CLASSIFICATION TREES FOR SURVIVAL DATA WITH COMPETING RISKS

by

Fiona M. Callaghan

M.S., Carnegie Mellon University, 2002

M.A., Carnegie Mellon University, 2000

B.A. (Hons.), University of Canterbury, 1998

B.Sc., University of Canterbury, 1997

Submitted to the Graduate Faculty of

the Department of Biostatistics

Graduate School of Public Health in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

GRADUATE SCHOOL OF PUBLIC HEALTH

This dissertation was presented

by

Fiona M. Callaghan

It was defended on

April 15th, 2008

and approved by

Dr. Mark Roberts, M.D., M.P.P.
Professor, Department of Medicine, School of Medicine
University of Pittsburgh

Dr. Abdus Wahed Ph.D.
Assistant Professor, Department of Biostatistics, Graduate School of Public Health
University of Pittsburgh

Dr. Lisa Weissfeld Ph.D.
Professor, Department of Biostatistics, Graduate School of Public Health
University of Pittsburgh

Dissertation Advisors:

Dr. Chung-Chou Ho Chang Ph.D.
Research Assistant Professor, Department of Medicine, School of Medicine,
University of Pittsburgh,

Dr. Stewart Anderson Ph.D.
Professor, Department of Biostatistics, Graduate School of Public Health,
University of Pittsburgh

CLASSIFICATION TREES FOR SURVIVAL DATA WITH COMPETING RISKS

Fiona M. Callaghan, PhD

University of Pittsburgh, 2008

Classification trees are the most popular tool for categorizing individuals into groups and subgroups based on particular outcomes of interest. To date, trees have not been developed for the competing risk situation where survival times are recorded and more than one outcome is possible. In this work we propose three classification trees to analyze survival data with multiple competing risk outcomes, using both univariate and multivariate techniques, respectively. After we describe the method used in growing and pruning the classification trees for competing risks, we demonstrate the performance with simulations in a variety of competing risk model configurations, and compare the competing risk trees to currently available tree-based methods. We also illustrate their use by analyzing survival data concerning patients who had end-stage liver disease and were on the waiting list to receive a liver transplant.

Public Health Significance: Competing risks are common in longitudinal studies. The classification tree for competing risks will provide more accurate estimates of risk in distinct subpopulations than the current tree techniques can provide.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to thank my advisors – Dr Joyce Chang and Dr Stewart Anderson – and the committee for their help and support in producing this dissertation. I would also like to thank the Institute for Clinical Research Education and the Department of Biostatistics for their support both financial and otherwise. My time at the University of Pittsburgh has been an enjoyable one and this is due in no small part to the people I have worked with and the friends I have made over the years. Finally, I would like to thank my family and, above all, my partner Matthew Jackson for all his love and support.

1.0 INTRODUCTION

In the United States, the number of patients who have a serious liver disease and require a liver transplant is far greater than the number of organs available for transplantation. It is therefore important to allocate organs in an optimal manner. One goal is to identify groups at risk from death while on the waitlist (pretransplant death). The method currently used to allocate organs to patients on the waiting list for a liver transplant is based on scores derived from a model for end-stage liver disease (MELD). This model is a Cox proportional hazards model that treats death before receiving a transplant (pretransplant death) as the event of interest and treats competing events as censored. Patients with a higher risk of pretransplant death have higher MELD scores (derived from the likelihood based on the MELD model) and are thus more likely to receive a liver transplant.

One of the weaknesses of using a Cox proportional hazards model in estimating the likelihood of pretransplant death is that it does not account for the various types of outcomes that explain why a patient is removed from the waiting list. For example, although it accounts for removal due to pretransplant death, it does not account for removal due to receipt of a transplant or due to a health improvement that makes a transplant unnecessary. Failure to account for these competing risks may result in biased survival estimates. There are numerous methods for analyzing competing risks, but all of them involve restrictive assumptions, some are difficult to interpret in a clinical situation, and none are classification techniques, which would be the most appropriate techniques for identifying risk groups for organ waiting lists and organ allocation problems.

In recent years, the classification and regression tree (CART) methods of Breiman et al. [2] have been applied to survival data in several fields, including finance, engineering, and health sciences. CART methods can be used to break data observations into similar groups

by means of a series of binary (yes/no) questions or covariates. Although CART has been applied to univariate and multivariate survival data, it has not been applied to data with competing risks. The goal of this study is to use various features of existing trees for survival and nonsurvival data to develop a family of classification trees for data with competing risks. Specifically, the objectives are to develop several methods for growing, pruning, and selecting trees and to propose a schema for comparing the performance of different trees through extensive simulations, to compare the performance of trees that do and do not take competing risks into account, and to create available macros for the competing risk trees in R.

2.0 MOTIVATION: THE LIVER TRANSPLANT DATA

About 16,900 adults are currently on the waiting list for a liver transplant in the United States. Because this number greatly exceeds the number of organs that will become available, it is crucial to allocate organs in a highly effective and appropriate manner. Scores derived from the model for end-stage liver disease (MELD [21]) are currently used to prioritize the order in which liver transplant candidates who are adults (18 years or older) will receive transplants as donor organs become available. The MELD score is used to predict the likelihood that a patient will die while awaiting a liver transplant (pretransplant death). This likelihood is in turn used to rank the patient on the list of candidates awaiting liver transplantation. The use of the MELD score in the organ allocation system is designed to allow available organs to be given to transplant candidates according to the severity of their liver disease, rather than the length of time that they have been on the waiting list.

The MELD uses a Cox proportional hazards regression model [6] and currently includes the following covariates for each patient: the serum bilirubin level, the creatinine level, and the international normalized ratio (INR) for prothrombin time. When the Cox model is used to estimate the probability of pretransplant death via MELD scores, it treats transplant and removal from the waiting list for other reasons (e.g. improvement in medical condition) as censored events. In other words, it only takes into account the hazard rate of death before receiving a transplant and ignores the hazard rates of other competing events. This is problematic because the presence of competing events may fundamentally alter the probability of pretransplant death. Additionally, the events other than pretransplant death may be of interest to the physician for their own sake, and not just as a confounding factor.

There is a need for a method to predict the risk of pretransplant death that also accounts for the effect of the other risks, namely "recovery before transplant" and "transplant". If we

ignore the competing risk aspect of the data and use techniques designed for one outcome, we get biased estimates of risk. A secondary goal is to not only adjust for the presence of other competing risk events, but also to predict the risk for more than one kind of event simultaneously, when more than one kind of event is of interest to the physician. For instance, we may no longer wish to simply adjust for the recovery outcome, but we may want to predict it alongside the pretransplant death outcome.

When each subject at at risk from more than one kind of event, we say that competing risks are acting on each subject. More formally, a situation is said to involve *competing risks* when each observation can fail from one of J ($J \geq 2$) causes and, furthermore, the occurrence of one of these events *precludes* the observation of any of the other events (Klein et al. [17]). This is the definition of competing risks that we will be using in this work.

One common method for analyzing competing risks data is a Cox model stratified on different competing risk events. This model is based on the event-specific hazard function. In general, hazard functions are more difficult to interpret in a medical setting than quantities such as the cumulative incidence function or the survival function. Additionally, it is often the case that the effect of a covariate on the cause-specific hazard is quite different to the effect the covariate has on the cumulative incidence probability [24, 14]. As a result, an analysis of covariate effects using the cause-specific hazard function can be misleading if the goal is to predict probability of risk. Additionally, the Cox model assumes the proportional hazards assumption which can be difficult to verify and interpret.

An alternative to the event-specific hazard is the cumulative incidence function (CIF). It is defined as the probability of an individual experiencing an event by time t when other competing risks are acting on the individual. The cumulative incidence function models the more "real world" situation and it is directly related to crude survival rates, so it is essential to decision makers. However, many situations require more than just a summary measure; some form of regression would be useful. To estimate failure in the presence of competing risks, Fine and Gray [10] proposed a proportional hazards model for the CIF, and this regression model has been used widely in medical research (e.g., Guardiola [15]; Rocha et al. [28]; Clave et al [5]; Farag et al., [9]). Despite the merits of using a regression model based on CIF when competing risks are present, this type of model has several limitations when

applied to the problem of estimating survival rates for patients who are awaiting an organ transplant. For example, the model would be complicated to expand and difficult to interpret if we wanted it to account for time-varying and nonlinear covariate effects and include possible interaction effects among covariates. This type of model usually requires potentially restrictive assumptions, such as the proportional hazards assumption. Furthermore, regression models such as the Fine and Gray model do not provide a clear method for classifying patients into groups. Many problems are essentially classification problems where we wish to assign patients to groups with different levels of risk.

It would be very useful to have a method that could answer the following questions: What is the most likely outcome for this patient? Which treatment option should be used, given the patient's medical history? What are the subgroups in the the population with most or least risk of pretransplant death (after accounting for the competing risks nature of the data)? A classification method would be more suited to this problem than a regression method. The classification and regression tree methodology adapted for competing risks that we will develop in this research is an attempt to address these problems.

CART-style decision trees are a very popular tool in survival analysis. Like regression techniques (such as the familiar Cox model) the researcher can use survival trees to create a model that predicts the outcome using a range of covariates. However, survival trees have several advantages over traditional regression techniques. Firstly, tree methods do not require a functional form relating an outcome to the covariates. Survival trees do not require a proportional hazards assumption, and in general require less assumptions than regression techniques. They are more suited to classification of outcomes, which is a common goal for survival analysis, for instance when we wish to predict treatment options or type of outcome given a set of covariates. In this way, survival trees can be more suited to clinical practice than regression. They are also easier to explain and interpret than regression based on hazards, because they can be based on a range of non-hazard measures of risk (e.g. survival curve, cumulative incidence function) and they can be easily represented graphically in the form of a binary tree. They can also handle time-dependent covariates and high-level interactions with relative ease. However, to date, there have been no survival trees developed for competing

risks. The purpose of this work is to extend the area of survival trees to the competing risk setting.

For this dissertation, we will use the database from the Organ Procurement and Transplantation Network (OPTN) portion of the United Network for Organ Sharing (UNOS) database. The OPTN/UNOS database has the advantage of providing data on a national sample of liver transplant candidates. However, because this database does not include laboratory information for years earlier than 2002, it is necessary for us to also use clinical and laboratory data from a more detailed database, the University of Pittsburgh Medical Center (UPMC) database. Together the two databases will provide us with the information we need to develop and evaluate our models.

UNOS is a U.S. nonprofit organization that manages the nation's organ transplant waiting lists, matches organs with the patients who need them, and maintains a national database of pretransplant and follow-up information on every transplant recipient. Before the year 2002, the information included each patient's socio-demographic data, etiology-based category of liver disease, date of joining the transplant list, and date and reason for leaving the transplant list. Beginning in 2002, the information collected also included the results of serologic tests (e.g., serum creatinine and bilirubin levels) and liver function tests.

The most recent data available from OPTN/UNOS consists of information about candidates who were on the liver transplant waiting list at any time between 1995 and 2004. Table 1 shows the number of adult candidates who were removed from the waiting list during this period because of one of the following: (a) receipt of a transplant, (b) the occurrence of death before receipt of a transplant, or (c) other reason (e.g. an improvement in health that caused a change in the need or desire for a transplant).

Table 1: Number of Adult Candidates Who Were Removed from the Liver Transplant Waiting List (1995-2004).

	Year									
Reason for Removal	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
Transplant	3,298	3,393	3,487	3,816	4,117	4,311	4,527	4,812	5,137	4,332
Death	752	916	1,081	1,315	1,753	1,708	1,918	1,796	1,715	1,223
Other	488	706	637	671	791	1,111	1,217	2,497	1,603	1,055

Source: www.optn.org/data/

3.0 LITERATURE REVIEW

3.1 CLASSIFICATION METHODS

There are many techniques that have been designed to discriminate observations into groups. The purpose of this dissertation is to develop a classification technique for competing risks data, which combines features of multinomial data and time-to-event data. We are interested in identifying groups of individuals that have similar survival patterns over time. When we have a competing risks situation, each subject can experience one of a number of different events and we are interested in the time to the event, as well as the type of event. The nature of the outcome means that the current techniques for classification are not appropriate. We wish to develop a method of classification that accommodates features unique to competing risk data.

In the following, we look at some of the most commonly used techniques used for classification of continuous, binary or multinomial random variables, and the discussion of classification as it applies to survival analysis will be left until after the discussion of competing risks and time-to-event data (see Section 3.3).

Logistic regression is a method commonly used to estimate odds, odds ratios, and probabilities for binary outcomes and continuous or categorical predictors. Additionally, logistic regression can be used for classification by using the predicted probability to indicate whether an observation is more or less likely to be a "success" or a "failure". If we let p_i be the predicted probability of success for the ith observation (with $i = 1, ..., n$), then for some cut-off value p we can classify any observation using the following rule: if $\hat{p}_i \geq p$ then the ith observation is classified as a success; if $\hat{p}_i < p$ then the ith observation is classified as a failure. Logistic regression is useful for classification of binary outcomes because it is a fairly

straight-forward method and it can utilize categorical and continuous information to predict the outcome.

Potentially, we have an infinite number of possible decision rules corresponding to the infinitely many possible cutoff values for p. To help us choose the best classification rule, we can calculate the sensitivity, Pr(classify as success|observed success), and specificity, Pr(classify as failure|observed failure), for each value of $p = \hat{p}_i$, based on our data. Ideally, we would like to have a sensitivity and specificity close to 1, but this is not possible – sensitivity increases as p increases, whereas specificity decreases with increasing p. We choose a value of p that corresponds to acceptable levels of sensitivity and specificity. The ROC curve (receiver operating characteristic curve) is a plot of 1-specificity versus sensitivity. The area under this curve is a statistic used to give an overall measurement of how well the logistic regression model predicts the outcome, or, in other words, how well it classifies the observations as successes or failures.

An important weakness of logistic regression involves regression's usefulness in modeling conditional relationships among the predictor variables. If the effect of one variable depends upon the values of another variable then we model this by way of an interaction effect between two predictors, and we proceed similarly for 3- or 4-way dependencies. However, these dependencies can be difficult to identify and the inclusion of many interactions or multi-level interactions leads to a complicated model that can be difficult to fit and to interpret. However, the main limitation for logistic regression is that can only be used for binary outcomes and cannot be used to handle the complexities of time-to-event data. Recall that competing risks data generally has multiple possible outcomes and time-to-event information. We can only apply logistic regression to survival data if all the subjects are observed for the same amount of time and we only have two possible outcomes (success and failure).

To accommodate multiple outcomes we could use multinomial logistic regression, which is form of regression designed to handle the situation where there are three or more categorical outcomes. Suppose we have $i = 1, ..., N$ subjects and each will have one of $j = 1, ..., J$ possible outcomes. With multinomial logistic regression we can use continuous and categorical information about each subject to generate predicted probabilities $\hat{p}_{ij}$ that subject

i will experience event j. Once we have these probabilities, we can assign any number of classification rules, for instance we may say that if $\hat{p}_{ij}$ is the largest predicted probability for subject i, then we predict outcome j for that individual. There has been some research into ROC-style curves for more than two outcomes, and multinomial logistic regression is a technique that might be used to handle competing risks data, but, as with logistic regression, we would need to have observed all the subjects for the same length of time, which is a serious limitation. Furthermore, even if patients were observed for the same amount of time, neither the logistic nor multinomial models take into account the failure-time data.

3.2 CLASSIFICATION AND REGRESSION TREES

Regression trees were introduced by Morgan and Sonquist [23], and became very popular with the introduction of CART (Classification and Regression Trees) by Breiman, Friedman, Olshen and Stone [2]. Regression trees help us to identify groups of subjects with similar outcomes, based on the subjects' covariates. They require very few statistical assumptions, can handle many kinds of data type and they are relatively easy to interpret.

The central idea behind CART is to use binary partitions to create mutually exclusive groups of observations that are homogeneous with respect to the outcome. We start with the covariate space $\mathcal{X}$ and split this into two *daughter nodes*, say $\mathcal{X}_1$ and $\mathcal{X}_2$, with the goal of trying to make the subsets more homogeneous than the previous set. We further split $\mathcal{X}_1$ and $\mathcal{X}_2$ into two parts to get $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4$ and continue doing this until we have a collection of subsets $\mathcal{X}_1, ..., \mathcal{X}_k$ of $\mathcal{X}$ such that all or most of the subjects in each subset are very "similar", by some measurement.

In the language of trees, a subset $\mathcal{X}_k$ of $\mathcal{X}$ defined by a binary split is called a *node*. The final subsets that we end up with when we have finished splitting the tree are known as *terminal nodes*. An *internal node* is any node of a tree that is not a terminal node. The *root node* is the first node of the tree, before we have split the data, and is equivalent to $\mathcal{X}$. The terminal nodes form a partition of $\mathcal{X}$. Each terminal subset is designated by a class label, and there may be two or more terminal nodes with the same class label. A *branch* of

T is a tree that has an internal node of T as a root node, and contains all of the subsequent daughter nodes below that node in T. Finally, a *subtree* T^* of T shares the root node of T, but may not share all the subsequent branches of T. See Figure 1.

The CART algorithm involves calculating the change in homogeneity from parent node to daughter nodes for each potential split, in order to pick the best split at each node of the tree. This is measured with the change in impurity function ϕ. CART focuses on minimizing within-node difference (impurity). A common measure of impurity is within sums of squares of the outcome, if the outcome y is continuous. For example, for node p, $\phi(p) = \sum_{i \in p}(y_i - \bar{y})^2$. The change in impurity from parent node p to the left daughter node L and right daughter node R using split s is therefore $\phi(p, s) = \phi(p) - \phi(L) - \phi(R)$. The aim of the CART algorithm is actually to *maximize* this reduction in impurity at each step.

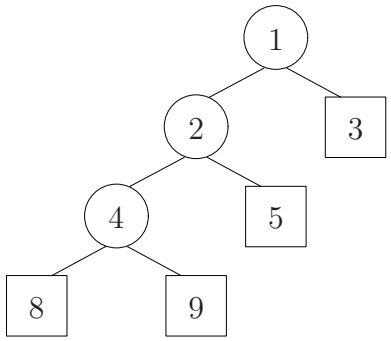
In general, there are four main steps to growing a tree. The first step involves selecting an appropriate measure of "homogeneity" or "impurity" for the outcome. The second step is deciding on a methodical way of growing the tree and some kind of stopping-rule (for instance, we stop when the sample size is very small in each terminal node, or the observations in each terminal node have identical covariates). The third step is to prune the large tree that we grew in the previous step, in such a way that we generate an ordered list of trees from largest to smallest (root node only). In the fourth step (tree selection) we have a criteria to choose the "best" tree; the goal is to find the simplest tree that still categorizes the observations effectively.

In the following sections, we will look at some of the different applications of within-node classification trees and between-node classification trees, firstly for non-time-to-event data and then incorporating survival times, multiple event survival times and finally competing risks.

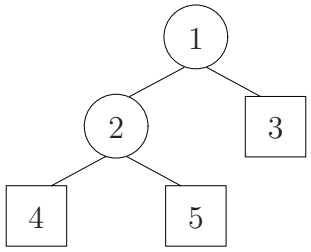
3.3 SURVIVAL TREES

In recent years, there have been many applications of regression trees to survival analysis. Survival analysis techniques are very common in medical research and there is also a desire

A.



B.



C.

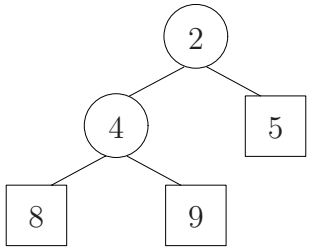


Figure 1: A tree, subtree and branch.

A. A tree. Node 1 is the root node; Nodes 1, 2 and 4 are internal nodes; and Nodes 3, 5, 8, and 9 are terminal nodes. The numbering of the nodes is as follows: the left daughter node of node n is node $2n$ and the right daughter node is node $2n + 1$. **B.** A subtree of the tree in A. **C.** A branch of the tree in A.

on the part of physicians to have decision rules for diagnosis. Hence, these two factors have provided a strong motivation for developing regression trees for survival analysis. In the area of univariate survival analysis (one subject, one possible outcome, all independent observations) there have been two main approaches.

The first approach is analogous to CART method of Breiman, et al. [2] The goal at each step when growing the tree is to maximize homogeneity of the outcome within each node. Many authors have taken this approach, including Gordan and Olshen [13], Davis and Anderson [7], Therneau, Grambsch, and Fleming [33], Molinaro et al. [22] and LeBlanc and Crowley [18]. Typically, the continuous outcome used in CART is replaced with martingale residuals [33] or with deviance [18], and then sums of squares are formed by squaring and summing the residuals. Another technique developed recently by Jin et al. [16] involves calculating the sums of squares within a node using the variance of the survival times.

A second method has been to grow the tree maximizing between-node separation. This is known as the "goodness-of-split" method, and generally uses a two-sample statistic, such as the log-rank statistic, to measure between-node heterogeneity. For each potential split, a two-sample statistic is calculated, and the "best" split is the one with the largest (or most significant) statistic. Some authors who have developed and extended this technique include Ciampi et. al.[4], Segal [30], and LeBlanc and Crowley [19]. This method of growing a tree departs from the CART algorithm in growing the tree and in the methods for pruning and selection.

Recently, survival trees have been used to model correlated, multivariate outcomes. Correlated outcomes generally occur in one of two possible ways. In the first case, each subject can experience multiple events, such as recurrence of disease in a cancer study or recidivism in crime statistics. The events occur serially in time and this kind of data is called *multiple event time data*. The second kind of data is called *clustered failure time data* and it occurs when the observations are naturally clustered, for instance when investigating disease among families or time to decay of teeth, where we have multiple observations per individual. These events occur a "parallel" manner in time.

Frailty models and Wei-Lin-Weissfeld marginal models are two approaches for handling correlated data in survival analysis that have been adapted to regression trees. Su and Fan

[32] used gamma distributed frailty models to come up with a likelihood ratio test splitting rule, while Gao, Manatunga and Chen [11] used a Wald test splitting rule and gamma distributed frailty. Both sets of authors have also built trees with the Wei-Lin-Weissfeld marginal method. Su and Fan [31] and Fan et al. [8] used a robust log rank statistic derived from the marginal approach for the splitting rule, and Gao, Manatunga and Chen [12] used the Wei-Lin-Weissfeld method to develop a survival regression tree with the property of having proportional hazards over the whole tree, rather than just between daughter nodes. All of these trees employ a maximizing-between-node-difference approach. A general "road map" for how one might construct a multivariate regression tree for survival analysis using the CART approach has been proposed by Molinaro et al. [22]. This approach involves substituting the uncensored survival times into an appropriate loss function (eg. within sums of squares). Censoring is adjusted for by incorporating IPCW or inverse probability of censoring weights in the loss function.

None of these approaches considers the usual competing risks situation, where each subject experiences only one event that could be one of several mutually exclusive types. In Section 3.4 we review common techniques for analyzing competing risks data.

3.4 COMPETING RISKS

Competing risk data occurs when we have time-to-event data with more than one kind of outcome. When we have competing risk data, the occurrence of the competing risk event *precludes* any other kind of event from happening. An alternative definition that is sometimes used does not require the outcomes to be mutually exclusive (we can observe more than one kind of event per individual). In this proposal, we will assume the former definition (time-to-event data with mutually exclusive types of outcomes) of Klein and Moeschberger [17] and Kalbfleisch and Prentice [26]. Typically, there is an "event of interest" (for instance, time to transplant) that is the primary focus of the research, and competing risk events (for instance, time to withdrawal from the study due to side-effects), that prevent us from observing the true time to the event of interest for every subject. We may also have a "true censoring"

outcome: observations that are censored due to drop-out or to non-study related reasons. However, it is not necessary to "privilege" one kind of event over another; we may have a number of events of equal interest. In either case, if we want to estimate the probability of an event, we will have to account for or adjust for the presence of the other events. There is some debate in the literature over the best estimator to use when dealing with competing risks data. The most common candidates are the cause-specific hazard function, and the cumulative incidence function.

3.4.1 Summary Curves for Competing Risks

The following discussion uses the notation of Klein and Moeschberger [17]. Let $X_j, j = 1, ..., J$ be the potential, unobservable time to occurrence of the jth competing risk. What we observe is the time to the first failure from any event $T = \min_j(X_1, ..., X_J)$ and δ, an indicator which records which of the J competing risks caused the subject to fail ($\delta = j$ if $T = X_j$). We can think of each type of event as having a *cause-specific hazard rate for risk* j defined as,

$$\lambda_j(t) = \lim_{\Delta t \to 0} \frac{P[t \leq T < t + \Delta t, \delta = j | T \geq t]}{\Delta t}$$

where T is the time to the event j. The cause specific hazard is the rate at which subjects who have not yet experienced any of the competing risks are experiencing event j. The overall hazard rate for the time to failure, T, is the sum of the J cause-specific hazard rates,

$$\lambda_T(t) = \sum_{j=1}^{J} \lambda_j(t)$$

There are two important disadvantages to the cause-specific hazard rate. The first is problem is interpretability: it is a difficult quantity to translate into something meaningful for physicians. This is why some kind of probability is usually used (such as the survival function or cumulative incidence function) to summarize the chance of a particular competing risk occurring. The second problem relates to covariates. Covariates can have a very different effect on the cause-specific hazard function compared to, say, the cumulative incidence function. This can be a problem because it is more natural to think of a covariates effect on the risk or probability of an event, rather than the conditional rate of change of the probability

(hazard) and therefore it is easy to confuse the impact of a covariate on the hazard rate with the impact of the covariate on the probability.

At first glance, a survival function seems like a good candidate to model competing risks because it is a function that is used in other areas of survival analysis and the Kaplan-Meier curve is a familiar and relatively simple estimator. Let $S(t_1, ..., t_J) = P[X_1 > t_1, ..., X_J > t_J]$ be the *joint survival function of the J competing risks*. Then the *net survival function for risk j*, $S_j(t)$, is the marginal survival function found from the joint survival function by taking $t_i = 0$ for all $i \neq j$, or

$$S_j(t) = S(0, ..., t_j, ..., 0) = P[X_1 > 0, ..., X_j > t_j, ..., X_J > 0]$$

The interpretation of the net survival function is the probability of failure from event j in a world where other competing risks are not present. Generally this is not a useful quantity to estimate because it does not reflect the real world situation.

The *cumulative incidence function for risk j* (also known as "crude probability" and "cause-specific sub-distribution function") is defined to be

$$F_j(t) = P[T \leq t, \delta = j] = \int_0^t h_j(u) \exp\{-H_T(u)\} du$$

where $H_T(t) = \sum_{j=1}^{J} \int_0^t h_j(u) du$ is the cumulative hazard rate of T. The cumulative incidence function is interpreted as the probability of failure by time T from event j in a world where all the competing risks are acting on the individual. This is a useful interpretation that is more easily applicable to the real world than that of the net survival function, and this accounts for much of the popularity of the cumulative incidence function in modeling competing risks.

$F_j(t)$ depends on the hazard rates of the other competing risk events. This leads to another advantage of the cumulative incidence function. Since the cause-specific hazard rates can be directly estimated from the observed data, $F_j(t)$ is also directly estimable without making assumptions about the joint distribution of the potential failure times. It should be noted that $F_j(t)$ is non-decreasing and $F_j(0) = 0$, but it is not a true distribution function since $\lim_{t\to\infty} F_j(t) = P[\delta = j] < 1$, hence the reason for $F_j(t)$ also being called "sub-distribution" function.

There are two other main candidates to model competing risks data: the *partial crude sub-distribution function* and the *conditional probability function*. The partial crude sub-distribution function $F_j^J(t)$ is very similar to the cumulative incidence function, except that a subset of the competing risks are removed from consideration. The interpretation of this function is the probability of death from risk j in a world where only some of the competing risks are acting on the individual.

Formally, let $\mathbf{J}$ be the set of risks that an individual can fail from, and $\mathbf{J}^c$ be the set of risks that are removed. Let $T^J = \min(X_j, j \in \mathbf{J})$. The *partial crude sub-distribution function* is defined as,

$$F_j^J(t) = P[T^J \leq t, \delta = j], j \in \mathbf{J}$$

The other estimator that is sometimes used is the *conditional probability function*, $CP_j(t)$ which is described by Pepe and Mori [25] as the probability of experiencing event j by time t given the subject has not experienced any other event by time t. For simplicity, suppose that there is one event of interest X_1 and one competing risk event X_2 (however, all the results and definitions can be generalized to J possible risks). Then $CP_1(t)$ is defined as,

$$P[X_1 \leq t, X_1 < X_2 | \{X_2 < t, X_1 > X_2\}^c] = \frac{F_1(t)}{F_1^c(t)}$$

where $F_1^c(t)$ denotes the complement of $F_1(t)$.

3.4.2 Regression Analysis for Competing Risks

There are two popular methods for regression analysis when competing risks are present: regression on cause-specific hazards using the competing risks analogue to the Cox proportional hazards model, and the regression model for the cumulative incidence function proposed by Fine and Gray [10].

In proportional hazards regression on the event-specific hazards we model the cause-specific hazard for cause j for a subject with covariate vector Z as

$$\lambda_j(t|Z) = \lambda_{j0} \exp(\beta_j^T Z)$$

where λ_{j0} is the baseline cause-specific hazard rate for cause j and β_j is the covariate effects on cause j. This is a straight-forward application of stratified Cox-regression, however the covariate effects require careful interpretation. Under the usual Cox regression situation (in the absence of competing risks) two survival functions S_1 and S_2 based on covariate values Z_1 and Z_2 are related with the following formula,

$$S_2 = S_1^{\exp(\beta^T(Z_2 - Z_1))}.$$

But this relationship does not hold for cumulative incidence functions when competing risks are present. The reason for this is because cumulative incidence functions depend on the cause-specific hazards for all the causes. Hence the effect of a covariate on the cumulative incidence function for cause j depends not only on how the covariate effects the cause-specific hazard for j, but also how the covariate effects the cause-specific hazard for the other causes, as well as the effect of the covariate on the baseline hazards for all the causes. As a result, the effect of a covariate on the cumulative incidence function can be quite different to the effect of the covariate on the cause-specific hazard function, which can lead to confusion.

Fine and Gray [10] have developed a method for regression on the cumulative incidence function directly. To do this they proposed an alternative to the cause-specific hazard called the *sub-distribution hazard* which is defined as follows,

$$\begin{aligned} \gamma_j(t) &= \lim_{\Delta t \to 0} \frac{P[t \leq T < t + \Delta t, \delta = j | T \geq t \cup (T \leq t \cap \delta \neq j)]}{\Delta t} \\ &= -\frac{d \log(1 - F_j(t))}{dt} \end{aligned}$$

where $F_j(t)$ is the cumulative incidence function for cause j. The difference between the cause-specific hazard and the sub-distribution hazard lies in the risk set: for the cause-specific hazard a subject is removed from the risk set after experiencing any event by time t; for the sub-distribution hazard for cause j, only subjects who have experienced event j by time t are removed, all other subjects remain in the risk set, including those who have experienced some other type of event by time t. To avoid bias in the estimates, subjects that experience events other than type j, are kept in the risk set, but the "degree" to which they belong in the risk set can be less than 1, and is given by a weighting function that decreases

with time. The weighting function is an inverse probability of censoring (IPCW) weighting function.

The main advantage to the Fine and Gray regression is that it uses cumulative incidence functions directly, and so there is less confusion in interpreting the covariate effects when translating from cause-specific hazards than in the Cox model. However the technique also has some disadvantages. The most notable drawback is the assumption that a subject that has failed from another cause is treated as still being at risk from cause j, when this may in fact be biologically impossible, for example when we are dealing with different causes of death. Additionally, this technique requires a proportional hazards assumption and, like other regression techniques, it does not give a clear method for classification of the subjects into outcome groups. Finally, this technique is suited to the situation where researchers are interested in one event type only, and wish only to adjust for the effect of the other events, but not to the situation where more than one event is of interest.

4.0 OUTLINE: CLASSIFICATION TREES FOR COMPETING RISKS

The aim of this thesis is to develop several new within-node and between-node trees for competing risks data, as outlined in Table 2.

The "single event" and "composite event" tree types refers to whether or not the device used to create each split discriminates based on one event only, or the differences among all the events jointly. When survival data involve competing risks, under the "single event" categories, we will build up the tree by focusing on the event of interest and treating other competing risks as nuisance events that have to be adjusted for. Under the "composite event" categories, we propose to build up the tree by accounting for all events simultaneously.

The "between-node" and "within-node" tree types refer to the method that is used to split the data into two parts at each node of the tree. Traditionally, between-node trees use some kind of two-sample test to measure discrimination between two groups and within-node trees are formed by calculating the reduction in "impurity" in going from the "parent node" to the two "child nodes" via an impurity function. Survival trees were developed using the two-sample tests common to survival analysis e.g. log-rank test (Segal, [30]). These trees have the advantage that the measure for discrimination is familiar and has a clinical interpretation. CART and the first trees adapted for survival analysis (Gordon and Olshen [13], Therneau, Grambsch, and Fleming [33]) were originally developed using within-node measures of impurity. This approach has the advantage that it can utilize the machinery associated with CART.

In Chapter 5 we propose two types of univariate trees for competing risks. One univariate tree is a between-node tree based on the difference between cumulative incidence functions. The other univariate tree uses event-specific martingale residuals as the basis of a within-node tree. Both of these tree types address the problem of identifying risk subgroups when there

Table 2: Proposed survival trees for competing risks

Tree-type	Single Event	Composite Event
Between-node	1. Univariate Test CIF	2. Multivariate Test CIF
Within-node	3 a. Event-specific residual 3 b. Variance of survival time (future research)	4. Within-node multivariate tree (future research)

is one competing risk event of primary interest. In Chapter 6 we propose a classification tree for competing risks based on maximizing the between-node difference of the CIF with regard to more than one competing risk event. This tree is designed to address the problem of identifying subgroups when multiple competing risk events are of interest to the researcher. In Chapter 7 we look at two further tree techniques for competing risks – a within-node multivariate tree and a variance-based univariate within-node tree – that are left to future research.

5.0 UNIVARIATE CLASSIFICATION TREES FOR COMPETING RISKS

5.1 INTRODUCTION

The classification and regression tree (CART), first introduced by Breiman et al. [2], is a popular statistical method for classifying subjects into progressively smaller subgroups based on recursive binary partitioning of the covariate space. While traditional regression methods are useful for estimating and testing covariate effects and for predicting outcomes, tree-based classification methods can also be used for identifying important covariates and prediction, and trees are particularly good at providing information about the conditional relationships among the covariates. In addition, tree-based methods have several advantages over regression methods: they require fewer assumptions (tree methods are often non-parametric in nature), can handle more types of data, and provide a straightforward rule for classifying observations. In the field of medicine, the series of yes/no questions generated by the tree-based method lends itself better to decision rules for diagnosis and treatment.

There have been two main approaches to developing tree-based methods. The first approach involves growing a tree to maximize the within-node homogeneity of the outcome. The goal is to identify groups of observations with similar outcomes, and one advantage of this method is that we are able to assign a measure of "dissimilarity" or "impurity" to each of the subgroups of individuals identified by the method. This is the approach used in CART, and the well established existing machinery (for tree selection, prediction) developed for CART can be used. In addition, a measure of impurity is associated with each subgroup defined by a within-node tree. The second approach involves growing a tree to maximize the between-node heterogeneity. Splits are chosen that produce the maximum difference between the subgroups. Although the second approach requires different methodology, the

advantage is that it uses a measure of between-group discrimination that is usually much easier to interpret. An example of a between-node measure of difference is the log-rank test statistic.

Trees for survival data use both between- and within-node approaches. For univariate survival data, within-node methods are exemplified by Gordan and Olshen [13], Davis and Anderson [7], Therneau, Grambsch, and Fleming [33], LeBlanc and Crowley [18], and Molinaro, Dudoit, and van der Laan [22]. Univariate tree methods based on between-node heterogeneity were developed and extended by many authors including Ciampi et al. [3], Segal [30], and LeBlanc and Crowley [19]. There are far fewer methods for multivariate survival data. Almost all the multivariate methods have been based on between-node heterogeneity, with the exception of Molinaro et al. [22] who proposed a general within-node homogeneity approach for both univariate and multivariate data. The multivariate methods proposed by Su and Fan [31, 32] and Gao, Manatunga, and Chen [11, 12] concentrated on between-node heterogeneity and used the results of regression models. Specifically, for recurrent event data and clustered event data, Su and Fan [32] used likelihood-ratio tests while Gao et al. [11] used Wald tests from a gamma frailty model to maximize the between-node heterogeneity. Su and Fan [31] and Fan et al. [8] used a robust log-rank statistic while Gao et al. [12] used a robust Wald test from the marginal failure-time model of Wei, Lin, and Weissfeld [35].

The techniques developed for univariate and multivariate survival trees cannot be applied to data with competing risks because they do not take into account that the occurrence of competing events preclude the occurrence of other events. For example, in the case of a patient who has end-stage liver disease and is on the waiting list to receive a donated liver for transplantation, a patient can be removed from the waiting list because of death or several reasons other than death, including a change in health status that makes the patient no longer eligible for a transplant. These reasons can be treated as competing risks because their occurrence could fundamentally alter the probability that death, the main outcome of interest, will occur.

In Chapter 5 we propose two novel classification trees for data with competing risks. The first is a tree that uses Gray's [14] two-sample test of cumulative incidence function to measure between-node heterogeneity. The second is a tree that uses event-specific martingale

residuals to measure within-node homogeneity, and we propose two impurity functions based on these residuals. In Sections 5.2 and 5.3 we outline the methods for splitting, growing, pruning, and selecting the final tree. In Section 5.4, we describe the simulations that we performed to compare our new trees with each other and with a univariate between-node tree based on the log-rank test (LeBlanc and Crowley, [19]). In Section 5.5, we illustrate our new techniques by analyzing data from patients who had end-stage liver disease and were on the waiting list for receipt of a liver transplant. In Section 5.6, we discuss implementation and future directions of our methods.

5.2 TREE TO MAXIMIZE BETWEEN-NODE HETEROGENEITY

Two stages are involved in constructing a tree: growing and pruning. In this section, we describe the methods used in both stages of building a tree that maximizes between-node heterogeneity for data with competing risks.

5.2.1 The Growing Stage

We began by selecting a splitting function based on an event-specific cumulative incidence function (CIF). A CIF, also known as a subdistribution function, is a marginal probability of failure for an event of interest and is used to summarize data with competing risks. After splitting the covariate, we used a method proposed by Gray [14] to compare the two CIFs. This method extended the rank-based tests for data with a single event (e.g. the log-rank test, the generalized Wilcoxon test, and the Fleming-Harrington test) to apply to data with competing risks. We repeated the process of splitting and comparing CIFs until we found the two groups with the maximum between-node heterogeneity.

For any split, we let $k = 1, 2$ denote the indicator of the two groups. Recall that in the competing risk setting, "failures" or "events" are categorized into several mutually exclusive types. Suppose there are J different event types ($j = 1, ..., J$). Without loss of generality, we assume that the event of interest is event 1 from this point forward and events $j = 2, ..., J$

are referred to as the competing risk events. Let T_{ik}^0, C_{ik} and $\delta_{ik}^0 \in \{1, \ldots, J\}$ be the true failure time, true censoring time and indicator of the true event type, respectively, for the ith individual ($i = 1, \ldots, n_k$) in group $k = 1, 2$. For each individual i in a set of data, we observe (T_{ik}, δ_{ik}), where $T_{ik} = \min(T_{ik}^0, C_{ik})$ and $\delta_{ik} = \delta_{ik}^0 I(T_{ik} \leq C_{ik})$. We assume that observations from different individuals within a group are independent and identically distributed, but we assume that the underlying competing risk processes for each individual are not independent. We also assume that the censoring time C_{ik} is independent of the failure time T_{ik}^0 for any event j.

For event type j and group k, the CIF is defined as $F_{jk}(t) = P(T_{ik}^0 \leq t, \delta_{ik}^0 = j)$. For each possible split of a covariate for a node, we need to test whether the CIFs are equal, or, equivalently, we need to test $H_0 : F_{11} = F_{12} = F_1^0$, where F_1^0 is an unspecified CIF and where the event of interest is of type 1. We assume that $F_{jk}(t)$ is continuous with subdensity $f_{jk}(t)$ with respect to the Lebesgue measure. We also define $S_k(t) = P(T_{ik}^0 > t) = 1 - \sum_j F_{jk}(t)$ as the survival function for individuals in group k. Without loss of generality, we can combine all competing events into one, thereby making $J = 2$. The Gray rank-based statistic can therefore be simplified as

$$z = \int_0^\tau K(t) \left[\left\{ 1 - \hat{F}_{11}(t-) \right\}^{-1} d\hat{F}_{11}(t) - \left\{ 1 - \hat{F}_{12}(t-) \right\}^{-1} d\hat{F}_{12}(t) \right], \tag{5.1}$$

where

$$\begin{aligned} \widehat{F}_{jk}(t) &= \int_0^t \widehat{S}_k(u-) Y_k^{-1}(u) dN_{jk}(u), \\ N_{jk}(t) &= \sum_{i=1}^{n_k} I(T_{ik} \leq t, \delta_{ik} = j), \\ Y_k(t) &= \sum_{i=1}^{n_k} I(T_{ik} \geq t), \end{aligned}$$

and where $\widehat{S}_k(t-)$ is the left-hand limit of the Kaplan-Meier survival estimator and $K(t)$ is a weight function with certain restrictions discussed below.

Suppose the improper random variable for event 1 failure times is defined as $X_{ik} = T_{ik}^0$ when $\delta_{ik}^0 = 1$ and $X_{ik} = \infty$ when $\delta_{ik}^0 \neq 1$. Then the quantity

$$R_k(t) = \frac{I(\tau_k \geq t) Y_k(t) \widehat{G}_{1k}(t-)}{\widehat{S}_k(t-)}$$

is an estimator of the expected number of X_{ik}'s that are still at risk at time t in group k when they are censored by C_{ik}(Gray, [14]). Therefore, the quantity $L(t)R_1(t)R_2(t)/\{R_1(t)+R_2(t)\}$ can be interpreted as an average of the number at risk multiplied by $L(t)$. We define the weight function in equation 5.1 to be this quantity and specify different forms of $L(t)$ to give different emphasis on the time differences (e.g., early difference and late difference). Gray specified $L(t)$ with the form

$$L(t) = \left\{\widehat{G}_1^0\right\}^{\rho}, \tag{5.2}$$

where $-1 \leq \rho \leq 1$, and

$$\widehat{G}_1^0 = 1 - \widehat{F}_1^0(t) = 1 - n^{-1}\int_0^t \widehat{h}_{\cdot}^{-1}(u)dN_{1\cdot}(u),$$

where $n = n_1 + n_2$, where $\widehat{h}_k(t) = n^{-1}I(t \leq \tau_k)Y_k(t)/\widehat{S}_k(t-)$, and where $\widehat{h}_{\cdot} = \widehat{h}_1 + \widehat{h}_2$.

The quantities τ_k are fixed times satisfying certain technical conditions (see Theorem 1 in Gray [14]), and can be interpreted as the largest possible event times for each group. Parameter ρ controls the assignment of weights with respect to time. If ρ is close to 1, more weight is given to the early time differences between the two CIFs. If ρ is close to -1, more weight is given to the late time differences.

To compare two CIFs after a split, we use the fact that $z_1^2/n\sigma_{11}^2$ follows a chi-square distribution with 1 degree of freedom asymptotically, where

$$\begin{aligned}\sigma_{kk'}^2 &= \sum_{r=1}^{2}\int_0^{\tau_k\wedge\tau_{k'}} a_{kr}(t)a_{k'r}(t)h_r^{-1}(t)dF_1^0(t) \\ &\quad + \sum_{r=1}^{2}\int_0^{\tau_k\wedge\tau_{k'}} b_{2kr}(t)b_{2k'r}(t)h_r^{-1}(t)dF_{2r}^0(t)\end{aligned} \tag{5.3}$$

and where

$$\begin{aligned}a_{kr}(t) &= d_{1kr}(t) + b_{1kr}(t), \\ b_{jkr}(t) &= \left\{I(j=1) - G_1^0(t)/S_r^0(t)\right\}\left\{c_{kr}(\tau_k) - c_{kr}(t)\right\}, \\ c_{kr}(t) &= \int_0^t d_{1kr}(u)d\Gamma_1^0(u), \text{and} \\ d_{jkr}(t) &= I(j=1)K_k^0(t)\left\{I(k=r) - h_r(t)/h_{\cdot}(t)\right\}/\Gamma_1^0(t).\end{aligned}$$

These quantities can be consistently estimated by substituting the following: $\widehat{h}_k(t)$ for $h_k(t)$; $\widehat{F}_1^0(t)$ for $F_1^0(t)$; $\widehat{F}_{jk}(t)$ for $F_{2r}^0(t)$; $\widehat{S}_r^0(t-)$ for $S_r^0(t-)$; $n^{-1}K_k$ for K_k^0; and $\widehat{\Gamma}_1^0(t) = \int_0^t [R_\cdot(u)]^{-1} dN_{1\cdot}(u)$ for $\Gamma_1^0(t)$.

To choose the split that corresponds to the largest Gray two-sample statistic, we compute the statistic for all possible splits on all the covariates for the whole sample, $\mathcal{L}$. If a covariate Z is continuous or ordinal, then we look at all the possible cutpoints c that divide the sample into two groups, $Z \leq c$ and $Z > c$. If a covariate is nominal, then we consider all the possible ways of forming the two groups. We repeat this process for all subsequent branches of the tree until we reach one of the predefined stopping criteria. The usual criteria are that the sample size in each terminal node is very small, the subgroups are homogeneous with respect to the covariates, or F_1 cannot be estimated (e.g., the subgroup contains no observations of the event of interest).

By using the splitting method, we can grow the largest possible tree, T_0, from the data. This tree will capture most the possible structures of the data and the dependencies among the covariates.

5.2.2 The Pruning Stage

The largest grown tree, T_0, is designed to perfectly or almost perfectly predict every case in a data set. However, if most or all of the splits of T_0 are essentially based on random noise, then the prediction results would be poor if we used T_0 to predict subgroups for another data set. To balance the trade-off between the fidelity and overfitting of the current data set, the usual approach is to prune the tree. We begin by removing the branches that result from random noise contained in the data used to build the largest grown tree (the training data). Then we build a sequence of subtrees, calculate their estimated amount of between-node difference, and select the subtree that maximizes total amount of discrimination with respect to the validation data.

To create a sequence of pruned subtrees from T_0, we use the complexity penalty method proposed by LeBlanc and Crowley [19]. Let T^i and T^t denote the sets of internal and terminal nodes of the tree T, respectively. Suppose $|\cdot|$ denotes the cardinality of a set—that

is, $|T^i|$ is equal to the total number of internal nodes, and $|T^t|$ is equal to the total number of terminal nodes for tree T. Let $G(T)$ be the amount of discrimination for tree T based on the training data, and let $\alpha|T^i|$ be the penalty that measures the complexity of T, where $\alpha > 0$ represents the tradeoff between the tree's prognostic information (or discrimination) and the tree's complexity. Then pruning the weakest branch of T is equivalent to maximizing the complexity penalty function so that

$$T^* = \arg\min_{T^* \prec T} G(T) - \alpha|T^i|.$$

where $\prec$ means "is a subtree of."

In our case, let $G(h)$ be the maximum Gray's two-sample statistic for node h, and let $G(T) = \sum_{h \in T^i} G(h)$ be the sum of these maximum statistics over all internal nodes. Let the complexity penalty function be denoted as $G_\alpha(T) = G(T) - \alpha|T^i|$. Our goal is to prune the tree branch that has the smallest $G_\alpha(T)$ value and is thus the weakest branch. For each node h, we need to solve the equation $G_\alpha(T_h) = G(T_h) - \alpha|T_h^i| = 0$ for α. In this case,

$$\alpha^* = \frac{G(T_h)}{|T_h^i|}.$$

As α increases from 0, we can find the first point of α at which the split makes a net negative contribution to the total amount of prognostic information in the tree.

We prune the split with the smallest $G(T_h)/|T_h^i|$ and then repeat this process until we have an ordered list of optimally pruned subtrees from the smallest tree (the root node tree, T_M) to the largest tree (T_0):

$$T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0,$$

and the corresponding ordered list of α^*'s is

$$\infty = \alpha_M^* > \alpha_{M-1}^* > ... > \alpha_1^* > \alpha_0^* = 0.$$

Now that we have constructed a sequence of subtrees and have calculated their estimated amount of discrimination, we need to select the final pruned tree to use. The tree of choice is the one that best maximizes the total amount of discrimination when applied to the validation data.

For validation, three main methods can be used: test-sample validation, cross-validation, and bootstrap. The test-sample validation method uses the current data set (the training data set) to grow and prune trees, and it uses an external data set (the validation data set) to assess the performance of the trees. The problem with this method is that an external data set is not always available. The other two methods do not require an external set. The cross-validation method divides the current data set into V subsets with nearly equal sample sizes. It removes one subset at a time and uses it as the validation data set. It uses the remaining subsets as the training data sets for the purpose of growing and pruning trees. After cycling through all the subsets in this manner, it combines the results by averaging the performance at each step. Although the cross-validation method yields results that are less biased than those of the test-sample validation method, it requires a larger sample size and tends to give estimates with higher variability (Breiman et al., [2]). The bootstrap method is another technique with less variability. However, it usually has greater bias than the cross-validation method (Breiman et al., [2]), and hence we propose using cross-validation.

For the cross-validation method, we grow an optimally pruned sequence of trees $T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0$ on the entire data $\mathcal{L}$. Then we divide the data set $\mathcal{L}$ into V groups $\mathcal{L}_v$ $(v = 1, ..., V)$, all of which have nearly equal sample size. We define $\mathcal{L}_{(v)} = \mathcal{L} - \mathcal{L}_v$. For each v, we use $\mathcal{L}_{(v)}$ to grow a nested sequence of optimally pruned trees

$$T_M^v \prec T_{M-1}^v \prec ... \prec T_1^v \prec T_0^v, \tag{5.4}$$

which we obtain by pruning using the corresponding $\alpha_m^{'} = \sqrt{\alpha_m^* \alpha_{m+1}^*}$'s, Note that the geometric mean of the α_m^*'s is the value of α that corresponds to the maximum $G_\alpha(T_m)$ statistic for $\alpha_m^* \leq \alpha < \alpha_{m+1}^*$ (Breiman et al., [2]). For data set $\mathcal{L}_v$, we calculate the sum of between-node statistics $G(T_m^v)$. To estimate $G(T_m)$, we average the sum over V sets

$$G^{CV}(T_m) = \frac{1}{V} \sum_{v=1}^{V} G(T_m^v) \tag{5.5}$$

and choose the $\alpha = \alpha^*$ that maximizes this average.

Our final tree, $T(\alpha_c)$, is chosen from the original list of optimally pruned subtrees grown on the whole sample $\mathcal{L}$, where α_c is some fixed value of the complexity penalty. The $T(\alpha_c)$

is the smallest optimally pruned subtree that has the maximum cross-validation adjusted discrimination $G_{\alpha}^{CV}(T)$ for $\alpha = \alpha_c$. This means $T(\alpha_c) \in \{T_M, ..., T_0\}$ is chosen such that

$$T(\alpha_c) = \arg \max_{T_m \in \{T_M, ..., T_0\}} G^{CV}\{T_m\} - \alpha_c |T_m^i|.$$

5.3 TREE TO MAXIMIZE WITHIN-NODE HOMOGENEITY

To design a tree that maximizes within-node homogeneity, we require a homogeneity measure that is appropriate for data with competing risks. Ordinary martingale residuals have been used for univariate survival data (Therneau et al. [33]), but these residuals do not take multiple types of events into account. Therefore, we propose a modified residual for use in the splitting procedure.

5.3.1 The Growing Stage

5.3.1.1 Sum-of-squares impurity function. We propose to use the event-specific martingale residual as an outcome measure for event 1 risk. For individual i in group k, this residual for the event of interest ($j = 1$) is defined as

$$\widehat{M}_{ik}^1 = I_{\{\delta_{ik}=1\}} - \widehat{\Lambda}_0^1(T_{ik}),$$

where $\widehat{\Lambda}_0^1(\cdot)$ is the estimated cause-specific cumulative hazard function. This function can be calculated by

$$\widehat{\Lambda}_0^1(T_{ik}) = \int_0^{T_{ik}} \frac{dN_{1k}(s)}{Y_k(s)},$$

where $N_{1k}(t)$ and $Y_k(t)$ are the number of event 1 and number of risks at time t for group k, respectively.

For a node h, we propose using an impurity function $\phi_{SS}(h)$ based on the sum of the squared martingale variations,

$$\phi_{SS}(h) = \sum_{i \in h} \left(\widehat{M}_{ih}^1 - \overline{M}_h^1 \right)^2, \tag{5.6}$$

where $\overline{M}_h^1 = \frac{1}{N_h} \sum_{i \in h} \widehat{M}_{ih}^1$.

Let s be a possible split for the parent node P, and let L and R be the corresponding left child node and right child node, respectively, for this split. To obtain the split that maximizes the within-node homogeneity, we need to find an s that has the largest reduction in within-node impurity from the parent node P. That is, we need to find an s that maximizes the impurity function $\Phi_{SS}(s, P) = \phi_{SS}(P) - \phi_{SS}(L) - \phi_{SS}(R)$. This process can be simplified as

$$\Phi_{SS}(s, P) = \frac{N_L N_R}{N_P} \left(\overline{M}_L^1 - \overline{M}_R^1 \right)^2.$$

We repeat this procedure for all subsequent nodes until we reach the stopping criteria described in Section 5.2.1. This recursive procedure grows the largest possible tree, T_0.

5.3.1.2 Absolute value impurity function.

The metric used by Therneau et al. [33] (and one that is often used in CART) is the sum-of-squares or $\mathcal{L}_2$ metric. However, time-to-event data or any skewed data is often better summarized using an absolute value or $\mathcal{L}_\infty$ because it is less susceptible to extreme values. Therefore we also propose using the following impurity function based on event-specific martingale residuals and the absolute value function:

$$\phi_{ABS}(h) = \sum_{i \in h} \left| \widehat{M}_{ih}^1 - \overline{M}_h^1 \right|. \tag{5.7}$$

As before, we maximize the reduction in the impurity function, $\Phi_{ABS}(s, P) = \phi_{ABS}(P) - \phi_{ABS}(L) - \phi_{ABS}(R)$, for each split when growing the tree T_0.

5.3.2 The Pruning Stage

The following holds for either impurity function $\phi(h)$ described above.

As before, we adjust for overfitting at the pruning stage. For pruning, we use the cost-complexity algorithm presented by Breiman et al. [2] to obtain an ordered list of optimally pruned subtrees, substituting our impurity function for node h, $\phi(h)$, based on event-specific residuals. The weakest branch to be pruned is defined by the following cost-complexity function of a tree T,

$$\phi_\alpha(T) = \sum_{h \in T^t} \phi(h) + \alpha |T^t|, \tag{5.8}$$

where α is a nonnegative complexity parameter. This quantity represents the total amount of impurity or "dissimilarity" among the individuals in the final subgroups (terminal nodes of the tree T) with respect to their event-specific martingale residuals, after being penalized for the number of terminal nodes.

The tree that minimizes the cost-complexity function (5.8) is the optimally pruned subtree. As α increases, there are "jumps" whereby a smaller subtree becomes the minimizing tree for a new value of α, say α^*. The α^*'s are defined as

$$\alpha^* = \frac{\phi(h) - \phi(T_h)}{|T_h^t| - 1},$$

where $\phi(T) = \sum_{h \in T^t} \phi(h)$ and where T_h is the tree that has root node h. The branch with the smallest cost-complexity is pruned recursively until we have a decreasing sequence of smallest optimally pruned subtrees

$$T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0$$

and the sequence of corresponding α^*'s is

$$\infty = \alpha_M^* > \alpha_{M-1}^* > ... > \alpha_1^* > \alpha_0^* = 0.$$

In order to use cross-validation to select the final tree, we grow an optimally pruned sequence of trees $T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0$ on the entire data $\mathcal{L}$. Then we divide the data set $\mathcal{L}$ into V groups $\mathcal{L}_v$ $(v = 1, ..., V)$, all of which have nearly equal sample size. We define $\mathcal{L}_{(v)} = \mathcal{L} - \mathcal{L}_v$. For each v, we use $\mathcal{L}_{(v)}$ to grow a nested sequence of optimally pruned trees

$$T_M^v \prec T_{M-1}^v \prec ... \prec T_1^v \prec T_0^v,$$

which we obtain by pruning T_0^v using $\alpha_m^{'} = \sqrt{\alpha_m^* \alpha_{m+1}^*}$.

For each tree T_m^v we calculate the sum of the squared martingale variations for data set $\mathcal{L}_v$, to get a more realistic estimate of the total impurity of each tree. We average the sum over V sets and choose the tree that minimizes the impurity, after adjusting for the number

of terminal nodes. Recall that the total impurity of a tree is $\phi(T) = \sum_{h \in T^t} \phi(h)$, where $\phi(h) = \sum_{i \in h}(M_i^1 - \overline{M}_h^1)^2$. We let

$$\phi(\mathcal{L}_1; \mathcal{L}_2, T) = \sum_{h \in T^t} \sum_{i \in h} \left\{ (\widehat{M}_i^1 - \overline{M}_h)^2 \right\}$$

be the value of ϕ for tree T when T is grown with $\mathcal{L}_2$ but evaluated on $\mathcal{L}_1$. Then we can calculate

$$\phi\left\{\mathcal{L}_v; \mathcal{L}_{(v)}, T_m^v\right\} = \sum_{h \in T^{vt}(\alpha)} \sum_{i \in h} \left\{ (\widehat{M}_i^1 - \overline{M}_h^1)^2 \right\}$$

for every value of v and take the average,

$$\phi^{CV}(T_m) = \frac{1}{V} \sum_{v=1}^{V} \phi\left\{\mathcal{L}_v; \mathcal{L}_{(v)}, T_m^v\right\}.$$

Our final tree, $T(\alpha_c)$, is chosen from the original list of optimally pruned subtrees grown on the whole sample $\mathcal{L}$. The $T(\alpha_c)$ is the smallest optimally pruned subtree that has the minimum cross-validated adjusted $\phi_\alpha(T)$ for $\alpha = \alpha_c$. This means it is

$$T(\alpha_c) = \arg \min_{T_m \in \{T_0, \ldots, T_M\}} \phi^{CV}(T_m) + \alpha_c |T_m^t|.$$

5.4 SIMULATIONS

In this section, we discuss a series of simulations designed to investigate how accurately the splitting method detects cutpoints of a covariate and how well the tree procedures detect the data structure. We compare the performance of four trees: our maximum between-node heterogeneity tree based on Gray's test (the "between-node tree"), our maximum within-node homogeneity tree based on an event-specific martingale residual impurity function with sums-of-squares (the "within-node SS tree"), our maximum within-node homogeneity tree based on an event-specific martingale residual impurity function with absolute deviations (the "within-node ABS tree"), and a between-node tree based on the naive log-rank test that was proposed by LeBlanc and Crowley [19] for univariate survival data (the "LR tree").

5.4.1 Performance of Splitting Methods

We generated data from four different models in which the CIF for the event of interest (event 1) was dependent on a covariate Z at a known cutpoint c_1. We used the between-node tree, within-node sum-of-squares tree, within-node absolute-value tree and LR tree to estimate the cutpoint, $\widehat{c}_1$, from the one-split tree grown from the data. The censoring rates were generated with the same distribution as the events 1 and 2, depending on whether the model were based on the exponential or log normal distributions. We repeated the simulation with low censoring (final proportion of censored events 10%) and moderate censoring (final proportion of censored events 50%). In all the models, covariate Z was generated from a standard uniform distribution.

Table 3 shows the setups for the four different models used in the simulation. The models had different cutpoints for changing CIFs. Each model was generated with different event 1 failure rates for $Z \leq c_1$ and for $Z > c_1$. These rates were chosen to approximate the real life situation of patients on the waiting list for a liver transplant. In models 1, 2, and 3, the failure times for event 1, event 2, and true censoring were generated from exponential distributions. In model 4, the failure times were generated from a log normal distribution with shape parameter $\sigma = 1$ and with different location parameters for $Z \leq c_1$ and for $Z > c_1$. In models 1, 2, and 4, different cutpoints for c_1 and c_2 were used. In model 3, the same cutpoints for c_1 and c_2 were used to investigate whether a change of cutpoint affects the accuracy of the estimates. Final proportions for event 2 varied between about 20% and 70%, depending on censoring rates and the rates of event 1.

The sample size for each data set was 200, and we generated 2000 data sets for each model configuration. To compare the performance of the three trees, we used four measures: the mean estimate $\overline{\widehat{c}_1}$ (mean), the mean bias $\overline{\widehat{c}_1 - c_1}$ (bias), the standard deviation of the $\widehat{c}_1$'s (SD), and the square root of the mean square error,

$$RMSE = \sqrt{\frac{1}{2000}\sum_{i=1}^{2000}(c_1 - \widehat{c}_1)^2}.$$

Table 4 summarizes the results for the one-split simulations. Here, Uni = Univariate tree for competing risks based on Gray's test, LR = univariate survival tree method based

Table 3: Description of simulations for splitting, univariate methods

Model	Cutpoint Event 1 c_1	Cumulative Incidence Function Event 1 $F_1(t\|Z \le c_1)$	$F_1(t\|Z > c_1)$	Cutpoint Event 2 c_2	Cumulative Incidence Function Event 2 $F_2(t\|Z \le c_2)$	$F_2(t\|Z > c_2)$	
Low censoring 10%							
1	0.5	exp(0.2)	exp(0.4)	0.3	exp(0.7)	exp(0.5)	
2	0.3	exp(0.4)	exp(0.2)	0.5	exp(0.5)	exp(0.7)	
3	0.5	exp(0.2)	exp(0.4)	0.5	exp(0.7)	exp(0.5)	
		$F_1(t\|Z \le c_1)$	$F_1(t\|Z > c_1)$		$F_2(t\|Z \le c_2)$	$F_2(t\|c_2 < Z \le c_1)$	$F_2(t\|Z > c_1)$
4	0.5	log norm(1.25)	log norm(0.5)	0.3	log norm(0.1)	log norm(0.5)	log norm(0.25)
Moderate censoring 50%							
1	0.5	exp(0.1)	exp(0.3)	0.3	exp(0.4)	exp(0.2)	
2	0.3	exp(0.3)	exp(0.1)	0.5	exp(0.2)	exp(0.4)	
3	0.5	exp(0.1)	exp(0.3)	0.5	exp(0.4)	exp(0.2)	
		$F_1(t\|Z \le c_1)$	$F_1(t\|Z > c_1)$		$F_2(t\|Z \le c_2)$	$F_2(t\|c_2 < Z \le c_1)$	$F_2(t\|Z > c_1)$
4	0.5	log normal(2)	log normal(0.75)	0.3	log norm(0.25)	log norm(0.75)	log norm(0.25)

Description of models used in simulations for the performance of splitting, univariate methods for competing risks

on log-rank test, MR-SS = univariate competing risk tree based on martingale residuals and sum-of-squares metric, MR-ABS = univariate competing risk tree based on martingale residuals and absolute value metric. The number of simulations is $B = 2000$, and sample size for each simulation is 200.

For models 1, 3, and 4, the mean of the $\widehat{c}_1$'s was used to compare the performance. For model 2, because the distribution of estimates for the cutpoints was right skewed, the median of the $\widehat{c}_1$'s is a more appropriate measure of central tendency. The between-node tree and the within-node trees provided more accurate estimates of c_1 than the LR tree did. This was expected because the LR tree does not take competing risks into account. In general, the between-node method (that takes into account competing risks via Gray's test) and the within-node methods provided the least biased estimates, with the within-node tree with the absolute value impurity function often performing the best. Distributions of the estimated cutpoints selected by the various methods for low censoring can be seen in Figure 2 and for moderate censoring in Figure 3. The distributions are all uni-modal, centered around the

Table 4: Estimated cutpoint from various methods, where one event of interest

	Low censoring 10%				**Moderate censoring 50%**			
Measure	Uni.	LR	MR-SS	MR-ABS	Uni.	LR	MR-SS	MR-ABS
Model 1: exponential, $c_1 = 0.5, c_2 = 0.3$								
Mean	0.5280	0.5396	0.5265	**0.5190**	**0.5225**	0.5516	0.5406	0.5432
Median	0.5133	0.5162	0.5111	**0.5099**	**0.5088**	0.5187	0.5161	0.5199
Bias	0.0280	0.0396	0.0265	**0.0190**	**0.0225**	0.0516	0.0406	0.0432
SD	0.1693	0.1644	0.1626	**0.1000**	0.1196	0.1287	0.1260	**0.0811**
RMSE	0.1716	0.1690	0.1647	**0.1017**	0.1216	0.1386	0.1323	**0.0919**
Model 2: exponential, $c_1 = 0.3, c_2 = 0.5$								
Mean	0.3361	**0.3320**	0.3391	0.3456	0.3042	0.2859	0.2886	**0.3016**
Median	**0.2968**	0.2951	0.2962	0.3116	0.2940	0.2832	0.2857	**0.2961**
Bias	0.0361	**0.0320**	0.0391	0.0456	0.0042	-0.0141	-0.0114	**0.0016**
SD	0.1733	0.1766	0.1770	**0.1203**	0.1083	0.1171	0.1126	**0.0763**
RMSE	0.1770	0.1795	0.1812	**0.1286**	0.1084	0.1179	0.1132	**0.0763**
Model 3: exponential, $c_1 = 0.5, c_2 = 0.5$								
Mean	0.5477	0.5390	0.5292	**0.5223**	**0.5380**	0.5472	0.5408	0.5418
Median	0.5169	0.5155	0.5134	**0.5114**	**0.5136**	0.5158	0.5150	0.5208
Bias	0.0477	0.0390	0.0292	**0.0223**	**0.0380**	0.0472	0.0408	0.0418
SD	0.1375	0.1656	0.1609	**0.0998**	0.1086	0.1330	0.1270	**0.0828**
RMSE	0.1455	0.1701	0.1635	**0.1022**	0.1150	0.1411	0.1334	**0.0927**
Model 4: log normal, $c_1 = 0.5, c_2 = 0.3$								
Mean	**0.5157**	0.5452	0.5228	0.5174	0.5274	0.5363	0.5180	**0.5165**
Median	**0.5028**	0.5140	0.5073	0.5073	0.5090	0.5121	**0.5066**	0.5071
Bias	**0.0157**	0.0452	0.0228	0.0174	0.0274	0.0363	0.0180	**0.0165**
SD	0.1554	0.1161	0.1024	**0.0608**	0.1106	0.0735	0.0536	**0.0371**
RMSE	0.1562	0.1246	0.1049	**0.0632**	0.1139	0.0820	0.0565	**0.0406**

Accuracy measures for the estimated cutpoint, $\widehat{c}_1$, from various univariate methods. The median is best estimate for skewed distributions of $\widehat{c}_1$ when $c_1 = 0.3$. Bold-faced values highlight the best result for each model and outcome measure.

true cutpoint, with a skewed distribution for model 2 where the true cutpoint is not in the middle of the $[0, 1]$ interval ($c_1 = 0.3$).

5.4.2 Performance of Methods to Detect Data Structure

Table 5 shows the model configurations that we used to investigate the ability of a tree to detect the structure in data with competing risks. Models 1 and 2 were generated from exponential distributions, and models 3 and 4 were generated from log normal distributions. For each model, two covariates (Z_1 and Z_2) were related to the survival times, and four other covariates (Z_3–Z_6) were independent of the survival times. Covariates Z_1, Z_3, and Z_4 were binary variables generated from a binomial distribution with $p = 0.5$. Covariates Z_2, Z_5, and Z_6 were variables generated from a discrete uniform distribution over the points 0, 0.25, 0.5, 0.75, and 1. For all models, the final censoring rate was 50%. Models 1 and 3 had an interaction effect between Z_1 and Z_2, so a tree with three terminal nodes would be expected to capture this data structure. Models 2 and 4 had additive hazards for event 1 based on Z_1 and Z_2, so a tree with four terminal nodes would be expected to capture this data structure.

We constructed 100 trees with a sample size of $N = 500$ per model in each simulation. We began by comparing the performance of trees in terms of three different measures. The first measure was $|T^t|$, the number of terminal nodes in each tree. The second measure was the number of times both Z_1 and Z_2 were chosen (this is referred to as the "inclusive" signal for variable selection), and the third measure was the number of times that only Z_1 and Z_2 were chosen (this is referred to as the "exclusive" signal for variable selection). Next, to compare predictive ability of the trees, we generated validation data sets with 500 observations from the same model (but with no censoring) for each simulation. We calculated the mean absolute difference between the event 1 failure time for each observation in a validation data set and the median event 1 failure time predicted by the tree. We calculated the mean absolute error for event 1 times and event 2 times, respectively, as

$$\text{MAE}_1 = \frac{1}{N_1} \sum_{h=1}^{|T^t|} \sum_{i=1}^{N_{1h}} |T_{i1h} - \widehat{\tau}_{1h}|,$$

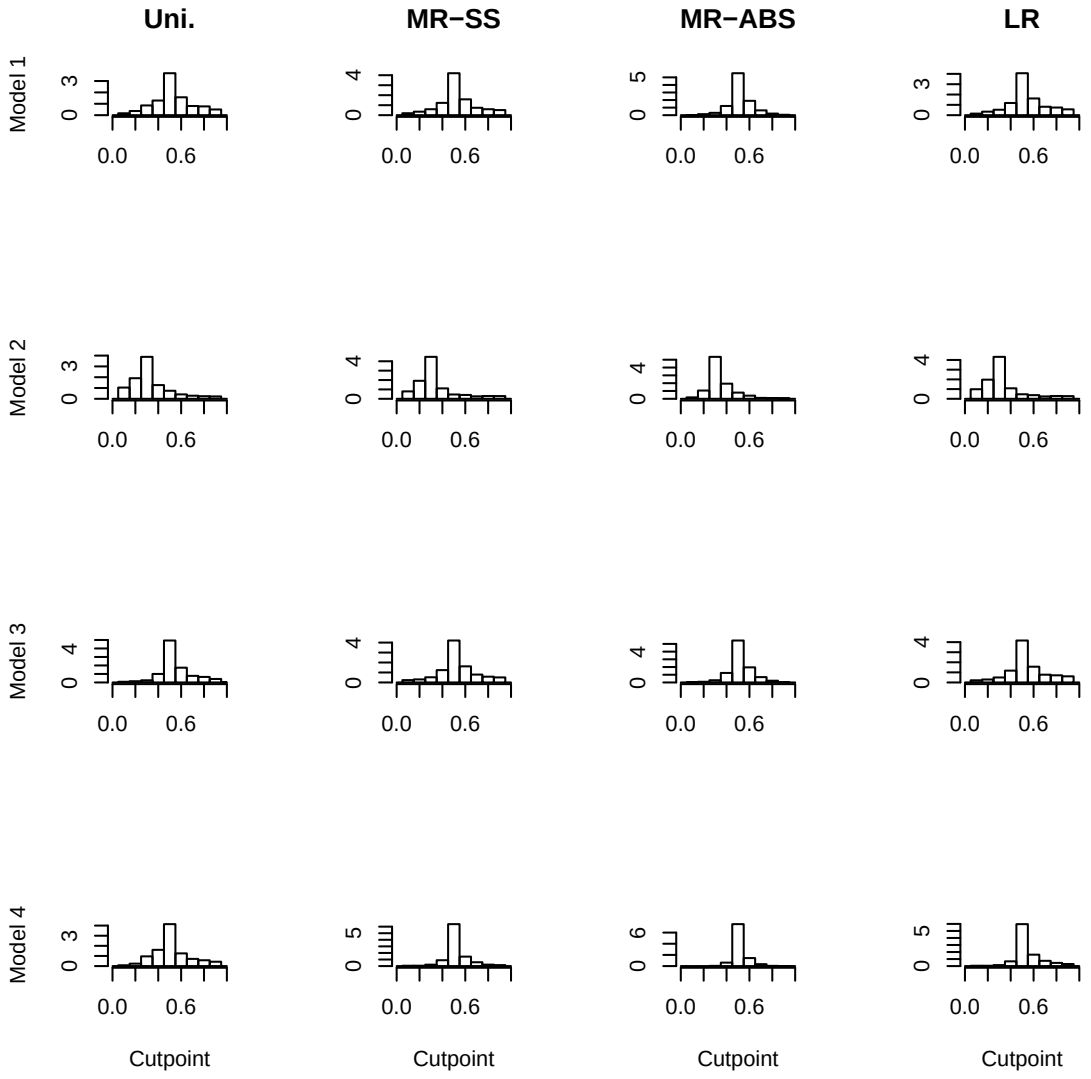


Figure 2: Univariate methods, distribution of estimated cutpoints, 10% censoring

Distribution of estimated cutpoints $\hat{c_1}$ for Models 1-4 for four methods: Between-node (Uni), Within-node sums-of-squares impurity (MR-SS), Within-node absolute value impurity (MR-ABS) and Log-Rank method (LR), using 10% censoring.

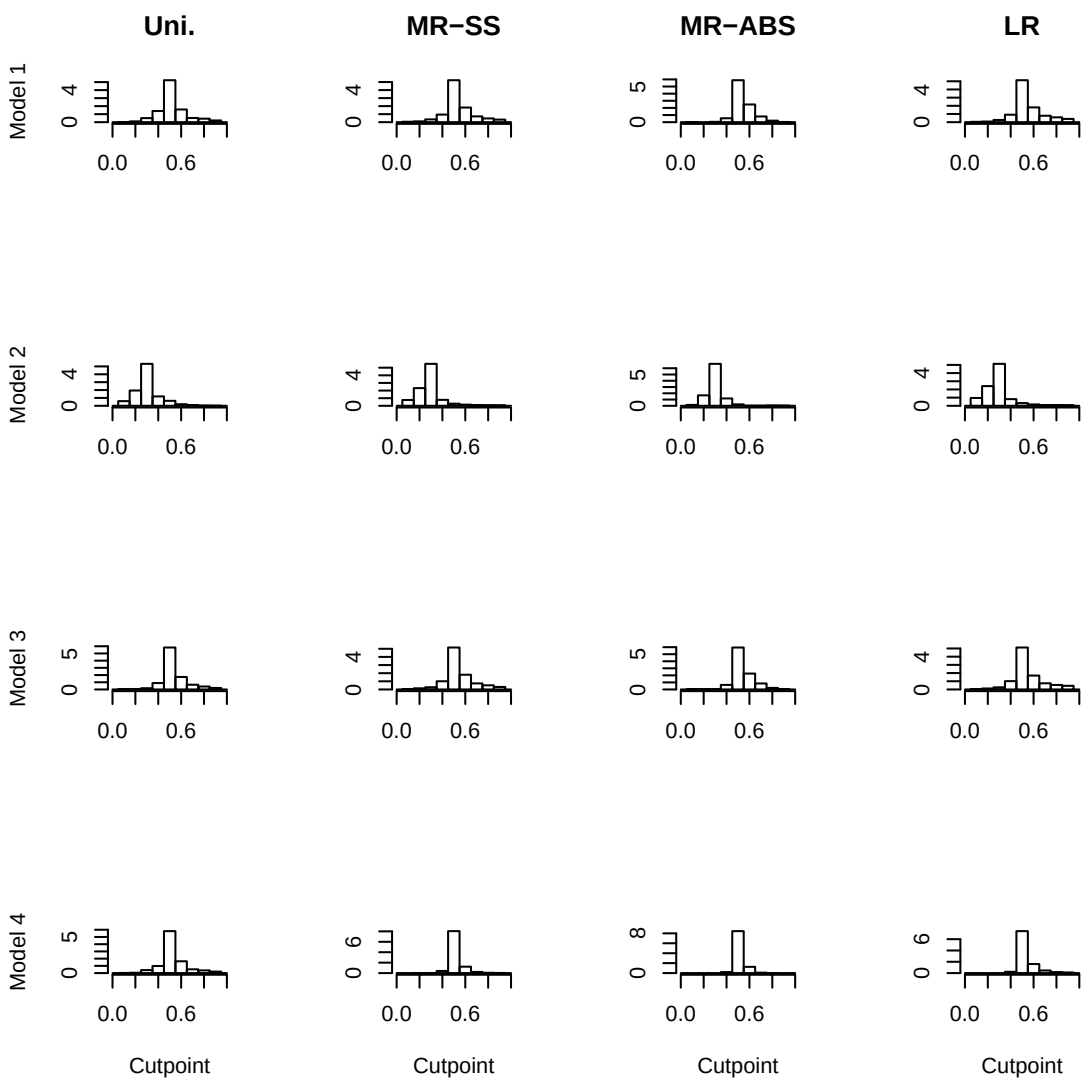


Figure 3: Univariate methods, distribution of estimated cutpoints, 50% censoring

Distribution of estimated cutpoints $\hat{c}$ for Models 1-4 for four methods: Between-node (Uni), Within-node sums-of-squares impurity (MR-SS), Within-node absolute value impurity (MR-ABS) and Log-Rank method (LR), using 50% censoring.

Table 5: Description of models used in univariate simulations for data structure

Model	Cumulative incidence function $F_1(t\|Z_1, Z_2)$	Expected $\|T^t\|$
1	Exponential	3
	$\lambda = 0.1 + 0.35I\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$	
2	Exponential	4
	$\lambda = 0.05 + 0.2I(Z_1 > 0.5) + 0.2Z_2$	
3	Log normal	3
	$\mu = 2 - 1.5\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$, $\sigma = 1$	
4	Log normal	4
	$\mu = 2 - 0.85I(Z_1 > 0.5) - 0.85Z_2$, $\sigma = 1$	

$$\text{MAE}_2 = \frac{1}{N_2} \sum_{h=1}^{|T^t|} \sum_{i=1}^{N_{2h}} |T_{i2h} - \widehat{\tau}_{2h}|,$$

where N_{1h} is the number of type 1 events in node h, N_1 is the total number of type 1 events, T_{i1h} is the ith failure time for event 1 in terminal node h, and $\widehat{\tau}_{1h}$ is the median event 1 failure time based on the training data for terminal node h. This function represents the difference between the observed event 1 failure time (T_{i1h}) and the predicted event 1 failure time ($\widehat{\tau}_{1h}$), for an observation in terminal node h. The quantities for event 2 are defined similarly. We used $\alpha = 1$ for the complexity penalty and the weighting parameter $\rho = -1$ in the Gray's test. We allowed the minimum number of observations in each terminal node to be 20. We used 10-fold cross-validation method to select the final tree.

Table 6 demonstrate the results of tree selection. The four methods used were: Between Uni = univariate between-node competing risks (Gray test), Survival Uni = univariate survival (log-rank), univariate within-node competing risks sums-of-squares impurity function = Within-SS, univariate within-node competing risks absolute value impurity function = Within-ABS. The within-node trees did the best in detecting the data structure. In general,

both the between-node and within-node trees outperformed the survival tree, which tends to have high variability with respect to the number of terminal nodes. The within-node trees tended to overfit a little (more terminal nodes than needed to capture data structure), while the between-node tree and the LR tree tended to underfit. In general, the within-node tree was more conservative in terms of selecting more variables than the between-node tree. The inclusive signal (percent of times Z_1 and Z_2 were selected) and exclusive signals (percent of times *only* Z_1 and Z_2 were selected) were the highest for the within-node methods. Comparing within-node methods, Within-SS (sum-of-squares impurity function) performed slightly better in terms of percent terminal nodes and the inclusive/exclusive signals. The between-node and within-node trees used in conjunction should be helpful in identifying important prognostic factors, particularly when there are interaction effects among the covariates.

An identical simulation situation was run, but with $N = 1000$ (instead of $N = 500$) and the results are shown in Table 7. All other parameters for the simulation were identical to those in the previous simulation (Table 6). We see similar results to the situation for $N = 500$, with the notable exception that the performance of both of the within-node methods appears to improve for the multiplicative models (where 3 terminal nodes ideally chosen for these models). For the multiplicative models, 3 terminal nodes are chosen 80-90% of the time when $N = 1000$ compared to around 30% when $N = 500$. The inclusive and exclusive signals increase accordingly to around 90-100%. This implies that the performance of the within-node trees improves with sample size.

5.5 APPLICATION TO DATA FROM THE LIVER TRANSPLANT WAITLIST

To illustrate the use of two competing risk tree methods (a between-node and within-node method), we applied them to a data set concerning patients awaiting a liver transplant. In the United States, patients who have end-stage liver disease and meet the eligibility requirements for a transplant are placed on the waiting list for a donated liver. The United Network for Organ Sharing (UNOS) manages the waiting list, matches donated organs with

Table 6: Investigating tree structure with one event-of-interest, N=500

	Percent terminal nodes in final tree							Variable Selection		Average	
Method	1	2	3	4	5	6	≥ 7	Inc	Exc	MAE1	MAE2
Model 1: Exponential $\lambda = 0.1 + 0.35I\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$											
Between Uni	0	94	**4**	2	0	0	0	4	2	1.5257	**1.5218**
Survival Uni	0	86	**3**	3	3	2	3	9	0	1.5257	1.5230
Within-SS	0	0	**31**	29	20	17	3	**100**	**36**	**1.5230**	1.5255
Within-ABS	1	0	**29**	29	31	10	0	98	33	1.5240	1.5316
Model 2: Exponential $\lambda = 0.05 + 0.2I(Z_1 > 0.5) + 0.2Z_2$											
Between Uni	3	89	5	**3**	0	0	0	7	6	**1.5328**	1.5332
Survival Uni	6	70	3	**3**	2	4	12	24	5	1.5340	1.5337
Within-SS	0	5	24	**27**	27	8	9	**84**	**27**	1.5365	**1.5290**
Within-ABS	8	26	30	**27**	8	1	0	45	26	1.5372	1.5298
Model 3: Log normal $\mu = 2 - 1.5\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$, $\sigma = 1$											
Between Uni	0	94	**5**	1	0	0	0	4	3	**1.7088**	1.9842
Survival Uni	0	80	**8**	3	0	3	6	17	6	1.7095	1.9876
Within-SS	0	0	**19**	25	37	17	2	**100**	24	1.7132	**1.9620**
Within-ABS	0	0	**20**	30	36	12	2	99	**26**	1.7146	1.9671
Model 4: Log normal $\mu = 2 - 0.85I(Z_1 > 0.5) - 0.85Z_2$, $\sigma = 1$											
Between Uni	1	93	4	**2**	0	0	0	6	5	**1.3295**	1.4874
Survival Uni	0	78	4	**1**	2	2	13	21	4	1.3324	1.4902
Within-SS	0	3	21	**32**	23	16	5	**97**	37	1.3299	1.4841
Within-ABS	0	5	36	**28**	22	9	0	89	**38**	1.3336	**1.4805**

The bold-faced numbers represent either 1) the ideal number of nodes selected (3 for multiplicative hazard, 4 for additive hazard), or 2) the best result for each model. The variable selection measure "Inc" or "Inclusive signal" represents the percent of trees that selected both variables Z_1 and Z_2. The variable selection measure "Exc" or "Exclusive signal" represents the percent of trees that selected only Z_1 and Z_2.

Table 7: Investigating tree structure with one event-of-interest, N=1000

	Percent terminal nodes in final tree							Variable Selection		Average	
Method	1	2	3	4	5	6	≥ 7	Inc	Exc	MAE1	MAE2
Model 1: Exponential $\lambda = 0.1 + 0.35I\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$											
Between Uni	0	89	**3**	6	0	2	0	9	2	1.5148	1.5100
Survival Uni	0	62	**3**	4	3	1	27	35	3	1.5160	1.5110
Within-SS	0	1	**87**	10	2	0	0	99	**88**	1.5143	**1.5094**
Within-ABS	0	0	**82**	15	3	0	0	**100**	86	**1.5142**	1.5100
Model 2: Exponential $\lambda = 0.05 + 0.2I(Z_1 > 0.5) + 0.2Z_2$											
Between Uni	0	94	1	**3**	0	0	2	6	4	**1.5169**	**1.5352**
Survival Uni	0	42	0	**2**	2	1	53	58	0	1.5224	1.5375
Within-SS	0	13	63	**19**	4	1	0	**87**	**78**	1.5189	1.5357
Within-ABS	6	39	53	**1**	1	0	0	52	50	1.5177	1.5358
Model 3: Log normal $\mu = 2 - 1.5\{(Z_1 > 0.5) \cap (Z_2 > 0)\}$, $\sigma = 1$											
Between Uni	0	76	**7**	11	5	0	1	22	11	**1.7081**	2.0240
Survival Uni	1	51	**5**	14	5	3	21	44	4	1.7179	2.0251
Within-SS	0	0	**92**	5	2	0	1	**100**	**93**	1.7107	2.0177
Within-ABS	0	0	**85**	7	7	1	0	**100**	88	1.7111	**2.0176**
Model 4: Log normal $\mu = 2 - 0.85I(Z_1 > 0.5) - 0.85Z_2$, $\sigma = 1$											
Between Uni	0	86	4	**4**	1	3	2	14	6	1.3091	1.4898
Survival Uni	0	52	4	**2**	1	7	34	48	5	1.3138	1.4937
Within-SS	0	2	58	**21**	11	8	0	**98**	**77**	**1.3087**	**1.4792**
Within-ABS	0	1	73	**17**	9	0	0	**98**	**77**	1.3150	1.4804

The bold-faced numbers represent either 1) the ideal number of nodes selected (3 for multiplicative hazard, 4 for additive hazard), or 2) the best result for each model. The variable selection measure "Inc" or "Inclusive signal" represents the percent of trees that selected both variables Z_1 and Z_2. The variable selection measure "Exc" or "Exclusive signal" represents the percent of trees that selected only Z_1 and Z_2.

the patients who need them, and maintains the national database of pretransplant and follow-up information. When organs become available, UNOS allocates them to patients who have the highest risk of pretransplant death, based on scores derived from the model for end-stage liver disease (MELD). In this model, death is the event of interest. Censoring is applied to receipt of a transplant, removal from the list because of a loss to follow-up, or a change in the desire or need for a transplant. For reasons outlined in Sections 5.1 and 3.4 this approach to censoring may introduce bias in the results.

We used UNOS data regarding 1203 patients who were on the waiting list from April 1990 to June 1994 and were followed until April 1997. Of the 1203 patients, 342 experienced death before transplant (pretransplant death), 224 received a transplant, and 637 were removed from the list for other reasons. For our analysis, we treated pretransplant death as the event of interest (event 1), removal from the list for other reasons as competing risks (event 2), and lack of an event as true censoring (event 0). We used the following covariates for Z_1 through Z_{14}, respectively: age, serum albumin level, serum alkaline phosphatase level, serum creatinine level, diabetes (no/yes), dialysis for renal disease (no/yes), muscle wasting (no/yes), prothrombin time, serum glutamic-oxaloacetic transaminase (SGOT) level, total bilirubin level, total protein level, ulcerative colitis (no/yes), history of variceal bleeding (no/yes), and white blood cell count. For each tree, we ranked the terminal nodes from the highest risk (I) to the lowest risk (VIII) of event 1, based on the probability of 100-day failure for event 1.

We used the between-node tree method for competing risks based on Gray's test and the within-node tree method based on sums-of-squares (the within-node method based on absolute value impurity performed very similarly in the simulations to the within-SS tree). Figures 4 and 5 shows the final trees selected by the between-node and within-node-SS methods, Table 8 compares the summary data for the terminal nodes of these trees, and Figures 6 and 7 plots the CIFs for the nodes.

The final between-node tree (Figure 4) had four terminal nodes, splitting on the albumin level (albumin $\leq$3.05 vs. $>$3.05 g/dL), the bilirubin level ($\leq$22.1 vs. $>$22.1 μmol/L), and age ($\leq$44.4 vs. $>$44.4 years). Node 5 (albumin $\leq$3.05 and bilirubin $>$22.1) had the highest risk, followed by node 9 (albumin $\leq$3.05, bilirubin $\leq$22.1, and age $>$44.4), node 8 (albumin

$\leq$3.05, bilirubin $\leq$22.1, and age $\leq$44.4), and node 3 (albumin >3.05). This suggests that the combination of low albumin and high bilirubin levels places patients at the highest risk of death while on the waiting list. The combination of low albumin and bilirubin levels and older age also places patients at elevated risk of death. However, a high albumin level has a protective effect.

Like the between-node tree (Figure 4), the within-node tree (Figure 5) chose the albumin level of 3.05 as the first split. But in addition to splitting on albumin, bilirubin, and age, the within-node tree split on prothrombin time (measured in seconds) and white blood cell count (measured as cells 10^3/L). The final within-node tree had 8 terminal nodes, with node 11 showing the highest risk and node 7 showing the lowest risk. The results suggest that the highest risk is for patients who have a low albumin level ($\leq$3.05), high bilirubin level (>19.85), and high prothrombin time (>15.05). The lowest risk is for patients with a very high albumin level (>3.45). In general, the risk was highest for older versus younger patients (node 9 vs. 8 and node 21 vs. 20) and for individuals with high versus low white blood cell counts (node 13 vs. 12). Because three of the four highest levels of risk (I, II, and IV) occurred in the tree branch associated with low albumin level and high prothrombin time, the combination of these findings appears to greatly increase the risk of death.

5.6 DISCUSSION

Classification trees for survival data are useful for categorizing patients into several groups with similar risks. Some trees have been developed for multivariate survival data, but none of these trees can be applied to competing risks. We proposed two tree-based techniques specifically designed for data with competing risks. The tree that maximizes the between-node difference is based on Gray's test for CIFs, while the tree that minimizes the within-node impurity takes advantage of the highly successful CART techniques developed by Breiman et al. [2].

In general, we found that both of the new trees performed better than the classification tree that failed to take competing risks into account. The within-node tree tended to overfit

Table 8: Summary of the terminal nodes generated from the between-node and the within-node univariate trees for data from the liver transplant waiting list

Terminal Node	**Sample size (proportion)**				**Median Survival Time**		Risk Group
	Total	Event 1	Event 2	Censored	Event 1	Event 2	
Between-node tree							
3	596	122(0.21)	135(0.23)	339(0.57)	144	374	IV
5	40	22(0.55)	2(0.05)	16(0.40)	28.5	188.5	I
8	163	32(0.20)	37(0.23)	94(0.58)	80	369	II
9	404	166(0.41)	50(0.12)	188(0.47)	82.5	363.5	III
Within-node tree							
4	309	46(1.00)	80(0.23)	183(0.57)	171.5	375	VIII
10	267	64(0.55)	52(0.05)	151(0.40)	139.5	367	VII
11	20	12(0.20)	3(0.23)	5(0.58)	76.5	15	IV
12	135	21(0.41)	31(0.12)	83(0.47)	136	388	VI
13	252	100(0.21)	32(0.23)	120(0.57)	103	351.5	V
28	133	42(0.55)	21(0.05)	70(0.40)	67.5	314	III
29	53	32(0.20)	5(0.23)	16(0.58)	36	168	II
15	34	25(0.41)	0(0.12)	9(0.47)	22	—	I

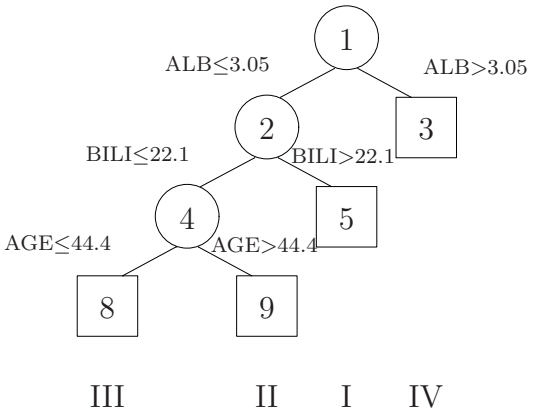


Figure 4: The between-node tree for data from the liver transplant waiting list.

The between-node tree for data from the liver transplant waiting list. Roman numerals represent the 100-day risk (failure probability) for each terminal node, with risk ranging from highest (I) to lowest (VIII). AGE = age in years, ALB = serum albumin level, BILI = total bilirubin level, PTP = prothrombin time, and WBC = white blood cell count.

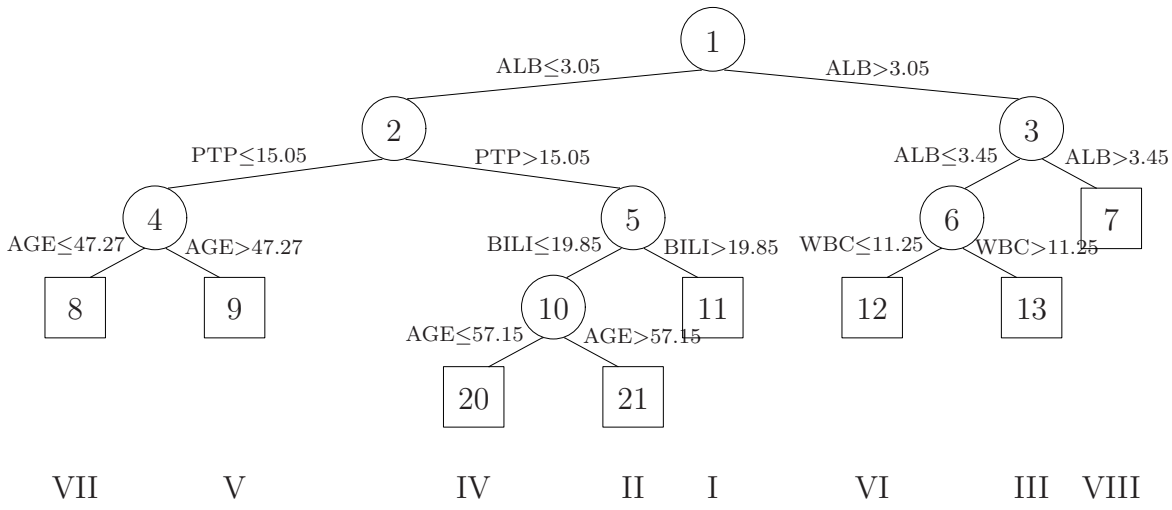


Figure 5: The within-node tree for data from the liver transplant waiting list.

The within-node tree for data from the liver transplant waiting list. Roman numerals represent the 100-day risk (failure probability for pretransplant death) for each terminal node, with risk ranging from highest (I) to lowest (VIII). AGE = age in years, ALB = serum albumin level, BILI = total bilirubin level, PTP = prothrombin time, and WBC = white blood cell count.

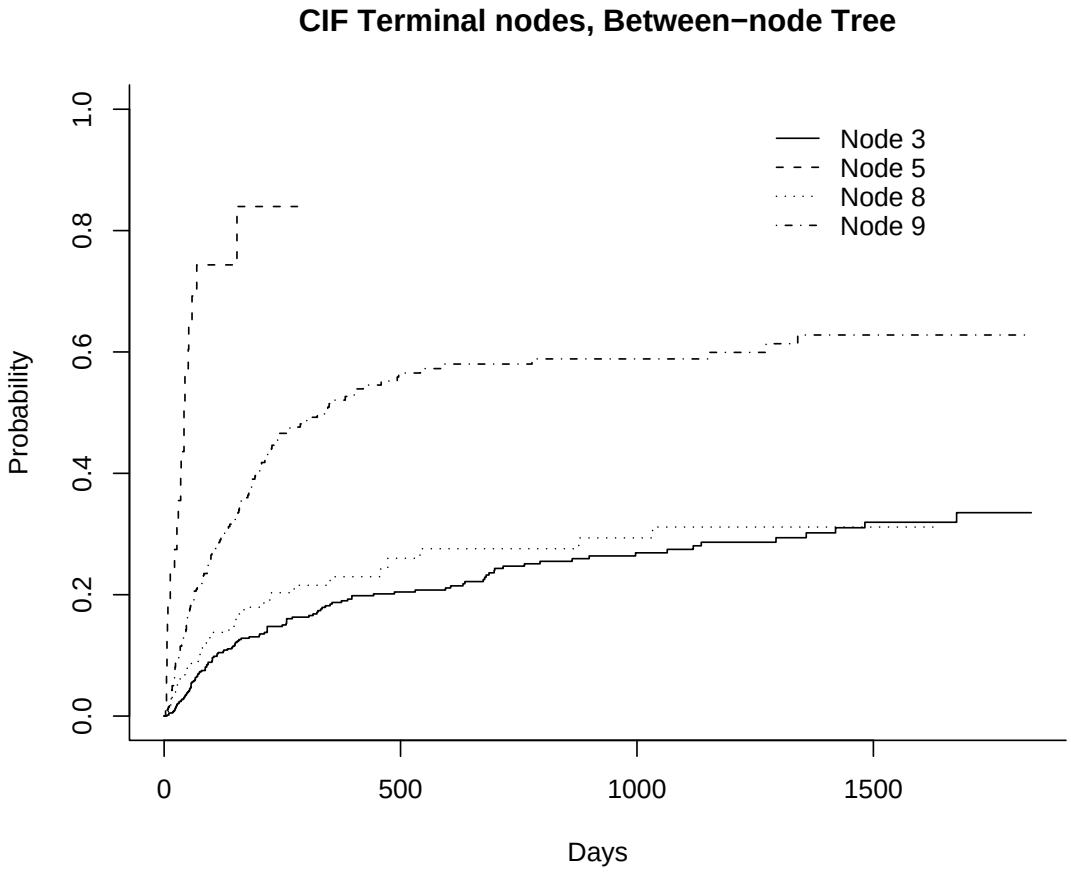


Figure 6: CIF of pre-transplant death for each terminal node of the between-node tree.

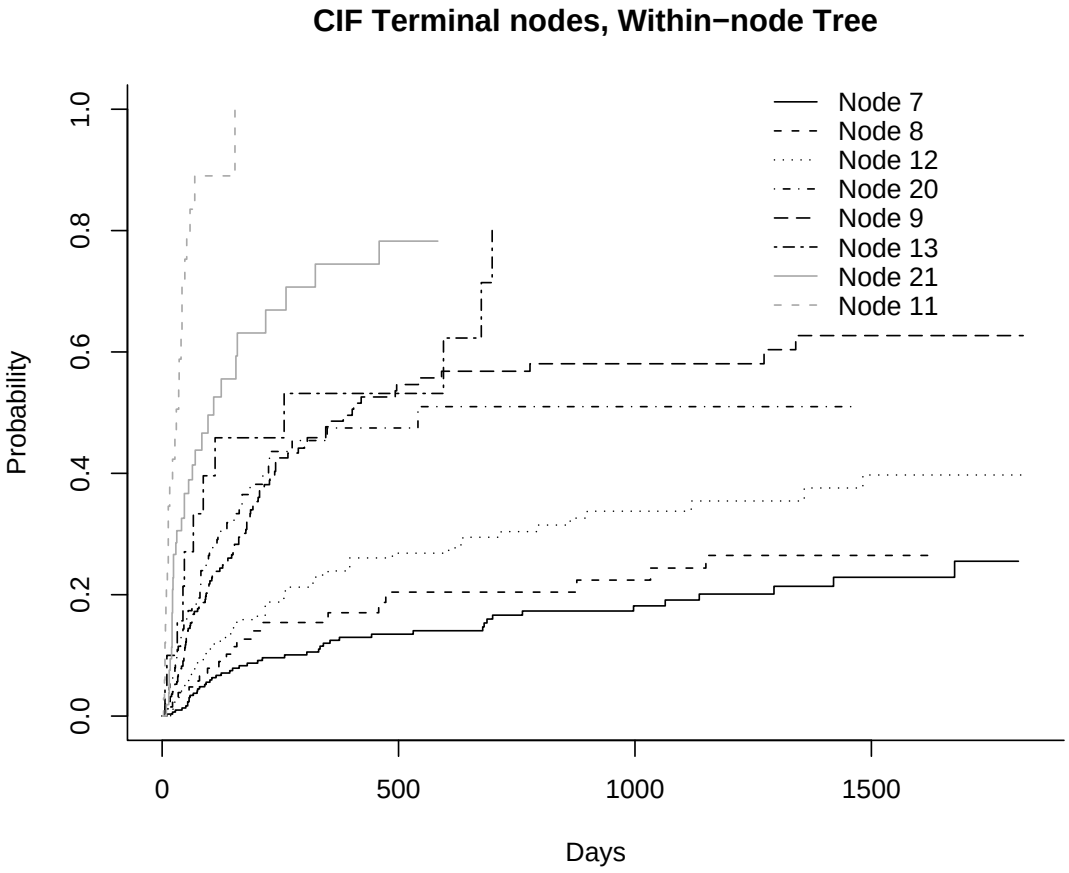


Figure 7: CIF of pre-transplant death for each terminal node of the within-node tree.

the data, while the between-node tree tended to underfit the data. The within-node tree methods performance improved further when the sample size increased. Together, however, these trees can be used not only to identify subgroups in complex data sets but also to identify situations in which the effect of one covariate is conditional on the level of another covariate. In the future, we will explore other ways of creating the impurity function. We will also expand our work to the competing risk-adjusted rank-based test for the equality of survival functions.

6.0 MULTIVARIATE TREE METHOD

Three stages are involved in constructing a tree: growing, pruning and tree selection. In this chapter, we describe the methods used in the stages of building a multivariate tree that maximizes between-node heterogeneity for data with multiple competing risks of interest.

6.1 INTRODUCTION

Univariate trees that account for competing risks are not appropriate for the situation where there is more than one event of interest. With regard to the liver transplant example, removal from the waiting list due to death or removal due to transplant are both outcomes of considerable interest to physicians. The physician may want to identify subgroups of patients that have similar risk with respect to both events. When only one of these outcomes is considered at a time (using the univariate competing risk tree), the subgroups generated by the univariate tree are likely to be heterogeneous with respect to the risk of the other events. Furthermore, a patient may fall into two different risk categories depending on which tree generated the subgroups, making it unclear which tree to choose to base the final decision. It would be helpful to have a classification technique that discriminates based on the risk of both events simultaneously.

In this chapter we introduce a novel multivariate classification tree for data with competing risks for the situation where more than one competing risk is of interest to the researcher. This method uses a between-node, composite measure of the risk of two more events based on the multivariate test of cumulative incidence functions (Luo and Turnbull [20]). In Sec-

tions 6.2 and 6.3, we outline the methods for splitting, growing, pruning, and selecting the final tree. In Section 6.4, we describe the simulations that we performed to compare our new multivariate tree with univariate competing risk trees (Chapter 5), and with a univariate between-node tree based on log-rank tests (Le Blanc and Crowley [19]). In Section 6.5, we illustrate our new techniques by analyzing data from patients who had end-stage liver disease and were on the waiting list for receipt of a liver transplant. In Section 6.6, we discuss implementation and future directions of our methods.

6.2 THE GROWING STAGE

We begin by selecting a splitting function based on event-specific cumulative incidence functions (CIFs). The CIF, also known as a subdistribution function, is a marginal probability of failure for the event of interest. Instead of using the overall survival function, the CIF is used to summarize data with competing risks. After splitting the covariate, we use a method based on one proposed by Luo and Turnbull [20] to compare the CIFs for event 1 and 2 for group 1, with the CIFs for event 1 and 2 for group 2. In this method, Luo and Turnbull extend the rank-based tests for data with a single event (e.g., the log-rank test, the generalized Wilcoxon test, and the Fleming-Harrington test) to apply to data with multivariate competing risks of interest. For our method, we repeated the process of splitting and comparing CIFs until we find the two CIFs with the maximum between-node heterogeneity.

For any split, we let $k = 1, 2$ denote the two groups. Recall that in the competing risk setting, "failures" or "events" are categorized into several mutually exclusive types. Suppose there are $j = 1, ..., J$ different event types. Without loss of generality, we assume that there are two events of interest, $J = 2$, and censoring, but the following can be generalized to three or more events. Let T_{ik}^0 and C_{ik} be the true failure time and censoring time, respectively, for the ith individual ($i = 1, \ldots, N$) in group k. Let $\delta_{ik}^0 \in \{1, \ldots, J\}$ be the indicator of the true event type. For each individual i in a set of data, we observe (T_{ik}, δ_{ik}), where $T_{ik} = \min(T_{ik}^0, C_{ik})$ and $\delta_{ik} = \delta_{ik}^0 I(T_{ik} \leq C_{ik})$. We assume that observations from different individuals within a group are independent and identically distributed, but we assume that

the underlying competing risk processes for each individual are not independent. We also assume that the censoring time C_{ik} is independent of the failure time T^0_{ik} for any event j.

For event j and group k, the CIF is defined as $F_{jk}(t) = P(T^0_{ik} \leq t, \delta^0_{ik} = j)$. For each possible split of a covariate for a node, we need to test whether the CIFs are equal for each event j – that is, we need to test

$$\begin{aligned} H_0 : F_{11} &= F_{12} \\ F_{21} &= F_{22} \end{aligned}$$

versus

$$H_1 : F_{jk} \neq F_{jk'} \text{ for at least one } j \text{ and } k \neq k'.$$

We assume that $F_{jk}(t)$ is continuous with respect to the Lebesgue measure, with sub-density $f_{jk}(t)$. We also define $S_k(t) = P(T^0_{ik} > t) = 1 - \sum_j F_{jk}(t)$ as the survival function for event j and for individuals in group k. Let n_k be the sample size from the k^{th} sample. The j^{th} component of the multivariate rank-based test statistic is

$$X_j(\tau) = \sqrt{\frac{n_1 n_2}{n_1 + n_2}} \int_0^\tau \widehat{W}_j(s) d\Big[\widehat{F}_{j1}(s) - \widehat{F}_{j2}(s)\Big], \tag{6.1}$$

where

$$\begin{aligned} \widehat{F}_{jk}(s) &= \int_0^t \frac{\hat{S}_k(s)}{\bar{Y}_k(s)} d\bar{N}_{jk}(s) \\ \hat{S}_k(t) &= \exp\Big\{ -\sum_{j=1}^{J} \int_0^t \frac{1}{\bar{Y}_k(s)} d\bar{N}_{jk}(s) \Big\} \\ \bar{Y}_k(s) &= \frac{1}{n_k} \sum_{i=1}^{n_k} I(T_{ik} \geq t) \\ \bar{N}_{jk}(s) &= \frac{1}{n_k} \sum_{i=1}^{n_k} I(T_{ik} \leq t, \delta_{ik} = j) \end{aligned}$$

and $\widehat{W}_j(s)$ is a weight function that converges to some weight, $W_j(s)$, in probability. We consider weights in the same class as those proposed by Gray [14],

$$\widehat{W}(t) = (1 - \hat{F}_j(t))^r \tag{6.2}$$

where $\hat{F}_j(t)$ is the estimated CIF for event j when data from group 1 and group 2 are combined. The parameter r controls the assignment of the weights with respect to time. When $r = -1$, more weight is given to early differences; when $r = 1$, more weight is given to late differences. τ denotes the "horizon" time; a fixed, final time-point.

To compare two sets of CIFs after a split, we use the fact that $\mathbf{T} = (T_1, T_2) = (X_1/\sigma_1, X_2/\sigma_2)$ follows a bivariate normal distribution asymptotically under the null,

$$\begin{pmatrix} T_1 \\ T_2 \end{pmatrix} \sim \text{BivNorm}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right)$$

for some ρ. We can estimate σ_1, σ_2, the covariance C and correlation ρ with the following consistent estimators,

$$\begin{aligned}
\hat{\sigma}_1^2 &= \frac{n^{(1)}n^{(2)}}{n^{(1)}+n^{(2)}} \sum_{l=1}^{J} \int_0^\tau \left[\frac{\hat{A}_l^{(1)}(s;1,\tau,n^{(1)})^2}{n^{(1)}} d\bar{N}_l^{(1)}(s) + \frac{\hat{A}_l^{(2)}(s;1,\tau,n^{(2)})^2}{n^{(2)}} d\bar{N}_l^{(2)}(s) \right] \\
\hat{\sigma}_2^2 &= \frac{n^{(1)}n^{(2)}}{n^{(1)}+n^{(2)}} \sum_{l=1}^{J} \int_0^\tau \left[\frac{\hat{A}_l^{(1)}(s;2,\tau,n^{1})^2}{n^{(1)}} d\bar{N}_l^{(1)}(s) + \frac{\hat{A}_l^{(2)}(s;2,\tau,n^{(2)})^2}{n^{(2)}} d\bar{N}_l^{(2)}(s) \right] \\
\hat{C} &= \frac{n^{(1)}n^{(2)}}{n^{(1)}+n^{(2)}} \sum_{l=1}^{J} \int_0^\tau \left[\hat{A}_l^{(1)}(s;1,\tau,n^{(1)}) \hat{A}_l^{(1)}(s;2,\tau,n^{(1)}) \frac{d\bar{N}_l^{(1)}(s)}{n^{(1)}} \right. \\
&\quad \left. + \hat{A}_l^{(2)}(s;1,\tau,n^{(2)}) \hat{A}_l^{(2)}(s;2,\tau,n^{(2)}) \frac{d\bar{N}_l^{(2)}(s)}{n^{(2)}} \right] \\
\hat{\rho} &= \hat{C}/\hat{\sigma}_1\hat{\sigma}_2
\end{aligned}$$

where

$$\begin{aligned}
\hat{B}_j^{(m)}(u) &= \int_0^u \widehat{W}_j(s) \frac{\hat{F}^{(m)}(s)}{\bar{Y}^{(m)}(s)} d\bar{N}_j^{(m)}(s) \\
\hat{A}_l^m(s;j,t,n^m) &= [\hat{B}_j^{(m)}(t) - \hat{B}_j^{(m)}(s)] \frac{1}{\bar{Y}^{(m)}(s)}, \qquad \text{if } l \neq j \\
\hat{A}_l^m(s;j,t,n^m) &= [\hat{B}_j^{(m)}(t) - \hat{B}_j^{(m)}(s)] \frac{1}{\bar{Y}^{(m)}(s)} - \widehat{W}_j(s) \frac{\hat{F}^{(m)}(s)}{\bar{Y}^{(m)}(s)}, \qquad \text{if } l = j
\end{aligned}$$

In order to be able to compare the splits, we use a univariate measure of between-node heterogeneity based on Mahalanobis' distance,

$$G(s) = \mathbf{T}' \mathbf{C}^{-1} \mathbf{T} \tag{6.3}$$

where s represents a particular split of the data into two groups and $\mathbf{C}$ is the corresponding covariance matrix given by $\hat{C}, \hat{\sigma}_1^2$ and $\hat{\sigma}_2^2$.

To choose the split that corresponds to the largest two-sample statistic, we compute the statistic $G(s)$ for all possible splits s on all the covariates for the whole sample, $\mathcal{L}$. If a covariate Z is continuous or ordinal, then we look at all the possible cutpoints c that divide the sample into two groups, $Z \leq c$ and $Z > c$. If a covariate is nominal, then we consider all the possible ways of forming the two groups. We repeat this process for all subsequent branches of the tree until we reach one of the predefined stopping criteria. The usual criteria for the growing stage are that the sample size in each terminal node is very small, the subgroups are homogeneous with respect to the covariates, and F_{jk} cannot be estimated (e.g. the subgroup contains no j-type events).

By using the splitting method, we can grow the largest possible tree, T_0, from the data. This tree will capture most of the possible binary, heierachical structure of the data and the dependencies among the covariates.

6.3 THE PRUNING STAGE

The largest grown tree, T_0, is designed to perfectly or almost perfectly predict every case in a data set. However, if most or all of the splits of T_0 are essentially based on random noise, then the prediction results would be poor if we used T_0 to predict subgroups for another data set. To balance the trade-off between the fidelity and overfitting of the current data set, the usual approach is to prune the tree. We begin by removing the branches that result from random noise contained in the data used to build the largest grown tree (the training data). Then we build a sequence of subtrees, calculate their estimated discrimination, and select the subtree that maximizes total amount of discrimination with respect to to the validation data.

To create a sequence of pruned subtrees from T_0, we use the complexity penalty method proposed by LeBlanc and Crowley [19]. Let T^i and T^t denote the sets of internal and terminal nodes of the tree T, respectively. Suppose $|\cdot|$ denotes the cardinality of a set—that

is, $|T^i|$ is equal to the total number of internal nodes, and $|T^t|$ is equal to the total number of terminal nodes for tree T. Let $G(T)$ be the amount of discrimination for tree T based on the training data, and let $\alpha|T^i|$ be the penalty that measures the complexity of T, where $\alpha > 0$ represents the trade-off between the tree's prognostic information (or discrimination) and the tree's complexity. Then pruning the weakest branch of T is equivalent to maximizing the complexity penalty function so that

$$T^* = \arg\min_{T^* \prec T} G(T) - \alpha|T^i|.$$

where $\prec$ means "is a subtree of".

In our case, let $G(h)$ be the quantity defined in equation (6.3) for node h, and let $G(T) = \sum_{h \in T^i} G(h)$ be the sum of these maximum statistics over all internal nodes. Let the complexity penalty function be denoted as $G_\alpha(T) = G(T) - \alpha|T^i|$. Our goal is to prune the tree branch that has the smallest $G_\alpha(T)$ value and is thus the weakest branch. For each node h, we need to solve the equation $G_\alpha(T_h) = G(T_h) - \alpha|T_h^i| = 0$ for α. In this case,

$$\alpha^* = \frac{G(T_h)}{|T_h^i|}.$$

As α increases from 0, we can find the first point of α at which the split makes a net negative contribution to the total amount of prognostic information in the tree.

We prune the split with the smallest $G(T_h)/|T_h^i|$ and then repeat this process until we have an ordered list of optimally pruned subtrees from the smallest tree (the root node tree, T_M) to the largest tree (T_0):

$$T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0,$$

and the corresponding ordered list of α^*'s is

$$\infty = \alpha_M^* > \alpha_{M-1}^* > ... > \alpha_1^* > \alpha_0^* = 0.$$

Now that we have constructed a sequence of subtrees and have calculated their estimated discrimination, we need to select the final pruned tree to use. The tree of choice is the one that best maximizes the total amount of discrimination when applied to the validation data.

For validation, three main methods can be used: test-sample validation, cross-validation, and bootstrap. The test-sample validation method uses the current data set (the training data set) to grow and prune trees, and it uses an external data set (the validation data set) to assess the performance of the trees. The problem with this method is that an external data set is not always available. The other two methods do not require an external set. The cross-validation method divides the current data set into V subsets with nearly equal sample sizes. It removes one subset at a time and uses it as the validation data set. It uses the remaining subsets as the training data sets for the purpose of growing and pruning trees. After cycling through all the subsets in this manner, it combines the results by averaging the performance at each step. Although the cross-validation method yields results that are less biased than those of the test-sample validation method, it requires a larger sample size and tends to give estimates with higher variability (Breiman et al. [2]). The bootstrap method is another technique with less variability. However, it has greater bias than the cross-validation method [19] and hence we propose using cross-validation.

For the cross-validation method, we grow an optimally pruned sequence of trees $T_M \prec T_{M-1} \prec ... \prec T_1 \prec T_0$ on the entire data $\mathcal{L}$. Then we divide the data set $\mathcal{L}$ into V groups $\mathcal{L}_v$ $(v = 1, ..., V)$, all of which have nearly equal sample size. We define $\mathcal{L}_{(v)} = \mathcal{L} - \mathcal{L}_v$. For each v, we use $\mathcal{L}_{(v)}$ to grow a nested sequence of optimally pruned trees

$$T_M^v \prec T_{M-1}^v \prec ... \prec T_1^v \prec T_0^v, \tag{6.4}$$

which we obtain by pruning using the corresponding $\alpha_m^{'} = \sqrt{\alpha_m^* \alpha_{m+1}^*}$'s, Note that the geometric mean of the α_m^*'s is the value of α that corresponds to the maximum $G_\alpha(T_m)$ statistic for $\alpha_m^* \leq \alpha < \alpha_{m+1}^*$ (Breiman et al., [2]). For data set $\mathcal{L}_v$, we calculate the sum of between-node statistics $G(T_m^v)$. To estimate $G(T_m)$, we average the sum over V sets

$$G^{CV}(T_m) = \frac{1}{V} \sum_{v=1}^{V} G(T_m^v) \tag{6.5}$$

and choose the $\alpha = \alpha^*$ that maximizes this average.

Our final tree, $T(\alpha_c)$, is chosen from the original list of optimally pruned subtrees grown on the whole sample $\mathcal{L}$, where α_c is some fixed value of the complexity penalty. The $T(\alpha_c)$

is the smallest optimally pruned subtree that has the maximum cross-validation adjusted discrimination $G_{\alpha}^{CV}(T)$ for $\alpha = \alpha_c$. This means $T(\alpha_c) \in \{T_M, ..., T_0\}$ is chosen such that

$$T(\alpha_c) = \arg \max_{T_m \in \{T_M, ..., T_0\}} G^{CV}\{T_m\} - \alpha_c |T_m^i|.$$

6.4 SIMULATIONS

In this section, we discuss a series of simulations designed to investigate how accurately the splitting method detects cutpoints of a covariate and how well the tree procedures detect the data structure. In all simulations there are two event types, that is $J = 2$, plus true censoring. We compare the performance of five tree methods. The first tree is the maximum between-node heterogeneity tree based on a composite rank test of the CIFs (the "multivariate competing risks tree"). The second tree is the maximum between-node heterogeneity tree proposed in Chapter 5 based on a univariate rank test of the CIFs ("univariate between-node competing risks tree"). This method requires that we choose one event-of-interest, which we designated to be event 1. The third, fourth and fifth trees are a between-node tree based on the naive log-rank test that was proposed by LeBlanc and Crowley [19] for univariate survival data (the "LR tree"), the within-node univariate tree for competing risks based on event-specific martingale residuals and the sums-of-squares metric ("MR-SS") and the within-node univariate tree for competing risks based on event-specific martingale residuals and the absolute value metric ("MR-ABS"). These three trees require that the censoring indicator be a 0/1 indicator, i.e. we cannot use the competing risks censoring indicator which takes values $\delta = 0, 1, 2$. These methods were used for the situation where more than one event is of interest, and therefore is makes sense to construct a censoring indicator that is 1 when any event occurs (either event 1 or 2) and is 0 when true censoring occurs.

6.4.1 Performance of Splitting Methods

Table 9 shows the setups for the six different models used in our simulations to investigate splitting accuracy for the five types of trees. We used various tree methods described in

Section 6.4 to estimate the cutpoint, $\hat{c}$, from the one-split tree grown from the data. Each model was generated with different event 1 and 2 failure rates for $Z \leq c$ and for $Z > c$, where Z is distributed $U(0,1)$. These rates for events 1, 2 and censoring were chosen to approximate the real life situation of patients on the waiting list for a liver transplant. In most of the models, the failure times for event 1, event 2, and true censoring were generated from exponential distributions, with the exception of model 4 where the failure times were generated from a log normal distribution. We repeated the simulation with low censoring (10%) and moderate censoring (50%).

The same cutpoint $c = 0.5$ for five of the six models, was used for both the event 1 and the event 2 processes. One exception to this is model 5, where the hazards for event 1 and 2 are dependent on different cutpoints for the same covariate Z, i.e. the rate for event 1 is dependent on $Z \leq 0.5$ and $Z > 0.5$, whereas the rate for event 2 is dependent on $Z \leq 0.3$ and $Z > 0.3$. The purpose of this model is to investigate whether $\hat{c}$ lies between the two cutpoints or whether $\hat{c}$ tends to be closer to either 0.3 or 0.5. In cases where the true cutpoints for event 1 and event 2 differ, let c_j denote the true cutpoint for the rate of event $j = 1, 2$. For model 6, $c = 0.3$ (rather than $c = 0.5$) for both event 1 and 2 rates. Model 6 is important to consider because a tree-method that does *not* estimate c reliably may produce estimates that are distributed uniformly across the interval $[0, 1]$, and this will result in an average estimate of c of approximately 0.5, which would be misleading.

In models 1, 3 and 4, only the rate of event 2 depends on the covariate Z. In these models we would expect the multivariate method to perform better than the between-node tree method based on Gray's test ("Uni.") which focuses on changes in the event of interest (event 1) only. For the models 3 and 4, where there is only a weak change in the rate of event 2 and the event 1 rate does not depend on Z, we might expect only the multivariate method to do well. However, Gray's test is not designed to detect changes in rate of event 1 and a small difference in the rate of event 2 for $Z \leq c$ and $Z > c$ may be difficult to detect for methods that only take into considerate the overall rate of event 1 and 2 combined (LR, MR-SS, MR-ABS). Models 2, 5 and 6 have the rates of both events depending on Z. We may expect that the multivariate and univariate competing risk between-node methods to do well here, as both are designed to detect changes in event 1 (and the multivariate method

should detect that the rates of both event types are dependent on Z). We would not expect the other methods to do well, as the overall rate of event 1 and 2 does not change – as the rate of event 1 increases with $Z > c$, the rate of event 2 decreases when $Z > c$ resulting in the net effect of no change in total rate of all event types.

The sample size for each data set was 500, and we generated 1000 data sets for each model configuration. To compare the performance of the three trees, we used four measures: the mean estimate $\bar{\widehat{c}}$ (mean), the mean bias $\overline{\widehat{c} - c}$ (bias), and the standard deviation of the $\widehat{c}$'s (SD).

Tables 10 and 11 summarize the results for the one-split simulations. The tree methods are: between-node heterogeneity tree based on a composite rank test of the CIFs ("Mult."), between-node heterogeneity tree based on a univariate rank test of the CIFs ("Uni."), a between-node tree based on the naive log-rank test for univariate survival data (the "LR"), the within-node univariate tree for competing risks based on event-specific martingale residuals and the sums-of-squares metric ("MR-SS") and the within-node univariate tree for competing risks based on event-specific martingale residuals and the absolute value metric ("MR-ABS"). At first glance, the multivariate method does not appear to perform well with respect to the measures of mean and median. However, we can see that this is deceptive when we look at the histograms of the cutpoint estimates for each model (Figures 8-19). For instance, for Model 2 with low censoring, we see that the the multivariate and univariate competing risk methods clearly are choose the correct cutpoint of $c = 0.5$ more reliably than the LR, Within-SS and Within-ABS tree methods. The distribution of the cutpoint estimates for LR, Within-SS and Within-ABS tree methods are similar to a uniform or bath-tub distribution and therefore, as 0.5 is in the middle of the $[0, 1]$ interval, the average cutpoint estimate is close to $c = 0.5$.

In general, the multivariate (Mult.) and univariate (Uni.) methods tend to perform the best (in terms of central tendency around the correct cutpoint) for models where the both rates for event 1 and 2 are dependent on Z (models 2, 5 and 6) and for low censoring for all models. The multivariate method is the most sensitive to different values for c. It is the only method to produce an average cutpoint near 0.4 for model 5, when the true cutpoint for event 1 was $c_1 = 0.5$ and for event 2 $c_2 = 0.3$. The distributions for Model 5 are bi-modal,

Table 9: Description of simulations for splitting, multivariate method

	Event specific hazards				
	Event 1	(h_1)	Event 2	(h_2)	
Model	$Z \leq 0.5$	$Z > 0.5$	$Z \leq 0.5$	$Z > 0.5$	Description
1	1	1	1	2	Event 2 dependent only
2	1	2	2	1	Both events dependent on Z, opposite direction
3	2	2	0.5	1	Weak change in event 2
4	$\mu = 0$	$\mu = 0$	$\mu = 0.5$	$\mu = 1$	Weak change in event 2 Log norm,$\sigma = 1$
	$Z \leq 0.5$	$Z > 0.5$	$Z \leq 0.3$	$Z > 0.3$	
5	1	2	2	1	Both events dependent opposite direction, $c_1 \neq c_2$
	$Z \leq 0.3$	$Z > 0.3$	$Z \leq 0.3$	$Z > 0.3$	
6	1	2	2	1	Both events dependent, opposite direction $c = 0.3$

Description of models used in simulations for the performance of splitting.

around $c = 0.3$ and $c = 0.5$. The multivariate and univariate between-node competing risk methods were the only methods to produce an average cutpoint estimate near the true value of $c = 0.3$ for model 6. These results tend to persist under moderate censoring also.

Under moderate censoring, the multivariate and univariate competing risks methods tend not perform well in models 1, 3 and 4 where only the rate for event 2 is dependent on Z. In Model 3 and 4 (when the rate of event 1 does not depend on Z and the rate for event 2 is weakly dependent on Z) the martingale residual tree with the absolute value metric performs the best. The distribution for Model 6 (when the true cutpoint is $c = 0.3$) is a suitably skewed distribution for all methods centered around 0.3.

A feature of note in Figures 8-19 is the "bath-tub" shape of some of the distributions of $\hat{c}$. This a problem common to tree methods because of a phenomenon referred to as "end-cut preference". End-cut preference refers to the problem where extreme values of the covariate Z tend to be chosen as a cutpoint for Z. This is because most splitting statistics are unstable for smaller sample sizes, and extreme cutpoints tend to result in one group having a very small sample size. A large value of the splitting statistic may result simply due to noise in this case. Bath-tub shape distributions for $\hat{c}$ will also result in the mean estimated cutpoint being close to 0.5, when $Z \sim U(0, 1)$. Several bath-tub shape distributions can be seen in Figures 8-19.

Table 10: Estimated cutpoint, multiple events of interest, Models 1–4.

	Low censoring 10%					Moderate censoring 50%				
Measure	Mult.	Uni.	LR	MR-SS	MR-ABS	Mult.	Uni.	LR	MR-SS	MR-ABS
Model 1: Exponential, Event 2 dependent on Z, $c = 0.5$										
Mean	0.5143	0.4612	**0.5096**	0.4771	0.4878	0.4639	0.4679	0.5358	0.4933	**0.4998**
Median	0.5004	0.4843	0.4999	0.4932	0.4950	0.4398	0.4674	0.5062	0.4981	**0.4996**
Bias	0.0143	-0.0388	**0.0096**	-0.0229	-0.0122	-0.0361	-0.0321	0.0358	-0.0067	**-0.0002**
SD	0.1317	0.1614	0.0970	0.0954	**0.0541**	0.3622	0.2954	0.1553	0.1529	**0.0774**
RSME	0.1324	0.1659	0.0974	0.0980	**0.0555**	0.3638	0.2970	0.1592	0.1530	**0.0773**
Model 2: Exponential, Both events dependent on Z, $c = 0.5$										
Mean	0.5173	0.5083	0.4974	0.4546	**0.5009**	0.5081	0.5174	0.4898	**0.5076**	0.4928
Median	**0.5006**	0.5020	0.5169	0.4178	0.5106	**0.5021**	0.5035	0.4628	0.5147	0.4883
Bias	0.0173	0.0083	-0.0026	-0.0454	**0.0009**	0.0081	0.0174	-0.0102	0.0076	**-0.0072**
SD	0.1064	**0.0354**	0.3343	0.3298	0.2256	0.3199	**0.0800**	0.3376	0.3423	0.2193
RSME	0.1077	**0.0363**	0.3341	0.3327	0.2255	0.3199	**0.0818**	0.3375	0.3422	0.2193
Model 3: Exponential, Weak change event 2, $c = 0.5$										
Mean	**0.4908**	0.4497	0.5189	0.4639	0.4867	0.4196	0.4954	0.5097	0.4860	0.5004
Median	**0.5007**	0.4877	0.5062	0.4849	0.4922	0.2930	0.4993	0.5057	0.4887	0.4969
Bias	**-0.0092**	-0.0503	0.0189	-0.0361	-0.0133	-0.0804	-0.0046	0.0097	-0.0140	0.0004
SD	0.2218	0.1893	0.2462	0.2214	**0.1396**	0.3495	0.3110	0.2824	0.2539	0.1646
RSME	0.2219	0.1958	0.2468	0.2243	**0.1401**	0.3584	0.3109	0.2824	0.2541	0.1645
Model 4: Log normal, Weak change event 2, $c = 0.5$										
Mean	0.5159	0.5273	0.4834	0.5304	**0.5064**	0.5472	0.5312	0.4639	**0.4943**	0.4815
Median	**0.5020**	0.5064	0.4932	0.5136	0.5053	0.6140	0.5361	0.4769	**0.4970**	0.4935
Bias	0.0159	0.0273	-0.0166	0.0304	**0.0064**	0.0472	0.0312	-0.0361	**-0.0057**	-0.0185
SD	0.2452	0.2081	0.2316	0.2288	**0.1342**	0.3620	0.3053	0.2712	0.2677	**0.1505**
RSME	0.2455	0.2098	0.2321	0.2307	**0.1343**	0.3649	0.3068	0.2734	0.2677	**0.1515**

Accuracy measures for the estimated cutpoint, $\widehat{c}$, from various methods for multivariate data with absolute value function for martingale residual (MR-ABS) tree. Bold-faced values represent the best result for each outcome measure and model. $c_j, j = 1, 2$ indicates the cutpoint for event j when $c_1 \neq c_2$.

Table 11: Estimated cutpoint, multiple events of interest, Models 5–6.

	Low censoring 10%					**Moderate censoring 50%**				
Measure	Mult.	Uni.	LR	MR-SS	MR-ABS	Mult.	Uni.	LR	MR-SS	MR-ABS
Model 5: Exponential, Both events dependent on Z, $c_1 = 0.5, c_2 = 0.3$										
Mean	0.4612	0.4759	0.4754	**0.4783**	0.4760	0.4446	**0.4993**	0.4693	0.4765	0.4711
Median	0.4210	0.4925	0.5003	**0.5002**	0.5008	0.3826	**0.5004**	0.4969	0.4975	0.4983
Bias $c_1 = 0.5$	-0.0388	-0.0241	-0.0246	**-0.0217**	-0.0240	-0.0554	**-0.0007**	-0.0307	-0.0235	-0.0288
Bias $c_2 = 0.3$	**0.1612**	0.1759	0.1754	0.1783	0.1760	**0.1446**	0.1993	0.1693	0.1765	0.1711
SD	0.2076	**0.0806**	0.2458	0.2183	0.1430	0.3145	**0.1047**	0.2882	0.2748	0.1803
RSME	0.2111	**0.0841**	0.2469	0.2192	0.1449	0.3192	**0.1047**	0.2897	0.2757	0.1825
Model 6: Exponential, Both events dependent on Z, $c = 0.3$										
Mean	0.3297	**0.3111**	0.5149	0.5223	0.5025	0.3632	**0.3273**	0.5004	0.4933	0.5022
Median	**0.2998**	0.3015	0.5419	0.5423	0.4983	0.2824	**0.3054**	0.4871	0.4443	0.5020
Bias	0.0297	**0.0111**	0.2149	0.2223	0.2025	0.0632	**0.0273**	0.2004	0.1933	0.2022
SD	0.1676	**0.0403**	0.3296	0.3291	0.2172	0.3126	**0.0913**	0.3282	0.3305	0.2183
RSME	0.1701	**0.0418**	0.3933	0.3970	0.2969	0.3188	**0.0953**	0.3844	0.3827	0.2975

Accuracy measures for the estimated cutpoint, $\widehat{c}$, from various methods for multivariate data with absolute value function for martingale residual (MR-ABS) tree. Bold-faced values represent the best result for each outcome measure and model. $c_j, j = 1, 2$ indicates the cutpoint for event j when $c_1 \neq c_2$.

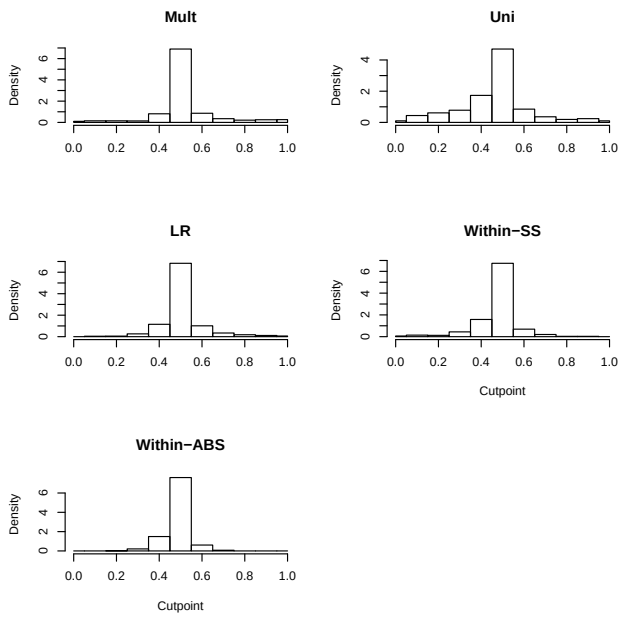


Figure 8: Distribution of estimated cutpoints, Model 1 Low Censoring

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 1 and low censoring.

6.4.2 Performance of Methods to Detect Data Structure

Table 12 shows the model configurations that we used to investigate the ability of a tree to detect the structure in data with competing risks. For each model, two covariates (Z_1 and Z_2) were related to the survival times, and four other covariates (Z_3–Z_6) were independent of the survival times. All the covariates were generated from a discrete uniform distribution over the points 0, 0.25, 0.5, 0.75, and 1. For all models, the final censoring rate was 50%. Models 1A–5A had an interaction effect between Z_1 and Z_2 (multiplicative hazard), so a tree with three terminal nodes would be expected to capture this data structure. Models 1B–5B had additive hazards for event 1 based on Z_1 and Z_2, so a tree with four terminal nodes would be expected to capture this data structure. All of the models were based on

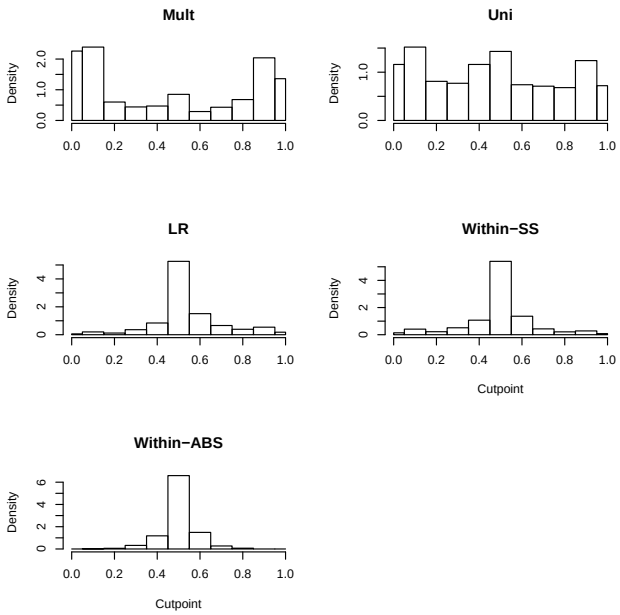


Figure 9: Distribution of estimated cutpoints, Model 1 Moderate Censoring

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 1 and moderate censoring.

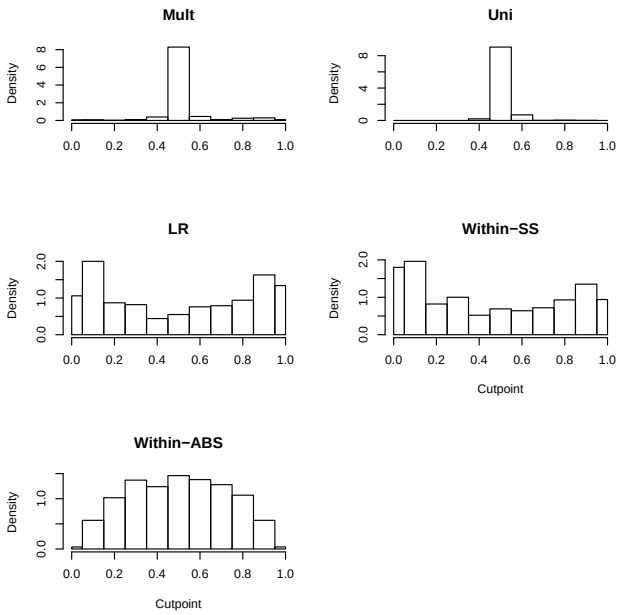


Figure 10: Distribution of estimated cutpoints, Model 2 Low Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 2 and low censoring.

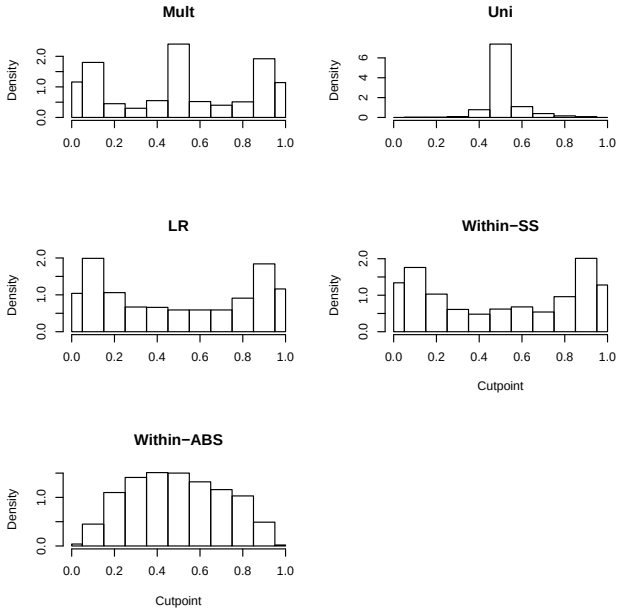


Figure 11: Distribution of estimated cutpoints, Model 2 Moderate Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 2 and moderate censoring.

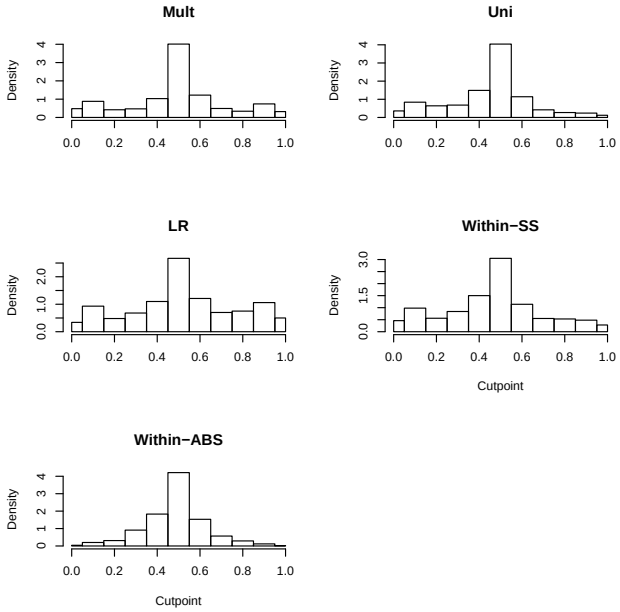


Figure 12: Distribution of estimated cutpoints, Model 3 Low Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 3 and low censoring.

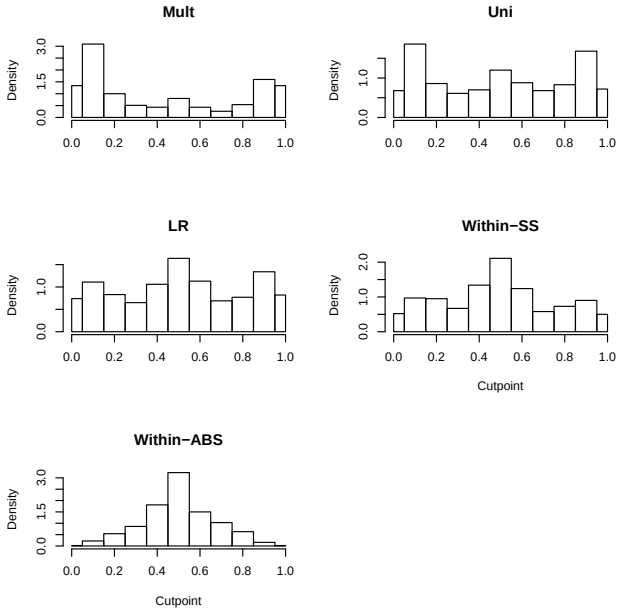


Figure 13: Distribution of estimated cutpoints, Model 3 Moderate Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 3 and moderate censoring.

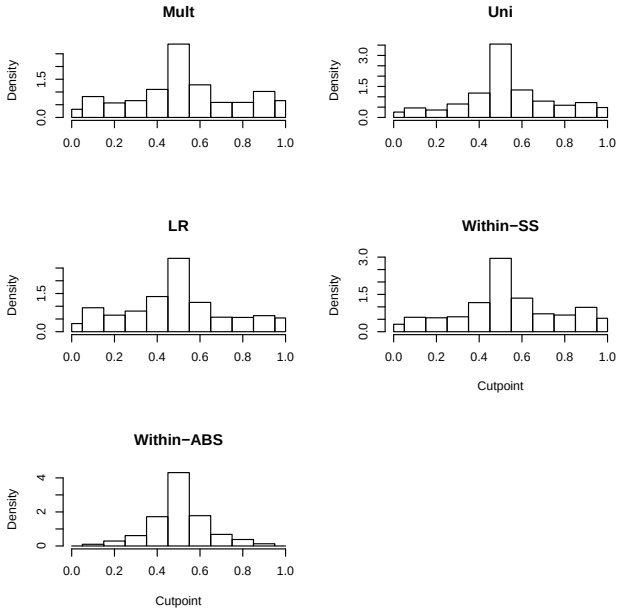


Figure 14: Distribution of estimated cutpoints, Model 4 Low Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 4 and low censoring.

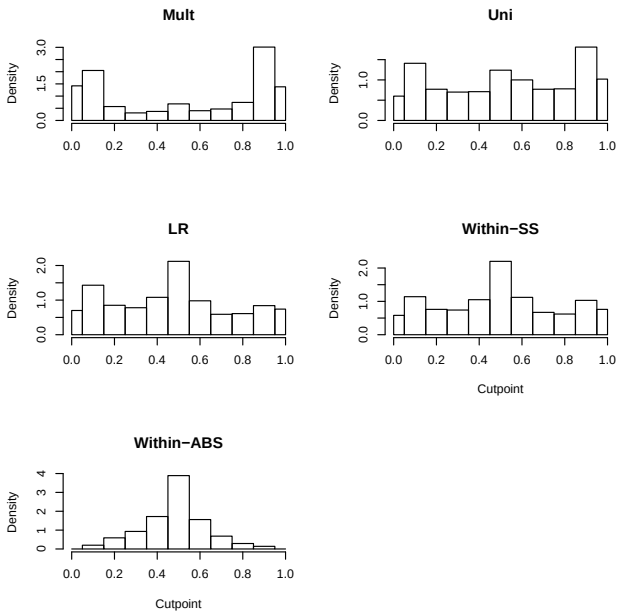


Figure 15: Distribution of estimated cutpoints, Model 4 Moderate censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 4 and moderate censoring.

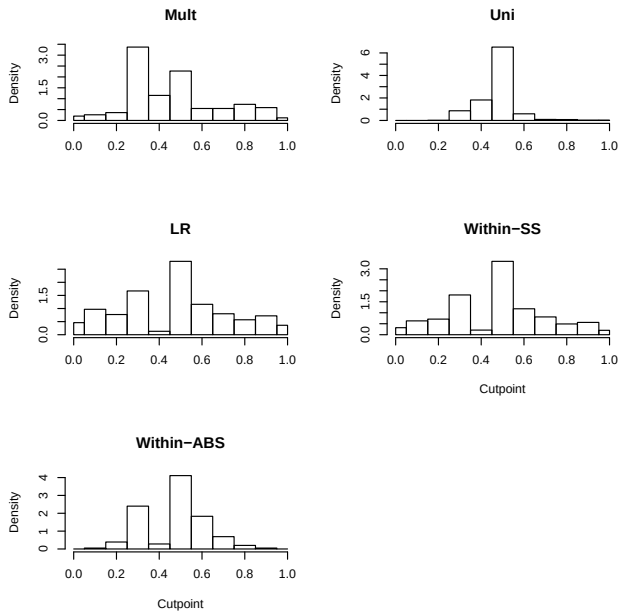


Figure 16: Distribution of estimated cutpoints, Model 5 Low Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 5 and low censoring.

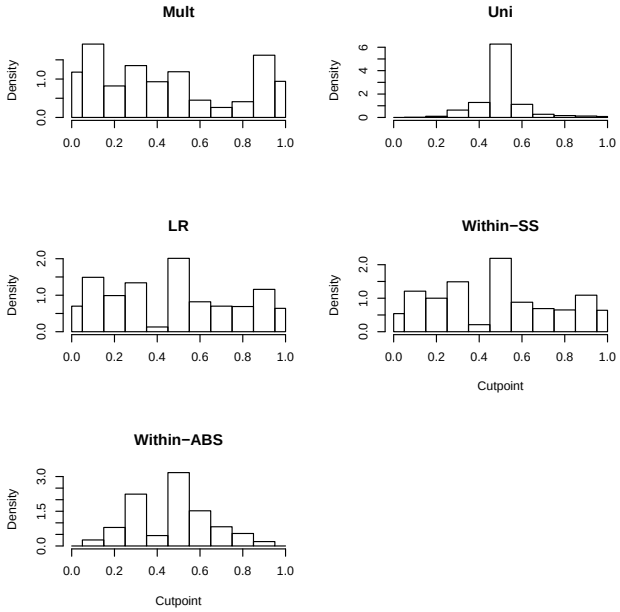


Figure 17: Distribution of estimated cutpoints, Model 5 Moderate Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 5 and moderate censoring.

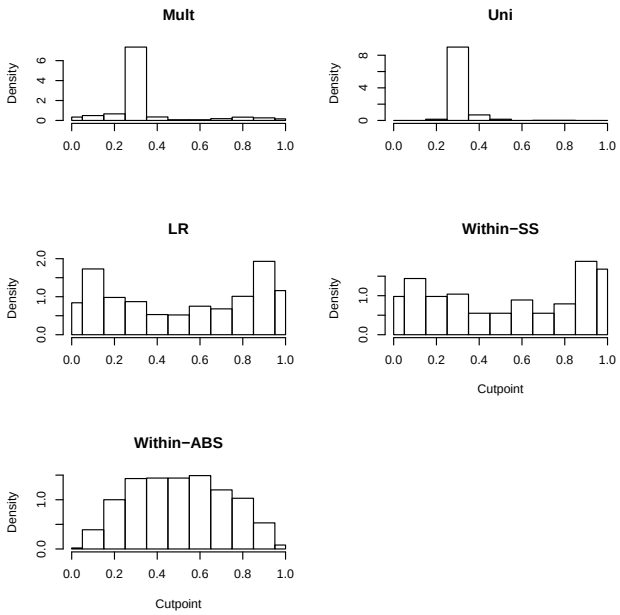


Figure 18: Distribution of estimated cutpoints, Model 6 Low Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 6 and low censoring.

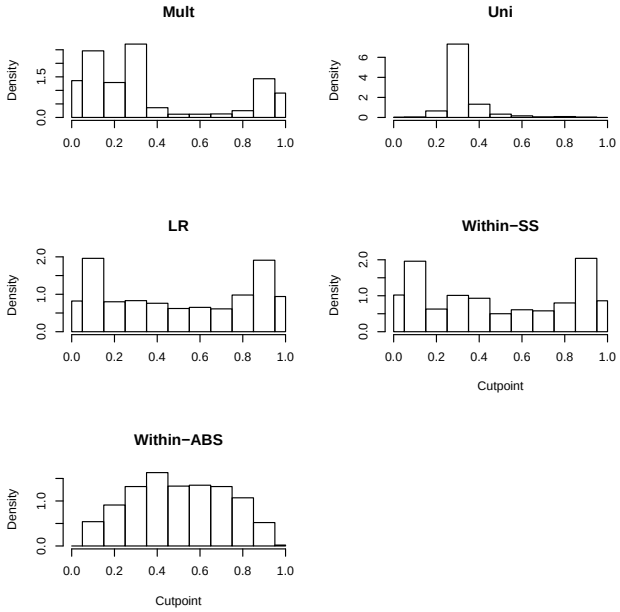


Figure 19: Distribution of estimated cutpoints, Model 6 Moderate Censoring.

The distribution of the estimated cutpoints from one-split trees for multivariate competing risks data for Model 6 and moderate censoring.

exponential distributions except for models 3A and 3B which were based on log normal distributions.

We constructed 100 trees with a sample size of $N = 1000$ per model in each simulation. We began by comparing the performance of trees in terms of three different measures. The first measure was $|T^t|$, the number of terminal nodes in each tree. The second measure was the number of times both Z_1 and Z_2 were chosen (this is referred to as the "inclusive" signal for variable selection), and the third measure was the number of times that only Z_1 and Z_2 were chosen (this is referred to as the "exclusive" signal for variable selection). Next, to compare predictive ability of the trees, we generated validation data sets with 1000 observations from the same model (but with no censoring) for each simulation. We used $\alpha = 1$ for the complexity penalty and the weighting parameter $\rho = -1$ to emphasize late differences in the multivariate test of the CIFs. We allowed the minimum number of observations in each terminal node to be 20. We used 5-fold cross-validation method to select the final tree. We calculated the mean absolute difference between the event 1 failure time for each observation in a validation data set and the median event 1 failure time predicted by the tree. We calculated the mean absolute error for event 1 failure times and event 2 failure times as

$$\begin{aligned} MAE_1 &= \frac{1}{N_1}\sum_{h=1}^{|T^t|}\sum_{i=1}^{N_{1h}}|T_{i1h} - \widehat{\tau}_{1h}| \\ MAE_2 &= \frac{1}{N_2}\sum_{h=1}^{|T^t|}\sum_{i=1}^{N_{2h}}|T_{i2h} - \widehat{\tau}_{2h}| \end{aligned}$$

where N_{jh} is the number of type j events in node h, N_j is the total number type j events, T_{ijh} is the ith failure time for event j in terminal node h, and $\widehat{\tau}_{jh}$ is the median event j failure time based on the training data for terminal node h.

The results for the multiplicative and additive models are shown in Tables 13 and 14, respectively. The tree methods are labeled as follows: "Between-Mult"=multivariate competing risk method, "Between-Uni"=univariate between node competing risk method based on Gray's test, "LR"= the between node univariate survival tree based on log-rank test, "Within-SS"=within-node competing risk tree based on event-specific martingale residuals

Table 12: Description of models used in simulations for data structure,1A–5B.

Model	Event specific hazards	Description
Conditional relationship Z_1, Z_2: 3 Terminal nodes expected		
1A	$h_1(t) = 2 - I(Z_1 > c \cap Z_2 > c)$	Hazards differ for both events,
	$h_2(t) = 1 + I(Z_1 > c \cap Z_2 > c)$	opposite direction, $c = 0.5$.
2A	$h_1(t) = 2$	Weak difference,
	$h_2(t) = 1 + I(Z_1 > c \cap Z_2 > c)$	event 2 only, $c = 0.3$
3A	$\mu_1(t) = 1 - I(Z_1 > c \cap Z_2 > c)$	Hazards differ for both events,
	$\mu_2(t) = I(Z_1 > c \cap Z_2 > c)$	opposite direction,
		Log normal $\sigma = 1, \rho = 0, c = 0.3$
4A	$h_1(t) = 2 - I(Z_1 > c_1 \cap Z_2 > c_2)$	Hazards differ for both events,
	$h_2(t) = 1 + I(Z_1 > c_1 \cap Z_2 > c_2)$	opposite direction,
		$c_1 = 0.3, c_2 = 0.5$
5A	$h_1(t) = 2 - I(Z_1 > c \cap Z_2 > c)$	Hazards differ for both events,
	$h_2(t) = 1 + I(Z_1 > c \cap Z_2 > c)$	opposite direction, $c = 0.3$
Independent relationship Z_1, Z_2: 4 Terminal nodes expected		
1B	$h_1(t) = 1 + I(Z_1 > c) + I(Z_2 > c)$	Hazards differ for both events,
	$h_2(t) = 3 - I(Z_1 > c) - I(Z_2 > c)$	opposite direction, $c = 0.5$.
2B	$h_1(t) = 2$	Weak difference,
	$h_2(t) = 1 + 0.5 \cdot I(Z_1 > c) + 0.5 \cdot I(Z_2 > c)$	event 2 only,$c = 0.3$
3B	$\mu_1(t) = 0.5 \cdot I(Z_1 > c) + 0.5 \cdot I(Z_2 > c)$	Hazards differ for both events,
	$\mu_2(t) = 1 - 0.5 \cdot I(Z_1 > c) - 0.5 \cdot I(Z_2 > c)$	Log normal, $\sigma = 1, \rho = 0$
4B	$h_1(t) = 1 + I(Z_1 > c) + I(Z_2 > c)$	Hazards differ for both events
	$h_2(t) = 3 - I(Z_1 > c) - I(Z_2 > c)$	$c = 0.3$
5B	$h_1(t) = 1 + I(Z_1 > c_1) + I(Z_2 > c_2)$	Hazards differ for both events
	$h_2(t) = 3 - I(Z_1 > c_1) - I(Z_2 > c_2)$	$c_1 = 0.3, c_2 = 0.5$

Table 13: Investigating tree structure multiplicative models, 1A-5A.

	Percent of terminal nodes in final tree							Variable Selection		Average	
Method	1	2	**3**	4	5	6	≥7	Inclusive	Exclusive	MAE1	MAE2
Model 1A: Exponential $\lambda_1 = 2 - I(Z_1 > c \cap Z_2 > c)\}$, $c = 0.5$ $\lambda_2 = 1 + I(Z_1 > c \cap Z_2 > c)\}$											
Between-Mult	46	29	**8**	7	4	2	4	15	4	0.2555	0.2540
Between-Uni	14	39	**14**	8	2	1	22	47	**17**	0.2556	0.2545
LR	12	8	**3**	2	4	1	70	**69**	0	0.2557	0.2543
Within-SS	97	3	**0**	0	0	0	0	0	0	**0.2553**	**0.2537**
Within-ABS	100	0	**0**	0	0	0	0	0	0	**0.2553**	0.2538
Model 2A: Exponential $\lambda_1 = 2$ $\lambda_2 = 1 + I(Z_1 > c \cap Z_2 > c), c = 0.3$											
Between-Mult	47	26	**9**	6	2	5	5	10	1	**0.2300**	0.2303
Between-Uni	38	34	**9**	4	0	2	13	12	0	0.2315	**0.2302**
LR	8	9	**3**	2	2	1	75	**75**	0	0.2317	0.2302
Within-SS	89	9	**2**	0	0	0	0	2	**2**	0.2319	0.2285
Within-ABS	99	1	**0**	0	0	0	0	0	0	0.2318	0.2285
Model 3A: Log normal $\mu_1 = 1 - I(Z_1 > c \cap Z_2 > c)$, $\sigma = 1, \rho = 0, c = 0.3$ $\mu_2 = I(Z_1 > c \cap Z_2 > c)$											
Between-Mult	5	33	**17**	20	11	5	9	57	23	0.7509	0.7892
Between-Uni	0	8	**27**	34	15	5	11	**91**	**39**	0.7491	**0.7866**
LR	7	10	**3**	4	2	1	73	66	0	0.7521	0.8065
Within-SS9	6	4	**0**	0	0	0	0	0	0	**0.7477**	0.8046
Within-ABS	100	0	**0**	0	0	0	0	0	0	0.7478	0.8048
Model 4A: Exponential $\lambda_1 = 2 - I(Z_1 > c_1 \cap Z_2 > c_2), c_1 = 0.3, c_2 = 0.5$ $\lambda_2 = 1 + I(Z_1 > c_1 \cap Z_2 > c_2)$											
Between-Mult	44	22	**6**	3	8	7	10	26	3	0.2539	**0.2526**
Between-Uni	5	41	**14**	9	4	3	24	51	**18**	0.2519	0.2551
LR	8	14	**4**	2	3	2	67	**63**	0	0.2516	0.2556
Within-SS	97	3	**0**	0	0	0	0	0	0	**0.2512**	0.2541
Within-ABS	100	0	**0**	0	0	0	0	0	0	**0.2512**	0.2540
Model 5A: Exponential $\lambda_1 = 2 - I(Z_1 > c \cap Z_2 > c), c = 0.3$ $\lambda_2 = 1 + I(Z_1 > c \cap Z_2 > c)$											
Between-Mult	33	34	**10**	6	3	1	13	28	8	0.2538	0.2550
Between-Uni	1	20	**25**	16	8	3	27	**75**	**25**	0.2545	0.2543
LR	5	11	**2**	4	4	1	73	73	0	0.2543	0.2557
Within-SS	98	2	**0**	0	0	0	0	0	0	**0.2534**	**0.2541**
Within-ABS	100	0	**0**	0	0	0	0	0	0	**0.2534**	**0.2541**

Investigating tree structure joint effect of Z_1, Z_2 on hazards, $N = 1000$, for multiplicative models when there is more than one event of interest. The bold-faced numbers represent either 1) the ideal number of nodes selected, or 2) the best result for each model. The variable selection measure "Inc" or "Inclusive signal" represents the percent of trees that selected both variables Z_1 and Z_2. The variable selection measure "Exc" or "Exclusive signal" represents the percent of trees that selected only Z_1 and Z_2.

Table 14: Investigating tree structure, additive models 1B–5B.

	Percent of terminal nodes in final tree							Variable Selection		Average	
Method	1	2	3	**4**	5	6	≥7	Inclusive	Exclusive	MAE1	MAE2
Model 1B: Exponential $\lambda_1 = 1 + I(Z_1 > c) + I(Z_2 > c)\}$, $c = 0.5$ $\lambda_2 = 3 - I(Z_1 > c) - I(Z_2 > c)$											
Between-Mult	24	50	6	**3**	4	3	10	21	4	**0.1903**	0.1904
Between-Uni	0	32	10	**13**	7	10	28	67	**18**	0.1911	0.1891
LR	8	9	2	**2**	0	1	78	**76**	0	0.1911	0.1893
Within-SS	99	1	0	**0**	0	0	0	0	0	**0.1903**	**0.1887**
Within-ABS	100	0	0	**0**	0	0	0	0	0	**0.1903**	**0.1887**
Model 2B: Exponential $\lambda_1 = 2$ $\lambda_2 = 1 + 0.5 \cdot I(Z_1 > c) + 0.5 \cdot I(Z_2 > c), c = 0.3$											
Between-Mult	44	30	10	**9**	0	1	6	10	**2**	0.2217	0.2113
Between-Uni	39	31	5	**5**	1	3	16	15	1	**0.2216**	0.2121
LR	6	16	1	**0**	1	1	75	**72**	0	**0.2216**	0.2118
Within-SS	83	17	0	**0**	0	0	0	0	0	0.2218	**0.2099**
Within-ABS	100	0	0	**0**	0	0	0	0	0	0.2218	**0.2099**
Model 3B: Log normal $\mu_1 = 0.5 \cdot I(Z_1 > c) + 0.5 \cdot I(Z_2 > c)$, $\sigma = 1, c = 0.3$ $\mu_2 = 1 - 0.5 \cdot I(Z_1 > c) - 0.5 \cdot I(Z_2 > c)$											
Between-Mult	21	51	4	**5**	4	5	10	24	4	**0.7529**	**0.7355**
Between-Uni	0	16	8	**23**	11	8	34	**84**	**22**	0.7622	0.7418
LR	8	11	3	**2**	1	3	72	70	1	0.7670	0.7475
Within-SS	94	5	0	**1**	0	0	0	0	0	0.7618	0.7388
Within-ABS	99	1	0	**0**	0	0	0	0	0	0.7604	0.7388
Model 4B: Exponential $\lambda_1 = 1 + I(Z_1 > c) + I(Z_2 > c)\}$, $c = 0.3$ $\lambda_2 = 3 - I(Z_1 > c) - I(Z_2 > c)$											
Between-Mult	24	50	3	**5**	6	4	8	22	2	**0.1917**	0.1907
Between-Uni	0	41	7	**12**	10	3	27	57	**13**	0.1925	0.1912
LR	11	11	6	**0**	1	2	69	**69**	0	0.1927	**0.1904**
Within-SS	97	2	0	**1**	0	0	0	0	0	0.1921	0.1911
Within-ABS	100	0	0	**0**	0	0	0	0	0	0.1921	0.1911
Model 5B: Exponential $\lambda_1 = 1 + I(Z_1 > c_1) + I(Z_2 > c_2), c_1 = 0.3$ $\lambda_2 = 3 - I(Z_1 > c_2) - I(Z_2 > c_2), c_2 = 0.5$											
Between-Mult	19	51	10	**6**	4	4	6	26	9	**0.1915**	0.1916
Between-Uni	2	35	11	**10**	11	4	27	63	**21**	0.1924	0.1911
LR	7	8	5	**9**	1	3	67	**67**	0	0.1920	0.1916
Within-SS	96	1	2	**1**	0	0	0	0	0	0.1919	**0.1910**
Within-ABS	100	0	0	**0**	0	0	0	0	0	0.1919	**0.1910**

Investigating tree structure joint effect of Z_1, Z_2 on hazards, $N = 1000$, for additive models when there is more than one event of interest. The bold-faced numbers represent either 1) the ideal number of nodes selected, or 2) the best result for each model. The variable selection measure "Inc" or "Inclusive signal" represents the percent of trees that selected both variables Z_1 and Z_2. The variable selection measure "Exc" or "Exclusive signal" represents the percent of trees that selected only Z_1 and Z_2.

using sums-of-squares impurity, "Within-ABS" = within-node competing risk tree based on event-specific martingale residuals using absolute value impurity.

Although the univariate between-node competing risk tree tends to perform the best, followed by the multivariate between-node competing risk tree, none of the tree methods perform well in terms of selecting the correct number of terminal nodes or variable selection.

A further simulation was carried out where the cumulative incidence function for event 1 (CIF_1) is held constant with respect to the covariates Z_1 and Z_2, but the CIFs for event 2 and censoring are dependent on Z_1 and Z_2. The simulations are described in Table 15 and the results are given in Table 16. The motivation for these simulations was to investigate whether the multivariate method would perform better when the CIF for event 2 and not simply the hazard was dependent on the covariates. The between-node tree based on Gray's test would not be expected to do well because it only detects differences in the CIF of event 1 and not event 2. However, the multivariate tree does not perform well, choosing the correct number of terminal nodes between $0-15\%$ of the time. The within-node methods perform the best. For Model 1 (choosing the correct number of terminal nodes $89-98\%$ of the time) but this does not hold for the other Models (correct number of terminal nodes $0-35\%$ of the time).

6.5 APPLICATION TO LIVER TRANSPLANT WAITLIST

To illustrate the use of the multivariate competing risk tree, we apply this method to a data set concerning patients awaiting a liver transplant. In the United States, patients who have end-stage liver disease and meet the eligibility requirements for a transplant are placed on the waiting list for a donated liver. The United Network for Organ Sharing (UNOS) manages the waiting list, matches donated organs with the patients who need them, and maintains the national database of pre-transplant and follow-up information. When organs become available, UNOS allocates them to patients who have the highest risk of pre-transplant death, based on scores derived from the model for end-stage liver disease (MELD). In this model, death is the event of interest. Censoring is applied to receipt of a transplant, removal

Table 15: Description of models used in simulations for data structure, constant CIF for event 1.

Model	Event specific hazards	Description
Conditional relationship Z_1, Z_2: Expect 3 Terminal nodes		
1C	$h_1(t) = 0.3$	Constant CIF event 1,
	$h_2(t) = 0.1 + 0.3I(Z_1 > c \cap Z_2 > c)$	CIF event 2 and censoring depend on covariates,
	$h_0(t) = 0.6 - 0.3I(Z_1 > c \cap Z_2 > c)$	exponential model, $c = 0.3$
2C	$\mu_1(t) = 0$	Constant CIF event 1,
	$\mu_2(t) = 0.5I(Z_1 > c \cap Z_2 > c)$	CIF event 2 and censoring depend on covariates,
	$\mu_0(t) = 0.5 - 0.5I(Z_1 > c \cap Z_2 > c)$	log normal model, $c = 0.3$
Independent relationship Z_1, Z_2: Expect 4 Terminal nodes		
1D	$h_1(t) = 0.3$	Constant CIF event 1,
	$h_2(t) = 0.1 + 0.2I(Z_1 > c) + 0.2I(Z_2 > c)$	CIF event 2 and censoring depend on covariates,
	$h_0(t) = 0.6 - 0.2I(Z_1 > c) - 0.2I(Z_2 > c)$	exponential model, $c = 0.3$
2D	$\mu_1(t) = 0.5$	Constant CIF event 1,
	$\mu_2(t) = 0.5I(Z_1 > c) + 0.5I(Z_2 > c)$	CIF event 2 and censoring depend on covariates,
	$\mu_0(t) = 1 - 0.5I(Z_1 > c) - 0.5I(Z_2 > c)$	log normal model, $c = 0.3$

Table 16: Investigating tree structure with constant CIF for event 1, Models 1C–2D

	Percent of terminal nodes in final tree							Variable Selection		Average	
Method	1	2	3	4	5	6	≥7	Inclusive	Exclusive	MAE1	MAE2
Model 1C: Exponential $h_1(t) = 0.3$, $c = 0.3$											
$h_2(t) = 0.1 + 0.3I(Z_1 > c \cap Z_2 > c)$											
$h_0(t) = 0.6 - 0.3I(Z_1 > c \cap Z_2 > c)$											
Mult	10	40	**15**	10	5	6	14	46	15	1.5210	1.2613
Uni	22	35	**9**	7	4	2	21	31	4	**1.5000**	1.2816
LR	0	8	**4**	2	3	1	82	91	4	1.5370	1.2487
Within-SS	2	4	**89**	5	0	0	0	94	89	1.5334	1.2385
Within-ABS	1	0	**98**	1	0	0	0	**99**	**98**	1.5335	**1.2382**
Model 2C: Log normal $\mu_1(t) = 0$, $c = 0.3$											
$\mu_2(t) = 0.5I(Z_1 > c \cap Z_2 > c)$											
$\mu_0(t) = 0.5 - 0.5I(Z_1 > c \cap Z_2 > c)$											
Mult	28	38	**13**	5	4	3	9	26	9	1.4402	1.1736
Uni	18	43	**8**	4	1	1	25	28	1	**1.4265**	1.1423
LR	1	8	**2**	0	1	1	87	**90**	1	1.4995	1.1165
Within-SS	8	54	**35**	3	0	0	0	34	33	1.4955	**1.1050**
Within-ABS	13	51	**34**	2	0	0	0	36	**34**	1.4954	1.1058
Model 1D: Exponential $h_1(t) = 0.3$, $c = 0.3$											
$h_2(t) = 0.1 + 0.2I(Z_1 > c) + 0.2I(Z_2 > c)$											
$h_0(t) = 0.6 - 0.2I(Z_1 > c) - 0.2I(Z_2 > c)$											
Mult	54	27	4	**4**	2	4	5	11	2	**1.6189**	1.5839
Uni	21	28	14	**3**	1	1	32	37	3	1.6474	1.5470
LR	4	6	2	**3**	0	2	83	**87**	1	1.6614	**1.5264**
Within-SS	70	16	10	**2**	2	0	0	10	**7**	1.6645	1.5326
Within-ABS	73	21	6	**0**	0	0	0	5	5	1.6651	1.5327
Model 2D: Log normal $\mu_1(t) = 0.5$, $c = 0.3$											
$\mu_2(t) = 0.5I(Z_1 > c) + 0.5I(Z_2 > c)$											
$\mu_0(t) = 1 - 0.5I(Z_1 > c) - 0.5I(Z_2 > c)$											
Mult	48	25	8	**7**	5	2	5	22	8	**0.8024**	0.7469
Uni	12	50	6	**3**	2	0	27	32	4	0.8207	0.7281
LR	0	6	0	**1**	0	2	91	**94**	0	0.8322	0.7220
Within-SS	4	43	51	**2**	0	0	0	53	52	0.8221	**0.7175**
Within-ABS	2	35	63	**0**	0	0	0	63	**63**	0.8226	0.7178

Investigating tree structure joint effect of Z_1, Z_2 on hazards keeping the CIF of event 1 constant when there are multiple events of interest. The bold-faced numbers represent either 1) the ideal number of nodes selected, or 2) the best result for each model. The variable selection measure "Inc" or "Inclusive signal" represents the percent of trees that selected both variables Z_1 and Z_2. The variable selection measure "Exc" or "Exclusive signal" represents the percent of trees that selected only Z_1 and Z_2.

from the list because of a loss to follow-up, or a change in the desire or need for a transplant. For reasons outlined in Section 3.4, this approach to censoring may introduce bias in the results. Classification trees that address competing risk were applied to this data in Chapter 5. However, this method focused on the situation where there was only one event of interest.

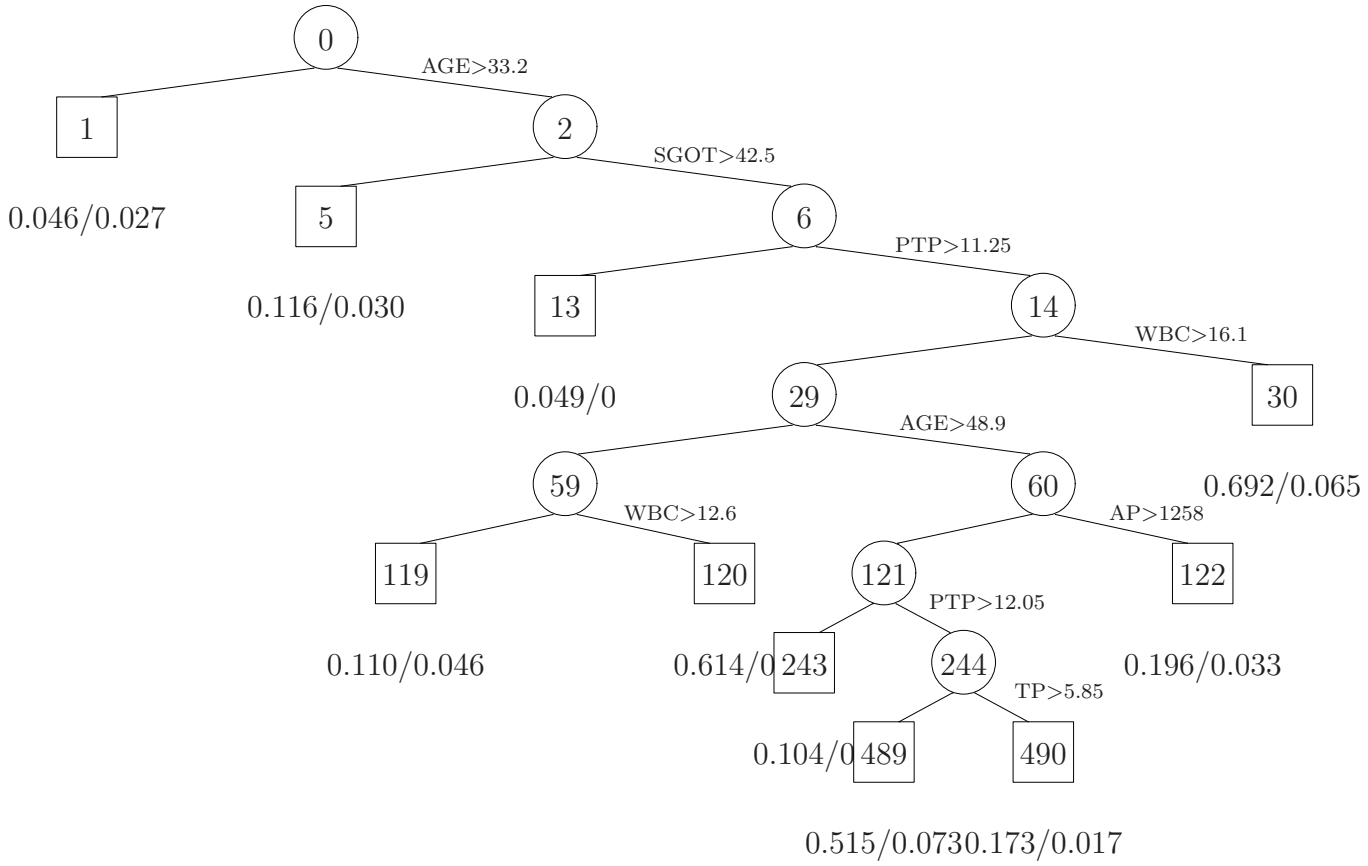
We used the UNOS data consisting of 1203 patients who were on the waiting list from April 1990 to June 1994 and were followed until April 1997. Of the 1203 patients, 342 died before transplant (pre-transplant death), 224 received a transplant, and 637 were removed from the list for other reasons. For our analysis, we treated pre-transplant death as the event of interest (event 1), removal from the list for other reasons as competing risks (event 2), and lack of an event as true censoring (event 0). We used the following covariates for Z_1 through Z_{14}, respectively: age, serum albumin level, serum alkaline phosphatase level, serum creatinine level, diabetes (no/yes), dialysis for renal disease (no/yes), muscle wasting (no/yes), prothrombin time, serum glutamic-oxaloacetic transaminase (SGOT) level, total bilirubin level, total protein level, ulcerative colitis (no/yes), history of variceal bleeding (no/yes), and white blood cell count. We ranked the 10 terminal nodes from the highest risk (1) to the lowest risk (10) of event 1, based on the probability of 100-day failure for event 1, and we repeated this for the corresponding risk of event 2.

Figure 20 shows the final trees selected by the multivariate tree method, Table 17 compares the summary data for the terminal nodes of this tree, and Figure 21 shows the plots the CIFs for the nodes.

The final multivariate tree (Figure 20) had ten terminal nodes, splitting on age ($\leq$33.2 vs. $>$33.2 years, and splitting again at $\leq$48.9 vs. $>$48.9 years), SGOT level ($\leq$42.5 vs. $>$42.5 units/L), prothrombin time ($\leq$11.25 vs. $>$11.25 seconds and $\leq$12.05 vs. $>$12.05 seconds), white blood cell count ($\leq$16.1 vs. $>$16.1 10^3/L and $\leq$12.6 vs. $>$12.6 10^3/L), alkaline phosphatase ($\leq$1258 vs. $>$1258 units IU/L), and total protein ($\leq$5.85 vs. $>$5.85 gm/dL). These are very different variables from those identified by a classification tree method for competing risks when only pre-transplant death was the event of interest. In that analysis, albumin level, bilirubin level and age were the three variables selected to identify distinct risk groups for pre-transplant death. The MELD (based on a Cox proportional hazards regression) uses creatinine, bilirubin and prothrombin time to identify subgroups.

The group of highest risk of 100 day event 1 failure (pre-transplant death) is in node 30 ($p = 0.692$) which is defined by higher age (AGE$>$ 33.2), SGOT levels (SGOT$>$ 42.5), prothrombin time (PTP$>$ 11.25) and white blood cell count (WBC$>$ 16.1). This node is also associated with high risk for event 2 ($p = 0.065$). In general, the factors associated with a high risk for event 1 as indicated by the tree are high age, high SGOT levels, high prothrombin time, high white blood cell count, low alkaline phosphatase levels, and low total protein levels. The one exception to this pattern is the split on prothrombin time at node 121. Here, the lower prothrombin time (PTP$\leq$ 12.05) has a higher risk for event 1 (0.614) than the other terminal nodes that follow from this split (0.104 node 243, 0.515 node 489, 0.173 node 490). This effect occurs in the branch defined by higher age group (AGE$>$ 48.9) and lower alkaline phosphatase level (AP$\leq$ 1258). It seems that the tree indicates that the risk of elevated prothrombin time changes depending on age and alkaline phosphatase level. In contrast, the group with the lowest risk for pre-transplant death is node 1 (AGE$\leq$ 33.2, $p = 0.046$).

For event 2 (transplant and other known reasons for removal from waitlist), the node associated with highest risk of 100 day failure (probability of experiencing event 2 within 100 days is 0.073) is node 489 which is defined by the following splits: AGE$>$ 33.2, SGOT$>$ 42.5, PTP$>$ 11.25 (prothrombin time), WBC$\leq$ 16.1 (white blood cell count), AGE$>$ 48.9, AP$\leq$ 1258 (alkaline phosphatase level), PTP$>$ 12.05 (prothrombin time), TP$\leq$ 5.85 (total protein). This node is also associated with high pre-transplant death risk ($p = 0.515$). In general, high prothrombin time and low total protein are risk factors for removal from the waitlist due to reasons other than transplant. The relationship of the risk of event 2 with the other covariates is not straightforward, and there appear to be many conditional relationships between the covariates that affect the risk of event 2.



10	6	7	9	2	8	3	5	4	1
6	5	3	9=	9=	9=	1	7	4	2

Figure 20: The multivariate classification tree for patients on liver transplant waitlist.

The 100 day failure probabilities for event 1 and event 2 are given under each terminal node (p_1/p_2). The risk ranking (1-10) for event 1 and 2 for each terminal node are given under the tree, with ties indicated by the "=" sign, e.g. "9=". Note that the probability of failing from event 2 is equal to zero for nodes 13, 120 and 243. AGE = age in years, SGOT = glutamic-oxaloacetic transaminase (SGOT) level, PTP = prothrombin time, WBC = white blood cell count, AP = alkaline phosphatase level, and TP = total protein.

6.6 DISCUSSION

Classification trees for survival data are useful for categorizing patients into several groups with similar risks. Some trees have been developed for multivariate survival data, but none of these trees can be applied to competing risks. We proposed a between-node tree-based technique specifically designed for data with competing risks where more than one event is of interest. The tree that maximizes the between-node difference is based on a composite measure of the cumulative incidence functions of the events of interest, and we compared this to univariate competing risk trees proposed in Chapter 5 and an existing univariate survival tree.

In general, we found that the multivariate method was good at detecting the correct cutpoint for covariates related to the survival times. However, when it came to detecting data structure, no method appeared to be able to select the correct covariates or the correct number of terminal nodes with reliability. The one exception was the univariate within-node methods in the situation where the CIF for event 1 was independent of the covariates and the CIF for event 2 was dependent on the covariates with a multiplicative hazard. In that case the within-node methods performed very well.

In the future, we will explore other ways of detecting between-group difference when multiple competing risks are of interest. We will also expand our work to the competing risk-adjusted rank-based test for the equality of survival functions.

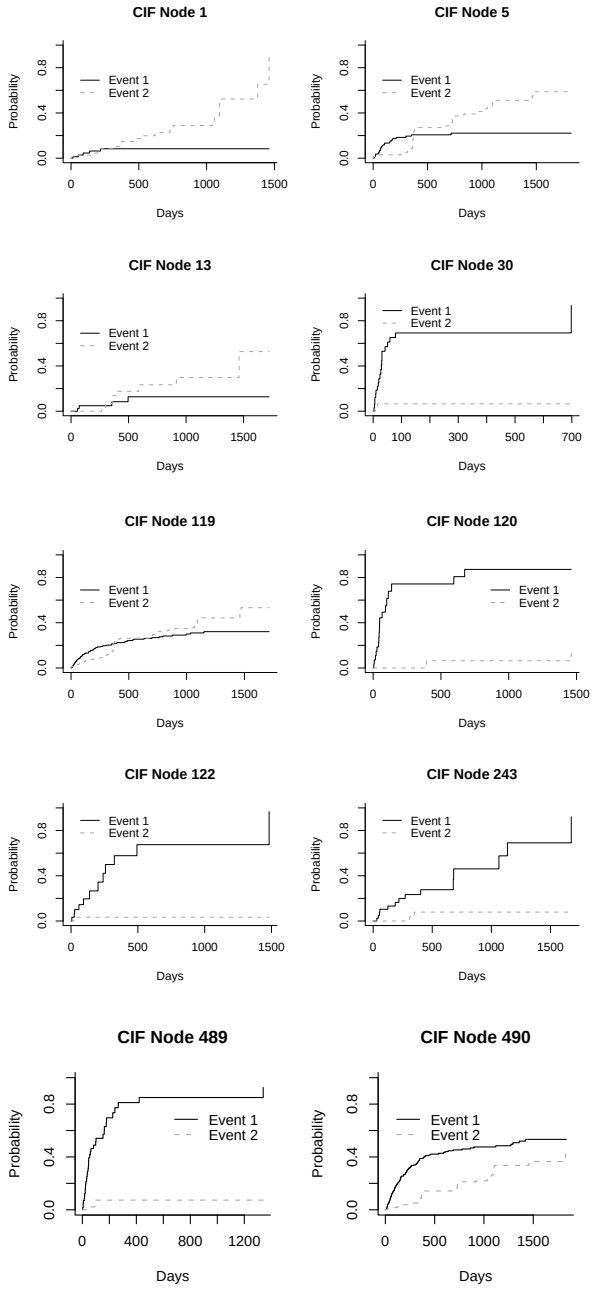


Figure 21: CIFs of both events for each terminal node of the multivariate tree.

The cumulative incidence functions (CIFs) of the event of interest for each terminal node generated from the multivariate tree for liver transplant data. Event 1 outcome is pretransplant death, Event 2 outcome is transplant or other outcome.

Table 17: Liver transplant wait-list data, multivariate tree summary

Terminal Node		**N and 100 day failure probability**				Risk Group Event 1	Risk Group Event 2
		Total	Event 1	Event 2	Censoring		
1	N	86	5	22	59		
	100-day risk		0.0457	0.0266		10	6
5	N	135	25	40	70		
	100-day risk		0.1163	0.0299		6	5
13	N	42	4	9	29		
	100-day risk		0.0488	0		9	8=
30	N	33	20	2	11		
	100-day risk		0.6921	0.0651		1	2
119	N	364	74	83	207		
	100-day risk		0.1099	0.0458		7	3
120	N	28	18	2	8		
	100-day risk		0.6143	0		2	8=
122	N	30	12	1	17		
	100-day risk		0.1959	0.0333		4	4
243	N	39	14	2	23		
	100-day risk		0.1040	0		8	8=
489	N	59	37	3	19		
	100-day risk		0.5152	0.0731		3	1
490	N	387	133	60	194		
	100-day risk		0.1726	0.0168		5	7
Total	N	1203	342	224	637		

7.0 FUTURE WORK

7.1 DISTANCE FUNCTIONS: BETWEEN- AND WITHIN-NODE

The first avenue for future research is the investigation of different distance functions for within-node homogeneity or between-node heterogeneity.

With regard to between-node methods, any kind of two-sample statistic can be used. One future avenue of research for competing risks, is to develop a tree-method based on a two-sample test of the event-specific hazard. For reasons outlined in Section 3.4, analysis based on hazards and crude probabilities provide a full picture of the competing risk data.

In the case of continuous outcomes and within-node methods, almost any non-negative function can be used as an impurity function. More formally, if we let $h = \{y_i : i = 1, ..., n\}$ denote a node or set of observations, where $y_i \in \mathbb{R}^1$ denote the ith real-valued observation in node h, then the impurity function $\phi(h)$ has to satisfy the following properties,

1. $\phi(h) \geq 0$,
2. $\phi(h) = 0$ when outcomes $y_i \in h$ are identical, i.e. when $y_i = y_j$ for all i, j

The distributional properties of an impurity function are not required in order to use it as the basis for a tree-method. In fact, that almost any impurity function can be chosen because it is interpretable or applicable to the outcome without requiring distributional results is one of the attractive features of tree methods. There is a lot of scope for investigating different impurity functions for competing risks. In sections 7.2 we propose a method based on a univariate variance-based impurity and in Section 7.3 we discuss the challenges in developing an impurity function for multivariate methods for competing risks.

7.2 VARIANCE-BASED WITHIN-NODE UNIVARIATE TREE

In this section we present a CART-style competing risk tree where the continuous outcome is based directly on the survival time. Specifically, we describe a measure of within-node impurity $\phi(h)$ based on the variance of survival times. The main advantage of a variance-based tree is complexity; the variance of the survival time is a relatively straight-forward way to measure impurity and it is computationally less complicated than the residual-based or test-based (log-rank) approaches. We begin this section by outlining the tree for univariate outcomes as proposed by Jin, et al. [16], and then we develop the tree for competing risks using variance of survival times. After we define $\phi(h)$ for the competing risk situation, the rest of the tree machinery is identical to that of section 5.3, because they are both within-node, CART-style trees.

7.2.1 Review: Variance-based Univariate Survival Tree.

The following is a description of the univariate survival tree based on variance of survival time as proposed by Jin, et al. [16]. Define the truncated mean survival time $m(\tau)$ for node h to be

$$m_h(\tau) = \int_0^\tau S_h(x)dx$$

where the survival times are truncated at τ and S_h is the survival function restricted to the observations in node h. The impurity function for node h used for the non-competing risk survival tree is

$$\phi(h) = (m_h(\tau) - m(\tau))^2 p_h$$

where p_h is the probability of an observation being in node h and $m(\tau)$ is the truncated mean for the whole sample, $m(\tau) = \int_0^\tau S(x)dx$. The measure of within-node impurity for a tree T is

$$\phi(T) = \sum_{h \in T^t} \phi(h)$$

The corresponding split function for parent node p into daughter nodes L and R is

$$\phi_{var}(s, p) = \phi(L) + \phi(R) - \phi(p)$$

The estimates of m, m_h and p_h are denoted $\hat{m}$, $\hat{m}_h$ and $\hat{p}_h$, and are the corresponding Kaplan-Meier estimators and the observed proportion of observations in each node, respectively. Jin et al, modify the pruning algorithm a little, but otherwise rest of the univariate CART machinery goes through in the usual manner.

7.2.2 Growing, pruning and selection for variance-based tree.

We can adapt the univariate survival tree described in Section 7.2.1 to the competing risks situation by considering the cumulative incidence function restricted to type 1 events. For simplicity, suppose we have two competing risk events $j = 1, 2$ and $i = 1, ..., D$ unique event times, $t_1 < ... < t_D$. Let d_{ij} be number of j-type events at time i and Y_i number of subjects at risk at time i. We propose the mean of the cumulative incidence function for event 1 to be

$$m_1(\tau) = \int_0^\tau F_1(x)dx$$

The estimate for this is

$$\hat{m}_1 = \int_0^\tau CIF_1(u)du$$

where

$$\begin{aligned} CIF_1(t) &= \sum_{t_i \le t} \frac{d_{i1}}{Y_i} KM_{12}(t_i) \\ KM_{12} &= KM_1(t)KM_2(t) \\ KM_j(t) &= \prod_{t_i \le t} \left(1 - \frac{d_{ij}}{Y_i}\right) \end{aligned}$$

$KM_j(t)$ is the Kaplan-Meier estimator for j-type events. Recall that this quantity estimates the net survival function for event j. Let KM_{12} denote the Kaplan-Meier function obtained by considering any failures of any kind as an event and it estimates the probability of surviving all causes of failure by time t. Now we let the subscript "h" denote the same quantities as above, but restricted to the observations in node h. Then we have the corresponding

impurity and split functions,

$$\begin{aligned} \phi_1(h) &= (m_{1h}(\tau) - m_1(\tau))^2 p_h \\ \phi_1(T) &= \sum_{h \in T^t} \phi_1(h) \\ \phi_{1var}(s,p) &= \phi_1(L) + \phi_1(R) - \phi_1(p) \end{aligned}$$

We obtain the estimates of these functions by substituting $\hat{m}_{1h}$, $\hat{m}_1$ and $\hat{p}_h$.

7.3 WITHIN-NODE MULTIVARIATE TREE FOR COMPETING RISKS

Molinaro, et al. [22] outline a general schema for constructing univariate and multivariate survival trees using the within-node approach of the CART algorithm. This paper is the only attempt in the literature to construct a multivariate survival tree using the CART algorithm; the other methods adopt a between-node approach with a two-sample statistic that accounts for the correlation in some manner, e.g. robust log-rank test or likelihood ratio test based on a frailty model.

The goal of Molinaro et al. [22] is to unify the approach to constructing survival trees and CART. With other CART-style, within-node survival trees the measure of performance (impurity function ϕ) varies from tree to tree in a somewhat ad hoc fashion. The impurity function is usually chosen because it is a good metric for handling censoring in failure-time data, rather than because it associated with a loss function that is of most interest to the researcher. The general approach of Molinaro et al. [22] is to construct a loss function for the full data (without regard to censoring) and then adjust the loss function for censoring using inverse probability of censoring weights (IPCW). The resulting loss function for the observed (censored) data has the same expected value as the loss function for the full data, and, when there is no censoring, the loss function reduces to the usual sums-of-squares impurity function used in regression trees. This adjusted loss function as the basis for the impurity function $\phi(h)$. The rest of the CART algorithm goes through in the usual manner. The IPCW originally proposed by Robins and Rotnitzky [27] is used.

We begin by describing the schema for the survival tree for univariate outcomes. The first step is to decide on a *parameter of interest* ψ for generating splits. This is usually the mean, median or mean of the log survival times. Molinaro et al. [22] then create a loss function that is used to measure the performance based on the full data (as if no censoring were present). The authors illustrate their schema using the mean of the log survival time ($Z = \log T$) and quadratic loss,

$$L(X, \psi) = (Z - \psi(W))^2$$

where W are non-time-dependent covariates and $\psi(W)$ is the conditional mean given the time-independent covariates, $\psi(W) = E[Z|W]$. Formally, let X represent the *full data structure*, $X \equiv \bar{X}(T) = \{X(t) = (R(t), L(t)) : 0 \leq t \leq T\}$ where $X(t)$ is a multivariate stochastic process, $R(t) \equiv I(T \leq t)$, $L(t)$ is the covariate process and T is a random survival time. Let $L(0)$ denote non-time-dependent (baseline) covariates with $W \subseteq L(0)$.

The IPCW observed data loss function for quadratic loss is

$$L(O, \psi|\eta_n) = (Z - \psi(W))^2 \frac{\Delta}{\bar{G}_n(T|X)}$$

where $\bar{G}_n(T|X)$ denotes a consistent estimator of $\bar{G}_0(\cdot|X) = Pr(C > c|X)$, the *censoring survivor function*, i.e. the conditional survivor function for the censoring time C given full data X, $\Delta \equiv I(T \leq C)$ is the *censoring indicator* and η_n are any nuisance parameters associated with $\bar{G}_n(T|X)$. Let O represent the *observed data structure*, $O \equiv (\tilde{T}, \Delta, \bar{X}(\tilde{T}))$ where $\tilde{T} \equiv \min(T, C)$ of the survival time T and censoring variable C. If the assumption of *coarsening at random* (CAR) holds and we have no time-dependent covariates ($L = L(0)$), then $\bar{G}_0(\cdot|X)$ can be estimated with $\bar{G}_n(T|L(0))$ which is a function of the observed data only. CAR holds if the censoring distribution only depends on the observed process $\bar{X}(c)$. Results of van der Laan and Robins [34] guarantee that the expected value of the full data loss function is the same as the observed data loss function. We could estimate $\bar{G}_n(T|X)$ with the Kaplan-Meier function. Note that $L(O, \psi|\eta_n)$ is equal to zero when $\Delta = 0$. The corresponding impurity function (estimate of risk) $\phi(h)$ for node h is

$$\phi(h) = \frac{1}{n} \sum_{i=1}^{n_h} L(O_i, \psi|\eta_n)$$

In the multivariate setting, where each individual has m outcomes, the IPCW observed data loss function for quadratic loss is

$$L(O, \psi|\eta_n) = (Z - \psi(W))^T \Omega(W)(Z - \psi(W)) \frac{\Delta}{\bar{G}_n(T|X)} \tag{7.1}$$

where $\psi(W)$ and Z are $m \times 1$ vectors, and $\Omega(W)$ is a symmetric $m \times m$ matrix. A reasonable choice for Ω is the inverse of the conditional covariance matrix $\Sigma(W)$ of the outcome Z given the covariates W, $\Sigma(W) = E_0\big[(Z - \psi(W))(Z - \psi(W))^T|W\big]$. The impurity function is $\phi(h) = \frac{1}{n_h} \sum_{i=1}^{n_h} L(O_i, \psi|\eta_n)$.

When applying this approach to competing risks, it is not clear how to bridge the gap between the full data loss function and the observed loss function. Suppose that we consider a competing risk situation with two types of events, $j = 1, 2$. If we assume that an individual cannot experience two kinds of events, then the full data structure would involve the observation of exactly one event time for every subject. Let ζ_1 be the set of event 1 log failure times and ζ_2 be the set of event 2 log failure times. We propose that a reasonable candidate for the quadratic full data loss function is

$$L(X, \psi) = \begin{cases} (Z - \psi_1(W))^2 & \text{for } Z \in \zeta_1 \\ (Z - \psi_2(W))^2 & \text{for } Z \in \zeta_2 \end{cases}$$

where $\psi_1(W) = E[Z_1|W]$ and $\psi_2(W) = E[Z_2|W]$ are the mean times to type 1 and type 2 events respectively, conditioned on time-independent covariates W. The corresponding observed loss function is

$$L(O, \psi) = \begin{cases} (Z - \psi_1(W))^2 \frac{\Delta}{\bar{G}_n(T|X)} & \text{for } Z \in \zeta_1 \\ (Z - \psi_2(W))^2 \frac{\Delta}{\bar{G}_n(T|X)} & \text{for } Z \in \zeta_2 \end{cases} \tag{7.2}$$

where Δ indicates the presence of any event and $\bar{G}_n(T|X)$ is based on the "true censoring" i.e. where no event of either type is observed. The estimate of the risk or impurity function for node h is

$$\phi(h) = \frac{1}{|\zeta_1|} \sum_{\{i:Z_i \in \zeta_1\}} (Z_i - \psi_1(W))^2 \frac{\Delta_i}{\bar{G}_n(T_i|X_i)} + \frac{1}{|\zeta_2|} \sum_{\{i:Z_i \in \zeta_2\}} (Z_i - \psi_2(W))^2 \frac{\Delta_i}{\bar{G}_n(T_i|X_i)} \tag{7.3}$$

One advantage of the impurity function in (7.3) is that it is easy to see the contribution that each competing risk makes to the overall impurity. We would have to verify that the the

expected value of the full loss function is equal to the expected value of the observed loss function, as Molinaro et al. [22] has done for univariate and multivariate survival situations.

A disadvantage to the formulation of the loss function in equation (7.2) that there is no accounting for correlation, as there is in the composite between-node competing risk tree based on the Luo and Turnbull statistic. There does not seem to be a way around this whilst keeping to the general schema. For instance, one could argue that "full data" structure is one where two events are observed for every person. This does not make much medical sense for events such as death, but we could imagine a scenario where the occurrence of one type of event prevents us from observing the time to the second event, even though a finite, second event time does exist. In this case, the full data quadratic loss function would be more like the multivariate loss function in equation (7.1). However, with competing risk data we only observe (at most) one event. The general approach of Molinaro et al. is to remove the censored observations from the loss function and "replace" them with the IPCW. In this case, we would essentially have a univariate outcome adjusted with the IPCW and the loss function would reduce to equation (7.2). We might consider using a different method to calculate the IPCW. If it is possible for a person to experience a second event after the first event, even though the second failure time is not observed, then the occurrence of one event can be considered to be censoring the other event's failure time. In this case we might define $\bar{G}_1(T|X)$ to be the censoring survivor function where event 2 failures times are considered as censored event times, in addition to the "true" censoring times. A similar definition would hold for $\bar{G}_2(T|X)$. Again, we would have to verify that the expected value of the loss function for the full data and observed data are equal.

7.4 OTHER ISSUES

One further modification of the tree methods (that is not related to the choice of distance function) is the choice of the cost-complexity function for within-node methods or the complexity penalty function for between-node methods. These functions are designed to ensure parsimony in the tree model by balancing discrimination with the number of nodes in the

tree. They are used at the pruning steps of the tree methods in order to remove branches of the tree with little discriminatory power given the number of nodes they add to the tree (see sections 5.2.2 and 5.3.2). Both functions are based on the same principle as the Akaike Information Criterion (AIC) [1] and Schwarz Bayesian Information Criterion (BIC) [29] which are designed reflect the trade-off between likelihood associated with a regression model and the number of predictors. The cost-complexity and complexity-penalty functions reflect the trade-off between total homogeneity or heterogeneity and number of terminal or internal nodes. This suggests further improvement of the tree pruning algorithms based on other measures of the trade-off between discrimination and complexity based on the likelihood. However, one of the advantages of the tree-based methods would be lost, namely that they are often non-parametric and do not depend on specifying a model, on which a likelihood would need to be based.

APPENDIX

SIMULATION PROGRAM

```
# CIF multiple test, Luo and Turnbull 1999, Statistica Sinica
# generate random data
library(survival)
library(cmprsk)
library(rpart)
library(mvpart)


####
####
# The estimate of Luo Turnbull test
# tsim=survival time, dsim= 0/1/2 event indicator, r= weights for test
####
####
multest<-function(tsim, dsim,group,r){
cif<-cuminc(ftime=tsim,fstatus=dsim,group=group)
n1<-sum(group==1)
n2<-sum(group==0)
# event 1
ciftotal<-cuminc(ftime=tsim,fstatus=dsim)
W1<-(1-timepoints(ciftotal, (dsim==1)*tsim)$est[1,])^r
I11<-timepoints(cif, (dsim==1)*tsim)$est[1,]
I12<-timepoints(cif, (dsim==1)*tsim)$est[2,]
dI11<-I11
dI12<-I12
for(i in 2:length(I11)){
dI11[i]<-I11[i]-I11[i-1]
dI12[i]<-I12[i]-I12[i-1]
}
for(i in 1:length(dI11)){if(is.na(dI11[i])==T)dI11[i]<-0}
for(i in 1:length(dI12)){if(is.na(dI12[i])==T)dI12[i]<-0}
X1<-sqrt(n1*n2/(n1+n2))*sum(W1*(dI11-dI12))
# event 2
W2<-(1-timepoints(ciftotal, (dsim==2)*tsim)$est[2,])^r
I21<-timepoints(cif, (dsim==2)*tsim)$est[3,]
I22<-timepoints(cif, (dsim==2)*tsim)$est[4,]
dI21<-I21
dI22<-I22
for(i in 2:length(I21)){
dI21[i]<-I21[i]-I21[i-1]
dI22[i]<-I22[i]-I22[i-1]
}
for(i in 1:length(dI21)){if(is.na(dI21[i])==T)dI21[i]<-0}
for(i in 1:length(dI22)){if(is.na(dI22[i])==T)dI22[i]<-0}
X2<-sqrt(n1*n2/(n1+n2))*sum(W2*(dI21-dI22))
stat<-c(X1,X2)
stat
}
#multest(tsim, dsim, group, r)

addon<-function(B, N){
test<-N;
output<-B;
```

```
if(sum(N>1)!=0){
for(i in 1:length(N)){
if(N[i]>1){
output[i]<-B[i]/N[i];
}
}
}
output
}
#addon(B11,E11)


addon<-function(B, N){
test<-N;
output<-B;
if(sum(N>1)!=0){
for(i in 1:length(N)){
if(N[i]>1){
output[i]<-B[i]/N[i];
}
}
}
output
}

multest<-function(tsim, dsim,group,r){
cif<-cuminc(ftime=tsim,fstatus=dsim,group=group)
n1<-sum(group==1)
n2<-sum(group==0)
# event 1
ciftotal<-cuminc(ftime=tsim,fstatus=dsim)
W1<-(1-timepoints(ciftotal, (dsim==1)*tsim)$est[1,])^r
I11<-timepoints(cif, (dsim==1)*tsim)$est[1,]
I12<-timepoints(cif, (dsim==1)*tsim)$est[2,]
dI11<-I11
dI12<-I12
for(i in 2:length(I11)){
dI11[i]<-I11[i]-I11[i-1]
dI12[i]<-I12[i]-I12[i-1]
}
for(i in 1:length(dI11)){if(is.na(dI11[i])==T)dI11[i]<-0}
for(i in 1:length(dI12)){if(is.na(dI12[i])==T)dI12[i]<-0}
X1<-sqrt(n1*n2/(n1+n2))*sum(W1*(dI11-dI12))
# event 2
W2<-(1-timepoints(ciftotal, (dsim==2)*tsim)$est[2,])^r
I21<-timepoints(cif, (dsim==2)*tsim)$est[3,]
I22<-timepoints(cif, (dsim==2)*tsim)$est[4,]
dI21<-I21
dI22<-I22
for(i in 2:length(I21)){
dI21[i]<-I21[i]-I21[i-1]
dI22[i]<-I22[i]-I22[i-1]
}
for(i in 1:length(dI21)){if(is.na(dI21[i])==T)dI21[i]<-0}
for(i in 1:length(dI22)){if(is.na(dI22[i])==T)dI22[i]<-0}
X2<-sqrt(n1*n2/(n1+n2))*sum(W2*(dI21-dI22))
stat<-c(X1,X2)
stat
}

#multest(tsim, dsim, group, -1)


ciftest<-function(tsim, dsim,Z,cutpoint,r){
group<-Z<=cutpoint;
cif<-cuminc(ftime=tsim,fstatus=dsim,group=group);
ciftotal<-cuminc(ftime=tsim,fstatus=dsim);
n1<-sum(group==F);
n2<-sum(group==T);
## Event 1 group1
test11<-survfit(Surv(tsim,(dsim==1)^2), subset=(group==F));
Y11<-test11$n.risk;
E11<-test11$n.event;
W11<-(1-timepoints(ciftotal, tsim*(group==F))$est[1,])^r
W11<-W11[-1]
I11<-timepoints(cif, tsim*(group==F))$est[1,]
dI11<-I11
for(i in 2:length(I11)){
dI11[i]<-I11[i]-I11[i-1]
}
```

```
I11<-I11[-1]
dI11<-dI11[-1]
for(i in 1:length(I11)){if(is.na(I11[i]))I11[i]<-0}
for(i in 1:length(dI11)){if(is.na(dI11[i]))dI11[i]<-0}
B11<-W11*dI11*(E11>0)^2
B11c<-cumsum(B11)
B11T<-sum(B11)
#A11
B11Tvec<-B11T*rep(1, length(B11))
A111<-(B11Tvec-B11c)/(Y11/n1)-addon(B11,E11)*n1
A112<-(B11Tvec-B11c)/(Y11/n1)
## A121 and A122
test12<-survfit(Surv(tsim,(dsim==1)^2), subset=(group==T))
Y12<-test12$n.risk
E12<-test12$n.event
W12<-(1-timepoints(ciftotal, tsim*(group==T))$est[1,])^r
W12<-W12[-1]

I12<-timepoints(cif, tsim*(group==T))$est[2,]
dI12<-I12
for(i in 2:length(I12)){
dI12[i]<-I12[i]-I12[i-1]
}
I12<-I12[-1]
dI12<-dI12[-1]
for(i in 1:length(I12)){if(is.na(I12[i]))I12[i]<-0}
for(i in 1:length(dI12)){if(is.na(dI12[i]))dI12[i]<-0}

B12<-W12*dI12*(E12>0)^2
B12c<-cumsum(B12)
B12T<-sum(B12)
B12Tvec<-B12T*rep(1, length(B12))
A121<-(B12Tvec-B12c)/(Y12/n2)-addon(B12,E12)*n2
A122<-(B12Tvec-B12c)/(Y12/n2)
############### A21: event 2 group 1, Ajml = event j, group m, l

test21<-survfit(Surv(tsim,(dsim==2)^2), subset=(group==F))
Y21<-test21$n.risk
E21<-test21$n.event
W21<-(1-timepoints(ciftotal, tsim*(group==F))$est[2,])^r
W21<-W21[-1]
I21<-timepoints(cif, tsim*(group==F))$est[3,]
dI21<-I21
for(i in 2:length(I21)){
dI21[i]<-I21[i]-I21[i-1]
}
I21<-I21[-1]
dI21<-dI21[-1]
for(i in 1:length(I21)){if(is.na(I21[i]))I21[i]<-0}
for(i in 1:length(dI21)){if(is.na(dI21[i]))dI21[i]<-0}

B21<-W21*dI21*(E21>0)^2

B21c<-cumsum(B21)
B21T<-sum(B21)
B21Tvec<-B21T*rep(1, length(B21))
A212<-(B21Tvec-B21c)/(Y21/n1)-addon(B21,E21)*n1
A211<-(B21Tvec-B21c)/(Y21/n1)
############### A22: event 2 group 1, Ajml = event j, group m, l

test22<-survfit(Surv(tsim,(dsim==2)^2), subset=(group==T))
Y22<-test22$n.risk
E22<-test22$n.event
W22<-(1-timepoints(ciftotal, tsim*(group==T))$est[2,])^r
W22<-W22[-1]
I22<-timepoints(cif, tsim*(group==T))$est[4,]
dI22<-I22
for(i in 2:length(I22)){
dI22[i]<-I22[i]-I22[i-1]
}
I22<-I22[-1]
dI22<-dI22[-1]
for(i in 1:length(I22)){if(is.na(I22[i]))I22[i]<-0}
for(i in 1:length(dI22)){if(is.na(dI22[i]))dI22[i]<-0}
B22<-W22*dI22*(E22>0)^2
B22c<-cumsum(B22)
B22T<-sum(B22)
B22Tvec<-B22T*rep(1, length(B22))
A222<-(B22Tvec-B22c)/(Y22/n2)-addon(B22,E22)*n2
A221<-(B22Tvec-B22c)/(Y22/n2)
```

```
sigma1<-(n1*n2/(n1+n2))*(sum((A111^2/n1)*E11/n1)+
sum((A121^2/n2)*E12/n2)+sum((A112^2/n1)*E21/n1)+
sum((A122^2/n2)*E22/n2))

sigma2<-(n1*n2/(n1+n2))*(sum((A211^2/n1)*E21/n1)+
sum((A221^2/n2)*E22/n2)+sum((A212^2/n1)*E21/n1)+
sum((A222^2/n2)*E22/n2))

##### Covariance
#l=1
covar<-(n1*n2/(n1+n2))*(sum(A111*A211*E11/n1^2)+
sum(A121*A221*E12/n2^2)+
sum(A112*A212*E21/n1^2)+
sum(A122*A222*E22/n2^2))
covmat<-matrix(c(sigma1, covar, covar, sigma2), nrow=2)
teststat<-matrix(multest(tsim, dsim, group, r), nrow=2)
HoetT<-t(teststat)%*%solve(covmat)%*%teststat
HoetT
}

#multest(tsim, dsim, group, -1)
#ciftest(tsim, dsim, Z1,group, r)
#multest(tsim, dsim, Z1>min(Z1), -1)



############################################################
#################################################################

# program that computes bootstrap estimates
# for Gray tree, logrank and event-specific residual tree
#################################
#################################
# Tree programs, Gray, LR, Martingale residual trees, Multivariate tree
#########################################################

#####################################################################
## ss is subset of data that will be used to calculate the test statistic
## cuminc has a 'subset' option.
# takes in nodevec, the vector that states which obs is in which current node
maxsplit<-function(nodevec,node,x1,t,delta,ss2){
ss<-node==nodevec;
i<-1;
output<-rep(0,2);
#for(i in 1:length(x1)){
for(i in 1:length(unique(x1))){
if(nodevec[i]==node){
temp5<-sort(unique(x1));
temp3<-temp5[i];
temp4<-temp5[i+1];
z<-x1<=temp3;
temp<-cuminc(t,delta,z,subset=ss, rho=-1)$Test[1];
if(is.null(temp)==T|sum(z)<=ss2|sum(1-z)<=ss2){temp=0};
if(temp>output[1]){
# take the median of the cutpoint and the next highest value
temp2<-median(c(temp3, temp4));
output<-c(temp,temp2)
}
}
}
output
}
#maxsplit(nd,0,x1,t,delta,ss2)

#########################################################################
# maxsplit2
# input: covariates (cbind of x vectors), time=t, delta= event indicator , node, nodevec is the vector
# that states which obs in in which node
# output:  the best split over all covariates,
# statistic, cutpoint, variable (1,2,3.. column of the covariates matrix)
# and the node 0,1,2,3....
maxsplit2<-function(nodevec,covariates,t,delta, node,ss2){
temp<-c();
temp2<-c();
output<-rep(0,4);
for(j in 1:dim(covariates)[2]){
temp<-maxsplit(nodevec,node, covariates[,j],t,delta,ss2);
#temp2<-c(temp[1],temp[2],j,node);
#temp3<-cbind(output, temp2);
```

```
if(temp[1]>output[1]){output<-c(temp[1],temp[2],j,node)}
}
#temp3
output
}
#maxsplit2(nd,covariates,t,delta,0,ss2)
#####################################################################
# grow tree
# input:  covariates, t, delta
# output:  list of splits (stat, cutoff, var, node)
# and a vector called 'node' will say which node each obs is in.
# sampsize is the min number of obs in each node
growtree<-function(covariates, t, delta, sampsize,ss2){
node<-rep(0,length(delta));
nodelist<-c(0);
output<-c();
i<-1;
while(i <=length(nodelist)){
#check that there are event 1,2
# call is ss test (sample size test)
sstest1<-sum((node==nodelist[i])*(delta==1));
sstest2<-sum((node==nodelist[i])*(delta==2));
if(sstest1>0&sstest2>0&sum(node==nodelist[i])>=sampsize){
temp<-maxsplit2(node,covariates,t,delta,nodelist[i],ss2);
if(temp[1]>0){
nodelist<-c(nodelist,2*nodelist[i]+1,2*nodelist[i]+2);
node<-(node==nodelist[i])*(covariates[,temp[3]]<=temp[2])*(2*nodelist[i]+1)+
(node==nodelist[i])*(covariates[,temp[3]]>temp[2])*(2*nodelist[i]+2)+node*(node!=nodelist[i]);
output<-cbind(output,temp)
}
}
#output<- nodelist;
i<-i+1
}
#temp
output
#nodelist
#node
}
#x<-growtree(covariates,t,delta,sampsize,ss2)

#####################################################################
## pruning the tree

# need to calculate G(T) and # internal nodes beneath current node
#+1
# Take output from 'growtree'
# Start at end of list
# Basic idea:  does current node have children on list?
# if yes then find children nodes and add their G(T) and # nodes
# if no then G(T)=stat, # nodes=1
# takes in growtree object
# output growtree object with 2 rows : one with G(T) and # int nodes
prunetree<-function(obj){
# n is number of internal nodes in list
n<-dim(obj)[2];
# combines the split statistic, row of 1s (will be no. int nodes), and node number
output<-t(cbind(obj[1,],matrix(1,n), obj[4,]));
for(i in 1:n){
j<-n-i+1;
currentnode<-obj[4,j];
allnodes<-obj[4,];
if(sum(allnodes==(2*currentnode+1)|allnodes==(2*currentnode+2))>0){
for(k in 1:(n-j)){
l<-n-k+1;
# if the current node j is parent of node l, then add stats from node l
if(obj[4,l]==(2*currentnode+1)|obj[4,l]==(2*currentnode+2)){
output[1,j]<-output[1,j]+output[1,l];
output[2,j]<-output[2,j]+output[2,l];
}
}
}
# if there are daughter nodes on list then need to find them and add G(T)
# and no. nodes to intnode
}
output
}
#y<-prunetree(x)
#y
# We calculate G(T)/number of internal nodes = cost complexity
```

```
# Find int node with smallest cost complexity
# prune this node and all child nodes (remove from the tree list)
# store this new pruned tree somewhere
# then use 'pruntree' to prune this new tree object.

#####################################################################
# descendant will
#input:  node to prune
# output: vector stating all child nodes that will be pruned (pruned=T, not pruned=F)
descendant<-function(pruneobj, node){
pruned<-pruneobj[3,]==node;
for(i in 1:dim(pruneobj)[2]){
if(pruned[i]==T){
pruned<-pruned==T|pruneobj[3,]==(2*pruneobj[3,i]+1)|pruneobj[3,]==(2*pruneobj[3,i]+2)
}
}
pruned
#pruneobj
}
#descendant(yMult,4)

#####################################################################
# ancestor indicates which nodes are descendants of current node
# input: node, and tree obj
# output:  vector T/F ancestor of current node
ancestor<-function(pruneobj, node){
currentnode<-node;
parent<-pruneobj[3,]<0;
n<-dim(pruneobj)[2];
for(i in 1:n){
j<-n-i+1;
#if node  is odd, and currentnode is parent, and in the list
if(floor(currentnode/2)!=currentnode/2){
if(((currentnode-1)/2)==pruneobj[3,j]){
currentnode<-pruneobj[3,j];
parent[j]<-T
}
}
# if the node is even
else{
if(((currentnode-2)/2)==pruneobj[3,j]){
currentnode<-pruneobj[3,j];
parent[j]<-T
}
}
}
parent
}
#ancestor(yMult,1)
#############################################################################
# program to actually prune
# takes pruneobj with intnodes, G(t),node
# outputs same object with:
# prune order, and min alpha's

# tobeprune is T if node not yet pruned, F is node is pruned, updates during algorithm
# pruneorder records the step that a node was pruned on
# alphalist records the alpha used to prune the node
# alpha is list of alphas at each prune step, updates during algorithm
# currentnode records last node to be pruned, updates during algorithm

prune2<-function(pruneobj){
output<-c();
n<-dim(pruneobj)[2];
temp<-t(matrix(1,n));
tobeprune<-pruneobj[3,]>=0;
GTlist<-pruneobj[1,];
maxGT<-matrix(0,1,n);
# maxGT records the G(T) of each pruned subtree
intnodelist<-pruneobj[2,];
# no of internal nodes in each pruned subtree
intnode<-matrix(0,1,n);
pruneorder<-matrix(0,1,n);
alphalist<-matrix(0,1,n);
alpha<-GTlist/intnodelist;
minalpha<-min(alpha/tobeprune);
currentnode<-0;
inodes<-pruneobj[3,];
j<-0;
while(sum(tobeprune)!=0){
```

```
j<-j+1;
stepnum<-j*temp;
firstmin<-T;
minalphavec<-minalpha*temp;
for(i in 1:n){
alpha<-GTlist/intnodelist;
minalpha<-min(alpha/tobeprune)
# firstmin, just records the first time got into if statement , in case of ties
# if(alpha[i]==minalpha&firstmin==T){
if((alpha[i]/tobeprune[i])==minalpha&firstmin==T){
# firstmin, makes sure only use one alpha, in case of ties
firstmin<-F;
currentnode<-pruneobj[3,i];
temp1<-descendant(pruneobj, currentnode);
# tobeprune<-tobeprune&(!temp1);
maxGT<-maxGT+temp1*max(GTlist*tobeprune)*(maxGT==0);
intnode<-intnode+temp1*max(intnodelist*tobeprune)*(intnode==0);
tobeprune<-tobeprune&(!temp1);
pruneorder<-pruneorder+j*temp1*(pruneorder==0);
alphalist<-alphalist+minalpha*temp1*(alphalist==0);
temp2<-ancestor(pruneobj,currentnode);
GTlist<-GTlist-temp2*GTlist[i];
intnodelist<-intnodelist-temp2*intnodelist[i];
}
}
cnode<-currentnode*temp;
# output<-rbind(output,stepnum,alpha,pruneorder,alphalist, cnode)
# output<-rbind(output,GTlist,maxGT);
}
output<-rbind(inodes,pruneorder, alphalist, maxGT, intnode);
output
}
#y<-prune2(y)
yMult<-prune2(yMult)
#Outputs the 1) node number of large tree To
# 2) step that the branch with the node as the root was pruned
# 3) alpha used to prune that branch
# 4) max GT on that subtree pruned at step
# 5) number of internal nodes in that sub tree at step



#################################################################
# Tree Selection
# first need to create function 'prune3' that takes a large Tree and a
# particular alpha, and outputs the tree pruned to that alpha
# Input:  alpha, tree info
# Output: G(T) of the pruned tree (not G_alpha_T)

# function, maxsplitcomp
# this takes list of pruned sub trees and an alpha
# and calculates max G alpha T over the trees.
# this is the GT of the tree pruned with alpha

prune3<-function(alpha,subtrees){
GalphaT<-subtrees[4,]-alpha*subtrees[5,];
temp<-max(GalphaT);
for(i in 1:dim(subtrees)[2]){
if(GalphaT[i]==temp){output<-subtrees[4,i]}
}
output
}
#prune3(1,yMult)
################################################
#Function to calculate bootstrap adjustment
# need to create a sub function that 'runs' data
# down  given tree rules, to get GT
# G(L1;L2 T)
#mydata[sample(dim(mydata)[1], 10),] sample 10 rows

# takes 'treedata' from the growtree function

rundowntree<-function(treedata,newcov, newt, newd){
n<-dim(treedata)[2];
node<-0;
newstat<-matrix(0,1,n);
nodelist<-matrix(0,1,dim(newcov)[1]);
for(i in 1:n){
# for(i in 1:8){
temp2<-0
```

```
# update current node (scalar)
node<-treedata[4,i];
# update the variable that we will split on (goes with node)
currvar<-treedata[3,i];
# temp is the binary split on the variable and cutoff
temp<-newcov[,currvar]<=treedata[2,i];
# ss indicates the subset of data corresponding to current node
ss<-nodelist==node;
# temp2 is the current split statistic
# must change to 0 if it is -1 or null
if(sum(ss*newd)==0){temp2<-0}
if(sum(ss*temp*(newd==1))>0&sum(ss*!temp*(newd==1))>0&
sum(ss*temp*(newd==2))>0&sum(ss*!temp*(newd==2))>0){
temp2<-cuminc(newt,newd,temp,subset=ss, rho=-1)$Test[1]
}
if((is.null(temp2)==T)){temp2<-0};
if(temp2<0){temp2<-0};
newstat[i]<-temp2;
# nodelist tells us which data point is in which node (vector)
nodelist<-(nodelist==node)*(temp)*(2*node+1)+
(nodelist==node)*(!temp)*(2*node+2)+nodelist*(nodelist!=node)
}
newstat
}
#rundowntree(treedata,newcov, newt, newd)
#rundowntree(xMult,covariates,tsim,dsim)
#########################################################################

# omb1 calculates the first part of obm
# G(Lb;Lb,Tmb) using alphadash=sqrt(alpham*alpham+1)
# input:  pruned list from large tree, the bootstrap sample
# ovariates, time, delta, and samplesize for splitting
# output:  a list of G(T), one for each alphamdash,
# chosen from list of bootstrap pruned trees, as max Galphadash_m(T)


obm1<-function( origprunedtree, bsprunedtree,bcov,bt,bd, sampsize){
# generate the alpha-dash_m's
temp1<-sort(unique(origprunedtree[3,]))
temp2<-c(temp1,Inf)
temp3<-c(0,temp1)
alphadash<-sqrt(temp2*temp3)
# grow large tree using bootstrap sample
#Tb0<-growtree(bcov, bt, bd, sampsize)
# prune bs tree
#pruneTb0<-prune2(prunetree(Tb0))
GTbm<-matrix(0,1,length(alphadash))
# for every alphadash m, find the G(T)
for(i in 1:length(alphadash)){
GTbm[i]<-prune3(alphadash[i],bsprunedtree)
}
# now find G(original data; bootstrap data, Tbm)
#GTbmorig
# need a function that takes a Large Tree and pruning order
# and can caluclate GT for each pruned subtree
GTbm[length(alphadash)]<-0;
GTbm
#alphadash
}
#####################################################################
# need a function where, input: tree data, and prune step
# output: tree data with every node prunestep or greater removed
# treedata2 is the rbind of orign tree data adn pruned data
removeprune<-function(treedata2, prunestep){
treedata3<-c()
for(i in 1:dim(treedata2)[2]){
if(treedata2[6,i]>=prunestep){treedata3<-cbind(treedata3,treedata2[,i])}
}
treedata3
}
#####################################################################
# obm2 calculates the other part of obm
# G(L;Lb,Tm)
# input:
# large bootstrap tree, pruned bs tree, original data (time, delta, cov)
# and orignal large pruned tree (to calc alphadashm's)

obm2<-function(bstree, bsprune, ot, od, ocov,prunedtree){
# generate the alpha-dash_m's
temp1<-sort(unique(prunedtree[3,]))
```

```
temp2<-c(temp1,Inf)
temp3<-c(0,temp1)
alphadash<-sqrt(temp2*temp3)
btreedata<-rbind(bstree,bsprune);
GTmbmatrix<-c();
#listGTL<-c();
for(i in 1:max(btreedata[6,])){
temptree<-removeprune(btreedata,i);
GTL<-sum(rundowntree(temptree,ocov,ot,od));
# need to find vector of G_alphadash(T)
Galphadash<-GTL-alphadash*(max(temptree[9,]));
Galphadash[length(Galphadash)]<-0;
GTmbmatrix<-rbind(GTmbmatrix,Galphadash);
# listGTL<-c(listGTL, GTL)
}
GTmb<-matrix(0,1,dim(GTmbmatrix)[2]);
for(i in 1: dim(GTmbmatrix)[2]){
temp4<-which.max(GTmbmatrix[,i]);
GTmb[i]<-GTmbmatrix[temp4,1]
}
GTmb[dim(GTmbmatrix)[2]]<-0;
#now I need something to find max in each column of GTmbmatrix
# then whichever row the max is in, use the number from the first column
#GTmbmatrix
#listGTL
GTmb
}


#####################################################################
##now need a bootstrap function
## create B bootstrap samples
## output obm1 - obm2 for each bs sample - matrix
## input:  original data, B no. bs samples, min samp size required
# for splitting, orig pruned tree, alphac=2-4
## output: an adjusted G alpha T m for each pruned original tree
bootstraptree<-function(id,covariates,t,d,B, sampsize, origprunetree,alphac,ss2){
wbmmatrix<-c();
#obm1matrix<-c();
#obm2matrix<-c();
X<-cbind(id,t,d,covariates);
for(i in 1:B){
bsX<-X[sample(dim(X)[1], length(id), replace=T),];
bscov<-bsX[,-1:-3]
bst<-bsX[,2];
bsd<-bsX[,3];
bsbigtree<-growtree(bscov,bst,bsd,sampsize,ss2);
temp2<-prunetree(bsbigtree);
bstree<-prune2(temp2);
OBM1<-obm1(origprunetree,bstree,bscov,bst,bsd,sampsize);
OBM2<-obm2(bsbigtree, bstree, t,d, covariates,origprunetree);
# both OBM's go from big to small.
wbm<-OBM2-OBM1;
wbmmatrix<-rbind(wbmmatrix,wbm);
# obm1matrix<-rbind(obm1matrix,OBM1);
# obm2matrix<-rbind(obm2matrix,OBM2);
i<-i+1
}
wm<-colMeans(wbmmatrix);
GLLm<-c(sort(unique(origprunetree[4,]), decreasing=T),0);
pruneorder<-origprunetree[2,];
inodes<-origprunetree[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
#inodes<-c(subset(inodes,duplicated(pruneorder)==F),0);
Galphahatm<-GLLm+wm-alphac*inodes;
Galphahatm
#OBM1
#bsbigtree
#bsX
#GLLm
#wm
#alphac
#inodes
#obm1matrix
#obm2matrix
}


#######################################
#######################################
```

```
# program in
# input: info to pick final tree (w)
# pruned large tree (z)
#a<-as.matrix(z[,z[2,]>=which.max(w)])
# program term node calculates teh number of terminal nodes

termnode<-function(z){
term<-0;
temp<-term;
for(i in 1:length(z[1,])){
node1<-2*z[1,i]+1;
node2<-node1+1;
if(sum(z[1,]==node1)==0){term<-term+1};
if(sum(z[1,]==node2)==0){term<-term+1}
}
term
}

##############################################
# covused produces a vector indicating if
# each covariate is used in the tree
##############################################

covused<-function(x,z,covariates){
numcov<-length(covariates[1,]);
covlist<-matrix(seq(1:numcov),1,numcov);
temptree<-rbind(z,x);
temp<-matrix(0,1,numcov);
a<-as.matrix(temptree[,temptree[2,]>=which.max(w)])
for(i in 1:length(a[1,])){
temp<-temp+(covlist==a[8,i])
}
output<-(temp>0);
output
}
#covused(x,z,covariates)


############################################################
############################################################
############################################################
# grow tree log rank ############################
############################################################
############################################################
############################################################
## ss is subset of data that will be used to calculate the test statistic
## cuminc has a 'subset' option.
# takes in nodevec, the vector that states which obs is in which current node
maxsplitLR<-function(nodevec,node,x1,t,delta,ss2){
ss<-node==nodevec;
i<-1;
output<-rep(0,2);
test<-c(0,0);
#for(i in 1:length(x1)){
for(i in 1:length(unique(x1))){
if(nodevec[i]==node){
# z<-x1<=x1[i];
temp5<-sort(unique(x1));
temp3<-temp5[i];
temp4<-temp5[i+1];
z<-x1<=temp3;
if((sum(z*ss)>0) & (sum((1-z)*ss)>0)){
temp<-survdiff(Surv(t,delta)~z,subset=ss)[5];
if(is.null(temp)==T|sum(z)<=ss2|sum(1-z)<=ss2){temp=0};
temp<-as.numeric(temp);
output<-as.numeric(output);
if(temp>output[1]){
# temp2<-median(c(x1[i], sort(x1)[which.max(sort(x1)==x1[i])+1]));
temp2<-median(c(temp3, temp4));
output<-rbind(temp,temp2)
}
}
}
}
output
}


#####################################################################
# maxsplit2
```

```
# input: covariates (cbind of x vectors), time=t, delta= event indicator , node, nodevec is the vector
# that states which obs in in which node
# output:  the best split over all covariates,
# statistic, cutpoint, variable (1,2,3.. column of the covariates matrix)
# and the node 0,1,2,3....
maxsplit2LR<-function(nodevec,covariates,t,delta, node,ss2){
temp<-c();
temp2<-c();
output<-rep(0,4);
for(j in 1:dim(covariates)[2]){
temp<-maxsplitLR(nodevec,node, covariates[,j],t,delta,ss2);
#temp2<-c(temp[1],temp[2],j,node);
#temp3<-cbind(output, temp2);
if(temp[1]>output[1]){output<-c(temp[1],temp[2],j,node)}
}
#temp3
output
}
#maxsplit2LR(nd,covariates,t,(delta==1)^2,0,ss2)
#######################################################################
# grow tree
# input:  covariates, t, delta
# output:  list of splits (stat, cutoff, var, node)
# and a vector called 'node' will say which node each obs is in.
# sampsize is the min number of obs in each node
growtreeLR<-function(covariates, t, delta, sampsize,ss2){
node<-rep(0,length(delta));
nodelist<-c(0);
output<-c();
i<-1;
while(i <=length(nodelist)){
# while(i <=2){
#check that there are event 1,2
# call is ss test (sample size test)
sstest1<-sum((node==nodelist[i])*(delta==1));
# sstest2<-sum((node==nodelist[i])*(delta==2));
if(sstest1>0&sum(node==nodelist[i])>=sampsize){
temp<-maxsplit2LR(node,covariates,t,delta,nodelist[i],ss2);
if(temp[1]>0){
nodelist<-c(nodelist,2*nodelist[i]+1,2*nodelist[i]+2);
node<-(node==nodelist[i])*(covariates[,temp[3]]<=temp[2])*(2*nodelist[i]+1)+
(node==nodelist[i])*(covariates[,temp[3]]>temp[2])*(2*nodelist[i]+2)+node*(node!=nodelist[i]);
output<-cbind(output,temp)
}
}
#output<- node;
i<-i+1
}
#temp
output
#nodelist
#node
}


#####################################################################
## pruning the tree

# need to calculate G(T) and # internal nodes beneath current node
#+1
# Take output from 'growtree'
# Start at end of list
# Basic idea:  does current node have children on list?
# if yes then find children nodes and add their G(T) and # nodes
# if no then G(T)=stat, # nodes=1
# takes in growtree object
# output growtree object with 2 rows : one with G(T) and # int nodes
prunetreeLR<-function(obj){
# n is number of internal nodes in list
n<-dim(obj)[2];
# combines the split statistic, row of 1s (will be no. int nodes), and node number
output<-t(cbind(obj[1,],matrix(1,n), obj[4,]));
for(i in 1:n){
j<-n-i+1;
currentnode<-obj[4,j];
allnodes<-obj[4,];
if(sum(allnodes==(2*currentnode+1)|allnodes==(2*currentnode+2))>0){
for(k in 1:(n-j)){
l<-n-k+1;
# if the current node j is parent of node l, then add stats from node l
```

```
if(obj[4,l]==(2*currentnode+1)|obj[4,l]==(2*currentnode+2)){
output[1,j]<-output[1,j]+output[1,l];
output[2,j]<-output[2,j]+output[2,l];
}
}
}
# if there are daughter nodes on list then need to find them and add G(T)
# and no. nodes to intnode
}
output
}
#y<-prunetreeLR(x)
#y
# We calculate G(T)/number of internal nodes = cost complexity
# Find int node with smallest cost complexity
# prune this node and all child nodes (remove from the tree list)
# store this new pruned tree somewhere
# then use 'pruntree' to prune this new tree object.

#########################################################################
# descendant will
#input:  node to prune
# output: vector stating all child nodes that will be pruned (pruned=T, not pruned=F)
descendantLR<-function(pruneobj, node){
pruned<-pruneobj[3,]==node;
for(i in 1:dim(pruneobj)[2]){
if(pruned[i]==T){
pruned<-pruned==T|pruneobj[3,]==(2*pruneobj[3,i]+1)|pruneobj[3,]==(2*pruneobj[3,i]+2)
}
}
pruned
#pruneobj
}

#descendantLR(y,2)

#########################################################################
# ancestor indicates which nodes are descendants of current node
# input: node, and tree obj
# output:  vector T/F ancestor of current node
ancestorLR<-function(pruneobj, node){
currentnode<-node;
parent<-pruneobj[3,]<0;
n<-dim(pruneobj)[2];
for(i in 1:n){
j<-n-i+1;
#if node  is odd, and currentnode is parent, and in the list
if(floor(currentnode/2)!=currentnode/2){
if(((currentnode-1)/2)==pruneobj[3,j]){
currentnode<-pruneobj[3,j];
parent[j]<-T
}
}
# if the node is even
else{
if(((currentnode-2)/2)==pruneobj[3,j]){
currentnode<-pruneobj[3,j];
parent[j]<-T
}
}
}
parent
}
#ancestorLR(y,5)
#############################################################################
# program to actually prune
# takes pruneobj with intnodes, G(t),node
# outputs same object with:
# prune order, and min alpha's

# tobeprune is T if node not yet pruned, F is node is pruned, updates during algorithm
# pruneorder records the step that a node was pruned on
# alphalist records the alpha used to prune the node
# alpha is list of alphas at each prune step, updates during algorithm
# currentnode records last node to be pruned, updates during algorithm

prune2LR<-function(pruneobj){
output<-c();
n<-dim(pruneobj)[2];
temp<-t(matrix(1,n));
```

```
tobeprune<-pruneobj[3,]>=0;
GTlist<-pruneobj[1,];
maxGT<-matrix(0,1,n);
# maxGT records the G(T) of each pruned subtree
intnodelist<-pruneobj[2,];
# no of internal nodes in each pruned subtree
intnode<-matrix(0,1,n);
pruneorder<-matrix(0,1,n);
alphalist<-matrix(0,1,n);
alpha<-GTlist/intnodelist;
minalpha<-min(alpha/tobeprune);
currentnode<-0;
inodes<-pruneobj[3,];
j<-0;
while(sum(tobeprune)!=0){
j<-j+1;
stepnum<-j*temp;
firstmin<-T;
minalphavec<-minalpha*temp;
for(i in 1:n){
alpha<-GTlist/intnodelist;
minalpha<-min(alpha/tobeprune)
# firstmin, just records the first time got into if statement , in case of ties
if((alpha[i]/tobeprune[i])==minalpha&firstmin==T){
# if(alpha[i]==minalpha&firstmin==T){
# firstmin, makes sure only use one alpha, in case of ties
firstmin<-F;
currentnode<-pruneobj[3,i];
temp1<-descendantLR(pruneobj, currentnode);
# tobeprune<-tobeprune&(!temp1);
maxGT<-maxGT+temp1*max(GTlist*tobeprune)*(maxGT==0);
intnode<-intnode+temp1*max(intnodelist*tobeprune)*(intnode==0);
tobeprune<-tobeprune&(!temp1);
pruneorder<-pruneorder+j*temp1*(pruneorder==0);
alphalist<-alphalist+minalpha*temp1*(alphalist==0);
temp2<-ancestorLR(pruneobj,currentnode);
GTlist<-GTlist-temp2*GTlist[i];
intnodelist<-intnodelist-temp2*intnodelist[i];
}
}
cnode<-currentnode*temp;
# output<-rbind(output,stepnum,alpha,pruneorder,alphalist, cnode)
# output<-rbind(output,GTlist,maxGT);
}
tobeprune
j
output<-rbind(inodes,pruneorder, alphalist, maxGT, intnode);
output
}
#z<-prune2LR(y)
#z
#Outputs the 1) node number of large tree To
# 2) step that the branch with the node as the root was pruned
# 3) alpha used to prune that branch
# 4) max GT on that subtree pruned at step
# 5) number of internal nodes in that sub tree at step



#################################################################
# Tree Selection
# first need to create function 'prune3' that takes a large Tree and a
# particular alpha, and outputs the tree pruned to that alpha
# Input:  alpha, tree info
# Output: G(T) of the pruned tree (not G_alpha_T)

# function, maxsplitcomp
# this takes list of pruned sub trees and an alpha
# and calculates max G alpha T over the trees.
# this is the GT of the tree pruned with alpha

prune3LR<-function(alpha,subtrees){
GalphaT<-subtrees[4,]-alpha*subtrees[5,];
temp<-max(GalphaT);
for(i in 1:dim(subtrees)[2]){
if(GalphaT[i]==temp){output<-subtrees[4,i]}
}
output
}
#prune3LR(2,z)
```

```
##############################################
#Function to calculate bootstrap adjustment
# need to create a sub function that 'runs' data
# down  given tree rules, to get GT
# G(L1;L2 T)
#mydata[sample(dim(mydata)[1], 10),] sample 10 rows

# takes 'treedata' from the growtree function

rundowntreeLR<-function(treedata,newcov, newt, newd){
n<-dim(treedata)[2];
node<-0;
newstat<-matrix(0,1,n);
nodelist<-matrix(0,1,dim(newcov)[1]);
newd<-(newd==1)^2;
for(i in 1:n){
# update current node (scalar)
node<-treedata[4,i];
# update the variable that we will split on (goes with node)
currvar<-treedata[3,i];
# temp is the binary split on the variable and cutoff
temp<-newcov[,currvar]<=treedata[2,i];
# ss indicates the subset of data corresponding to current node
ss<-nodelist==node;
# temp2 is the current split statistic
# must change to 0 if it is -1 or null
# temp2<-cuminc(newt,newd,temp,subset=ss,rho=-1)$Test[1];
temp2<-0;
if(sum(temp*ss)>0&sum((1-temp)*ss)>0){
temp2<-survdiff(Surv(newt,newd)~temp,subset=ss)$chisq[1]}
if((is.null(temp2)==T)){temp2<-0};
if(temp2<0){temp2<-0};
newstat[i]<-temp2;
# nodelist tells us which data point is in which node (vector)
nodelist<-(nodelist==node)*(temp)*(2*node+1)+
(nodelist==node)*(!temp)*(2*node+2)+nodelist*(nodelist!=node)
}
newstat
}
#rundowntreeLR(x,covariates,t,delta2)
#########################################################################

# omb1 calculates the first part of obm
# G(Lb;Lb,Tmb) using alphadash=sqrt(alpham*alpham+1)
# input:  pruned list from large tree, the bootstrap sample
# ovariates, time, delta, and samplesize for splitting
# output:  a list of G(T), one for each alphamdash,
# chosen from list of bootstrap pruned trees, as max Galphadash_m(T)
################################
#######################################
#######################################


obm1LR<-function( origprunedtree, bsprunedtree,bcov,bt,bd, sampsize){
# generate the alpha-dash_m's
temp1<-sort(unique(origprunedtree[3,]))
temp2<-c(temp1,Inf)
temp3<-c(0,temp1)
alphadash<-sqrt(temp2*temp3)
# grow large tree using bootstrap sample
#Tb0<-growtree(bcov, bt, bd, sampsize)
# prune bs tree
#pruneTb0<-prune2(prunetree(Tb0))
GTbm<-matrix(0,1,length(alphadash))
# for every alphadash m, find the G(T)
for(i in 1:length(alphadash)){
GTbm[i]<-prune3(alphadash[i],bsprunedtree)
}
# now find G(original data; bootstrap data, Tbm)
#GTbmorig
# need a function that takes a Large Tree and pruning order
# and can caluclate GT for each pruned subtree
GTbm[length(alphadash)]<-0;
GTbm
#alphadash
}

#################################################################
# need a function where, input: tree data, and prune step
# output: tree data with every node prunestep or greater removed
```

```
# treedata2 is the rbind of orign tree data adn pruned data
removepruneLR<-function(treedata2, prunestep){
treedata3<-c()
for(i in 1:dim(treedata2)[2]){
if(treedata2[6,i]>=prunestep){treedata3<-cbind(treedata3,treedata2[,i])}
}
treedata3
}

#####################################################################
# obm2 calculates the other part of obm
# G(L;Lb,Tm)
# input:
# large bootstrap tree, pruned bs tree, original data (time, delta, cov)
# and orignal large pruned tree (to calc alphadashm's)

obm2LR<-function(bstree, bsprune, ot, od, ocov,prunedtree){
# generate the alpha-dash_m's
temp1<-sort(unique(prunedtree[3,]))
temp2<-c(temp1,Inf)
temp3<-c(0,temp1)
alphadash<-sqrt(temp2*temp3)
btreedata<-rbind(bstree,bsprune);
GTmbmatrix<-c();
#listGTL<-c();
for(i in 1:max(btreedata[6,])){
temptree<-removepruneLR(btreedata,i);
GTL<-sum(rundowntreeLR(temptree,ocov,ot,od));
# need to find vector of G_alphadash(T)
Galphadash<-GTL-alphadash*(max(temptree[9,]));
Galphadash[length(Galphadash)]<-0;
GTmbmatrix<-rbind(GTmbmatrix,Galphadash);
# listGTL<-c(listGTL, GTL)
}
GTmb<-matrix(0,1,dim(GTmbmatrix)[2]);
for(i in 1: dim(GTmbmatrix)[2]){
temp4<-which.max(GTmbmatrix[,i]);
GTmb[i]<-GTmbmatrix[temp4,1]
}
GTmb[dim(GTmbmatrix)[2]]<-0;
#now I need something to find max in each column of GTmbmatrix
# then whichever row the max is in, use the number from the first column
#GTmbmatrix
#listGTL
GTmb
}

test<-obm2(xMult,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)

#####################################################################
##now need a bootstrap function
## create B bootstrap samples
## output obm1 - obm2 for each bs sample - matrix
## input:  original data, B no. bs samples, min samp size required
# for splitting, orig pruned tree, alphac=2-4
## output: an adjusted G alpha T m for each pruned original tree
bootstraptreeLR<-function(id,covariates,t,d,B, sampsize, origprunetree,alphac,ss2){
wbmmatrix<-c();
#obm1matrix<-c();
#obm2matrix<-c();
X<-cbind(id,t,d,covariates);
for(i in 1:B){
bsX<-X[sample(dim(X)[1], length(id), replace=T),];
bscov<-bsX[,-1:-3]
bst<-bsX[,2];
bsd<-bsX[,3];
bsbigtree<-growtreeLR(bscov,bst,bsd,sampsize,ss2);
temp2<-prunetreeLR(bsbigtree);
bstree<-prune2LR(temp2);
OBM1<-obm1LR(origprunetree,bstree,bscov,bst,bsd,sampsize);
OBM2<-obm2LR(bsbigtree, bstree, t,d, covariates,origprunetree);
# both OBM's go from big to small.
wbm<-OBM2-OBM1;
wbmmatrix<-rbind(wbmmatrix,wbm);
# obm1matrix<-rbind(obm1matrix,OBM1);
# obm2matrix<-rbind(obm2matrix,OBM2);
i<-i+1
}
wm<-colMeans(wbmmatrix);
GLLm<-c(sort(unique(origprunetree[4,]), decreasing=T),0);
```

```
pruneorder<-origprunetree[2,];
inodes<-origprunetree[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
#inodes<-c(subset(inodes,duplicated(pruneorder)==F),0);
Galphahatm<-GLLm+wm-alphac*inodes;
Galphahatm
#GLLm
#wm
#alphac
#inodes
#obm1matrix
#obm2matrix
}




#######################################
#######################################
# program in
# input: info to pick final tree (w)
# pruned large tree (z)
#a<-as.matrix(z[,z[2,]>=which.max(w)])
# program term node calculates teh number of terminal nodes

termnode<-function(z){
term<-0;
temp<-term;
len<-length(z[1,]);
if(len==0){term<-0}
else{
for(i in 1:len){
node1<-2*z[1,i]+1;
node2<-node1+1;
if(sum(z[1,]==node1)==0){term<-term+1};
if(sum(z[1,]==node2)==0){term<-term+1}
}
}
term
}

############################################
# covused produces a vector indicating if
# each covariate is used in the tree
############################################

covused<-function(x,z,covariates,a){
numcov<-length(covariates[1,]);
covlist<-seq(1:numcov);
temptree<-rbind(z,x);
temp<-matrix(0,1,numcov);
#a<-as.matrix(temptree[,temptree[2,]>=which.max(w)])
if(length(a[1,])==0){output<-0}
else{
for(i in 1:length(a[1,])){
temp<-temp+(covlist==a[8,i])
}
}
output<-(temp>0);
output
}
#covused(x,z,covariates,a)

#########################################################
#########################################################
#########################################################
#########################################################
# Mart resid tree
#########################################################
#########################################################
#########################################################
#########################################################



covusedMR<-function(tempMR, covariates){
temp<--matrix(0,1,length(covariates[1,]));
 temp[1]<-sum(tempMR$fram$var=="covariates.Z1")>0;
```

```
 temp[2]<-sum(tempMR$fram$var=="covariates.Z2")>0;
 temp[3]<-sum(tempMR$fram$var=="covariates.Z3")>0;
 temp[4]<-sum(tempMR$fram$var=="covariates.Z4")>0;
 temp[5]<-sum(tempMR$fram$var=="covariates.Z5")>0;
 temp[6]<-sum(tempMR$fram$var=="covariates.Z6")>0;
temp
}
#covusedMR(tempMR,covariates)
#covused(x,z,covariates)


##############################################
##############################################
# Generating data
############################################################
# rpart getting MAE
############################################################


##############################################
##############################################

maeMR<-function(tempMR,X2,X, told, tnew){
del1<-X$dsim
del2<-X2$dsim2;
del<-X2$dsim2;
# calculates the mean prediction for each terminal node,
# we use this as labels for terminal nodes.
pred2<-predict(tempMR,newdata=X2, type="matrix");
# tsim<-X$tsim;
# this calculates the medians per terminal node
pred1<-predict(tempMR)
upred2<-sort(unique(pred2))
upred1<-sort(unique(pred1))
temp<-by(told[del1==1],factor(pred1[del1==1]),function(x) median(x));
temp1<-temp[!is.na(match(upred1,upred2))]
# temp<-by(tnew,factor(pred2),function(x) median(x));
# this takes the new predictions for the new data, and uses
# the labels of the medians.
temp2<-as.numeric(as.vector(factor(pred2, labels=temp1)));
# tsim2<-X2$tsim2;
maeMR1<-mean(abs(tnew[del2==1]-temp2[del2==1]));
maeMR1
}

##############################################
##############################################
# mae for LR and Gray
##############################################
##############################################
####Want:
####Input: x (growtree), y and z,w, and ax (use w to pick tree x) and az.+orig data
## assigns final node to old data set
## calcuates median for each final node for old data
## assigns final node to new data set
## assigns median for each node for new data
## calculates MAE
## fun1= take tree data (x z) and data , and assign final nodes.


#####################################################################
#####################################################################
# function to produce list assigning each observation to a final node
#####################################################################
#####################################################################
# takes in a = a tree chosen with w, with z and x attached
# X= data to add the nodel list to.
# output: a vector length X, that lists terminal nodes.
#####################################################################
# nodelist is the actual list of nodes (short)
# node will be the same length as tsim, with which node each
# observation is  in.
# nodelist<-c(nodelist,2*nodelist[i]+1,2*nodelist[i]+2);

termnodelist<-function(a,X){
nodelist<-a[1,];
covariates<-X[,4:9];
node<-rep(0,length(X[,1]));
for(i in 1:length(nodelist)){
currentnode<-nodelist[i];
```

```
node<-node*(node!=currentnode)+(covariates[,a[8,i]]<=a[7,i])*(node==currentnode)*(2*nodelist[i]+1)+
(covariates[,a[8,i]]>a[7,i])*(node==currentnode)*(2*nodelist[i]+2);
}
node
}
#termnodelist(a,X)
#########################################################3
###########################################################


###############################################
################################################
# median node function
# see above for explantion
# input: a tree, old data, new data
# output: for each observation of new data, it assigns
# the corresponding prediction (median time) from original tree.
###############################################
###############################################
mediannode<-function(a,X, X2,told){
tnode<-termnodelist(a,X);
del1<-X$dsim;
del2<-X2$dsim2;
# tsim<-X$tsim;
tnode2<-termnodelist(a,X2);
temp<-by(told[del1==1],factor(tnode[del1==1]),function(x) median(x));
tnodeu<-sort(unique(tnode))
tnode2u<-sort(unique(tnode2))
temp3<-sort(unique(tnode[del1==1]))
temp2<-rbind(temp3,temp)
test<-c();
for(i in 1: length(temp3)){
if(sum(temp3[i]==tnode2u)==0){
test<-c(test,temp3[i])
}
}
if(is.null(test)==F){
for(i in 1:length(test)){
temp2<-temp2[,temp2[1,]!=test[i]]
}
}
test<-c();
for(i in 1: length(tnode2u)){
if(sum(tnode2u[i]==temp3)==0){
test<-c(test,tnode2u[i])
}
}
if(is.null(test)==F){
for(i in 1:length(test)){
temp2<-cbind(temp2,c(test[i],0))
}
}
o<-order(temp2[1,])
#temp2<-temp2[1,]
#temp<-temp2[2,];
temp<-temp2[2,][o];
meds<-factor(tnode2, labels=temp) ;
meds
}

#########################################################
#########################################################
# calculate MAE for a new set of data
#########################################################
# input: a= tree info, X=old data X2=new data
# output:  the MAE for new data
#########################################################
maeGRLR<-function(a,X,X2,told,tnew){
del2<-X2$dsim2;
del1<-X$dsim;
if(length(a)==0){mae<-mean(abs(tnew[del1==1]-median(told[del1==1])))}
else{
# tnode<-termnodelist(a,X);
# mednode<-as.numeric(mediannode(a,X,X2,told));
mednode<-as.numeric(as.vector(mediannode(a,X,X2,told)));
# tsim2<-X2$tsim;
mae<-mean(abs(tnew[del2==1]-mednode[del2==1]))
}
mae
}
```

```
#maeGRLR(a,X,X2,X$tsim,X2$tsim2)



#############################################################################
# cif test with the optin of subsetting
#############################################################################
ciftestss<-function(tsim, dsim,Z,cutpoint,r,subgroup){
temptsim<-tsim
tempdsim<-dsim
tempZ<-Z
tsim<-subset(tsim, subgroup)
dsim<-subset(dsim, subgroup)
Z<-subset(Z, subgroup)
group<-Z<=cutpoint;
cif<-cuminc(ftime=tsim,fstatus=dsim,group=group);
ciftotal<-cuminc(ftime=tsim,fstatus=dsim);
n1<-sum(group==F);
n2<-sum(group==T);
## Event 1 group1
test11<-survfit(Surv(tsim,(dsim==1)^2), subset=(group==F));
Y11<-test11$n.risk;
E11<-test11$n.event;
W11<-(1-timepoints(ciftotal, tsim*(group==F))$est[1,])^r
if(length(E11)<length(W11)){
E11<-c(0,E11);
Y11<-c(Y11[1],Y11)
}
#W11<-W11[-1]
I11<-timepoints(cif, tsim*(group==F))$est[1,]
dI11<-I11
#new
dI11[1]<-0
for(i in 2:length(I11)){
dI11[i]<-I11[i]-I11[i-1]
}
#I11<-I11[-1]
#dI11<-dI11[-1]

for(i in 1:length(I11)){if(is.na(I11[i]))I11[i]<-0}
for(i in 1:length(dI11)){if(is.na(dI11[i]))dI11[i]<-0}
B11<-W11*dI11*(E11>0)^2
B11c<-cumsum(B11)
B11T<-sum(B11)
#A11
B11Tvec<-B11T*rep(1, length(B11))
A111<-(B11Tvec-B11c)/(Y11/n1)-addon(B11,E11)*n1
A112<-(B11Tvec-B11c)/(Y11/n1)
## A121 and A122
test12<-survfit(Surv(tsim,(dsim==1)^2), subset=(group==T))
Y12<-test12$n.risk
E12<-test12$n.event
W12<-(1-timepoints(ciftotal, tsim*(group==T))$est[1,])^r
if(length(E12)<length(W12)){
E12<-c(0,E12);
Y12<-c(Y12[1],Y12)
}
#W12<-W12[-1]

I12<-timepoints(cif, tsim*(group==T))$est[2,]
dI12<-I12
dI12[1]<-0
for(i in 2:length(I12)){
dI12[i]<-I12[i]-I12[i-1]
}
#I12<-I12[-1]
#dI12<-dI12[-1]
for(i in 1:length(I12)){if(is.na(I12[i]))I12[i]<-0}
for(i in 1:length(dI12)){if(is.na(dI12[i]))dI12[i]<-0}

B12<-W12*dI12*(E12>0)^2
B12c<-cumsum(B12)
B12T<-sum(B12)
B12Tvec<-B12T*rep(1, length(B12))
A121<-(B12Tvec-B12c)/(Y12/n2)-addon(B12,E12)*n2
A122<-(B12Tvec-B12c)/(Y12/n2)
############### A21: event 2 group 1, Ajml = event j, group m, l

test21<-survfit(Surv(tsim,(dsim==2)^2), subset=(group==F))
Y21<-test21$n.risk
```

```
E21<-test21$n.event
W21<-(1-timepoints(ciftotal, tsim*(group==F))$est[2,])^r
if(length(E21)<length(W21)){
E21<-c(0,E21);
Y21<-c(Y21[1],Y21)
}
#W21<-W21[-1]
I21<-timepoints(cif, tsim*(group==F))$est[3,]
dI21<-I21
dI21[1]<-0
for(i in 2:length(I21)){
dI21[i]<-I21[i]-I21[i-1]
}
#I21<-I21[-1]
#dI21<-dI21[-1]
for(i in 1:length(I21)){if(is.na(I21[i]))I21[i]<-0}
for(i in 1:length(dI21)){if(is.na(dI21[i]))dI21[i]<-0}

B21<-W21*dI21*(E21>0)^2

B21c<-cumsum(B21)
B21T<-sum(B21)
B21Tvec<-B21T*rep(1, length(B21))
A212<-(B21Tvec-B21c)/(Y21/n1)-addon(B21,E21)*n1
A211<-(B21Tvec-B21c)/(Y21/n1)
############### A22: event 2 group 1, Ajml = event j, group m, l

test22<-survfit(Surv(tsim,(dsim==2)^2), subset=(group==T))
Y22<-test22$n.risk
E22<-test22$n.event
W22<-(1-timepoints(ciftotal, tsim*(group==T))$est[2,])^r
if(length(E22)<length(W22)){
E22<-c(0,E22)
Y22<-c(Y22[1],Y22)
}
#W22<-W22[-1]
I22<-timepoints(cif, tsim*(group==T))$est[4,]
dI22<-I22
dI22[1]<-0
for(i in 2:length(I22)){
dI22[i]<-I22[i]-I22[i-1]
}
#I22<-I22[-1]
#dI22<-dI22[-1]
for(i in 1:length(I22)){if(is.na(I22[i]))I22[i]<-0}
for(i in 1:length(dI22)){if(is.na(dI22[i]))dI22[i]<-0}
B22<-W22*dI22*(E22>0)^2
B22c<-cumsum(B22)
B22T<-sum(B22)
B22Tvec<-B22T*rep(1, length(B22))
A222<-(B22Tvec-B22c)/(Y22/n2)-addon(B22,E22)*n2
A221<-(B22Tvec-B22c)/(Y22/n2)

sigma1<-(n1*n2/(n1+n2))*(sum((A111^2/n1)*E11/n1)+
sum((A121^2/n2)*E12/n2)+sum((A112^2/n1)*E21/n1)+
sum((A122^2/n2)*E22/n2))

sigma2<-(n1*n2/(n1+n2))*(sum((A211^2/n1)*E21/n1)+
sum((A221^2/n2)*E22/n2)+sum((A212^2/n1)*E21/n1)+
sum((A222^2/n2)*E22/n2))

##### Covariance
#l=1
covar<-(n1*n2/(n1+n2))*(sum(A111*A211*E11/n1^2)+
sum(A121*A221*E12/n2^2)+
sum(A112*A212*E21/n1^2)+
sum(A122*A222*E22/n2^2))
covmat<-matrix(c(sigma1, covar, covar, sigma2), nrow=2)
teststat<-matrix(multest(tsim, dsim, group, r), nrow=2)
HoetT<-t(teststat)%*%solve(covmat)%*%teststat
#tsim<-temptsim
#dsim<-tempdsim
#Z<-tempZ
HoetT
}

#multest(tsim, dsim, Z1<=0.5, -1)
#ciftest(tsim, dsim, Z1,0.5, r)
#subgroup<-Z1>=min(Z1)
#ciftestss(tsim, dsim, Z1,0.5, r, subgroup)
```

```
#ciftestss(nhxdays, d, age_ev,40, r, subgroup)
#subgroup<-Z1<=0.7
#ciftestss(tsim, dsim, Z1,0.5, r, subgroup)

################################################################
## ss is subset of data that will be used to calculate the test statistic
## cuminc has a 'subset' option.
# takes in nodevec, the vector that states which obs is in which current node
#maxsplit(nd,0,x1,t,delta,ss2)
#ciftest(tsim, dsim, Z1,0.5, r)

maxsplitMult<-function(nodevec,node,x1,t,delta,ss2,r){
ss<-node==nodevec;
i<-1;
output<-rep(0,2);
#for(i in 1:length(x1)){
for(i in 1:length(unique(x1*ss))){
if(nodevec[i]==node){
temp5<-sort(unique(x1*ss));
temp3<-temp5[i];
temp4<-temp5[i+1];
z<-x1<=temp3;
# temp<-cuminc(t,delta,z,subset=ss, rho=-1)$Test[1];
# temp<-ciftest(t,delta,x1,z,r);
temp<-0;
# if(sum(z)>ss2&sum(1-z)>ss2){temp<-ciftestss(tsim,dsim,Z1,temp3,r,ss)};
if(sum(z*ss)>ss2&sum((1-z)*ss)>ss2&sum(ss*(delta==1)*z)>0&sum(ss*(delta==1)*(1-z))>0&sum(ss*(delta==2)*z)>0&sum(ss*(delta==2)*(1-z))>0){
temp<-ciftestss(t,delta,x1,temp3,r,ss)
}
if(is.na(temp)==T|sum(z)<=ss2|sum(1-z)<=ss2){temp<-0};
if(temp>output[1]){
# take the median of the cutpoint and the next highest value
temp2<-median(c(temp3, temp4));
output<-c(temp,temp2)
}
}
}
output
}
#maxsplitMult<-function(nodevec,node,x1,t,delta,ss2,r){

#maxsplitMult(node,0,Z1,tsim,dsim,20,-1)
#maxsplitMult(nodevec,0,age_ev,nhxdays,d,20,-1)

#####################################################################
# next thing to do is maxsplit2, growtree etc. all as Mult.
#####################################################################
# maxsplit2Mult
# input: covariates (cbind of x vectors), time=t, delta= event indicator , node, nodevec is the vector
# that states which obs in in which node
# output:  the best split over all covariates,
# statistic, cutpoint, variable (1,2,3.. column of the covariates matrix)
# and the node 0,1,2,3....
maxsplit2Mult<-function(nodevec,covariates,t,delta, node,ss2,r){
temp<-c();
temp2<-c();
output<-rep(0,4);
for(j in 1:dim(covariates)[2]){
temp<-maxsplitMult(nodevec,node, covariates[,j],t,delta,ss2,r);
#temp2<-c(temp[1],temp[2],j,node);
#temp3<-cbind(output, temp2);
if(temp[1]>output[1]){output<-c(temp[1],temp[2],j,node)}
}
#temp3
output
}
#maxsplit2Mult(nd,covariates,tsim,dsim,0,ss2,r)


#########################################################
#subgroup<-Z1>=min(Z1)
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
#  maxsplitMult actually works.
#ciftestss(tsim,dsim,Z1,0.5,r,subgroup)
#ciftestss(tsim200,dsim200,cov200[,1],0.3,r,cov200[,1]>=.1)
#subgroup<-Z1>=0.2
#ciftestss(tsim,dsim,Z1,0.5,r,subgroup)
#####################################################################
#####################################################################
```

```
#####################################################################
# growtreeMult
# input:  covariates, t, delta
# output:  list of splits (stat, cutoff, var, node)
# and a vector called 'node' will say which node each obs is in.
# sampsize is the min number of obs in each node
growtreeMult<-function(covariates, tim, delta, sampsize,ss2,r){
node<-rep(0,length(delta));
nodelist<-c(0);
output<-c();
i<-1;
while(i <=length(nodelist)){
#check that there are event 1,2
# call is ss test (sample size test)
sstest1<-sum((node==nodelist[i])*(delta==1));
sstest2<-sum((node==nodelist[i])*(delta==2));
if(sstest1>1&sstest2>1&sum(node==nodelist[i])>=sampsize){
temp<-maxsplit2Mult(node,covariates,tim,delta,nodelist[i],ss2,r);
if(temp[1]>0){
nodelist<-c(nodelist,2*nodelist[i]+1,2*nodelist[i]+2);
node<-(node==nodelist[i])*(covariates[,temp[3]]<=temp[2])*(2*nodelist[i]+1)+
(node==nodelist[i])*(covariates[,temp[3]]>temp[2])*(2*nodelist[i]+2)+node*(node!=nodelist[i]);
output<-cbind(output,temp)
}
}
#output<- nodelist;
i<-i+1
}
#temp
output
#nodelist
#node
}

#sampsize<-10
#xMult<-growtreeMult(covariates,tsim,dsim,sampsize,ss2,r)
#xMult
# grows pretty quickly , 1 minute??
#################################################################
# need to do pruneMult etc.
#####################################################################
## pruning the tree

# need to calculate G(T) and # internal nodes beneath current node
#+1
# Take output from 'growtree'
# Start at end of list
# Basic idea:  does current node have children on list?
# if yes then find children nodes and add their G(T) and # nodes
# if no then G(T)=stat, # nodes=1
# takes in growtree object
# output growtree object with 2 rows : one with G(T) and # int nodes
prunetreeMult<-function(obj){
# n is number of internal nodes in list
n<-dim(obj)[2];
# combines the split statistic, row of 1s (will be no. int nodes), and node number
output<-t(cbind(obj[1,],matrix(1,n), obj[4,]));
for(i in 1:n){
j<-n-i+1;
currentnode<-obj[4,j];
allnodes<-obj[4,];
if(sum(allnodes==(2*currentnode+1)|allnodes==(2*currentnode+2))>0){
for(k in 1:(n-j)){
l<-n-k+1;
# if the current node j is parent of node l, then add stats from node l
if(obj[4,l]==(2*currentnode+1)|obj[4,l]==(2*currentnode+2)){
output[1,j]<-output[1,j]+output[1,l];
output[2,j]<-output[2,j]+output[2,l];
}
}
}
# if there are daughter nodes on list then need to find them and add G(T)
# and no. nodes to intnode
}
output
}


#yMult<-prunetreeMult(xMult)
#yMult
```

```
#zMult<-prune2(yMult)
#zMult
# We calculate G(T)/number of internal nodes = cost complexity
# Find int node with smallest cost complexity
# prune this node and all child nodes (remove from the tree list)
# store this new pruned tree somewhere
# then use 'pruntree' to prune this new tree object.
#########################################################################
#########################################################################
# GTL<-sum(rundowntreeMult(temptree,ocov,ot,od));
rundowntreeMult(temptree,ocov,ot,od)

rundowntreeMult<-function(treedata,newcov, newt, newd){
n<-dim(treedata)[2];
node<-0;
newstat<-matrix(0,1,n);
nodelist<-matrix(0,1,dim(newcov)[1]);
#newd<-(newd==1)^2;
for(i in 1:n){
# update current node (scalar)
node<-treedata[4,i];
# update the variable that we will split on (goes with node)
currvar<-treedata[3,i];
# temp is the binary split on the variable and cutoff
temp<-newcov[,currvar]<=treedata[2,i];
cutoff<-treedata[2,i];
# ss indicates the subset of data corresponding to current node
ss<-nodelist==node;
# temp2 is the current split statistic
# must change to 0 if it is -1 or null
# temp2<-cuminc(newt,newd,temp,subset=ss,rho=-1)$Test[1];
temp2<-0;
if(sum(temp*ss)>0&sum((1-temp)*ss)>0&sum(ss*(newd==1))>1&sum(ss*(newd==2))>1){
# temp2<-survdiff(Surv(newt,newd)~temp,subset=ss)$chisq[1]}
temp2<-ciftestss(newt,newd,newcov[,currvar],cutoff,r,ss)
}
if((is.null(temp2)==T)|is.na(temp2)==T){temp2<-0};
if(temp2<0){temp2<-0};
newstat[i]<-temp2;
# nodelist tells us which data point is in which node (vector)
nodelist<-(nodelist==node)*(temp)*(2*node+1)+
(nodelist==node)*(!temp)*(2*node+2)+nodelist*(nodelist!=node)
}
newstat
}
#rundowntreeLR(x,covariates,t,delta2)

#ciftestss(tsim,tsim,Z1,0.5,r,ss)
#rundowntreeMult(x,covariates,t,delta2)
#########################################################################
#omb2Mult
###############these things need to be changed
#GV<-obm2Mult(xcvMult,zcvMult,nhxdays[cvlist==v], d[cvlist==v],covariates[cvlist==v,],zMultliver)

obm2Mult<-function(bstree, bsprune, ot, od, ocov,prunedtree){
# generate the alpha-dash_m's
temp1<-sort(unique(prunedtree[3,]))
temp2<-c(temp1,Inf)
temp3<-c(0,temp1)
alphadash<-sqrt(temp2*temp3)
btreedata<-rbind(bstree,bsprune);
GTmbmatrix<-c();
#listGTL<-c();
for(i in 1:max(btreedata[6,])){
############## temptree<-removepruneLR(btreedata,i);
temptree<-removeprune(btreedata,i);
############## GTL<-sum(rundowntreeLR(temptree,ocov,ot,od));
GTL<-sum(rundowntreeMult(temptree,ocov,ot,od));
# need to find vector of G_alphadash(T)
Galphadash<-GTL-alphadash*(max(temptree[9,]));
Galphadash[length(Galphadash)]<-0;
GTmbmatrix<-rbind(GTmbmatrix,Galphadash);
# listGTL<-c(listGTL, GTL)
}
GTmb<-matrix(0,1,dim(GTmbmatrix)[2]);
for(i in 1: dim(GTmbmatrix)[2]){
temp4<-which.max(GTmbmatrix[,i]);
GTmb[i]<-GTmbmatrix[temp4,1]
}
GTmb[dim(GTmbmatrix)[2]]<-0;
```

```
#now I need something to find max in each column of GTmbmatrix
# then whichever row the max is in, use the number from the first column
#GTmbmatrix
#listGTL
GTmb
}

#test<-obm2Mult(xMult,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
#####################################################################
#Everything works for Mult so far -- rundowntree seems to work unalteredd,
# but I need to check that out.
# See whether obm2 works.  Need this for CV sims
#################################################################
#################################################################
#xMult<-growtreeMult(covariates,tsim,dsim,sampsize,ss2,r)
###obm2<-function(bstree, bsprune, ot, od, ocov,prunedtree){
#obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
#tsim200<-rexp(200)
#dsim200<-sample(c(0,1,2),200,replace=T)
#cov200<-cbind(rexp(200),rexp(200),sample(c(0,1),200,replace=T),sample(c(0,1),200,replace=T))
#growtreeMult(cov200,tsim200,dsim200,sampsize,ss2,r)

#V<-10
#v<-1
#v<-2
# cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
# GVlist<-c()
# for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GVMult<-obm2(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
# GVlist<-rbind(GVlist, GV)
# GVlist

#obm2(xMult,zMult,tsim,dsim,covariates,
# the cv should work.  I haven't done mae stuff.
#maeMult function
# need to go to one of the cv sim functions and work throught the mae bit
#########################################################
#########################################################
# calculate MAE for a new set of data
#########################################################
# input: a= tree info, X=old data X2=new data
# output:  the MAE for new data
#########################################################
#aMult<-rbind(zMult,xMult)
#mediannode(aMult,X,X2,tsim)
maeMult1<-function(a,X,X2,told,tnew){
del2<-X2$dsim2;
del1<-X$dsim;
if(length(a)==0){mae<-mean(abs(tnew[del1==1]-median(told[del1==1])))}
else{
# tnode<-termnodelist(a,X);
# mednode<-as.numeric(mediannode(a,X,X2,told));
mednode<-as.numeric(as.vector(mediannode(a,X,X2,told)));
# tsim2<-X2$tsim;
mae<-mean(abs(tnew[del2==1]-mednode[del2==1]))
}
mae
}
#maeMult1(aMult,X,X2,X$tsim,X2$tsim2)
maeMult2<-function(a,X,X2,told,tnew){
del2<-X2$dsim2;
del1<-X$dsim;
if(length(a)==0){mae<-mean(abs(tnew[del1==2]-median(told[del1==2])))}
else{
# tnode<-termnodelist(a,X);
# mednode<-as.numeric(mediannode(a,X,X2,told));
mednode<-as.numeric(as.vector(mediannode(a,X,X2,told)));
# tsim2<-X2$tsim;
mae<-mean(abs(tnew[del2==2]-mednode[del2==2]))
}
mae
}

#####################################
# mediannode2
```

```
#####################################
mediannode2<-function(a,X, X2,told){
tnode<-termnodelist(a,X);
del1<-X$dsim;
del2<-X2$dsim2;
# tsim<-X$tsim;
tnode2<-termnodelist(a,X2);
temp<-by(told[del1==2],factor(tnode[del1==2]),function(x) median(x));
tnodeu<-sort(unique(tnode))
tnode2u<-sort(unique(tnode2))
temp3<-sort(unique(tnode[del1==2]))
temp2<-rbind(temp3,temp)
test<-c();
for(i in 1: length(temp3)){
if(sum(temp3[i]==tnode2u)==0){
test<-c(test,temp3[i])
}
}
if(is.null(test)==F){
for(i in 1:length(test)){
temp2<-temp2[,temp2[1,]!=test[i]]
}
}
test<-c();
for(i in 1: length(tnode2u)){
if(sum(tnode2u[i]==temp3)==0){
test<-c(test,tnode2u[i])
}
}
if(is.null(test)==F){
for(i in 1:length(test)){
temp2<-cbind(temp2,c(test[i],0))
}
}
o<-order(temp2[1,])
#temp2<-temp2[1,]
#temp<-temp2[2,];
temp<-temp2[2,][o];
meds<-factor(tnode2, labels=temp) ;
meds
}
maeMult2(aMult,X,X2,X$tsim,X2$tsim2)


### mae for the mart resid tree but with event 2
#############################################
# MAE function for martingale resid tree, event 2
# input: tempMR=tree,X2= new data (covariates have to have
# same names as in X, X=olddata.
maeMR2<-function(tempMR,X2,X, told, tnew){
del1<-X$dsim
del2<-X2$dsim2;
del<-X2$dsim2;
# calculates the mean prediction for each terminal node,
# we use this as labels for terminal nodes.
pred2<-predict(tempMR,newdata=X2, type="matrix");
# tsim<-X$tsim;
# this calculates the medians per terminal node
pred1<-predict(tempMR)
temp<-by(told[del1==2],factor(pred1[del1==2]),function(x) median(x));
upred1<-sort(unique(pred1))
upred2<-sort(unique(pred2))
temp<-temp[!is.na(match(upred1,upred2))]
# temp<-by(tnew,factor(pred2),function(x) median(x));
# this takes the new predictions for the new data, and uses
# the labels of the medians.
temp2<-as.numeric(as.vector(factor(pred2, labels=temp)));
# tsim2<-X2$tsim2;
maeMR1<-mean(abs(tnew[del2==2]-temp2[del2==2]));
maeMR1
}


#maeMR(tempMR,X2,X2,X2$tsim2, X2$tsim2)
#test<-mediannode(a,X,X2,tsim)

#########################################################
#########################################################
# calculate MAE for a new set of data
#########################################################
```

```
# input: a= tree info, X=old data X2=new data
# output:  the MAE for new data
#########################################################
maeGRLR2<-function(a,X,X2,told,tnew){
del2<-X2$dsim2;
del1<-X$dsim;
if(length(a)==0){mae<-mean(abs(tnew[del1==2]-median(told[del1==2])))}
else{
# tnode<-termnodelist(a,X);
# mednode<-as.numeric(mediannode(a,X,X2,told));
mednode<-as.numeric(as.vector(mediannode2(a,X,X2,told)));
# tsim2<-X2$tsim;
mae<-mean(abs(tnew[del2==2]-mednode[del2==2]))
}
mae
}
#maeGRLR2(aMult,X,X2,X$tsim,X2$tsim2)
#maeGRLR(aLR,X,X2,X$tsim,X2$tsim2)
#a<-aLR
#told<-X$tsim
#tnew<-X2$tsim2
#########################################################
#########################################################
# Simulations:  univariate splits
#########################################################
#########################################################

sim1fun<-function(c1,c2,r1,r2,r3,r4,r5,r6,r7,B,ss,ss2){
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMR2<-c();
nodevec<-rep(0,ss);
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,r1)*(Z<=c1);
t1<-t1+rexp(ss,r2)*(Z>c1);
t2<-rexp(ss,r3)*(Z<=c2);
  t2<-t2+rexp(ss,r4)*(Z>c2);
t3<-rexp(ss,r5)*(Z<=c2);
  t3<-t3+rexp(ss,r6)*(Z>c2&Z<=c1);
t3<-t3+rexp(ss,r7)*(Z>c1);
  tsim<-pmin(t1,t2,t3);
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4])
tempMR2<-c(tempMR2,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2,dissim="man")$splits[4])
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMR2<-tempMR2
output<-cbind(outputGray,outputLR,outputMR,outputMR2);
output
}
#sim1fun2<-sim1fun(0.5,0.3,.2,.4,.7,.56,.1,.24,.04,2,200,20)
sim1fun2000<-sim1fun(0.5,0.3,.2,.4,.7,.56,.1,.24,.04,2000,200,20)
write.table(sim1fun2000,file="sim1fun2000")
sim2fun2000<-sim1fun(0.5,0.3,.1,.3,.5,.21,.4,.69,.49,2000,200,20)
write.table(sim2fun2000,file="sim2fun2000")

#########################################
# sim3fun is for c1=0.3, c2=0.5, sims 3 adn 4
#########################################
sim3fun<-function(c1,c2,r1,r2,r3,r4,r5,r6,r7,B,ss,ss2){
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMR2<-c();
nodevec<-rep(0,ss);
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,r1)*(Z<=c1);
```

```
t1<-t1+rexp(ss,r2)*(Z>c1);
t2<-rexp(ss,r3)*(Z<=c2);
  t2<-t2+rexp(ss,r4)*(Z>c2);
t3<-rexp(ss,r5)*(Z<=c1);
  t3<-t3+rexp(ss,r6)*(Z>c1&Z<=c2);
t3<-t3+rexp(ss,r7)*(Z>c2);
  tsim<-pmin(t1,t2,t3);
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4])
tempMR2<-c(tempMR2,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2,dissim="man")$splits[4])
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMR2<-tempMR2
output<-cbind(outputGray,outputLR,outputMR,outputMR2);
output
}
#sim3fun2<-sim3fun(0.3,0.5,.4,.2,.58,.7,.02,.22,.1,2,200,20)
sim3fun2000<-sim3fun(0.3,0.5,.4,.2,.58,.7,.02,.22,.1,2000,200,20)
write.table(sim3fun2000,file="sim3fun2000")

sim4fun2000<-sim3fun(0.3,0.5,.3,.1,.18,.5,.52,.72,.4,2000,200,20)
write.table(sim4fun2000,file="sim4fun2000")

sim5fun<-function(c1,r1,r2,r3,r4,r5,B,ss,ss2){
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMR2<-c();
nodevec<-rep(0,ss);
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,r1)*(Z<=c1);
t1<-t1+rexp(ss,r2)*(Z>c1);
t2<-rexp(ss,r3)*(Z<=c1);
  t2<-t2+rexp(ss,r4)*(Z>c1);
t3<-rexp(ss,r5);
  tsim<-pmin(t1,t2,t3);
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4])
tempMR2<-c(tempMR2,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2,dissim="man")$splits[4])
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMR2<-tempMR2;
output<-cbind(outputGray,outputLR,outputMR, outputMR2);
output
}
#sim5fun2<-sim5fun(0.5,.2,.4,.7,.5,.1,2,200,20)

sim5fun2000<-sim5fun(0.5,.2,.4,.7,.5,.1,2000,200,20)
write.table(sim5fun2000,file="sim5fun2000")

sim6fun2000<-sim5fun(0.5,.1,.3,.4,.2,.5,2000,200,20)
write.table(sim6fun2000,file="sim6fun2000")

# sim 7fun log normal
sim7fun<-function(c1,c2,r1,r2,r3,r4,r5,r6,r7,r8,B,ss,ss2){
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMR2<-c();
```

```
nodevec<-rep(0,ss);
for(i in 1:B){
Z<-runif(ss);
t1<-rlnorm(ss,r1,1)*(Z<=c1);
t1<-t1+rlnorm(ss,r2,1)*(Z>c1);
t2<-rlnorm(ss,r3,1)*(Z<=c2);
  t2<-t2+rlnorm(ss,r4,1)*(Z>c2&Z<=c1);
t2<-t2+rlnorm(ss,r5,1)*(Z>c1);
t3<-rlnorm(ss,r6,1)*(Z<=c2);
  t3<-t3+rlnorm(ss,r7,1)*(Z>c2&Z<=c1);
t3<-t3+rlnorm(ss,r8,1)*(Z>c1);
  tsim<-pmin(t1,t2,t3);
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4])
tempMR2<-c(tempMR2,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2,dissim="man")$splits[4])
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMR2<-tempMR2;
output<-cbind(outputGray,outputLR,outputMR,outputMR2);
output
}
#sim7fun(0.5,0.3,1.25,0.5,.1,.5,.25,1.5,1.25,2,2,200,20)
#sim7fun<-function  (c1, c2, r1,  r2, r3,r4, r5,r6, r7,  r8,B,ss,ss2){
sim7fun2000<-sim7fun(0.5,0.3,1.25,0.5,.1,.5,.25,1.5,1.25,2,2000,200,20)
write.table(sim7fun2000,file="sim7fun2000")

sim8fun2000<-sim7fun(0.5,0.3,2,0.75,.25,.75,.25,2,1.5,2,2000,200,20)
write.table(sim8fun2000,file="sim8fun200")


##########################################################3
############################################################
# Univariate Tree simulations
############################################################
############################################################
#sim1cv4<-sim1treeCV(5,500,.1,.35,.5,.5,0,20,20,10,1)
#sim1treeCV<-function(B,ss,r1,r2,r9, c1,c2,ss1,ss2,V, alphac){
#r1<-0.1;r2<-0.35;r9<-0.5;c1<-.5;c2<-0.5;V<-10;ss1<-20;ss2<-20;alphac<-1;ss<-1000;
#sim1uniabstree(2,1000,.1,.35,.5,.5,0,20,20,10,1)
########################################
#sim1 from uni paper but with ABS tree
########################################
sim1uniabstree<-function(B,ss,r1,r2,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
MRabsstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1+r2*(Z1>c1& Z2>c2));
t2<-rexp(ss,1-r9-r1-r2*(Z1>c1& Z2>c2));
t3<-rexp(ss,r9);
# t1<-rexp(ss,r1);
# t2<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
# t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
# d2sim<-(dsim==1|dsim==2)^2
d2sim<-(dsim==1)^2
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
```

```
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
# temp2MR<-coxph(Surv(tsim,d2sim)~1);
# resids<-temp2MR$residuals;
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR3<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
#
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
```

```
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1+r2*(Z1>c1& Z2>c2));
t22<-rexp(ss,1-r9-r1-r2*(Z1>c1& Z2>c2));
# t3<-rexp(ss,r9);
# t12<-rexp(ss,r1);
# t22<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
d2sim2<-(dsim2==1)^2
# d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR3<-c(tempMR3,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR3);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

#########################33
# uni tree model 2, with abs tree
############################
#sim2uniabstree(2,500,.05,.2,.2,.5,.5,0,20,20,10,1)
#r1<-.05;r2<-.2;r3<-.2;r9<-.5;c1<-0.5;c2<-0;ss1<-20;ss2<-20;V<-10;alphac<-1;ss<-500;
#sim2cv1<-sim2treeCV(5,500,.05,.2,.2,.5,.5,0,20,20,10,1)
# t1<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
# t2<-rexp(ss,1-r9-r1-r2*(Z1>c1)-r3*(Z2>c2));
# t3<-rexp(ss,r9);
sim2uniabstree<-function(B,ss,r1,r2,r3,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
MRabsstat<-c();
```

```
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t2<-rexp(ss,1-r9-r1-r2*(Z1>c1)-r3*(Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
# d2sim<-(dsim==1|dsim==2)^2
d2sim<-(dsim==1)^2
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
```

```
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
# temp2MR<-coxph(Surv(tsim,d2sim)~1);
# resids<-temp2MR$residuals;
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR3<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
#
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t22<-rexp(ss,1-r9-r1-r2*(Z1>c1)-r3*(Z2>c2));
# t12<-rexp(ss,r1+r2*(Z1>c1& Z2>c2));
# t22<-rexp(ss,1-r9-r1-r2*(Z1>c1& Z2>c2));
# t3<-rexp(ss,r9);
# t12<-rexp(ss,r1);
# t22<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
d2sim2<-(dsim2==1)^2
# d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR3<-c(tempMR3,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR3);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
```

```
#output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

##################################################
# uni tree model 3, with abs tree. log normal model
##################################################
# sim5cv2<-sim5treeCV(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)

#sim3uniabstree(2,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)

sim3uniabstree<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
MRabsstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1-r2*(Z1>c1&Z2>c2),1);
t2<-rlnorm(ss,r3+r4*(Z1>c1&Z2>c2),1);
t3<-rlnorm(ss,r9,1);

tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
# d2sim<-(dsim==1|dsim==2)^2
d2sim<-(dsim==1)^2
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
```

```
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
# temp2MR<-coxph(Surv(tsim,d2sim)~1);
# resids<-temp2MR$residuals;
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR3<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
#
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rlnorm(ss,r1-r2*(Z1>c1&Z2>c2),1);
t22<-rlnorm(ss,r3+r4*(Z1>c1&Z2>c2),1);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
d2sim2<-(dsim2==1)^2
# d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
```

```
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR3<-c(tempMR3,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR3);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}


#####################################################
# uni tree model 4, with abs tree. log normal model
#####################################################
#sim2uniabstree(2,500,.05,.2,.2,.5,.5,0,20,20,10,1)
#r1<-2;r2<-.85;r3<-.85;r4<-.6;r5<-1;r6<-.9;r7<-1.9;r8<-.4;
#r9<-.5;c1<-0.5;c2<-0;ss1<-20;ss2<-20;V<-10;alphac<-1;ss<-500;
# sim4cv6<-sim4treeCV(5,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)
#sim4uniabstree(2,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)

sim4uniabstree<-function(B,ss,r1,r2,r3,r5,r6,r7,r8,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
MRabsstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1-r2*(Z1>c1)-r3*(Z2>c2),1);
t2<-rlnorm(ss,r5*(Z1<=c1&Z2<=c2)+r6*(Z1<=c1&Z2>c2)+
r7*(Z1>c1&Z2<=c2)+r8*(Z1>c1&Z2>c2),1);
t3<-rlnorm(ss,r9,1);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
# d2sim<-(dsim==1|dsim==2)^2
d2sim<-(dsim==1)^2
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
```

```
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
# temp2MR<-coxph(Surv(tsim,d2sim)~1);
# resids<-temp2MR$residuals;
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR3<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
#
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rlnorm(ss,r1-r2*(Z1>c1)-r3*(Z2>c2),1);
t22<-rlnorm(ss,r5*(Z1<=c1&Z2<=c2)+r6*(Z1<=c1&Z2>c2)+
r7*(Z1>c1&Z2<=c2)+r8*(Z1>c1&Z2>c2),1);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
```

```
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
d2sim2<-(dsim2==1)^2
# d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR3<-c(tempMR3,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR3);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

#sim1uniabstree(2,1000,.1,.35,.5,.5,0,20,20,10,1)
sim1uniabstree25<-sim1uniabstree(25,1000,.1,.35,.5,.5,0,20,20,10,1)
write.table(sim1uniabstree25,file="sim1uniabstree25")
sim1uniabstree50<-sim1uniabstree(25,1000,.1,.35,.5,.5,0,20,20,10,1)
write.table(sim1uniabstree50,file="sim1uniabstree50")
sim1uniabstree75<-sim1uniabstree(25,1000,.1,.35,.5,.5,0,20,20,10,1)
write.table(sim1uniabstree75,file="sim1uniabstree75")
sim1uniabstree100<-sim1uniabstree(25,1000,.1,.35,.5,.5,0,20,20,10,1)
write.table(sim1uniabstree100,file="sim1uniabstree100")

sim2uniabstree25<-sim2uniabstree(25,500,.05,.2,.2,.5,.5,0,20,20,10,1)
write.table(sim2uniabstree25,file="sim2uniabstree25")
sim2uniabstree50<-sim2uniabstree(25,500,.05,.2,.2,.5,.5,0,20,20,10,1)
write.table(sim2uniabstree50,file="sim2uniabstree50")
sim2uniabstree75<-sim2uniabstree(25,500,.05,.2,.2,.5,.5,0,20,20,10,1)
write.table(sim2uniabstree75,file="sim2uniabstree75")
sim2uniabstree100<-sim2uniabstree(25,500,.05,.2,.2,.5,.5,0,20,20,10,1)
write.table(sim2uniabstree100,file="sim2uniabstree100")

sim3uniabstree25<-sim3uniabstree(25,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree25,file="sim3uniabstree25")
sim3uniabstree50<-sim3uniabstree(25,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree50,file="sim3uniabstree50")
sim3uniabstree75<-sim3uniabstree(25,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree75,file="sim3uniabstree75")
sim3uniabstree100<-sim3uniabstree(25,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree100,file="sim3uniabstree100")

sim4uniabstree25<-sim4uniabstree(25,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)
write.table(sim4uniabstree25,file="sim4uniabstree25")
sim4uniabstree50<-sim4uniabstree(25,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)
write.table(sim4uniabstree50,file="sim4uniabstree50")
sim4uniabstree75<-sim4uniabstree(25,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)
write.table(sim4uniabstree75,file="sim4uniabstree75")
sim4uniabstree100<-sim4uniabstree(25,500,2,.85,.85,.6,1,.9,1.9,.4,.5,0,20,20,10,1)
write.table(sim4uniabstree100,file="sim4uniabstree100")
# 6.37-
sim3uniabstree25500<-sim3uniabstree(25,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree25500,file="sim3uniabstree25500")

sim3uniabstree5500<-sim3uniabstree(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree5500,file="sim3uniabstree5500")
sim3uniabstree10500<-sim3uniabstree(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree10500,file="sim3uniabstree10500")
```

```
sim3uniabstree15500<-sim3uniabstree(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree15500,file="sim3uniabstree15500")
sim3uniabstree20500<-sim3uniabstree(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree20500,file="sim3uniabstree20500")
sim3uniabstree252500<-sim3uniabstree(5,500,2,1.5,1,1,0.5,.5,0,20,20,10,1)
write.table(sim3uniabstree252500,file="sim3uniabstree252500")


#########################################################3
############################################################
# Multivariate cutpoint simulations
##########################################################
##########################################################

#ss<-500;ss2<-20;cut1<-0.5;r1<-1;r2<-1;r3<-0.2677415;r<--1;
event2dep10<-function(cut1,r1,r2,r3,B,ss,ss2){
# event 2 dep only
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,r1);
t2<-rexp(ss,1+r2*(Z>cut1));
t3<-rexp(ss,r3);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2)$splits[4])
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMRabs<-tempMRabs;
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}
#event2dep10(cut1,r1,r2,r3,   ,B,ss,ss2)
# n=1000,B=2,6.35-<6.45
# n=100,B=1 v little time, n=500, B=1 11.05-10.06 1min
 #event2dep10(0.5,1,1,0.2677415,1,500,10)
# 6.55pm-
#sim1split5001<-event2dep10(0.5,1,1,0.2677415,10,500,10)

#event2dep10(0.5,1,1,0.2677415,2,500,20)
#event2dep10(0.5,1,1,2.45,2,500,20)


#sim1splitLomultABS<-event2dep10(0.5,1,1,0.2677415,1000,500,20)
#sim1splitHimultABS<-event2dep10(0.5,1,1,2.45,1000,500,20)

sim1splitHimultABS800<-event2dep10(0.5,1,1,2.45,200,500,20)
write.table(sim1splitHimultABS800,file="sim1splitHimultABS800")
sim1splitHimultABS1000<-event2dep10(0.5,1,1,2.45,200,500,20)
write.table(sim1splitHimultABS1000,file="sim1splitHimultABS1000")

sim2split<-function(cut1,r1,r2,r3,B,ss,ss2){
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
```

```
t1<-rexp(ss,1+r1*(Z>cut1));
t2<-rexp(ss,1+r2*(Z<=cut1));
t3<-rexp(ss,r3);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4])
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMRabs<-tempMRabs;
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}
#sim2split100<-sim2split(0.5, 1, 1, 1/3,100,500,20)
#sim2split<-function(cut1,r1,r2,r3,B,ss,ss2){
#sim2split(0.5,1,1,1/3,2,500,20)

#sim2splitLomultABS<-sim2split(0.5,1,1,1/3,1000,500,20)

sim2splitLomultABS200<-sim2split(0.5,1,1,1/3,200,500,20)
write.table(sim2splitLomultABS200,file="sim2splitLomultABS200")
sim2splitLomultABS400<-sim2split(0.5,1,1,1/3,200,500,20)
write.table(sim2splitLomultABS400,file="sim2splitLomultABS400")
sim2splitLomultABS600<-sim2split(0.5,1,1,1/3,200,500,20)
write.table(sim2splitLomultABS600,file="sim2splitLomultABS600")
sim2splitLomultABS800<-sim2split(0.5,1,1,1/3,200,500,20)
write.table(sim2splitLomultABS800,file="sim2splitLomultABS800")
sim2splitLomultABS1000<-sim2split(0.5,1,1,1/3,200,500,20)
write.table(sim2splitLomultABS1000,file="sim2splitLomultABS1000")


#sim2splitHimultABS<-sim2split(0.5,1,1,3,1000,500,20)


event2weak<-function(cut1,r1,r2,r3,B,ss,ss2){
# event 2 dep only
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,r1);
t2<-rexp(ss,0.5+r2*(Z>cut1));
t3<-rexp(ss,r3);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4]);
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
```

```
outputMRabs<-tempMRabs;
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}
#event2weak(0.5, 2,0.5,0.3032811353,100,500,20)
#event2weak<-function(cut1,r1,r2,r3,B,ss,ss2
#event2weak(0.5,2,0.5,0.3032811353,2,500,20)

sim3splitLomultABS<-event2weak(0.5,2,0.5,0.3032811353,1000,500,20)
sim3splitHimultABS<-event2weak(0.5,2,0.5,2.738612788,1000,500,20)

lnormsplit<-function(cut1,r1,r2,r3,B,ss,ss2){
# event 2 dep only
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
t1<-rlnorm(ss,r1,1);
t2<-rlnorm(ss,0.5+r2*(Z>cut1),1);
t3<-rlnorm(ss,r3,1);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2,cp=0)$splits[4]);
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMRabs<-tempMRabs;
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}
#sim6splitlo1001<-lnormsplit(0.5,0,0.5,1.5,100,500,20)
#lnormsplit<-function(cut1,r1,r2,r3,B,ss,ss2){
#lnormsplit(0.5,0,0.5,1.5,2,500,20)

#sim4splitLomultABS<-lnormsplit(0.5,0,0.5,1.5,1000,500,20)
#write.table(sim4splitLomultABS, file="sim4splitLomultABS")

#sim4splitHimultABS200<-lnormsplit(0.5,0,0.5,-0.25,200,500,20)
#write.table(sim4splitHimultABS200, file="sim4splitHimultABS200")
#sim4splitHimultABS400<-lnormsplit(0.5,0,0.5,-0.25,200,500,20)
#write.table(sim4splitHimultABS400, file="sim4splitHimultABS400")
#sim4splitHimultABS600<-lnormsplit(0.5,0,0.5,-0.25,200,500,20)
#write.table(sim4splitHimultABS600, file="sim4splitHimultABS600")

sim4splitHimultABS800<-lnormsplit(0.5,0,0.5,-0.25,200,500,20)
write.table(sim4splitHimultABS800, file="sim4splitHimultABS800")
sim4splitHimultABS1000<-lnormsplit(0.5,0,0.5,-0.25,200,500,20)
write.table(sim4splitHimultABS1000, file="sim4splitHimultABS1000")

cutoff3and5<-function(cut1,cut2,r1,r2,r3,B,ss,ss2){
# both events opposite direction
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,1+r1*(Z>cut1));
t2<-rexp(ss,1+r2*(Z<=cut2));
```

```
t3<-rexp(ss,r3);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,cp=0,minbucket=ss2)$splits[4]) ;
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMRabs<-tempMRabs;
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}

# sim7 low splits a=0.3040113919
# both events change opp direction
# r1=1, r2=1, r3 =0.30...
# for hi cens, r3=2.767792536
#cutoff3and5<-function(cut1,cut2,r1,r2,r3,B,ss,ss2){
#cutoff3and5(0.5,0.3,1,1,0.3040113919,2,500,20)

#sim5splitLomultABS<-cutoff3and5(0.5,0.3,1,1,0.3040113919,1000,500,20)
sim5splitLomultABS200<-cutoff3and5(0.5,0.3,1,1,0.3040113919,200,500,20)
write.table(sim5splitLomultABS200,file="sim5splitLomultABS200")
sim5splitLomultABS400<-cutoff3and5(0.5,0.3,1,1,0.3040113919,200,500,20)
write.table(sim5splitLomultABS400,file="sim5splitLomultABS400")
sim5splitLomultABS600<-cutoff3and5(0.5,0.3,1,1,0.3040113919,200,500,20)
write.table(sim5splitLomultABS600,file="sim5splitLomultABS600")
sim5splitLomultABS800<-cutoff3and5(0.5,0.3,1,1,0.3040113919,200,500,20)
write.table(sim5splitLomultABS800,file="sim5splitLomultABS800")
sim5splitLomultABS1000<-cutoff3and5(0.5,0.3,1,1,0.3040113919,200,500,20)
write.table(sim5splitLomultABS1000,file="sim5splitLomultABS1000")


#sim5splitHimultABS<-cutoff3and5(0.5,0.3,1,1,2.767792536,1000,500,20)

cutoff3<-function(cut1,r1,r2,r3,B,ss,ss2){
nodevec<-rep(0,ss);
tempgray<-c();
tempLR<-c();
tempMR<-c();
tempMRabs<-c();
tempMult<-c();
for(i in 1:B){
Z<-runif(ss);
t1<-rexp(ss,1+r1*(Z>cut1));
t2<-rexp(ss,1+r2*(Z<=cut1));
t3<-rexp(ss,r3);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3);
tempgray<-c(tempgray,maxsplit(nodevec,0,Z,tsim, dsim,ss2)[2]);
tempMult<-c(tempMult,maxsplitMult(nodevec,0,Z,tsim,dsim,ss2,r)[2]);
#maxsplitMult(nd,0,Z1,tsim, dsim,ss2,r)
# gray stats are calculated
# the LR stats are calculated
dsim2<-(dsim==1|dsim==2)^2;
tempLR<-c(tempLR,maxsplitLR(nodevec,0,Z,tsim, dsim2,ss2)[2]);
# the residual stats
temp2MR<-coxph(Surv(tsim,dsim2)~1);
resids<-temp2MR$residuals;
tempMR<-c(tempMR,rpart(resids~Z,method="anova", maxdepth=1,minbucket=ss2)$splits[4]);
tempMRabs<-c(tempMRabs,rpart(resids~Z,method="anova",minbucket=ss2,cp=0,maxdepth=1, dissim="man")$splits[4]);
}
outputGray<-tempgray;
outputLR<-tempLR;
outputMR<-tempMR;
outputMRabs<-tempMRabs;
```

```
outputMult<-tempMult;
output<-cbind(outputGray,outputLR,outputMR,outputMRabs,outputMult);
output
}


sim6splitLomultABS<-cutoff3(0.3, 1, 1, 1/3,1000,500,20)
sim6splitHimultABS<-cutoff3(0.3, 1, 1, 3,1000,500,20)


#################################################################
#  Multivariate Tree sims
#################################################################

sim1atreeABS<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
# r1=2,r2=0.5, r3=1, r4=0.5, r9=3...
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1-r2*(Z1>c1& Z2>c2));
t2<-rexp(ss,r3+r4*(Z1>c1& Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
```

```
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MR2 with absolute
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMRabs2<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1-r2*(Z1>c1& Z2>c2));
t22<-rexp(ss,r3+r4*(Z1>c1& Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t2<t1)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
```

```
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMRabs2<-c(tempMRabs2,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMRabs2);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

sim1atreeMultABS25<-sim1atreeABS(25,1000,2,1,1,1,3,0.5,0.5,20,20,5,1)
write.table(sim1atreeMultABS25, file="sim1atreeMultABS25")
sim1atreeMultABS50<-sim1atreeABS(25,1000,2,1,1,1,3,0.5,0.5,20,20,5,1)
write.table(sim1atreeMultABS50, file="sim1atreeMultABS50")
sim1atreeMultABS75<-sim1atreeABS(25,1000,2,1,1,1,3,0.5,0.5,20,20,5,1)
write.table(sim1atreeMultABS75, file="sim1atreeMultABS75")
sim1atreeMultABS100<-sim1atreeABS(25,1000,2,1,1,1,3,0.5,0.5,20,20,5,1)
write.table(sim1atreeMultABS100, file="sim1atreeMultABS100")

########################################3333
#sim2a, weak change event 2, c=0.3
########################################
sim2atreeMultABS<-function(B,ss,r1,r2,r3,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1);
t2<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
```

```
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MRabs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));

# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1);
t22<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
```

```
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
#sim5atree(B,ss,r1,r2,r3,r9, c1,c2,ss1,ss2,V, alphac){

#sim5atree25<-sim5atree(25,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)

#sim2atreeMultABS(2,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)

sim2atreeMultABS25<-sim2atreeMultABS(25,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)
write.table(sim2atreeMultABS25,file="sim2atreeMultABS25")
sim2atreeMultABS50<-sim2atreeMultABS(25,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)
write.table(sim2atreeMultABS50,file="sim2atreeMultABS50")
sim2atreeMultABS75<-sim2atreeMultABS(25,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)
write.table(sim2atreeMultABS75,file="sim2atreeMultABS75")
sim2atreeMultABS100<-sim2atreeMultABS(25,1000,2,1,0.5,3.235374207,0.3,0.3,20,20,5,1)
write.table(sim2atreeMultABS100,file="sim2atreeMultABS100")


########################################3333
#sim3a, rlnorm(n,0=mu,1=sig), multivariate and ABS function
########################################
sim3atreeMultABS<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1-r2*(Z1>c1&Z2>c2));
t2<-rlnorm(ss,r3+r4*(Z1>c1&Z2>c2));
t3<-rlnorm(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
```

```
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MRabs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
```

```
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rlnorm(ss,r1-r2*(Z1>c1&Z2>c2));
t22<-rlnorm(ss,r3+r4*(Z1>c1&Z2>c2));
# t32<-rlnorm(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
#c1=0.3,c2=0.3,r1=1,r2=1,r3=0,r4=1,r9=-0.25
#sim8atree<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){

#sim6atree25<-sim6atree(25,1000,1,1,0,1,-0.25,0.3,0.3,20,20,5,1)
#write.table(sim6atree25,file="sim6atree25")
sim3atreeMultABS25<-sim3atreeMultABS(25,1000,1,1,0,1,-0.25,0.3,0.3,20,20,5,1)
write.table(sim3atreeMultABS25,file="sim3atreeMultABS25")
sim3atreeMultABS50<-sim3atreeMultABS(25,1000,1,1,0,1,-0.25,0.3,0.3,20,20,5,1)
write.table(sim3atreeMultABS50,file="sim3atreeMultABS50")
sim3atreeMultABS75<-sim3atreeMultABS(25,1000,1,1,0,1,-0.25,0.3,0.3,20,20,5,1)
write.table(sim3atreeMultABS75,file="sim3atreeMultABS75")

#set.seed(0.6137893)
sim3atreeMultABS100<-sim3atreeMultABS(25,1000,1,1,0,1,-0.25,0.3,0.3,20,20,5,1)
write.table(sim3atreeMultABS100,file="sim3atreeMultABS100")

#####################################################################
#sim4a tree change both directions, 2 different cut points c1=0.3,c=0.5
#####################################################################
#########################################3333
#aka sim8a when we did not use ABS impurity function
########################################
sim4atreeMultABS<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
```

```
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1-r2*(Z1>c1&Z2>c2));
t2<-rexp(ss,r3+r4*(Z1>c1&Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
```

```
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2, dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
#
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1-r2*(Z1>c1&Z2>c2));
t22<-rexp(ss,r3+r4*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
#sim8atree<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
```

```
#sim8aatree25<-sim8atree(25,1000,2,1,1,1,3,0.3,0.5,20,20,5,1)
#write.table(sim8aatree25,file="sim8aatree25")
sim4atreeMultABS25<-sim4atreeMultABS(25,1000,2,1,1,1,3,0.3,0.5,20,20,5,1)
write.table(sim4atreeMultABS25,file="sim4atreeMultABS25")
sim4atreeMultABS50<-sim4atreeMultABS(25,1000,2,1,1,1,3,0.3,0.5,20,20,5,1)
write.table(sim4atreeMultABS50,file="sim4atreeMultABS50")
sim4atreeMultABS75<-sim4atreeMultABS(25,1000,2,1,1,1,3,0.3,0.5,20,20,5,1)
write.table(sim4atreeMultABS75,file="sim4atreeMultABS75")
sim4atreeMultABS100<-sim4atreeMultABS(25,1000,2,1,1,1,3,0.3,0.5,20,20,5,1)
write.table(sim4atreeMultABS100,file="sim4atreeMultABS100")

sim5atreeMultABS<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1-r2*(Z1>c1&Z2>c2));
t2<-rexp(ss,r3+r4*(Z1>c1&Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
```

```
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MR abs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1-r2*(Z1>c1&Z2>c2));
t22<-rexp(ss,r3+r4*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
```

```
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
sim5atreeMultABS25<-sim5atreeMultABS(25,1000,2,1,1,1,3,0.3,0.3,20,20,5,1)
write.table(sim5atreeMultABS25,file="sim5atreeMultABS25")
sim5atreeMultABS50<-sim5atreeMultABS(25,1000,2,1,1,1,3,0.3,0.3,20,20,5,1)
write.table(sim5atreeMultABS50,file="sim5atreeMultABS50")
sim5atreeMultABS75<-sim5atreeMultABS(25,1000,2,1,1,1,3,0.3,0.3,20,20,5,1)
write.table(sim5atreeMultABS75,file="sim5atreeMultABS75")
sim5atreeMultABS100<-sim5atreeMultABS(25,1000,2,1,1,1,3,0.3,0.3,20,20,5,1)
write.table(sim5atreeMultABS100,file="sim5atreeMultABS100")

sim1btreeMultABS<-function(B,ss,r1,r2,r3,r4,r5,r6,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t2<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss)
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
```

```
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MRabs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));

# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t22<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
```

```
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
#sim4btree<on(B,ss, r1,r2,r3,r4,r5,r6,r9, c1,c2,ss1,ss2,V, alphac){
# sim4btr(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
#sim4btree25<-sim4btree(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
#write.table(sim4btree25, file="sim4btree25")

sim1btreeMultABS25<-sim1btreeMultABS(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
write.table(sim1btreeMultABS25,file="sim1btreeMultABS25")
sim1btreeMultABS50<-sim1btreeMultABS(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
write.table(sim1btreeMultABS50,file="sim1btreeMultABS50")
sim1btreeMultABS75<-sim1btreeMultABS(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
write.table(sim1btreeMultABS75,file="sim1btreeMultABS75")
sim1btreeMultABS100<-sim1btreeMultABS(25,1000,1, 1, 1, 3, 1, 1, 4, 0.5,0.5,20,20,5,1)
write.table(sim1btreeMultABS100,file="sim1btreeMultABS100")

sim2btreeMultABS<-function(B,ss,r1,r2,r3,r4,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1);
t2<-rexp(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
```

```
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MRabs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));

# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
```

```
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1);
t22<-rexp(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
sim2btreeMultABS25<-sim2btreeMultABS(25,1000,2,1,0.5,0.5,3.685356312,0.3,0.3,20,20,5,1)
write.table(sim2btreeMultABS25, file="sim2btreeMultABS25")
sim2btreeMultABS50<-sim2btreeMultABS(25,1000,2,1,0.5,0.5,3.685356312,0.3,0.3,20,20,5,1)
write.table(sim2btreeMultABS50, file="sim2btreeMultABS50")
sim2btreeMultABS75<-sim2btreeMultABS(25,1000,2,1,0.5,0.5,3.685356312,0.3,0.3,20,20,5,1)
write.table(sim2btreeMultABS75, file="sim2btreeMultABS75")
sim2btreeMultABS100<-sim2btreeMultABS(25,1000,2,1,0.5,0.5,3.685356312,0.3,0.3,20,20,5,1)
write.table(sim2btreeMultABS100, file="sim2btreeMultABS100")

sim3btreeMultABS<-function(B,ss,r1,r2,r3,r4,r5,r6,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t2<-rlnorm(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
t3<-rlnorm(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
```

```
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MRabs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));

# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
```

```
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rlnorm(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t22<-rlnorm(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

sim3btreeMultABS25<-sim3btreeMultABS(25,1000,0,0.5,0.5,1,0.5,0.5,0,0.3,0.3,20,20,5,1)
write.table(sim3btreeMultABS25,file="sim3btreeMultABS25")
sim3btreeMultABS50<-sim3btreeMultABS(25,1000,0,0.5,0.5,1,0.5,0.5,0,0.3,0.3,20,20,5,1)
write.table(sim3btreeMultABS50,file="sim3btreeMultABS50")
sim3btreeMultABS75<-sim3btreeMultABS(25,1000,0,0.5,0.5,1,0.5,0.5,0,0.3,0.3,20,20,5,1)
write.table(sim3btreeMultABS75,file="sim3btreeMultABS75")
sim3btreeMultABS100<-sim3btreeMultABS(25,1000,0,0.5,0.5,1,0.5,0.5,0,0.3,0.3,20,20,5,1)
write.table(sim3btreeMultABS100,file="sim3btreeMultABS100")

sim4btreeMultABS<-function(B,ss,r1,r2,r3,r4,r5,r6,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
```

```
t1<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t2<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# MR abs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates))
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
```

```
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t22<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
sim4btreeMultABS25<-sim4btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.3,20,20,5,1)
write.table(sim4btreeMultABS25,file="sim4btreeMultABS25")
sim4btreeMultABS50<-sim4btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.3,20,20,5,1)
write.table(sim4btreeMultABS50,file="sim4btreeMultABS50")
sim4btreeMultABS75<-sim4btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.3,20,20,5,1)
write.table(sim4btreeMultABS75,file="sim4btreeMultABS75")
sim4btreeMultABS100<-sim4btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.3,20,20,5,1)
write.table(sim4btreeMultABS100,file="sim4btreeMultABS100")

sim5btreeMultABS<-function(B,ss,r1,r2,r3,r4,r5,r6,r9, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
```

```
MRstat<-c();
MRabsstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(2)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t2<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
t3<-rexp(ss,r9);
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
```

```
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
#MR abs
tempMRabs<-rpart(resids~covariates,method="anova",minbucket=ss2,dissim="man");
tempMR2abs<-c(length(unique(tempMRabs$where)),covusedMR(tempMRabs,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1+r2*(Z1>c1)+r3*(Z2>c2));
t22<-rexp(ss,r4-r5*(Z1>c1)-r6*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
tempMR2abs<-c(tempMR2abs,maeMR(tempMRabs,X2,X,X$tsim, X2$tsim2),maeMR2(tempMRabs,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
MRabsstat<-rbind(MRabsstat,tempMR2abs);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,MRabsstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
```

```
#sim5btreeMultABS(2,1000,1,1,1,3,1,1,4,0.3,0.5,20,20,5,1)

sim5btreeMultABS25<-sim5btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.5,20,20,5,1)
write.table(sim5btreeMultABS25,file="sim5btreeMultABS25")
sim5btreeMultABS50<-sim5btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.5,20,20,5,1)
write.table(sim5btreeMultABS50,file="sim5btreeMultABS50")
sim5btreeMultABS75<-sim5btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.5,20,20,5,1)
write.table(sim5btreeMultABS75,file="sim5btreeMultABS75")
sim5btreeMultABS100<-sim5btreeMultABS(25,1000,1,1,1,3,1,1,4,0.3,0.5,20,20,5,1)
write.table(sim5btreeMultABS100,file="sim5btreeMultABS100")

##################################################33
# Multivariate tree sims with constant CIF event 1
##################################################

sim1CIF<-function(B,ss,r1,r2,r3,r4,r5, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(1)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1);
t2<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
t3<-rexp(ss,r4-r5*(Z1>c1&Z2>c2));
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
```

```
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1);
t22<-rexp(ss,r2+r3*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
```

```
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
# r1=0.3,r2=0.1 r3=0.3,r4=0.6,r5=0.3
#sim1CIF<-function(B,ss,r1,r2,r3,r4,r5, c1,c2,ss1,ss2,V, alphac){
r<--1
#set.seed(1)
#sim1CIF1<-sim1CIF(1,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF1, file="sim1CIF1")

#set.seed(2)
#sim1CIF2<-sim1CIF(1,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF2, file="sim1CIF2")

#sim1CIF25<-sim1CIF(25,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF25, file="sim1CIF25")
#sim1CIF50<-sim1CIF(25,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF50, file="sim1CIF50")
#sim1CIF75<-sim1CIF(25,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF75, file="sim1CIF75")
#sim1CIF100<-sim1CIF(25,1000,0.3,0.1,0.3,0.6,0.3,0.3,0.3,20,20,5,1)
#write.table(sim1CIF100, file="sim1CIF100")


sim2CIF<-function(B,ss,r1,r2,r3,r4,r5, r6,r7,c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(1)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rexp(ss,r1);
t2<-rexp(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
t3<-rexp(ss,r5-r6*(Z1>c1)-r7*(Z2>c2));
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
```

```
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rexp(ss,r1);
t22<-rexp(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
```

```
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

sim2CIF25<-sim2CIF(25,1000,0.3,0.1,0.2,0.2,0.6,0.2,0.2,0.3,0.3,20,20,5,1)
write.table(sim2CIF25,file="sim2CIF25")
sim2CIF50<-sim2CIF(25,1000,0.3,0.1,0.2,0.2,0.6,0.2,0.2,0.3,0.3,20,20,5,1)
write.table(sim2CIF50,file="sim2CIF50")
sim2CIF75<-sim2CIF(25,1000,0.3,0.1,0.2,0.2,0.6,0.2,0.2,0.3,0.3,20,20,5,1)
write.table(sim2CIF75,file="sim2CIF75")
sim2CIF100<-sim2CIF(25,1000,0.3,0.1,0.2,0.2,0.6,0.2,0.2,0.3,0.3,20,20,5,1)
write.table(sim2CIF100,file="sim2CIF100")



sim3CIF<-function(B,ss,r1,r2,r3,r4,r5, c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(1)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1);
t2<-rlnorm(ss,r2+r3*(Z1>c1&Z2>c2));
t3<-rlnorm(ss,r4-r5*(Z1>c1&Z2>c2));
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
```

```
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t1<-rlnorm(ss,r1);
```

```
t2<-rlnorm(ss,r2+r3*(Z1>c1&Z2>c2));
# t3<-rlnorm(ss,r4-r5*(Z1>c1&Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}

#ss<-1000,c1<-0.3 c2<-0.3 r1<-0 r2<-0 r3<-0.5 r4<-0.5 r5<-0.5
#sim3CIF<-function(B,ss,r1,r2,r3,r4,r5, c1,c2,ss1,ss2,V, alphac){

#sim3CIF1<-sim3CIF(1,1000,0,0,0.5,0.5,0.5,0.3,0.3,20,20,5,1)

#sim3CIF25<-sim3CIF(25,1000,0,0,0.5,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim3CIF25,file="sim3CIF25")
#sim3CIF50<-sim3CIF(25,1000,0,0,0.5,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim3CIF50,file="sim3CIF50")
#sim3CIF75<-sim3CIF(25,1000,0,0,0.5,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim3CIF75,file="sim3CIF75")
#sim3CIF100<-sim3CIF(25,1000,0,0,0.5,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim3CIF100,file="sim3CIF100")

sim4CIF<-function(B,ss,r1,r2,r3,r4,r5, r6,r7,c1,c2,ss1,ss2,V, alphac){
graystat<-c();
LRstat<-c();
MRstat<-c();
Multstat<-c();
for(i in 1:B){
#set.seed(1)
Z1<-runif(ss);
# discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z2)/4
t1<-rlnorm(ss,r1);
t2<-rlnorm(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
t3<-rlnorm(ss,r5-r6*(Z1>c1)-r7*(Z2>c2));
tsim<-pmin(t1,t2,t3);
# vector with 0s 1s and 2s
dsim<-(t1<t2)*(t1<t3)+2*(t2<t1)*(t2<t3)
# the event indicator for LR test needs to be =1 if any event happens, and =0 ow.
d2sim<-(dsim==1|dsim==2)^2
# d2sim<-(dsim==1)^2 # just using old d2sim for a little while
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
```

```
# Z5<-sample(c(0,1),ss,replace=T);
# Z6<-sample(c(0,1),ss,replace=T);
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
nd<-rep(0,ss);
id<-seq(1:ss);
X<-data.frame(cbind(id,tsim,dsim,Z1,Z2,Z3,Z4,Z5,Z6,nd));
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
# Gray
x<-growtree(covariates,tsim,dsim,ss1,ss2);
y<-prunetree(x);
z<-prune2(y);
cvlist<-sample(rep(1:V, len=ss),ss,replace=F)
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcv<-growtree(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2)
ycv<-prunetree(xcv)
zcv<-prune2(ycv)
GV<-obm2(xcv,zcv,cvLv$tsim, cvLv$dsim,covariates[cvlist==v,],z)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-z[2,];
inodes<-z[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtree<-rbind(z,x);
a<-as.matrix(zxtree[,zxtree[2,]>=which.max(Galphahatm)]);
tempgray<-c(termnode(a),covused(x,z,covariates,a));
# LR
xLR<-growtreeLR(covariates,tsim,d2sim,ss1,ss2);
yLR<-prunetreeLR(xLR);
zLR<-prune2LR(yLR);
GVlist<-c()
for(v in 1:V){
cvLv<-X[cvlist==v,]
cvLV<-X[cvlist!=v,]
xcvLR<-growtreeLR(covariates[cvlist!=v,],tsim[cvlist!=v],d2sim[cvlist!=v],ss1,ss2)
ycvLR<-prunetreeLR(xcvLR)
zcvLR<-prune2LR(ycvLR)
GV<-obm2LR(xcvLR,zcvLR,tsim[cvlist==v], d2sim[cvlist==v],covariates[cvlist==v,],zLR)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zLR[2,];
inodes<-zLR[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
Galphahatm<-colMeans(GVlist)-alphac*inodes;
zxtreeLR<-rbind(zLR,xLR);
aLR<-as.matrix(zxtreeLR[,zxtreeLR[2,]>=which.max(Galphahatm)]);
tempLR<-c(termnode(aLR),covused(xLR,zLR,covariates,aLR));
# MR
temp2MR<-coxph(Surv(tsim,d2sim)~1);
resids<-temp2MR$residuals;
tempMR<-rpart(resids~covariates,method="anova",minbucket=ss2);
tempMR2<-c(length(unique(tempMR$where)),covusedMR(tempMR,covariates));
# Mult
#xMultliver<-growtreeMult(covariates,nhxdays,d,   20, 20,-1)
xMult<-growtreeMult(covariates,tsim,   dsim,ss1,ss2,r);
## when ss=500, this can take a while, 10.30..<10.38
yMult<-prunetreeMult(xMult);
zMult<-prune2(yMult);
GVlist<-c()
for(v in 1:V){
# cvLv<-X[cvlist==v,]
# cvLV<-X[cvlist!=v,]
xcvMult<-growtreeMult(covariates[cvlist!=v,],tsim[cvlist!=v],dsim[cvlist!=v],ss1,ss2,r)
ycvMult<-prunetreeMult(xcvMult)
zcvMult<-prune2(ycvMult)
GV<-obm2Mult(xcvMult,zcvMult,tsim[cvlist==v], dsim[cvlist==v],covariates[cvlist==v,],zMult)
GVlist<-rbind(GVlist, GV)
GVlist
}
pruneorder<-zMult[2,];
inodes<-zMult[5,];
inodes<-c(sort(unique(inodes),decreasing=T),0);
```

```
Galphahatm<-colMeans(GVlist)-(alphac*4)*inodes;
zxtreeMult<-rbind(zMult,xMult);
aMult<-as.matrix(zxtreeMult[,zxtreeMult[2,]>=which.max(Galphahatm)]);
tempMult<-c(termnode(aMult),covused(xMult,zMult,covariates,aMult));
#second validation data set: currently without censoring
Z1<-runif(ss);
## discrete uniform (0,.25,.5,.75,1)
Z1<-floor(5*Z1)/4;
# Z2<-sample(c(0,1),ss,replace=T);
Z2<-runif(ss);
Z2<-floor(5*Z1)/4;
t12<-rlnorm(ss,r1);
t22<-rlnorm(ss,r2+r3*(Z1>c1)+r4*(Z2>c2));
# t32<-rexp(ss,r9);
# tsim2<-pmin(t12,t22,t32);
tsim2<-pmin(t12,t22);
## vector with 0s 1s and 2s
  dsim2<-(t22<t12)+1
# dsim2<-(t12<t22)*(t12<t32)+2*(t22<t12)*(t22<t32)
# d2sim2<-(dsim2==1)^2
d2sim2<-(dsim2==1|dsim2==2)^2
# Z3<-floor(5*runif(ss))/4;
# Z4<-floor(5*runif(ss))/4;
Z3<-runif(ss);
Z3<-floor(5*Z3)/4;
Z4<-runif(ss);
Z4<-floor(5*Z4)/4;
Z5<-runif(ss);
Z5<-floor(5*Z5)/4;
Z6<-runif(ss);
Z6<-floor(5*Z6)/4;
covariates<-cbind(Z1,Z2,Z3,Z4,Z5,Z6);
nd2<-rep(0,ss);
id2<-seq(1:ss);
X2<-data.frame(cbind(id2,tsim2,dsim2,covariates,nd2));
## output
tempgray<-c(tempgray,maeGRLR(a,X,X2,X$tsim,X2$tsim2),maeGRLR2(a,X,X2,X$tsim,X2$tsim2));
tempLR<-c(tempLR, maeGRLR(aLR,X,X2,X$tsim,X2$tsim2),maeGRLR2(aLR,X,X2,X$tsim,X2$tsim2));
tempMult<-c(tempMult, maeMult1(aMult,X,X2,X$tsim,X2$tsim2), maeMult2(aMult,X,X2,X$tsim,X2$tsim2));
tempMR2<-c(tempMR2,maeMR(tempMR,X2,X,X$tsim, X2$tsim2),maeMR2(tempMR,X2,X,X$tsim, X2$tsim2));
graystat<-rbind(graystat,tempgray);
LRstat<-rbind(LRstat,tempLR);
MRstat<-rbind(MRstat,tempMR2);
Multstat<-rbind(Multstat,tempMult);
}
output<-rbind(graystat, LRstat, MRstat,Multstat);
#output<-rbind(graystat, LRstat,Multstat);
#output<-rbind(tempgray,tempLR,tempMR2);
output
}
#sim4CIF<-function(B,ss,r1,r2,r3,r4,r5, r6,r7,c1,c2,ss1,ss2,V, alphac){
#sim4CIF25<-sim4CIF(25,1000,0.5,1,0.5,0.5,0,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim4CIF25,file="sim4CIF25")
#sim4CIF50<-sim4CIF(25,1000,0.5,1,0.5,0.5,0,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim4CIF50,file="sim4CIF50")
#sim4CIF75<-sim4CIF(25,1000,0.5,1,0.5,0.5,0,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim4CIF75,file="sim4CIF75")
#sim4CIF100<-sim4CIF(25,1000,0.5,1,0.5,0.5,0,0.5,0.5,0.3,0.3,20,20,5,1)
#write.table(sim4CIF100,file="sim4CIF100")
```

BIBLIOGRAPHY

[1] H. Akaike. A new look at model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont California, 1984.

[3] A. Ciampi, J. Thiffault, J. P. Nakache, and B. Asselain. Stratification by stepwise regression, correspondence analysis and recursive partition. *Computational Statistics and Data Analysis*, 4:185–204, 1986.

[4] A. Ciampi, J. Thiffault, J.P. Nakache, and B. Asselain. Stratification by stepwise regression, correspondence analysis and recursive partition. *Computational Statistics and Data Analysis*, 4:185–204, 1986.

[5] E. Clave, V. Rocha, K. Talvensaari, M. Busson, C. Douay, M.L. Appert, C. Rabian, M. Carmagnat, F. Garnier, A. Filion, G. Socle, E. Gluckman, D. Charron, and A. Toubert. Prognostic value of pretransplantation host thymic function in hla-identical sibling hematopoietic stem cell transplantation. *Blood*, 6(105):2608–2613, 2005.

[6] D.R. Cox. Regression models and life tables (with discussion). *Journal of the Royal Statistics Society - Series B*, 34:187–220, 1972.

[7] R. Davis and J. Anderson. Exponential survival trees. *Statistics in Medicine*, 8:947–962, 1989.

[8] J. Fan, X. Su, R.A. Levine, M.E. Nunn, and M. LeBlanc. Trees for correlated survival data by goodness of split, with applications to tooth prognosis. *Journal of the American Statistical Association*, 101:959–967, 2006.

[9] S.S. Farag, B.J. Bolwell, P.J. Elder, M. Kalaycio, T. Lin, B. Pohlman, S. Penza, G. Marcucci, W. Blum, R. Sobecks, B.R. Avalos, J.C. Byrd, and E. Copelan. High-dose busulfan, cyclophosphamide, and etoposide does not improve outcome of allogeneic stem cell transplantation compared to bucy2 in patients with acute myeloid leukemia. *Bone Marrow Transplantation*, 7(35):653–661, 2005.

[10] J. P. Fine and R. J. Gray. A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association*, 94:496–509, 1999.

[11] F. Gao, A.K. Manatunga, and S. Chen. Identification of prognostic factors with multivariate survival data. *Computational Statistics and Data Analysis*, 45:813–824, 2004.

[12] F. Gao, A.K. Manatunga, and S. Chen. Developing multivariate survival trees with a proportional hazards structure. *Journal of Data Science*, 4:343–356, 2006.

[13] L. Gordon and R. Olshen. Tree-structured survival analysis. *Cancer Treatment Reports*, 69:1065–1069, 1985.

[14] R. Gray. A class of k-sample tests for comparing the cumulative incidence of a competing risk. *The Annals of Statistics*, 16(3):1141–1154, 1988.

[15] P. Guardiola. Effect of haart on liver-related mortality in patients with hiv/hcv coinfection. *Lancet*, 363((9408)):570, 2004.

[16] H. Jin, Y. Lu, K. Stone, and D. Black. Alternative tree-structured survival analysis based on variance of survival time. *Medical Decision Making*, 24:670–680, 2004.

[17] J.P. Klein and M.L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer, London, 2003. 2nd ed.

[18] M. LeBlanc and J. Crowley. Relative risk trees for censored survival data. *Biometrics*, 48:411–425, 1992.

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

[20] X. Luo and B. Turnbull. Comparing two treatments with multiple competing risks endpoints. *Statistica Sinica*, 9:985–997, 1999.

[21] M. Malinchoc, P. Kamath, F. Gordon, C. Peine, J. Rank, and P. Ter Borg. A model to predict poor survival in patients undergoing transjugular intrahepatic portosystemic shunts. *Hepatology*, 31:864–871, 2000.

[22] A. Molinaro, S. Dudoit, and M. van der Laan. Tree-based multivariate regression and density estimation with right censored data. *Journal of Multivariate Analysis*, 90:154–177, 2004.

[23] M. Morgan and J. Sonquist. Problems in the analysis of survey data and a proposal. *Journal of the American Statistical Association*, 58:415–434, 1963.

[24] M. S. Pepe. Inference for events with dependent risks in multiple endpoint studies. *Journal of the American Statistical Association*, 86(415):770–778, 1991.

[25] M.S. Pepe and M. Mori. Kaplan-meier, marginal or conditional probability curves in summarizing competing risks failure time data? *Statistics in Medicine*, 12:737–751, 1993.

[26] R.L. Prentice, J.D. Kalbfleisch, A.V. Peterson, N. Farewell V.T. Flournoy, and Breslow N.E. The analysis of failure times in the presence of competing risks. *Biometrics*, 34:541–554, 1978.

[27] J. Robins and A. Rotnitzky. *Recovery of information and adjustment for dependent censoring using surrogate markers*. Bikhauser, Basel, 1992. chapter in AIDS Epidemiology, Methodological Issues.

[28] V. Rocha, M. Labopin, G. Sanz, W. Arcese, R. Schwerdtfeger, A. Bosi, N. Jacobsen, T. Ruutu, M. de Lima, J. Finke, F. Frassoni, and E. Gluckman. Transplants of umbilical-cord blood or bone marrow from unrelated donors in adults with acute leukemia. *New England Journal of Medicine*, 22(351):2276–2285, 2004.

[29] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[30] M.R. Segal. Regression trees for censored data. *Biometrics*, 44:35–47, 1988.

[31] X. Su and J. Fan. Multivariate survival trees by goodness of split. *Technical Report 367. Department of Statistics, University of California, Davis.*, 2001.

[32] X. Su and J. Fan. Multivariate survival trees: A maximum likelihood approach based on frailty models. *Biometrics*, 60:93–99, 2004.

[33] T.M. Therneau, P.M. Grambsch, and T. Fleming. Martingale based residuals for survival models. *Biometrika*, 77:147–160, 1990.

[34] M. van der Laan and J. Robins. *Unified methods for censored longitudinal data and causality*. Springer, New York, 2003.

[35] L. J. Wei, D. Y. Lin, and L. A. Weissfeld. Regression analysis of multivariate incomplete failure time data by modelling marginal distributions. *Journal of the American Statistical Association*, 84:1065–1073, 1989.