Optimal Survival Trees: A Dynamic Programming Approach

Tim Huisman, Jacobus G. M. van der Linden, Emir Demirović

Delft University of Technology
T.J.Huisman-1@student.tudelft.nl, {J.G.M.vanderLinden, E.Demirovic}@tudelft.nl

Abstract

Survival analysis studies and predicts the time of death, or other singular unrepeated events, based on historical data, while the true time of death for some instances is unknown. Survival trees enable the discovery of complex nonlinear relations in a compact human comprehensible model, by recursively splitting the population and predicting a distinct survival distribution in each leaf node. We use dynamic programming to provide the first survival tree method with optimality guarantees, enabling the assessment of the optimality gap of heuristics. We improve the scalability of our method through a special algorithm for computing trees up to depth two. The experiments show that our method's run time even outperforms some heuristics for realistic cases while obtaining similar out-of-sample performance with the state-of-the-art.

Introduction

The aim of *survival analysis* is to study and predict the time until a singular unrepeated event occurs, such as death or mechanical failure. Applications include not only evaluating the effectiveness of medical treatment (Selvin 2008), but also, for example, recidivism risk estimation in criminology (Chung, Schmidt, and Witte 1991), fish migration analysis (Castro-Santos and Haro 2003) and studies on human migration and fertility (Eryurt and Koç 2012).

Survival analysis is challenging because the time of event of some instances is unknown, i.e., it is *censored*. This study focuses on the most common form of censoring: *right-censored* data, where the true time of the event is unknown, for example, because an instance survived beyond the end of the experiment record.

The application of survival analysis in the medical and other high-stake domains motivates the use of human-interpretable machine learning models, such as *decision trees* (Rudin 2019). A decision tree recursively partitions instances by their attributes into buckets, i.e., the leaves of the tree. In each of these leaf nodes, a survival curve can be computed, as shown, for example, in Fig. 1. The advantage of decision trees is that they can model complex nonparametric relations while remaining easy to understand (Freitas 2014; Carrizosa, Molero-Río, and Morales 2021). Davis and Anderson (1989) provided one of the first survival tree methods, by applying recursive splitting of censored survival data with an interface similar to CART (Breiman et al. 1984). Other similar greedy top-down induction heuristics were developed by LeBlanc and Crowley (1993), Su and Fan (2004), and Hothorn, Hornik, and Zeileis (2006), each applying different splitting techniques.

To improve the performance of survival trees, Bertsimas et al. (2022) recently proposed a local search method called Optimal Survival Trees (OST), based on the method proposed in (Dunn 2018; Bertsimas and Dunn 2019). Despite its name, OST does not provide a guarantee of global optimality, but iteratively finds local improvements to the tree structure and converges to a local optimum.

Trees that do provide a guarantee of global optimality over a training set for a given size limit are called optimal decision trees. Out-of-sample results for optimal decision trees for classification typically also show an improvement over heuristics (Bertsimas and Dunn 2017). Therefore, optimal decision tree methods can provide better performance, while producing smaller trees, which increases their interpretability (Piltaver et al. 2016). However, to the best of our knowledge, there are no globally optimal decision tree methods for survival analysis yet.

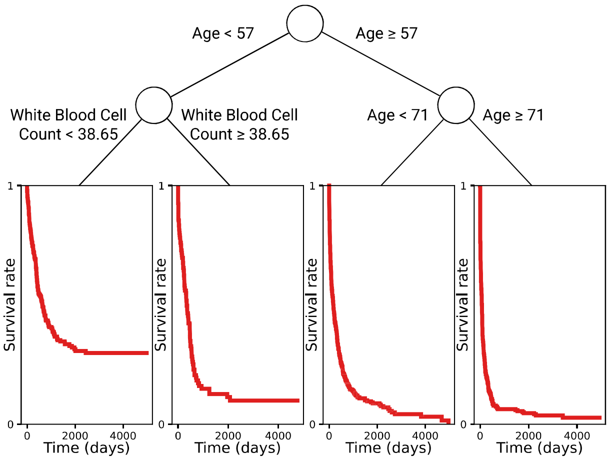

Figure 1: An example of a survival tree. Each leaf has a different survival distribution.

The challenge of finding optimal decision trees is scalability since it is an NP-Hard problem (Hyafil and Rivest 1976). Therefore, many optimal decision tree methods that optimize accuracy lack scalability. This includes methods based on mixed-integer programming (Bertsimas and Dunn 2017), constraint programming (Verhaeghe et al. 2020) and Satisfiability (Narodytska et al. 2018).

Better scalability results have been obtained by using dynamic programming (DP) because it directly exploits the recursive structure of the trees by treating subtrees as independent subproblems (Demirović et al. 2022). Van der Linden, De Weerdt, and Demirović (2023) show that these results also generalize beyond maximizing accuracy.

Our contributions are a first optimal survival tree algorithm called *SurTree*, based on a dynamic programming approach; an algorithm for trees of maximum depth two that greatly improves scalability; and a detailed experimental comparison of our new method with the local search method OST and the greedy heuristic CTree (Hothorn, Hornik, and Zeileis 2006). The first two contributions are inspired by the scalability improvements obtained for optimal classification trees with DP by Demirović et al. (2022). Our experiments show that SurTree's out-of-sample performance on average is better than CTree and similar to OST while outperforming OST in run time for realistic cases. Since SurTree is the first optimal method for survival trees, our method also helps assess the quality of heuristic solutions.

The following sections introduce related work and the preliminaries for our work. We then present our DP-based approach to optimal survival trees, evaluate it on synthetic and real data sets, and compare it with the state-of-the-art.

Related Work

Survival analysis Traditionally, many statistical approaches have been developed for dealing with censored data (Chung, Schmidt, and Witte 1991). This includes non-parametric approaches, such as the Kaplan-Meier method (Kaplan and Meier 1958) and the Nelson-Aalen estimator (Nelson 1972; Aalen 1978), semiparametric approaches, such as Cox proportional hazards regression (Cox 1972), and parametric approaches such as linear regression. Wang, Li, and Reddy (2019) provide an overview of the later use of machine learning for survival analysis, including survival trees, random survival forests (Ishwaran et al. 2008), support vector machines (Van Belle et al. 2011), and neural networks (Chi, Street, and Wolberg 2007).

Survival trees Since computing optimal decision trees is NP-Hard (Hyafil and Rivest 1976), traditionally most decision tree methods use *greedy top-down induction* based on some *splitting criterion*, such as Gini impurity or information gain (Breiman et al. 1984; Quinlan 1993). For survival trees, many such splitting criteria have been proposed. They can be divided into criteria that promote within-node homogeneity or between-node heterogeneity. Examples of the former are (Gordon and Olshen 1985; Davis and Anderson 1989; Therneau, Grambsch, and Fleming 1990) and (LeBlanc and Crowley 1992). Examples of the latter are (Ciampi et al. 1986; Segal 1988) and (LeBlanc and Crowley 1993). Other developments are presented by Molinaro, Dudoit, and Van der Laan (2004), who propose to weigh the uncensored data based on inverse-propensity weighting; Su and Fan (2004), who consider survival analysis for clustered events using a maximum likelihood approach based on frailty models; and Hothorn, Hornik, and Zeileis (2006), who introduce χ^2 tests as stopping criterion to prevent variable selection bias.

Recently, Bertsimas et al. (2022) presented OST (optimal survival trees), based on the coordinate-descent method proposed by Dunn (2018), by iteratively improving one node in the tree until a local optimum is reached. Because the problem is non-convex, they repeat this process several times to increase the probability of finding a good tree. Their results show that local search can outperform greedy heuristics such as (Therneau, Grambsch, and Fleming 1990) and (Hothorn, Hornik, and Zeileis 2006). However, despite its name, OST provides no guarantee of converging to the global optimum.

Optimal decision trees A popular approach for computing optimal decision trees is the use of general-purpose solvers. Bertsimas and Dunn (2017) and Verwer and Zhang (2017) showed how optimizing decision trees can be formulated as a mixed-integer programming (MIP) formulation. These MIP formulations were later improved by several others (Verwer and Zhang 2019; Zhu et al. 2020; Aghaei, Gómez, and Vayanos 2021). Verhaeghe et al. (2020) showed how constraint programming can be used to optimize trees. Narodytska et al. (2018) and Janota and Morgado (2020) presented a Satisfiability (SAT) approach for finding a perfect minimal tree. Hu et al. (2020) developed a maximum Satisfiability (MaxSAT) approach, while Shati, Cohen, and McIlraith (2021) extended the use of MaxSAT beyond binary predicates. However, as of yet, each of these approaches struggles to scale beyond small data sets and tree-size limits.

Recently, major improvements in scalability have been obtained using a dynamic programming (DP) approach, which as a result often outperforms the MIP, CP and (Max)SAT approaches by several orders of magnitude (Aglin, Nijssen, and Schaus 2020a; Demirović et al. 2022; Van der Linden, De Weerdt, and Demirović 2023). Nijssen and Fromont (2007) were one of the first to propose the use of DP for optimizing decision trees. Nijssen and Fromont (2010) showed how DP can also be used to optimize other objectives than accuracy, provided the objective is additive. Hu, Rudin, and Seltzer (2019); Lin et al. (2020); Aglin, Nijssen, and Schaus (2020a,b) and Demirović et al. (2022) improved the DP approach by introducing branching and bounding, new forms of caching, better bounds, and special procedures for trees of depth two. Van der Linden, De Weerdt, and Demirović (2023) prove that this DP approach can be applied to any separable optimization task, i.e., an optimization problem that can be independently solved for subtrees.

Since we do not know of any optimal survival tree method, and given the success of DP methods mentioned above, this study explores the use of DP for survival trees and the benefit of globally optimizing survival trees.

Preliminaries

Definitions and notation We aim to optimize a survival tree over a data set $\mathcal{D}$. This data set consists of instances that either experienced the event of interest or were censored. Each of these instances is described by a set of features. The following defines each of these terms:

An *event of interest*, or simply *event* or *death*, is the non-repeatable event for which the time until occurrence is measured within a trial. The *time-to-event* is the amount of time leading up to the event of interest from the beginning of the observation. In this study, we consider *right-censored* data, which means that for some instances the exact time-to-event is unknown, but lower bounded by some known time, for example, because a patient left the trial before the event of interest could be observed.

A data set $\mathcal{D}$, or a trial, consists of a set of *instances* $(t_i, \delta_i, \mathbf{fv}_i)$, each described by a feature vector $\mathbf{fv}_i$, a *censoring indicator* $\delta_i \in \{0, 1\}$ stating whether the event of interest was observed and a *time* $t_i > 0$. In case of censoring $(\delta_i = 0)$, t_i denotes the last time of observation. Otherwise, t_i denotes the time-to-event. The feature vector describes the instance by a set of features $\mathcal{F}$. Our method assumes that all features are binarized beforehand such that each feature is a binary predicate. We write $f \in \mathbf{fv}_i$ if the predicate holds for instance i or $\bar{f} \in \mathbf{fv}_i$ if it does not hold. We use $\mathcal{D}(f_i)$ and $\mathcal{D}(\bar{f}_i)$ to refer to all instances in $\mathcal{D}$ for which the predicate f_i is valid or not, respectively. Multiple feature splits can be stacked so that, for example, $\mathcal{D}(f_1, \bar{f}_2)$ refers to all instances for which f_1 holds and f_2 does not hold.

A *decision tree* partitions instances based on their features. We consider *binary* trees, where each node is either a *decision node* with two children, or a *leaf node*. Each decision node splits the data set on a certain feature. Each leaf node assigns a label to every instance that ends up at that leaf node. A *survival tree* (see Fig. 1) is a special type of decision tree that assigns in each leaf node not just a label, but a survival distribution that describes the survival odds after a certain amount of time.

Survival analysis background The goal of survival analysis is to accurately describe the *survival function*, which gives the probability of survival after a time t, denoted as $S(t) = P(T \geq t)$, with T the true time of the event (Wang, Li, and Reddy 2019). Its opposite is the cumulative death distribution function $F(t) = 1 - S(t)$, with its derivative, the death density function $f(t) = \frac{d}{dt}F(t)$.

One of the most used estimators for the survival function is the Kaplan-Meier estimator (Kaplan and Meier 1958):

$$\hat{S}(t) = \prod_{t' \leq t} \left(1 - \frac{d(t')}{n(t')}\right) \tag{1}$$

with $d(t')$ the number of deaths at time t' and $n(t')$ the number of survivors up and until time t':

$$d(t) = |\{(t_i, \delta_i, \mathbf{fv}_i) \in \mathcal{D} \mid t_i = t \wedge \delta_i = 1\}| \tag{2}$$

$$n(t) = |\{(t_i, \delta_i, \mathbf{fv}_i) \in \mathcal{D} \mid t_i \geq t\}| \tag{3}$$

The *hazard function* (also known by the force of mortality, the instantaneous death rate, or the conditional failure rate), given by $\lambda(t) = \frac{f(t)}{S(t)}$, indicates the frequency (or rate) of the event of interest happening at time t, provided that it has not happened before time t yet (Dunn and Clark 2009). Alternatively, it can be written as $\lambda(t) = -\frac{d}{dt} \ln S(t)$. The *cumulative hazard function* is the integral over the hazard function $\Lambda(t) = \int_0^t \lambda(u) du$, and thus the survival function $S(t)$ can be rewritten as:

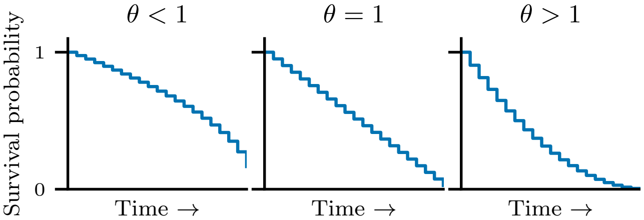

Figure 2: A visualization of how θ affects a survival distribution $\hat{S}(t)$. Every plot uses the same $\hat{\Lambda}(t)$, but use $\theta = 0.5$, $\theta = 1$ and $\theta = 2$ respectively.

$$S(t) = e^{-\Lambda(t)} \tag{4}$$

A commonly used estimator for the cumulative hazard function is the Nelson-Aalen estimator, which is defined analogously to the Kaplan-Meier estimator (Nelson 1972; Aalen 1978):

$$\hat{\Lambda}(t) = \sum_{t' \leq t} \frac{d(t')}{n(t')} \tag{5}$$

The Nelson-Aalen estimator of Eq. (5) in combination with Eq. (4) is what we will use for our method, as explained in the next section.

Method

We present *SurTree*, a dynamic programming approach to optimizing survival trees. First, we explain what loss function is minimized. Second, we show how DP can be used to find the global optimum for the loss function. Third, we present a special algorithm for trees of depth two that results in a significant increase in scalability.

Loss Function

The optimization of decision trees requires a target loss function. For computational efficiency, the loss function over a leaf node needs to be independent of other leaf nodes. Therefore, like (Bertsimas et al. 2022), we optimize the likelihood method from (LeBlanc and Crowley 1992). This method assumes that the survival function S_i for each instance i can be approximated by a *proportional hazard model*, described by multiplying the exponent in Eq. (4), that is, the baseline hazard model $\hat{\Lambda}(t)$, as estimated by the Nelson-Aalen estimator in Eq. (5), by some parameter θ_i:

$$\hat{S}_i(t) = e^{-\theta_i \hat{\Lambda}(t)} \tag{6}$$

Fig. 2 shows how θ_i changes the survival function $S_i(t)$.

LeBlanc and Crowley (1992) show that for a given data set $\mathcal{D}$ the estimate $\hat{\theta}$ with maximum likelihood is equal to:

$$\hat{\theta} = \frac{\sum_{(t_i,\delta_i,\mathbf{fv}_i)\in\mathcal{D}} \delta_i}{\sum_{(t_i,\delta_i,\mathbf{fv}_i)\in\mathcal{D}} \hat{\Lambda}(t_i)} \tag{7}$$

This means that for a single instance i, the saturated coefficient that perfectly maximizes the likelihood for that instance alone is given by:

$$\hat{\theta}_i^{sat} = \frac{\delta_i}{\hat{\Lambda}(t_i)} \tag{8}$$

The loss for a single instance is then defined as the difference between the log-likelihood of the instance's leaf node $\hat{\theta}$ and the log-likelihood of the instance's $\hat{\theta}_i^{sat}$. In the appendix, we show how this results in the following loss for a data set $\mathcal{D}$ that ends up in a leaf node with parameter $\hat{\theta}$:

$$\mathcal{L}(\mathcal{D},\hat{\theta}) = \sum_{(t_i,\delta_i,\mathbf{fv}_i)\in\mathcal{D}} \left(\hat{\Lambda}(t_i)\hat{\theta} - \delta_i \log \hat{\Lambda}(t_i) - \delta_i \log \hat{\theta} - \delta_i\right) \tag{9}$$

Dynamic Programming Approach

The loss function of Eq. (9) consists of several nonlinear terms that prevent it from being directly optimized with mixed-integer linear programming. However, it can be optimized with dynamic programming. The key change compared to a DP formulation for standard decision trees is the base case: instead of assigning a class based on the majority vote, we now optimize $\hat{\theta}$ such that the loss is minimized. We apply this to the DP formulation from (Demirović et al. 2022):

$$T(\mathcal{D},d,n) = \begin{cases} \min_{\hat{\theta}} \mathcal{L}(\mathcal{D},\hat{\theta}) & n = 0 \\ T(\mathcal{D},d,2^d-1) & n > 2^d - 1 \\ T(\mathcal{D},n,n) & d > n \\ \min\{T(\mathcal{D}(\bar{f}),d-1,n-i-1) & \\ \quad + T(\mathcal{D}(f),d-1,i) & \\ \quad : f \in \mathcal{F}, i \in [0,n-1]\} & \text{otherwise} \end{cases} \tag{10}$$

In this equation, subproblems are defined by the dataset $\mathcal{D}$, the (remaining) tree depth d, and branching node budget n. When $n = 0$, a leaf node is returned with a survival distribution given by θ for which the loss is minimized. When the depth or branching node budget exceeds what is possible according to the other budget, for example, when $d > n$, the budgets are updated accordingly. Otherwise, a branching node is optimized by looping over all possible branching features $f \in \mathcal{F}$ and all possible branching node budget distributions. The loss of the two subtrees is summed for each possible split, and the best possible split is returned.

The solutions to the subproblems $\langle \mathcal{D}, d, n\rangle$ are *cached.* Cached solutions are also used as lower bounds. Upper bounds (best solution so far) and lower bounds for a subtree search are used to terminate the search early.

To prevent overfitting, we use *hyper-tuning* to tune the depth and number of branching nodes. Alternatively, a cost-complexity parameter can be used to penalize adding more branching nodes. However, tuning the depth and number of nodes directly allows to reuse the cache, yielding a speed improvement, without loss of solution quality.

Trees of Depth Two

Demirović et al. (2022) developed a major scalability improvement for optimizing classification trees of maximum depth two. Instead of applying the splitting and recursing technique (as done similarly in Eq. (10)), which requires counting class occurrences for every possible leaf node, this algorithm precomputes the class occurrences by looping over all pairs of features in the feature vector $f_i, f_j \in \mathbf{fv}_k$ for each instance k. The counts can then be used to directly compute the misclassification score for each leaf node without having to examine the entire data set again. Van der Linden, De Weerdt, and Demirović (2023) show that this method can also be generalized to other optimization tasks, provided that a per-instance breakdown of the loss can be formulated.

Here, we provide a breakdown of the per-instance contribution to the costs, such that the same precomputing technique can be used for survival analysis. Pseudocode is provided in the appendix.

First, we split Eq. (9) into several summations:

$$\hat{\theta}\sum_i \hat{\Lambda}(t_i) - \sum_i \delta_i \log \hat{\Lambda}(t_i) - \log\hat{\theta}\sum_i \delta_i - \sum_i \delta_i \tag{11}$$

Then, by substituting Eq. (7) into the above formula, we get the following:

$$\begin{aligned} \mathcal{L}(\mathcal{D},\hat{\theta}) =& \frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)} \sum_i \hat{\Lambda}(t_i) - \sum_i \delta_i \log \hat{\Lambda}(t_i) \\ & - \log\left(\frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)}\right) \sum_i \delta_i - \sum_i \delta_i \\ =& -\sum_i \delta_i \log \hat{\Lambda}(t_i) - \log\left(\frac{\sum_i \delta_i}{\sum_i \hat{\Lambda}(t_i)}\right) \sum_i \delta_i \end{aligned} \tag{12}$$

Eq. (12) is expressed as a function of several sums over the instances. These sums can be precomputed in the same way as class occurrences are precomputed in (Demirović et al. 2022). Three sums need to be computed: the *event sum* ES, the *hazard sum* HS, and the *negative log hazard sum* NLHS.

$$\mathrm{ES}(f_i,f_j) = \sum_{(t_k,\delta_k,\mathbf{fv}_k)\in\mathcal{D}(f_i,f_j)} \delta_k \tag{13}$$

$$\mathrm{HS}(f_i,f_j) = \sum_{(t_k,\delta_k,\mathbf{fv}_k)\in\mathcal{D}(f_i,f_j)} \hat{\Lambda}(t_k) \tag{14}$$

$$\mathrm{NLHS}(f_i,f_j) = \sum_{(t_k,\delta_k,\mathbf{fv}_k)\in\mathcal{D}(f_i,f_j)} -\delta_k \log \hat{\Lambda}(t_k) \tag{15}$$

Eqs. (13)-(15) compute the event, hazard, and negative log hazard sum for the leaf node with data that satisfies feature f_i and f_j. The sums for the other leaf nodes can be computed as follows (similarly for HS and NLHS):

$$\mathrm{ES}(\overline{f_i}) = \mathrm{ES} - \mathrm{ES}(f_i) \tag{16}$$

$$\mathrm{ES}(f_i, \overline{f_j}) = \mathrm{ES}(f_i) - \mathrm{ES}(f_i, f_j) \tag{17}$$

$$\mathrm{ES}(\overline{f_i}, f_j) = \mathrm{ES}(f_j) - \mathrm{ES}(f_i, f_j) \tag{18}$$

$$\mathrm{ES}(\overline{f_i}, \overline{f_j}) = \mathrm{ES} - \mathrm{ES}(f_i) - \mathrm{ES}(f_j) + \mathrm{ES}(f_i, f_j) \tag{19}$$

Here, ES denotes the event sum over the whole dataset $\mathcal{D}$, while $\mathrm{ES}(f_i)$ denotes the event sum on the dataset $\mathcal{D}(f_i)$. Once these sums are computed for pairs of features, the final loss for each split and each possible leaf node can be computed from the sums:

$$\mathcal{L}(\mathcal{D}) = \mathrm{NLHS} - \mathrm{ES}\log\left(\frac{\mathrm{ES}}{\mathrm{HS}}\right) \tag{20}$$

Since we only explicitly count the values for when f_i and f_j hold, and derive the other cases implicitly through Eqs. (16)-(19), the run time is reduced from $O(|\mathcal{F}|^2|\mathcal{D}|)$ to $O(m^2|\mathcal{D}|)$, with m the maximum number of features that hold for any instance in $\mathcal{D}$. This is specifically advantageous when features hold sparingly. Non-sparse features are flipped to improve sparsity.

Experiments

The following introduces the experiment setup, the survival analysis metrics, a scalability analysis with an evaluation of the impact of our depth-two algorithm, and the out-of-sample performance of SurTree and two other methods.

Experiment Setup

Methods We implemented SurTree in C++ with a Python interface using the STreeD framework (Van der Linden, De Weerdt, and Demirović 2023).[1] In our experiment setup,[2] we compare SurTree with the Julia implementation of OST (Bertsimas et al. 2022) and the R implementation of CTree (Hothorn, Hornik, and Zeileis 2006). Each method is tuned using ten-fold cross-validation. For SurTree, we tune the depth and node budget. For CTree, we tune the confidence criterion. For OST, we tune the depth and, simultaneously, OST automatically tunes the cost-complexity parameter as part of its training. All experiments were run on an Intel i7-6600U CPU with 4GB RAM with a time-out of 10 minutes.

Data We evaluate both on synthetic data to measure the effect of censoring and of having more data, and on real data sets. The real data sets are taken from the SurvSet repository (Drysdale 2022). Since the results on the synthetic data show that the differences between the methods are more clearly visible for larger datasets, we limit our real data analysis to data sets with more than 2000 instances. We evaluate out-of-sample performance on the real data sets using five-fold cross-validation.

The synthetic data is generated according to the procedure described in (Bertsimas et al. 2022). First, we generate n feature vectors, with three continuous features, one binary feature, and two categorical features with three and five categories. Each of the features is uniformly distributed. Second, we randomly generate a survival tree T of depth five that splits on random features and assign a random distribution to each leaf node (see the appendix for a list of used distributions). Third, for each of the n instances, we 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, we assign the instance a random value u_i, uniformly distributed between 0 and 1. Fourth, we choose the lowest value for k such that for at most $c \cdot 100\%$ of the observations, $k(1-u_i^2) < t_i$ holds. Finally, for each observation for which $k(1-u_i^2) < t_i$, we set $t_i = k(1-u_i^2)$ and $\delta_i = 0$. For every other observation, we leave t_i and set $\delta_i = 1$.

We evaluate each method with a depth limit of four on five generated data sets for each combination of $n \in \{100, 200, 500, 1000, 2000, 5000\}$ and $c \in \{0.1, 0.5, 0.8\}$, each with a corresponding test set of 50,000 instances.

Preprocessing We use one-hot encoding to encode categorical variables. For categorical variables with more than ten categories, the least frequent categories or combined into an 'other' category. Because the dynamic programming approach of SurTree requires binary features, we binarize the numeric features using ten quantiles on all possible thresholds. Identical features and binary features that identify less than 1% of the data are removed. We evaluate CTree and OST on the numeric data and SurTree on the binarized data.

Survival Metrics

To evaluate the out-of-sample performance of all methods, we compare each method using two common metrics from the literature: *Harrell's C-index* (H_C) (Harrell et al. 1982), and the integrated Brier score (IB) (Graf et al. 1999).

Harrell's C-index Harrell's C-index measures the concordance score. Two instances are (dis)concordant if an earlier known death ($t_i < t_j$) for one instance means a (lower) higher risk of death for that instance ($\theta_i > \theta_j$). When $\theta_i = \theta_j$, the pair is said to have a *tied risk*. Since for censored observations, the time-to-event is not known, we can only compare pairs for which the instance with an *earlier time* is not censored. The number of concordant, discordant, and tied-risk pairs can be calculated using the following formulas respectively:

$$CC = \textstyle\sum_{i,j} \mathbb{1}(t_i < t_j)\mathbb{1}(\theta_i > \theta_j)\delta_i \tag{21}$$

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

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

Harrell's C-index is computed as follows:

$$H_C = \frac{CC + 0.5 \cdot TR}{CC + TR + DC} \tag{24}$$

The advantage of Harrell's C-index is that it does not make any parametric assumptions and that it works well for the proportional hazard model as used in this paper. Its disadvantage is that it does not take incomparable pairs into account, which is a problem when censoring is high. Note that a random predictor has an expected score of $H_C = 0.5$.

[1] https://github.com/AlgTUDelft/pystreed

[2] https://github.com/TimHuisman1703/streed_sa_pipeline

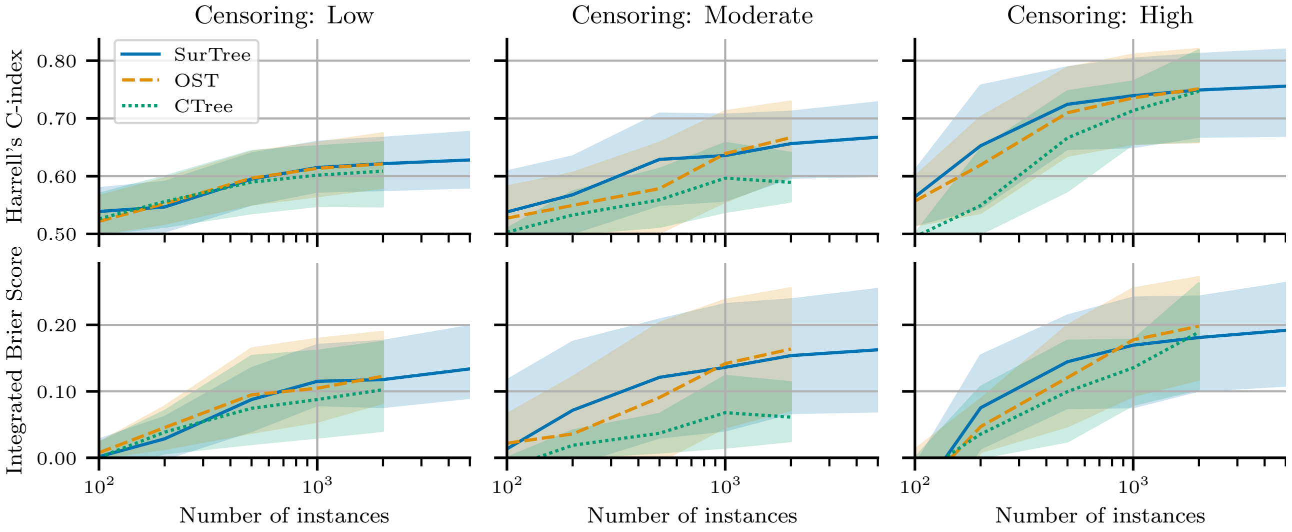

Figure 3: Harrell's C-index and the integrated Brier score on the synthetic data sets (except time-outs).

Integrated Brier score The Brier score (Brier 1950) is commonly used to evaluate probability forecasts, and measures the mean square prediction error. This measure can be used to evaluate survival distributions at a specific point in time. For evaluating the whole distribution, Graf et al. (1999) developed the integrated Brier score:

$$IB = \frac{\sum_i \int_{\underline{t}}^{t_i} \frac{(1-\hat{S}_i(t))^2}{\hat{G}(t)} dt + \delta_i \int_{t_i}^{\bar{t}} \frac{(\hat{S}_i(t))^2}{\hat{G}(t_i)} dt}{|\mathcal{D}|(\bar{t}-\underline{t})} \tag{25}$$

The integrated Brier score evaluates the Brier score over a time interval, with each time step weighed by the Kaplan-Meier estimator of the censoring distribution $\hat{G}(t)$. We compute the integrated Brier score using the test data over the time periods that fall within the 10% and 90% quantile of t_i in the test data, given by $\underline{t}$ and $\bar{t}$ respectively. For easier comparison, we report the normalized relative score $(1-IB/IB_0)$, with IB_0 the score obtained from the Kaplan-Meier estimator over the whole dataset.

The integrated Brier score also makes no parametric assumptions on the data. Another advantage is that it considers both the censored and non-censored data.

Scalability

Synthetic data To evaluate the scalability of each method, we compare the run time of each algorithm for increasing maximum depth and features. Each method is evaluated twice for five synthetic datasets with $n = 5000$ and $c = 0.5$. Once for the original setting ($f = 1$) with three continuous, one binary, and two categorical features, and once with double the number of features ($f = 2$): six continuous, two binary, and four categorical features. After binarization, this results in 39 and 78 binary features, respectively.

Fig. 4 shows that for $f = 1$ up to depth 5, and $f = 2$ up to depth 4, SurTree has a lower run time than OST. SurTree's run time scales exponentially with increasing maximum depth, whereas OST's run time scales linearly. OST has a relatively high run time for low depth because it randomly restarts its local search several times (by default 100 times) to improve the quality of the solution. In contrast, SurTree immediately finds the globally optimal solution. CTree surprisingly has approximately a constant run time for increasing depth and number of features.

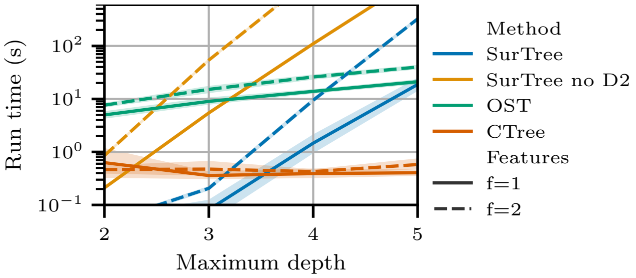

Figure 4: Run time performance for increasing depth, for 3 continuous, 1 binary, and 2 categorical features ($f = 1$) or 6 continuous, 2 binary, and 4 categorical features ($f = 2$).

Real data Despite SurTree being an optimal method, SurTree's average run time for optimizing trees of maximum depth three for the real data sets (including hyper-tuning) is lower than both CTree and OST. On average, it is more than 100 times faster than OST (geometric mean performance ratio). CTree's worse performance here must be attributed to the cross-validation method.

Depth-two algorithm Fig. 4 also shows the increase in scalability due to our algorithm for trees of depth two. On average, the depth-two algorithm reduces run time 45 times (geometric mean, not considering time-outs).

Out-of-Sample Results

Synthetic data Fig. 3 shows the performance on the synthetic data for an increasing number of instances. The results

					Harrell's C-index			Integrated Brier Score		
Data set	$\lvert\mathcal{D}\rvert$	Censoring (%)	$\lvert\mathcal{F}_{num}\rvert$	$\lvert\mathcal{F}\rvert$	CTree	OST	SurTree	CTree	OST	SurTree
Aids2	2839	38.0%	4	22	**0.53**	**0.53**	**0.53**	**0.01**	**0.01**	0.00
Dialysis	6805	76.4%	4	35	0.64	0.65	**0.66**	0.07	**0.09**	0.08
Framingham	4658	68.5%	7	60	0.67	0.67	**0.68**	0.09	**0.10**	**0.10**
Unempdur	3241	38.7%	6	45	**0.70**	0.69	0.69	**0.11**	0.10	0.10
Acath	2258	34.0%	3	21	0.59	0.58	**0.60**	**0.03**	0.02	**0.03**
Csl	2481	89.1%	6	42	**0.78**	0.76	0.75	**0.10**	**0.10**	**0.10**
Datadivat1	5943	83.6%	5	21	0.63	**0.64**	0.63	**0.08**	0.05	0.06
Datadivat3	4267	94.4%	7	30	0.65	0.63	**0.66**	0.02	0.02	**0.03**
Divorce	3371	69.4%	3	5	0.52	**0.53**	**0.53**	0.01	**0.02**	**0.02**
Flchain	6524	69.9%	10	60	**0.92**	**0.92**	**0.92**	0.65	0.65	**0.66**
Hdfail	52422	94.5%	6	27	-	-	**0.81**	-	-	**0.41**
Nwtco	4028	85.8%	7	17	**0.70**	**0.70**	0.69	0.12	**0.13**	**0.13**
Oldmort	6495	69.7%	7	33	0.64	**0.65**	0.63	**0.06**	0.05	0.05
Prostatesurvival	14294	94.4%	3	8	**0.75**	**0.75**	**0.75**	0.09	**0.10**	**0.10**
Rott2	2982	57.3%	11	50	0.68	0.68	**0.69**	0.12	**0.15**	0.14
Wins per metric					6	7	**10**	6	8	**9**
Average rank					2.07	2.03	**1.83**	2.21	1.97	**1.77**

Table 1: Out-of-sample Harrell's C-index and integrated Brier score for data sets from SurvSet (Drysdale 2022) for trees of maximum depth $d = 3$. $\lvert\mathcal{F}_{num}\rvert$ is the number of original features. $\lvert\mathcal{F}\rvert$ is the resulting number of binarized features.

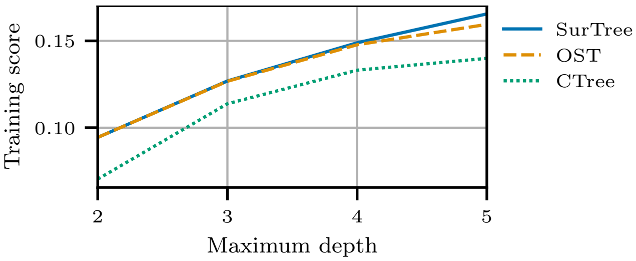

Figure 5: Normalized training loss for CTree, OST, and SurTree, when trained with binarized data.

are split for low, moderate, and high censoring. Since Harrell's C-index only measures performance for instances that are comparable, Harrell's C-index is slightly higher for high censoring. In general, each method performs better with more data, but both OST and CTree time out when $n = 5000$. Furthermore, these results show that both OST and SurTree perform significantly better than CTree, specifically for moderate and high censoring. OST and SurTree perform similarly, but a Wilcoxon signed rank test shows that SurTree has a better Harrell's C-index than OST for moderate and high censoring and few instances ($n = 200, 500$).

Real data Table 1 shows the out-of-sample H_C and IB scores for trees with a maximum depth of three. A Wilcoxon signed rank test reveals that both OST and SurTree perform significantly better (95% confidence) than CTree on both Harrell's C-index and the integrated Brier score. On average, SurTree performs slightly better than OST, but this difference is not statistically significant. Both OST and CTree resulted in one time-out (for Hdfail). For CTree, this is the result of a slow cross-validation algorithm in R based on Harrell's C-index that requires a quadratic number of comparisons. The appendix also shows results for trees with a maximum depth of four.

Training score Since SurTree is the first optimal survival tree method, we can now measure in reasonable time how far non-optimal methods are from the optimal solution, when comparing training scores on the same (binarized) data. Fig. 5 compares the mean training score of CTree, OST, and SurTree on five synthetic training data sets with $n = 5000$ and $c = 0.5$, without hyper-tuning. In this figure, the training score is the normalized loss, with 0 referring to the loss of a single leaf node and 1 referring to zero loss. The difference between SurTree and OST is greatest for $d = 5$, where SurTree's training score is 4% better than OST's. The difference between SurTree and CTree is greatest for $d = 2$, where SurTree's training score is 34% higher than CTree's.

Conclusion

We present SurTree, the first survival tree method with global optimality guarantees. The out-of-sample comparison shows it performs better than an existing greedy heuristic and similar to a state-of-the-art local search approach. SurTree uses dynamic programming and a special algorithm for trees of depth two resulting in run times even lower than the state-of-the-art local search method that does not provide optimality guarantees.

To improve the prediction quality, future work could explore the effect of fitting a Cox proportional hazards model in each leaf node, instead of only a single constant proportional hazard parameter (Cox 1972).

References

Aalen, O. 1978. Nonparametric Inference for a Family of Counting Processes. *The Annals of Statistics*, 6(4): 701–726.

Aghaei, S.; Gómez, A.; and Vayanos, P. 2021. Strong Optimal Classification Trees. *arXiv preprint arXiv:2103.15965*.

Aglin, G.; Nijssen, S.; and Schaus, P. 2020a. Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. In *Proceedings of AAAI-20*, 3146–3153.

Aglin, G.; Nijssen, S.; and Schaus, P. 2020b. PyDL8.5: a Library for Learning Optimal Decision Trees. In *Proceedings of IJCAI-20*, 5222–5224.

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

Bertsimas, D.; and Dunn, J. 2019. *Machine Learning Under a Modern Optimization Lens*. Belmont, MA: Dynamic Ideas.

Bertsimas, D.; Dunn, J.; Gibson, E.; and Orfanoudaki, A. 2022. Optimal Survival Trees. *Machine Learning*, 111(8): 2951–3023.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks.

Brier, G. W. 1950. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1): 1–3.

Carrizosa, E.; Molero-Río, C.; and Morales, D. R. 2021. Mathematical optimization in classification and regression trees. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 29(1): 5–33.

Castro-Santos, T.; and Haro, A. 2003. Quantifying migratory delay: a new application of survival analysis methods. *Canadian Journal of Fisheries and Aquatic Sciences*, 60(8): 986–996.

Chi, C.-L.; Street, W. N.; and Wolberg, W. H. 2007. Application of Artificial Neural Network-Based Survival Analysis on Two Breast Cancer Datasets. In *Proceedings of the annual AMIA Symposium*, 130–134.

Chung, C.-F.; Schmidt, P.; and Witte, A. D. 1991. Survival Analysis: A Survey. *Journal of Quantitative Criminology*, 7: 59–98.

Ciampi, A.; Thiffault, J.; Nakache, J.-P.; and Asselain, B. 1986. Stratification by stepwise regression, correspondence analysis and recursive partition: a comparison of three methods of analysis for survival data with covariates. *Computational Statistics & Data Analysis*, 4(3): 185–204.

Cox, D. R. 1972. Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2): 187–202.

Davis, R. B.; and Anderson, J. R. 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.; and Stuckey, P. J. 2022. MurTree: Optimal Classification Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26): 1–47.

Drysdale, E. 2022. SurvSet: An open-source time-to-event dataset repository. *arXiv preprint arXiv:2203.03094*.

Dunn, J. W. 2018. *Optimal Trees for Prediction and Prescription*. Ph.D. thesis, Massachusetts Institute of Technology.

Dunn, O. J.; and Clark, V. A. 2009. *Basic Statistics: A Primer for the Biomedical Sciences*. Hoboken, NJ: John Wiley & Sons.

Eryurt, M. A.; and Koç, İ. 2012. Internal migration and fertility in Turkey: Kaplan-Meier survival analysis. *International Journal of Population Research*, 2012.

Freitas, A. A. 2014. Comprehensible Classification Models – a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1): 1–10.

Gordon, L.; and Olshen, R. A. 1985. Tree-Structured Survival Analysis. *Cancer Treatment Reports*, 69(10): 1065–1069.

Graf, E.; Schmoor, C.; Sauerbrei, W.; and 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.; and Rosati, R. A. 1982. Evaluating the Yield of Medical Tests. *Jama*, 247(18): 2543–2546.

Hothorn, T.; Hornik, K.; and Zeileis, A. 2006. Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, 15(3): 651–674.

Hu, H.; Siala, M.; Hebrard, E.; and Huguet, M.-J. 2020. Learning Optimal Decision Trees with MaxSAT and its Integration in AdaBoost. In *IJCAI-PRICAI 2020*.

Hu, X.; Rudin, C.; and Seltzer, M. 2019. Optimal Sparse Decision Trees. In *Advances in NeurIPS-19*, 7267–7275.

Hyafil, L.; and Rivest, R. L. 1976. Constructing optimal binary decision trees is NP-complete. *Information processing letters*, 5(1): 15–17.

Ishwaran, H.; Kogalur, U. B.; Blackstone, E. H.; and Lauer, M. S. 2008. Random Survival Forests. *The Annals of Applied Statistics*, 2(3): 841–860.

Janota, M.; and Morgado, A. 2020. SAT-Based Encodings for Optimal Decision Trees with Explicit Paths. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT 2020)*, 501–518.

Kaplan, E. L.; and Meier, P. 1958. Nonparametric Estimation from Incomplete Observations. *Journal of the American Statistical Association*, 53(282): 457–481.

LeBlanc, M.; and Crowley, J. 1992. Relative Risk Trees for Censored Survival Data. *Biometrics*, 48(2): 411–425.

LeBlanc, M.; and Crowley, J. 1993. Survival Trees by Goodness of Split. *Journal of the American Statistical Association*, 88(422): 457–467.

Lin, J.; Zhong, C.; Hu, D.; Rudin, C.; and Seltzer, M. 2020. Generalized and Scalable Optimal Sparse Decision Trees. In *Proceedings of ICML-20*, 6150–6160.

Van der Linden, J. G. M.; De Weerdt, M. M.; and Demirović, E. 2023. Necessary and Sufficient Conditions for Optimal Decision Trees using Dynamic Programming. In *Advances in NeurIPS-23*.

Molinaro, A. M.; Dudoit, S.; and Van der Laan, M. J. 2004. Tree-based multivariate regression and density estimation with right-censored data. *Journal of Multivariate Analysis*, 90(1): 154–177.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning Optimal Decision Trees with SAT. In *Proceedings of IJCAI-18*, 1362–1368.

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

Nijssen, S.; and Fromont, E. 2007. Mining Optimal Decision Trees from Itemset Lattices. In *Proceedings of SIGKDD-07*, 530–539.

Nijssen, S.; and Fromont, E. 2010. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1): 9–51.

Piltaver, R.; Luštrek, M.; Gams, M.; and Martinčić-Ipšić, S. 2016. What makes classification trees comprehensible? *Expert Systems with Applications*, 62: 333–346.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5): 206–215.

Segal, M. R. 1988. Regression Trees for Censored Data. *Biometrics*, 44(1): 35–47.

Selvin, S. 2008. *Survival Analysis for Epidemiologic and Medical Research*. Cambridge: Cambridge University Press.

Shati, P.; Cohen, E.; and McIlraith, S. 2021. SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2021)*.

Su, X.; and Fan, J. 2004. Multivariate Survival Trees: A Maximum Likelihood Approach Based on Frailty Models. *Biometrics*, 60(1): 93–99.

Therneau, T. M.; Grambsch, P. M.; and Fleming, T. R. 1990. Martingale-based residuals for survival models. *Biometrika*, 77(1): 147–160.

Van Belle, V.; Pelckmans, K.; Van Huffel, S.; and Suykens, J. A. 2011. Support vector methods for survival analysis: a comparison between ranking and regression approaches. *Artificial Intelligence in Medicine*, 53(2): 107–118.

Verhaeghe, H.; Nijssen, S.; Pesant, G.; Quimper, C.-G.; and Schaus, P. 2020. Learning Optimal Decision Trees using Constraint Programming. *Constraints*, 25(3): 226–250.

Verwer, S.; and Zhang, Y. 2017. Learning decision trees with flexible constraints and objectives using integer optimization. In *Proceedings of CPAIOR-17*, 94–103.

Verwer, S.; and Zhang, Y. 2019. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. In *Proceedings of AAAI-19*, 1625–1632.

Wang, P.; Li, Y.; and Reddy, C. K. 2019. Machine Learning for Survival Analysis: A Survey. *ACM Computing Surveys (CSUR)*, 51(6): 1–36.

Zhu, H.; Murali, P.; Phan, D. T.; Nguyen, L. M.; and Kalagnanam, J. R. 2020. A Scalable MIP-based Method for Learning Optimal Multivariate Decision Trees. In *Advances in NeurIPS-20*, 1771–1781.