	.2030	.7011	.7254
			5	6.734 (5.824, 7.468)	26.56 (26.29, 26.63)	.2961	.3758	.7629	.7974
		0.8	2	1.101 (1.088, 1.169)	.0020 (.0010, .0020)	.0866	.0866	.6761	.6761
			3	2.105 (1.985, 2.317)	.0340 (.0335, .0340)	.1610	.1656	.7432	.7466
			4	3.306 (3.270, 3.788)	1.060 (1.056, 1.078)	.2574	.2862	.8094	.8222
			5	4.952 (4.751, 5.175)	25.42 (25.02, 25.56)	.3324	.4186	.8366	.8677
	100	0.1	2	2.595 (2.220, 2.961)	.0050 (.0050, .0055)	.0517	.0517	.5800	.5800
			3	4.435 (4.260, 4.956)	.2550 (.2515, .2585)	.1082	.1133	.6217	.6277
			4	7.135 (6.693, 7.312)	16.70 (16.58, 16.88)	.1881	.2264	.6689	.6875
			5	11.05 (10.84, 11.72)	$>$ *600*	.3254	-	.7232	-
		0.5	2	1.957 (1.865, 2.019)	.0050 (.0050, .0050)	.0587	.0587	.6196	.6196
			3	3.866 (3.562, 4.297)	.2490 (.2450, .2495)	.1174	.1225	.6690	.6769
			4	5.909 (5.527, 6.266)	16.39 (16.16, 16.60)	.2178	.2459	.7228	.7457
			5	9.408 (9.236, 9.453)	$>$ *600*	.3372	-	.7813	-
		0.8	2	1.563 (1.488, 1.601)	.0050 (.0040, .0050)	.0683	.0692	.6579	.6536
			3	2.796 (2.775, 3.007)	.2450 (.2405, .2470)	.1461	.1560	.7371	.7353
			4	4.784 (4.583, 4.854)	15.61 (15.40, 15.97)	.2441	.2847	.7957	.8189
			5	7.570 (7.255, 7.709)	$>$ *600*	.3458	-	.8444	-
1000	10	0.1	2	.6640 (.6235, .7130)	.0010 (.0001, .0015)	.0469	.0469	.5938	.5938
			3	1.132 (1.113, 1.248)	.0030 (.0020, .0030)	.0862	.0863	.6287	.6275
			4	1.925 (1.811, 1.982)	.0100 (.0100, .0115)	.1468	.1562	.6560	.6660
			5	2.927 (2.816, 2.981)	.0370 (.0355, .0380)	.2280	.2578	.6950	.7113
		0.5	2	.5630 (.5055, .5850)	.0010 (.0005, .0010)	.0442	.0442	.6034	.6034
			3	.9970 (.9080, 1.093)	.0030 (.0025, .0030)	.0987	.0987	.6599	.6599
			4	1.613 (1.536, 1.859)	.0100 (.0100, .0110)	.1683	.1732	.7089	.7139
			5	2.417 (2.404, 2.614)	.0340 (.0330, .0345)	.2383	.2555	.7454	.7564
		0.8	2	.4860 (.4375, .5165)	.0010 (.0005, .0010)	.0487	.0487	.6368	.6368

Table 1: A summary of the results obtained from the experiment.

				Runtime		Score		Harrell's C	
n	f	c	d	OST	SurTree	OST	SurTree	OST	SurTree
			3	.8970 (.8525, 1.035)	.0020 (.0020, .0030)	.1002	.1003	.7012	.7004
			4	1.345 (1.329, 1.379)	.0080 (.0080, .0095)	.1711	.1843	.7594	.7693
			5	2.101 (2.032, 2.157)	.0290 (.0290, .0300)	.2357	.2603	.7984	.8141
	50	0.1	2	2.725 (2.473, 2.942)	.0020 (.0020, .0030)	.0360	.0360	.5771	.5771
			3	4.847 (4.684, 4.963)	.0560 (.0545, .0570)	.0799	.0799	.6177	.6152
			4	7.502 (7.343, 7.639)	1.630 (1.603, 1.675)	.1345	.1506	.6505	.6573
			5	11.42 (11.09, 11.61)	35.50 (35.18, 35.84)	.2086	.2716	.6751	.7123
		0.5	2	2.223 (2.102, 2.299)	.0020 (.0020, .0030)	.0456	.0456	.6127	.6127
			3	3.974 (3.829, 4.249)	.0560 (.0540, .0575)	.0811	.0816	.6547	.6560
			4	6.468 (6.316, 6.901)	1.602 (1.573, 1.637)	.1351	.1583	.6929	.7079
			5	9.424 (9.318, 9.644)	34.86 (33.94, 35.22)	.2097	.2810	.7262	.7687
		0.8	2	1.808 (1.591, 1.851)	.0020 (.0020, .0030)	.0483	.0483	.6481	.6481
			3	3.291 (3.200, 3.470)	.0540 (.0530, .0550)	.1006	.1007	.7067	.7048
			4	5.264 (5.167, 5.585)	1.513 (1.501, 1.560)	.1838	.2055	.7712	.7879
			5	7.626 (7.394, 8.044)	33.35 (32.71, 33.40)	.2573	.3158	.8081	.8374
	100	0.1	2	4.923 (4.672, 5.364)	.0080 (.0075, .0085)	.0414	.0414	.5800	.5800
			3	9.726 (9.236, 10.09)	.3880 (.3855, .3920)	.0840	.0851	.6185	.6189
			4	14.37 (14.01, 14.84)	23.98 (23.90, 24.65)	.1380	.1603	.6480	.6627
			5	21.63 (20.90, 21.87)	$>$ *600*	.2063	-	.6749	-
		0.5	2	4.176 (3.870, 4.279)	.0080 (.0075, .0085)	.0418	.0418	.6022	.6022
			3	7.652 (7.582, 8.178)	.3850 (.3815, .3950)	.0934	.0942	.6623	.6613
			4	12.33 (11.64, 13.21)	23.31 (23.15, 23.43)	.1646	.1879	.7054	.7217
			5	18.07 (17.65, 18.54)	$>$ *600*	.2363	-	.7353	-
		0.8	2	2.981 (2.891, 3.174)	.0100 (.0075, .0105)	.0345	.0345	.6113	.6113
			3	5.650 (5.371, 6.117)	.3890 (.3785, .3910)	.0828	.0904	.6727	.6867
			4	9.168 (9.017, 9.247)	22.72 (22.67, 22.94)	.1452	.1813	.7338	.7594
			5	14.29 (14.07, 14.73)	$>$ *600*	.2267	-	.7816	-
5000	10	0.1	2	3.079 (2.991, 3.362)	.0040 (.0030, .0045)	.0337	.0337	.5709	.5709
			3	5.681 (5.575, 5.855)	.0120 (.0110, .0130)	.0698	.0698	.6116	.6116
			4	9.438 (9.274, 9.740)	.0470 (.0460, .0485)	.1333	.1427	.6547	.6599
			5	14.62 (14.48, 15.08)	.1560 (.1515, .1575)	.2006	.2376	.6830	.7024
		0.5	2	2.563 (2.397, 2.767)	.0040 (.0030, .0040)	.0268	.0268	.5856	.5856
			3	4.906 (4.710, 5.220)	.0110 (.0100, .0110)	.0609	.0619	.6337	.6345
			4	8.113 (7.530, 8.294)	.0440 (.0440, .0485)	.1102	.1186	.6759	.6828
			5	12.63 (12.46, 13.16)	.1450 (.1435, .1485)	.1804	.1982	.7226	.7358
		0.8	2	2.219 (2.049, 2.256)	.0040 (.0035, .0040)	.0433	.0433	.6295	.6295
			3	3.864 (3.789, 4.158)	.0100 (.0090, .0100)	.0905	.0906	.6912	.6883
			4	6.859 (6.582, 6.899)	.0380 (.0370, .0385)	.1398	.1446	.7372	.7415
			5	10.86 (10.75, 10.96)	.1200 (.1180, .1220)	.1860	.1998	.7685	.7785
	50	0.1	2	12.88 (12.42, 13.46)	.0110 (.0110, .0120)	.0206	.0206	.5623	.5623
			3	25.87 (25.52, 26.71)	.2250 (.2230, .2395)	.0506	.0506	.6065	.6065
			4	46.00 (44.29, 46.55)	5.695 (5.664, 5.814)	.1122	.1165	.6502	.6550
			5	73.73 (71.63, 75.86)	101.9 (99.11, 103.6)	.2069	.2317	.6910	.7037
		0.5	2	10.37 (9.542, 10.68)	.0110 (.0110, .0115)	.0263	.0263	.5848	.5848
			3	20.31 (19.49, 21.54)	.2200 (.2180, .2245)	.0685	.0689	.6412	.6413
			4	35.82 (34.61, 38.74)	5.622 (5.460, 5.732)	.1304	.1387	.6918	.6968
			5	60.61 (59.62, 62.58)	100.4 (99.49, 101.4)	.2045	.2382	.7295	.7527
		0.8	2	8.613 (8.241, 8.861)	.0110 (.0105, .0120)	.0254	.0254	.5993	.5993
			3	18.20 (17.14, 18.54)	.2230 (.2180, .2265)	.0608	.0608	.6562	.6562
			4	30.80 (29.54, 32.08)	5.620 (5.446, 5.686)	.1221	.1297	.7213	.7294

Table 1: A summary of the results obtained from the experiment.

				Runtime		Score		Harrell's C	
n	f	c	d	OST	SurTree	OST	SurTree	OST	SurTree
			5	47.93 (45.96, 50.16)	97.68 (97.40, 99.87)	.1735	.2042	.7565	.7798
	100	0.1	2	23.73 (22.65, 24.19)	.0320 (.0315, .0325)	.0365	.0376	.5797	.5810
			3	50.97 (49.00, 52.80)	1.526 (1.480, 1.544)	.0746	.0749	.6138	.6141
			4	88.77 (85.88, 93.16)	79.76 (78.39, 80.13)	.1383	.1475	.6545	.6610
			5	147.9 (142.8, 151.7)	$>$ *600*	.2160	-	.6876	-
		0.5	2	19.68 (18.53, 20.21)	.0320 (.0315, .0330)	.0380	.0380	.6000	.6000
			3	40.52 (40.22, 41.81)	1.563 (1.528, 1.599)	.0755	.0761	.6457	.6462
			4	71.92 (71.32, 74.00)	80.36 (80.29, 81.18)	.1422	.1524	.6988	.7071
			5	122.5 (117.7, 125.2)	$>$ *600*	.2398	-	.7524	-
		0.8	2	15.30 (15.24, 16.82)	.0320 (.0320, .0330)	.0388	.0388	.6309	.6309
			3	32.63 (32.50, 33.27)	1.546 (1.495, 1.573)	.0837	.0838	.6896	.6897
			4	60.44 (58.30, 60.58)	79.27 (78.73, 79.93)	.1550	.1654	.7478	.7572
			5	96.33 (93.30, 99.95)	$>$ *600*	.2416	-	.7981	-
10000	10	0.1	2	6.399 (6.109, 6.644)	.0080 (.0075, .0080)	.0338	.0338	.5764	.5764
			3	11.04 (10.77, 11.65)	.0220 (.0220, .0225)	.0607	.0611	.6067	.6082
			4	18.52 (18.27, 19.50)	.0970 (.0965, .1025)	.1140	.1186	.6540	.6545
			5	30.05 (29.15, 30.75)	.3210 (.3140, .3240)	.1817	.2136	.6814	.6992
		0.5	2	5.012 (4.964, 5.165)	.0080 (.0070, .0080)	.0281	.0281	.5891	.5891
			3	9.349 (9.287, 10.36)	.0220 (.0215, .0225)	.0689	.0696	.6426	.6430
			4	15.83 (15.46, 16.79)	.0930 (.0925, .0935)	.1165	.1289	.6864	.6953
			5	24.80 (24.42, 26.39)	.2950 (.2915, .3015)	.1772	.2075	.7198	.7412
		0.8	2	4.025 (3.973, 4.508)	.0070 (.0070, .0080)	.0388	.0388	.6237	.6237
			3	8.251 (7.778, 8.588)	.0180 (.0180, .0200)	.0799	.0804	.6808	.6826
			4	14.21 (13.65, 14.39)	.0760 (.0740, .0760)	.1423	.1565	.7331	.7460
			5	22.44 (21.07, 23.83)	.2330 (.2320, .2430)	.1983	.2218	.7685	.7857
	50	0.1	2	25.40 (24.07, 26.65)	.0220 (.0220, .0230)	.0271	.0271	.5750	.5750
			3	49.61 (48.90, 52.91)	.4350 (.4160, .4385)	.0553	.0553	.6092	.6092
			4	90.35 (86.24, 91.82)	10.89 (10.58, 11.05)	.1235	.1303	.6534	.6605
			5	151.3 (144.9, 154.0)	186.1 (182.5, 190.4)	.2002	.2194	.6903	.7028
		0.5	2	19.98 (19.21, 21.65)	.0220 (.0215, .0225)	.0337	.0337	.5980	.5980
			3	41.98 (40.63, 43.50)	.4320 (.4235, .4485)	.0688	.0706	.6448	.6485
			4	70.70 (68.73, 72.60)	10.87 (10.51, 11.18)	.1259	.1341	.6925	.7020
			5	123.0 (119.6, 131.5)	185.8 (185.2, 187.0)	.1838	.2136	.7256	.7466
		0.8	2	16.94 (15.93, 18.10)	.0220 (.0210, .0285)	.0441	.0441	.6294	.6294
			3	35.42 (33.83, 36.16)	.4270 (.4135, .4335)	.0843	.0843	.6822	.6822
			4	59.89 (59.48, 63.18)	10.35 (10.25, 10.43)	.1441	.1511	.7368	.7434
			5	101.7 (101.0, 103.0)	178.6 (176.2, 182.2)	.2017	.2260	.7767	.7939
	100	0.1	2	49.02 (47.81, 50.41)	.0650 (.0610, .0660)	.0273	.0273	.5633	.5633
			3	99.32 (96.33, 102.3)	2.963 (2.939, 3.038)	.0594	.0594	.6032	.6032
			4	172.3 (167.6, 176.6)	150.1 (148.4, 150.7)	.1084	.1220	.6435	.6528
			5	289.1 (283.4, 304.9)	$>$ *600*	.1742	-	.6759	-
		0.5	2	40.36 (38.64, 41.83)	.0620 (.0615, .0650)	.0356	.0356	.5929	.5929
			3	80.77 (79.08, 83.84)	2.903 (2.886, 2.958)	.0738	.0738	.6450	.6450
			4	142.9 (139.7, 144.9)	150.5 (148.5, 151.6)	.1296	.1410	.6948	.7051
			5	240.9 (236.9, 256.9)	$>$ *600*	.2121	-	.7500	-
		0.8	2	32.99 (29.07, 33.86)	.0630 (.0610, .0630)	.0325	.0325	.6161	.6161
			3	65.35 (62.77, 70.37)	3.003 (2.941, 3.031)	.0691	.0691	.6735	.6735
			4	116.8 (113.3, 123.1)	148.1 (147.1, 149.9)	.1229	.1315	.7235	.7321
			5	200.5 (195.6, 205.3)	$>$ *600*	.1767	-	.7628	-