Combined Ranking and Regression Trees for Algorithm Selection

Lukas Fehring

June 10, 2022

Version: Final Version

PADERBORN UNIVERSITY
The University for the Information Society

Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

INTELLIGENT SYSTEMS
Intelligent Systems Group (ISG)

Bachelor's Thesis

Combined Ranking and Regression Trees for Algorithm Selection

Lukas Fehring

1. Reviewer	Associate Professor Dr. Arunselvan Ramaswamy Department of Computer Science Paderborn University
2. Reviewer	Junior Professor Dr. Henning Wachsmuth Department of Computer Science Paderborn University
Supervisors	Associate Professor Dr. Arunselvan Ramaswamy and Junior Professor Dr. Henning Wachsmuth

June 10, 2022

Lukas Fehring
Combined Ranking and Regression Trees for Algorithm Selection
Bachelor's Thesis, June 10, 2022
Reviewers: Associate Professor Dr. Arunselvan Ramaswamy and Junior Professor Dr. Henning Wachsmuth
Supervisors: Associate Professor Dr. Arunselvan Ramaswamy and Junior Professor Dr. Henning Wachsmuth

Universität Paderborn
Intelligent Systems Group (ISG)
Department of Computer Science
Pohlweg 51
33098 and Paderborn

Abstract

It is well known there are good average algorithms to solve some problem classes. However, there is no best solver for all instances of said class. Rather, an algorithm's performance is highly dependent on the instance. In algorithm selection, one tries to find the best algorithm to solve a problem instance and therefore improve over good average solvers.
This is done by choosing the best instance solver (algorithm with the lowest runtime) based on the features of said instance. Since machine learning models can make predictions based on the instance's similarity to past instances, they are a good candidate to solve this problem. In past research, regression and ranking have been used to predict the best algorithm (mostly) as separate approaches. However, there already are machine learning models that combine ranking and regression. In this thesis, we introduce Hybrid Decision Trees. A hybrid decision tree is based on a decision tree with splits based on a convex combination of ranking and regression losses. In addition to the split, one can choose different components to impact the hybrid decision trees.
The quality of hybrid decision trees is evaluated on a selection of scenarios that cover different problem classes. Compared to some algorithm selection approaches (e.g. per algorithm decision tree regressors) hybrid decision trees perform adequately (on some scenarios). However, they rival neither the quality of e.g. survival trees nor the quality of preexisting hybrid models.
The main result of this thesis is that a convex combination of ranking and regression loss can improve the prediction quality over an algorithm selection model based solely on regression/on ranking.

Contents

1 Introduction of Hybrid Decision Trees

For most problem classes (sorting), there are various algorithms that can solve all problem instances (can sort all lists). These algorithms are called solvers and their performance depends on the problem instance (list that is to be sorted). While there are algorithms that are widely considered to be better solvers than others (Mergesort is considered to have better performance than Bubblesort), this is not true for all problem instances. However, these solvers do not perform best on all instances [Ker+18]. Since using the best performing algorithm can save resources the question of finding the best solver for specific problem instances is introduced. A trivial example that illustrates that. Mergesort has the same performance for every list and is considered a good solver. The performance of Bubblesort is dependent on the list. The algorithm performs well on some lists on poor on others. If one knew whether Bubblesort performs better or worse than Mergesort on a given list, one would always choose the better performing algorithm.

Because of this performance complimentarity, it is reasonable to decide which algorithm to use based on the input data for some problem classes. This problem is called Algorithm Selection [Ric76] and has been tackled with a wide variety of approaches in the past. In general, the research is focused on the area of machine learning [Ker+18; Hut+15; Xu+] with the general idea of training a model M to predict the best performing algorithm from a set of candidate algorithms (for some instance) [Hut+15].
To find the best algorithm, one often models Algorithm Selection as a regression, classification, or ranking problem.

The training data of a classification model is a set of problem instances and their respective best performing algorithms. Given an instance, a classification model then predicts which candidate algorithm performs best. Since the training data only consists of the best performing algorithms, the model can not utilize any additional data

(e.g. performance information of the candidate algorithms). Thus, this approach performs poorly compared to different approaches and is therefore not mentioned further [Tor+20].
When using regression, the general idea is to train a set of models. Each model M_i it trained to predict the performance of a separate algorithm A_i. Therefore the training data of M_i consists of instances and the respective performances of A_i on the instance. To predict which algorithm performs best, each model M_i predicts the performances of their respective algorithms and the best performing algorithm is selected as a prediction [Hut+15].
Ranking differs from classification and regression because the model predicts a ranking of algorithms. The training data consists of a set of instances and a respective quality ranking of all solvers. A higher ranking of A_1 than A_2 means that A_1 is predicted to perform better than A_2 [Hül+08].

As mentioned above a regression based model is based on determining the best solver by making performance predictions for all algorithms (with a separate model for each algorithm). The models are trained to make accurate performance predictions. While these predictions are a good starting point, the model does not directly predict which algorithm performs best. The disadvantage of this is shown in Figure 1.1. Here, the algorithm performance predictions of two models (model one, and model two) are compared. This is an example of algorithm selection with two candidate algorithms (algorithm A, and algorithm B) for one instance. The ground-truth performances indicated that algorithm B is a better solver (performs better) for the given instance. This is predicted correctly by model one. However, model two predicts algorithm A to have a better performance. Although a choice of algorithm based on model one would give the correct solution, model one's predictions are poorer than model two's in terms of the regression loss. This is the case as the difference between the algorithms' ground truth and predicted performances is larger. This shows that, while model one gives a better solution to the algorithm

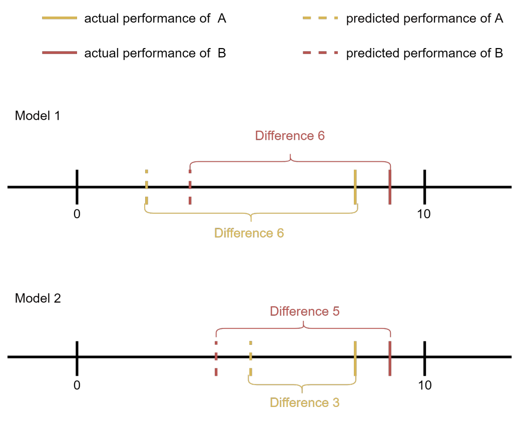


Fig. 1.1.: Illustration of the weakness of regression for algorithm selection. In this figure, two models are utilized to predict whether algorithm A or algorithm B is better suited for some problem instance.

selection problem, the performance predictions of model one are worse than the predictions of model two. Therefore, model one is the worse model in terms of regression.

In comparison to regression, ranking seems to be a better approach to the algorithm selection problem because the model is trained to predict a ranking of algorithms instead of their performance. To that end, the error used to train a model does not depend on the difference in runtime. Rather it depends on the ranking of algorithms. In Figure 1.1 one can see that model one predicts algorithm B to outperform algorithm A (opposite in prediction of model two). This means that model one is the better model in terms of ranking. Since having an inversion in a ranking of algorithm performances might be worse if the algorithms that are ranked wrong have a significant difference in their ground-truth performance, it is worth exploring whether the model's performance can be improved by utilizing performance and ranking information at the same time. [Han+20].

In this thesis, we will combine the approaches of Ranking and Regression to train a model M. This is motivated by the fact that both Regression and Ranking-based approaches have performed well in the past. However, they have inherent shortcomings explained above. The idea is to balance out these shortcomings by using a combined approach. To that end, we explore the idea of using a combined loss function to train a model as in “Hybrid Ranking and Regression for Algorithm Selection” [Han+20].

In contrast to the paper named above we choose "Binary Decision Tree" as our model. This model is a well-known approach in machine learning and has been successfully used in the past on both Regression and Label Ranking problems [Bre+84; CHH09]. Additionally, Decision Trees have been used as a model to solve the algorithm selection problems (with regression and random forests) [Hut+15] in the past. These past results make this approach very appealing. The resulting decision tree is called hybrid decision tree.

Thesis Overview The contents of this thesis can be split into three parts:
First, the algorithm selection problem is introduced and related work is explained (chapter 2).
Building on that, hybrid decision trees are introduced (chapter 3) as a candidate model to solve the algorithm selection problem. Note that hybrid decision trees are a very versatile model, and for most components various candidates are introduced.
The quality of hybrid decision trees is then evaluated. For this evaluation, first, the overall goals and research questions are discussed (chapter 4), and then hybrid decision trees are evaluated (chapter 5). The first evaluation step is utilizing ablation studies to find candidate hybrid decision tree configurations that are then compared to other algorithm selection models. Lastly, the research questions are answered based on the evaluation, the results are summarized, and directions for future research that builds on this thesis are introduced (chapter 6).

Problem Definition and Related Work

2

2.1 Problem Definition

As explained in chapter 1, the problem of Algorithm Selection is motivated by the observation that no algorithm performs best for each problem instance. Instead, the performance of an algorithm is highly dependent on the underlying problem instance. Suppose some good average solvers take hours, days, or even months to solve a specific problem instance, then there is a high incentive for pre-calculating an algorithm that performs better (on this particular instance) since this could save resources [Ker+18].

2.1.1 Algorithm Selection

The problem of selecting the best algorithm for a problem instance is called Algorithm Selection [Ric76]. Given a problem (e.g., sorting a list), there are various instances $\mathcal{I}$ of said problem (different lists) and various candidate algorithms $\mathcal{A} = \{A_1, ..., A_k\}$ that one may choose. To that end, one is interested in mappings

$$s : \mathcal{I} \rightarrow \mathcal{A} \tag{2.1}$$

called algorithm selectors that map problem instances (the set of possible instances $\mathcal{I}$ is called instance space) to the best algorithm from a set of candidate algorithms $\mathcal{A}$.
To that end, one introduces a performance function

$$m : \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R} \tag{2.2}$$

This function maps a combination of an instance I_n and algorithm A_i to a performance score that quantifies A_i's performance on I_n (note that given one Algorithm

A_i there is only one performance function m_i that is used to predict A_i's performance on all instances). For simplicity, we define $m_i(I_n) := m(I_n, A_i)$. For the algorithm A_j with the best performance on instance I_n, it holds $m(I_n, A_j) \geq m(I_n, A_i)$ for all other algorithms $A_i \in \mathcal{A}$.

A perfect selector $s^* \in \{s | s : \mathcal{I} \to A\}$ would be the oracle

$$s^*(I_n) \in \operatorname*{arg\,max}_{A_i \in \mathcal{A}} \mathbb{E}[m(I_n, A_i)] \tag{2.3}$$

for all $I_n \in \mathcal{I}$. The $\mathbb{E}$ is needed in Equation 2.3 as many algorithms contain randomness, which might lead to an algorithm not achieving the same performance on an instance I_n in 2 different runs [Tor+20]. Therefore, one should not just consider the performance of one run but the average performance of A_i on instance I_n.

Since it is impossible to calculate an algorithm's performance (without running it), the problem of choosing the best algorithm is not easily solved.

2.2 Related Work

There has been a wide variety of approaches applied to the algorithm selection problem [Ker+18; Hut+15; Xu+] which are relevant to this thesis. However, this thesis is mainly impacted by Ranking, Regression, and Combined Ranking and models. The related papers are noted below. In addition to these papers, some other publications influenced this thesis. Those are denoted throughout the thesis and can be found in the bibliography.

2.2.1 ASlib: A Benchmark Library for Algorithm Selection

"ASlib: A Benchmark Library for Algorithm Selection"[Bis+16] is a significant paper in the algorithm selection area. Among other things a standardized format for Algorithm Selection scenarios is proposed. This standardized format and the scenarios given in the associated repository enable a better comparison of different algorithm selection approaches. It has since been adopted by most other related papers cited in this thesis. Based on this format, there is a wide variety of datasets from different problem scenarios in the associated repository.

2.2.2 Algorithm Selection with Regression

The idea of regression has been incorporated into various algorithm selection approaches in the past. This is particularly true for the Selection of a SAT-Solver. While there exist very efficient SAT Solvers, it is possible that for a given input, one solver outperforms the other by a large margin. Therefore, past research already focused on this. In [Xu+] the well known approach "Satzilla07" was introduced. Here the authors use ridge Regression to predict the runtime of SAT solvers. While this approach is not state-of-the-art anymore, it is still shown that using Regression might be a good approach in some cases.

2.2.3 Algorithm Selection with Survival Trees

Survival Trees build on the Regression approaches but focus on dealing with incomplete training instances. As explained later, incomplete data may introduce a bias into a regression model if it is set to $\mu \cdot C$. To solve this problem, the authors combined Regression-based approaches in [Tor+20] with methods of survival analysis to train a Random survival forest that predicts a survival time.

2.2.4 Algorithm Selection with Combined Ranking and Regression

There has already been research on combined ranking and regression for algorithm selection in the past. In [Han+20], it has been shown that the combined approach of regression rand ranking performs better than other state-of-the-art approaches in the given scenarios. The authors used a combined ranking and regression loss as in Equation 3.57 to train linear/quadratic models and feed-forward neural networks. For example, they used the squared hinge ranking loss/ and mean squared error. However, the tendency was that a higher value of λ (more significant impact of ranking) leads to the best results.
This paper has motivated most of the work in this thesis.

2.2.5 Label Ranking

Label ranking approaches are well known in machine learning [VG10]. An example of this is research on rating the value of information to build a ranking [Val+09]. However, ranking is also prominent in other areas of machine learning, such as Algorithm Selection, and since this thesis deals with Algorithm Selection the related work mentioned here is focused on that.

Label Ranking Trees

Very closely related to the model of this thesis is the paper "Decision Tree and Instance-Based Learning for Label Ranking" [CHH09]. In this paper, the authors use a ranking loss based on the variance in the rankings of each label to train a model for label ranking scenarios (not specialized in the topic of algorithm selection). To this end, they need to define a "locally correct" order with the 'Expected Maximization' algorithm. While this paper showed that Label Ranking Trees do not perform as well as the other instance-based ranking approach it still showed that Label Ranking Trees perform just as well as other state-of-the-art approaches.

Label Ranking for Algorithm Selection

As mentioned above, there has been work on ranking for algorithm selection in the past. In [OHL15] the authors explain an approach called "Ranking-Based Algorithm Selection" to choose a solver from a set of solvers. This approach is evaluated on the 'SAT 2012' competition dataset (also used in this thesis) and performs "comparable or adequate" compared to "SATZILLA" on most sets from the SAT 2012 competition.

Methodology 3

In this thesis, we explore utilizing a convex combination of ranking losses and regression losses to train a binary decision tree. We call this model hybrid decision tree. However, hybrid decision trees are a very adaptable model in general, the utilized losses are just one component of hybrid decision trees. Therefore, we introduce various hybrid decision tree components in this chapter.

Firstly, we motivate hybrid decision trees (section 3.1) and then discuss the training data structure used to build the tree (section 3.2). After that, the model of hybrid decision trees is introduced (section 3.3). As mentioned above, binary decision trees are a versatile model with different initializations. Each group of components is explained in a separate section from section 3.4 to section 3.7.

3.1 Approach Design

As mentioned in section 2.1 the problem of algorithm selection deals with selecting the best performing algorithm for an instance I_n. A natural approach is predicting the performances of algorithms $A_1, ..., A_k$ on this instance. This can be formalized as making predictions on the performance function introduced in Equation 2.2. Supervised machine learning is an appropriate approach to this problem, as it allows to make predictions based on past data which can be generated/is generated already (refer to section 2.2.1).

As machine learning models don't work on the concrete instance I but rather its features, one needs to transform I into a vector of features $\boldsymbol{I} \in \mathbb{R}^n$. Even though feature calculation is not further explained in this thesis, one should understand that the intuition is to find numeric properties that provide information on I that correlate with the difficulty solving I with the candidate algorithms. However, not

every feature has to provide information for every algorithm. Example features that might be useful for the Boolean Satisfiability problem (SAT) ([Xu+]) are number of "and" operators $\wedge$, the number of "or" operators $\vee$.
As mentioned earlier $\boldsymbol{I}$ can be extracted from the instance I. However, we treat the instance and the vector extracted from the instance as synonyms.
To solve the problem of selecting one of the best algorithms (Equation 2.3) with machine learning, one approach is fitting a separate Regression model

$$M_i : \mathcal{I} \to \mathbb{R} \tag{3.1}$$

for each Algorithm $A_i \in \mathcal{A}$ to make performance predictions. These models $M_1, ..., M_k$ can be understood as one combined Model

$$M : \mathcal{I} \to \mathbb{R}^k \tag{3.2}$$

$$\mathbf{I} \mapsto \hat{\boldsymbol{y}} \qquad \text{with } \hat{\boldsymbol{y}} = (M_1(\boldsymbol{I}), \ldots M_k(\boldsymbol{I})) \in \mathbb{R}^k \tag{3.3}$$

This regression approach of training separate regressors (e.g., regression trees [Bre+84]) for each algorithm motivates the general direction of this thesis. The main difference is that instead of training individual models $M_1, ..., M_k$ for each algorithm, we choose to train one model M that makes performance predictions for all algorithms. To train this model M, a convex combination of a ranking loss and a regression loss is utilized.

3.2 Data Design

As briefly mentioned, we use the supervised learning model Decision Trees [Bre+84]. Decision trees are based on the idea of recursively splitting a set of data into two subsets. To explain how decision trees work in detail, we first introduce the structure of our training data.

The idea is to combine each problem instance I_n from $\mathcal{I}$ with a label that consists of the performance of all considered algorithms $A_1, ..., A_k$ as in “Hybrid Ranking and Regression for Algorithm Selection” [Han+20]. This results in the following data design

$$\mathcal{D} = \{(\boldsymbol{I}_n, \mathbf{y}_n)\}_{n=1}^N = \{(\boldsymbol{I}_n, (m'_1(\boldsymbol{I}_n), ..., m'_k(\boldsymbol{I}_n)))\}_{n=1}^N \tag{3.4}$$

$$\mathbf{y}_n = (m'_1(\boldsymbol{I}_n), ..., m'_k(\boldsymbol{I}_n)) \in \mathbb{R}^k \tag{3.5}$$

where $\mathbf{y}_n$ is referred to as the label of an instance I_n.

The functions

$$m'_i : \mathcal{I} \rightarrow \mathbb{R} \tag{3.6}$$

are introduced as an extension of the performance functions m_i introduced in Equation 2.2. There are two key differences in the behavior of m_i, and m'_i. Firstly, while $m_i(\boldsymbol{I}_n)$ is the performance of A_i on the instance I_n, $m'_i(\boldsymbol{I}_n)$ is scaled to the interval from [0,1]. This is done as ranking (introduced in section 3.6.3) and regression losses (introduced in section 3.6.2) often differ in their target domains with unintended consequences (further explained in section 3.6.5). This scaling results from our preliminary results, which are explained in section 5.3, and leads to the scaled ranking losses. The other difference is that in the case of incomplete performance information ($m_i(\boldsymbol{I}_n)$ not known), m' is set to a fixed value that exceeds the largest reachable value (refer to next paragraph) [Han+20].

3.2.1 Incomplete Labels

The incomplete performance information of an algorithm (introduced above) can result from various circumstances in generating training data. In this process, every algorithm needs to be run on every instance, and it might be that there are algorithms that perform poorly on some instances. This results in the following problem: Running all algorithms till completion could result in a very costly process of building training datasets. To solve this issue, one introduces a threshold C. All algorithms whose computation is not finished when the runtime reaches C are terminated. As a result, there are combinations of instances and algorithms with no exact/with censored performance information. The only knowledge about their performance is that the runtime exceeds C. The affected instances should not be removed from the set of instances for two reasons:

1. Instances that have incomplete labels should not be removed from consideration as some other algorithms might perform reasonably well on them.

2. The knowledge that an algorithm is terminated on an instance timeout is also important. If one excluded this instance from consideration, the training data would make no implications for the algorithm's performance on similar instances.

A similar logic shows that one should not remove the algorithms from consideration either:

- Algorithms that are terminated on some instances might be the best performing algorithms on others. Therefore, disregarding them would contradict the idea of finding the best solver for a problem instance.

Handling of Censored Information Therefore, an instance with censored performance data needs to be handled. To that end, we have a variety of options [Tor+20]. In this thesis we set their performance to

$$m'_i : \mathcal{I} \to \mathbb{R} \qquad m'(\boldsymbol{I}_n, A_i) = \begin{cases} m(\boldsymbol{I}_n, A_i) & , m(\boldsymbol{I}_n, A_i) < C \\ \mu \cdot C & , \text{else} \end{cases} \tag{3.7}$$

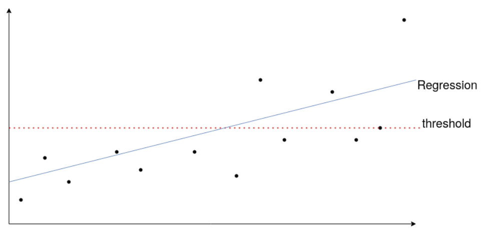
motivated by the par10 (penalized average runtime) loss [Tor+20]. This sets the performance of algorithms that are terminated because C is reached to worse performance than any algorithm that completes its calculation. This results in disregarding them as the choice for the best-performing algorithm and is commonly done with $\mu = 10$ (as in PAR10).
As the performances are scaled to $[0, 1]$, the final performance function

$$m'_i : \mathcal{I} \to [0, 1] \qquad m'(\boldsymbol{I}_n, A_i) = \begin{cases} \frac{m(\boldsymbol{I}_n, A_i)}{\mu \cdot C} & , m(\boldsymbol{I}_n, A_i) < C \\ 1 & , \text{else} \end{cases} \tag{3.8}$$

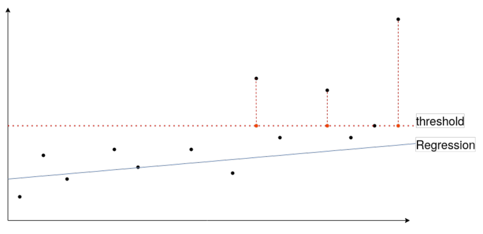
sets the performance of terminated algorithms is set to 1 and scales the performance of all other algorithms accordingly. Note that an increase of μ results in a decrease of performance of all algorithms that are not terminated.

The downside of this PAR-inspired approach is that it introduces a bias into the model, which is visualized in Figure 3.1. This figure shows how predictions change based on handling incomplete labels. To that end, there are two graphs Figure 3.1a and Figure 3.1b. In Figure 3.1a the algorithms are not stopped threshold. In Figure 3.1b the algorithms are stopped, and the model uses the threshold (with

$\mu = 1$) value instead. In Figure 3.1a the linear regression reaches significantly higher values than in Figure 3.1b.



(a) Linear Regression without Runtime Threshold C



(b) Linear Regression with Runtime Threshold C

Fig. 3.1.: Comparison of Linear Regressions that are built on performance information. In Figure 3.1a all algorithms are run till they finish their calculation. In Figure 3.1b algorithms that have not finished their calculation are stopped at a threshold C. This comparison shows that setting censored labels to a fixed value introduces a huge bias into a model.

The difference in the regression shows the bias introduced into a model if incomplete performance data is handled as explained above. Nevertheless, models which set those values to $\mu \cdot C$ are easily trained and perform reasonably well and are therefore used in this thesis for the hybrid decision trees introduced in the following section.

3.3 Binary Decision Trees

The explained above dataset $\mathcal{D}$ is then used to evaluate the quality of our model. To this end, the dataset is split into two parts. $\mathcal{D}_{train}$ is used to drain the model, and $\mathcal{D}_{test}$ is used to make performance predictions. In this thesis, we introduce a new model "hybrid decision tree " based on a standard Binary Decision Tree.
Binary Decision Trees have been used as a model in machine learning for various applications in the past [Bre+84; CHH09; Hut+15; RM07]. They are one of the most versatile and best-studied approaches in Data Science. This thesis will explore their use with a combined ranking and regression loss.
In the following paragraphs, we will discuss three points:

1. Basic Functionality of Decision Trees (section 3.3.1)

2. The process of building a tree (section 3.3.2)

3. The process of making a prediction (section 3.3.2)

3.3.1 Basic Functionality of Decision Trees

Binary decision trees are based on the idea of recursively dividing a dataset $\mathcal{D}$ into subsets $\mathcal{D}^+, \mathcal{D}^-$. The process of building a binary decision tree and utilizing a binary decision tree for predictions is explained in the following two paragraphs.

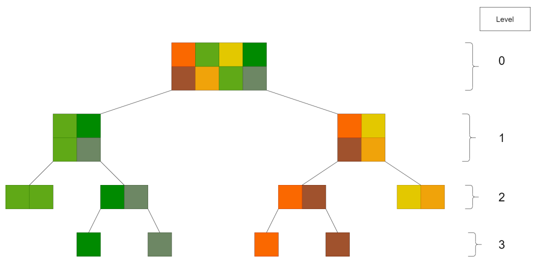


Fig. 3.2.: Binary Decision Tree - without features

Fundamental explanation of Binary Decision Trees In Figure 3.2 the process of building a tree is shown. Each rectangle represents one instance of the training dataset, whereas the rectangle's color represents the instance's label. With each level, the set of rectangles is split into two subsets of similarly colored rectangles. This splitting is repeated until a level of homogeneity is reached. This can either mean that there are only rectangles of the

same color or that the colors are very similar.
On each level, there are various sets of rectangles, called nodes. Given one node N, the rectangles of which it consists of are called the dataset of a node N and referred to as $\mathcal{D}_N$. The nodes that result from a split are the roots of their sub-trees.

Binary Decision Trees as a Machine Learning Model However, the above scenario was solely based on the colors, which is a simplification. In contrast to Figure 3.2, machine learning models do not work exclusively based on the labels. Instead, the split is chosen based on a set of features left out in Figure 3.2. There are three features utilized for each instance:

1. number $\in \mathbb{R}$
2. number even or odd $\in \{0.1\}$
3. number divisible by 4 $\in \{0.1\}$

The same tree is shown again in the Figure 3.3, but features and split points are also included. For simplicity, not all features are shown in the figure (features 2 and 3 can be calculated from feature 1).

If one wants to use a Decision tree to predict a label for a new instance I, one propagates the new instance through the tree starting at the root. With each level, one evaluates whether a leaf node is reached, in which case the label would be returned as a prediction. In Figure 3.3 the prediction would be a mixture of all the colors if a leaf node reached. If the current node is not a leaf node, one determines whether I belongs in the left or right sub-tree based on the split chosen for the dataset.

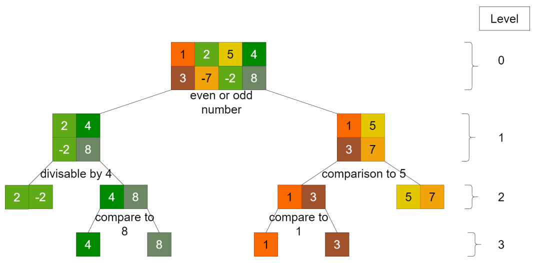


Fig. 3.3.: Binary Decision Tree - with features

3.3.2 Pseudocode

Building of a Binary Decision Tree From a Dataset

As explained earlier, the building procedure for binary decision trees is based on a recursive split of a dataset. This process is shown in more detail in Algorithm 1. The procedure calls in this algorithm will be explained further in the following sections.

Algorithm 1 build_decision_tree(self, $\mathcal{D}$)

```
1: $self.\mathcal{D} \leftarrow \mathcal{D}$
2: $self.label \leftarrow$ calculate_node_label($\mathcal{D}$)
3: **if** $\neg$ stopping_criterion($\mathcal{D}$) **then**
4:     $self.c, self.f \leftarrow$ find_splitting_criterion($\mathcal{D}$)
5:     $\mathcal{D}^+, \mathcal{D}^- \leftarrow$ split($\mathcal{D}, self.c, self.f$)
6:     $self.left\_child \leftarrow$ build_decision_tree($\mathcal{D}^-$)
7:     $self.right\_child \leftarrow$ build_decision_tree($\mathcal{D}^+$)
8: **end if**
9: **return** self
```

After the dataset is assigned and the label calculated (line 2, section 3.4), one checks whether the stopping criterion is reached (line 3, section 3.7). This determines whether or not the node self is a leaf and is the termination condition of the algorithm.
If this stopping criterion is not reached, the splitting criterion is calculated (line 4). This means that a routine is started that compares all possible ways to split the dataset (every feature, every splitting point), and the best (split according to a loss function introduced in section 3.5 is selected). This split is then saved as self.c (split point) and self.f (feature). Then the dataset is divided (all instances that have a value $< self.c$ at feature f are in the left sub-tree, all other instances in the right), and the recursive algorithm builds the left and right sub-trees (lines 5-7).
In the end, the binary tree is returned. Due to the recursive nature of this algorithm, an entire tree is built by this simple procedure.

Predicting with a Binary Decision Tree

Algorithm 2 predict(tree, $\boldsymbol{I}$)

```
1: if ¬ has_children(tree) then return $tree.label$
2: else
3:     $subtree \leftarrow$ select_subtree($tree.c, tree.f, \boldsymbol{I}$)
4: end if
5: return predict($subtree, I$)
```

When predicting a label for a new instance I, one starts at the root node and propagates it through the tree. To that end, the procedure Algorithm 2 first checks whether the given tree has any children or is a leaf (line 1). If it is a leaf, the label of the tree is returned. Otherwise, whether the instance $\boldsymbol{I}$ belongs in the left or right sub-tree is checked. If $\boldsymbol{I}$ at feature $self.f$ is smaller than $self.c$ the left sub-tree is selected. Otherwise, the right sub-tree is selected (line 3). Lastly, the algorithm is called recursively on the correct sub-tree, starting at either the left or the right child.

Short Summary

As seen above, the general concept of binary decision trees is adaptable to many different configurations. In addition to quantifying different splitting points with a convex combination of a ranking and regression loss function to find the best split, one can also utilize different node labels, stopping criteria,... Thus it makes sense to evaluate the performance of varying tree configurations. The binary decision tree specified throughout the following sections is called hybrid decision tree.

3.4 Node Label

When calling $predict$ on a hybrid decision tree with an instance I, the instance is propagated through the tree until a leaf node N is reached. Then the label of N is returned. This label is calculated as a representation of all the instances in $\mathcal{D}_N$.

To calculate the label of a node N, one aggregates the labels of all instances in the node's dataset $\mathcal{D}_N$ to one consensus label. In this thesis, labels are aggregated by algorithm-wise averaging the algorithm performance of the instances.

Let $\mathcal{D}_N$ be the set of instances and labels node N. Then, we define Y_N as the consensus label.

$$Y_N := (Y_{N_1}, Y_{N_2}, ..., Y_{N_k}) \tag{3.9}$$

Here the values

$$Y_{N_i} := \frac{1}{|\mathcal{D}_N|} \sum_{(\boldsymbol{I}_n, \boldsymbol{y}_n) \in \mathcal{D}_N} m_i'(\boldsymbol{I}_n) \tag{3.10}$$

represent the average performance of A_i on the instances in $\mathcal{D}_N$.

With this consensus label, one can predict both algorithm performances and an overall ranking.

3.5 Splitting Criterion

When choosing a splitting criterion for the dataset, one needs to consider two different problems. Firstly, we need to select a feature f to be split on, and secondly, we need to find a concrete splitting value c. We calculate the best split on each feature separately by iterating over all possible splitting criteria. The best split is then chosen from the list of best splits for each feature.

However, one does not include splitting points that would lead to unreasonable small datasets. This would lead to overfitting, and the depth of the tree would increase drastically. To that end, a parameter $min\_samples\_leaf$ is introduced that decreases the number of possible splitting points as points that would result in only $x < min\_sample\_leaf$ instances in one node are not considered.

To quantify the quality of a split of $\mathcal{D}_N$ it is divided into two subsets $\mathcal{D}_N^-$ and $\mathcal{D}_N^+$ that are supposed to be more homogeneous than $\mathcal{D}_N$. The homogeneity of the resulting datasets is quantified by a loss function $\mathcal{L}$ (further explained in section 3.6). This implies that calculating the homogeneity loss for both resulting datasets as a way to quantify the quality of a split. A loss function

$$\mathcal{L}_{split}^{\mathcal{D}_N} : \mathbb{R} \times F \to \mathbb{R} \tag{3.11}$$

that measures this homogeneity of a split is based on quantifying the homogeneity of the datasets that result from a given split at a feature F and a concrete splitting point in $\mathbb{R}$.

The first intuition of a split loss

$$\mathcal{L}_{split}^{\mathcal{D}_N}(c, f) = \mathcal{L}(\mathcal{D}^-) + \mathcal{L}(\mathcal{D}^+) \qquad \mathcal{D}^-, \mathcal{D}^+ \text{are the resulting partitions} \tag{3.12}$$

has the downside that even if the resulting datasets are of uneven size they have the same impact on the overall loss.

Therefore, the loss of a split

$$\mathcal{L}_{split}^{\mathcal{D}_N}(c, f) = \frac{|\mathcal{D}^-|}{|\mathcal{D}|}\mathcal{L}(\mathcal{D}^-) + \frac{|\mathcal{D}^+|}{|\mathcal{D}|}\mathcal{L}(\mathcal{D}^+) \tag{3.13}$$

is defined as the weighted sum of losses of the resulting datasets $\mathcal{D}^+$ and $\mathcal{D}^-$

3.6 Quantifying the Homogeneity of a Node's Dataset

As mentioned in section 3.5 to determine the best split of a dataset, one utilizes a loss function $\mathcal{L}$ that quantifies the homogeneity datasets that result from said split. This node-wise loss consists of a separate ranking loss $\mathcal{L}_{rank}$ and regression loss $\mathcal{L}_{regr}$. These losses utilize the basic ranking and regression loss functions.

To comprehensively explain this process, we explain the components in reversed order. We first introduce the structure of basic loss functions (section 3.6.1). Then regression loss functions (section 3.6.2), and ranking loss functions (section 3.6.3) that are utilized by $\mathcal{L}_{rank}$ and $\mathcal{L}_{regr}$ are introduced. Building on that, it is explained how those loss functions are used to measure the homogeneity of a dataset in terms of ranking $\mathcal{L}_{rank}$ and regression $\mathcal{L}_{regr}$ (section 3.6.4). Lastly, it is explained how these homogeneity losses are combined to one loss $\mathcal{L}$ (section 3.6.5).

3.6.1 Basics on Loss Functions

In essence, loss functions are used to evaluate the error of a prediction. Given a ground truth (correct) label $y \in Y$ and a predicted label $\hat{y} \in Y$, a loss function

$$\ell : Y \times Y \to \mathbb{R}_0^+ \tag{3.14}$$

assigns a value that quantifies the prediction's quality. Depending on the space of possible labels, Y one deals with different kinds of machine learning models. As explained earlier, this thesis focuses on a combined approach of ranking and regression. In regression, it holds that the labels are real numbers.

$$Y = \mathbb{R} \tag{3.15}$$

The space of labels Y equals the set of possible rankings over k labels in label ranking. This can be formalized as the set of all permutations over k labels. Since one is interested in a ranking of k algorithms, the labels are algorithms in this thesis. If one were to compare the performance of three algorithms A_1, A_2 and A_3 on a problem instance I, a possible ranking τ_I would be

$$A_1 \succ_I A_2 \succeq_I A_3 \tag{3.16}$$

and the space of possible rankings would be all possible orders of A_1, A_2, and A_3. It is important to note that two algorithms might also be tied in performance.

3.6.2 Regression Loss Function

One of the most basic loss functions for regression is the squared error [Bot18]

$$\ell : \mathbb{R} \times \mathbb{R} \to [0, 1] \tag{3.17}$$

$$\ell(y, \hat{y}) = (y - \hat{y})^2 \tag{3.18}$$

which is the squared difference of the ground truth value y and a predicted value $\hat{y}$. This is the only regression loss we consider in this thesis. However, since our training data has the structure described in Equation 3.4, one can not directly apply this loss in the context of hybrid decision trees (section 3.6.4).

3.6.3 Ranking Loss Function

Ranking losses are less intuitive than regression losses. Therefore, the explanation of ranking losses is divided into three parts:

1. The label ranking notation is introduced.
2. An example ranking loss is given.
3. The ranking losses utilized for hybrid decision trees are explained.

Label Ranking Notation

As mentioned above, we deal with label ranking in this thesis. Here one transforms the label of each instance into a ranking. The main idea is that given a set of labels (algorithms), one orders these labels (algorithms) according to some order (the performance measure $m_i'(I_n)$), where the label (algorithm) with the i-th highest rank is the i-th best label (i-th best-performing algorithm on I_n).
As a ranking loss disregards exact performance information, a separate ranking notation based on permutations is introduced as a level of abstraction. These ranking notations transform the problem into a ranking problem. This leads to a simplification in notation as one does not deal with performance information.
Given a node N we have a dataset $\mathcal{D}_N \subseteq \mathcal{I} \times \mathbb{R}^m$. From $(\boldsymbol{I}_n, \mathbf{y}_n) \in \mathcal{D}_N$, one derives a ranking r_i of algorithms by ordering them by Y_n from highest to lowest performance. This ranking can be formalized as a permutation τ_n over the algorithms

$$\mathcal{A} = \{A_1, ..., A_k\} \tag{3.19}$$

for each instance I_n. Here $\tau_n(l) = j$ means that A_l has the jth best performance on instance I_n [Hül+08].

Tie Handling in Ranking Calculation However, there might be algorithms $A_i, A_i + 1, ..., A_j$ for which it holds $m'_i(I_n) = m'_{i+1}(I_n) = ... = m'_j(I_n)$ for some problem instance I_n. Let A_{i-1} be the worse performing neighbor of them and A_{j+1} be the better performing neighbors then, the rank of the tied algorithms is

$$\tau(A_i) = \tau(A_j) = \frac{|\tau(A_{j+1}) - \tau(A_{i-1})|}{j - i} \tag{3.20}$$

set to the mid rank of the tied out algorithms since this is expected to give the best quality of models [Tor+22]. This is illustrated by the following example: Given 4 Algorithms $A_1, ..., A_4$ with performances

$$\begin{aligned} m'_1(I_n) &= 1 \\ m'_2(I_n) &= 7 \\ m'_3(I_n) &= 7 \\ m'_4(I_n) &= 12 \end{aligned}$$

on some instance I_n the resulting ranking of algorithms would be:

$$\begin{aligned} \tau_n(1) &= 1 \\ \tau_n(2) &= 2.5 \\ \tau_n(3) &= 2.5 \\ \tau_n(4) &= 4 \end{aligned}$$

However, we introduce an exception to the method of tie handling: If the tied algorithms have the worst performance, they are not set to the value explained above, but to the worst possible ranking. This means that if one modified to above example to have the performances

$$\begin{aligned} m'_1(I_n) &= 1 \\ m'_2(I_n) &= 12 \\ m'_3(I_n) &= 12 \\ m'_4(I_n) &= 7 \end{aligned}$$

the calculated ranking would be

$$\begin{aligned}\tau_n(1) &= 1\\ \tau_n(2) &= 4\\ \tau_n(3) &= 4\\ \tau_n(4) &= 2\end{aligned}$$

This is done since creating the ranking as explained in the beginning would result in

$$\begin{aligned}\tau_n(1) &= 1\\ \tau_n(2) &= 3.5\\ \tau_n(3) &= 3.5\\ \tau_n(4) &= 2\end{aligned}$$

A_2, and A_3 getting a ranking position that is better than the worst possible ranking. This is unintended as a tie in ranks for the worst performing algorithms is likely caused by the algorithms not finishing their calculation before the cutoff threshold C is reached.

Example Ranking Loss

As explained in section 3.6.1, loss functions are based on the idea of quantifying the difference between a correct ranking y and a predicted ranking $\hat{y}$. For example if one has 2 rankings

$$\tau_1 = (a \succ b \succ c) \tag{3.21}$$

(3.22)

and

$$\tau_2 = (c \succ a \succ b) \tag{3.23}$$

and uses the minimal number of transpositions as a loss loss function that measures of similarity of the rankings, the loss $\ell(\tau_1, \tau_2)$ is 2.

Ranking Error Loss Functions

As mentioned above, there are ranking error loss functions that are loosely based on the idea of calculating a distance of two rankings $\tau, \hat{\tau}$. This is done by exploring the difference between the ranks $\tau(i)$, $\hat{\tau}(i)$ for all candidate set algorithms $\mathcal{A}$ [HF10]. The ranking errors considered in this thesis include the Spearman's rank correlation

$$\ell : P \times P \to \mathbb{N} \tag{3.24}$$

$$\ell(\tau, \hat{\tau}) = \sum_{i=1}^{m} (\tau(i) - \hat{\tau}(i))^2 \tag{3.25}$$

and the Spearman's foot rule

$$\ell : P \times P \to \mathbb{N} \tag{3.26}$$

$$\ell(\tau, \hat{\tau}) = \sum_{i=1}^{m} |(\tau(i) - \hat{\tau}(i))| \tag{3.27}$$

which are the sum of absolute/squared rank distances. However, we use a slightly modified version of these errors that are scaled to $[0, 1]$ to reach similar behavior in our loss functions (section 3.5).
For the spearman rank correlation, we use

$$\ell(\tau, \hat{\tau}) = \sum_{i=1}^{m} \frac{(\tau(i) - \hat{\tau}(i))^2}{(k-1)^2} \tag{3.28}$$

and for the spearman footrule, we use

$$\ell(\tau, \hat{\tau}) = \sum_{i=1}^{m} \frac{|(\tau(i) - \hat{\tau}(i))|}{k-1} \tag{3.29}$$

as two rankings $\tau(i)$ and $\hat{\tau}(i)$ differ by at most $k - 1$ since 1 is the best possible rank and k is the worst possible rank.

Another loss function used in this thesis is the number of discordant pairs [Agr10] of the Kendalls Tau Measure

$$
\begin{aligned}
&\ell : P \times P \to \mathbb{N} && (3.30)\\
&\ell(\tau, \hat{\tau}) = |U| && (3.31)\\
&U = \{(i,j) : 1 \leq i < j \leq k, \qquad \tau(i) < \tau(j) \wedge \hat{\tau}'(i) > \hat{\tau}'(j) && (3.32)\\
&\qquad\qquad\qquad\qquad\qquad\quad \vee \tau(i) > \tau(j) \wedge \hat{\tau}'(i) < \hat{\tau}'(j)\} && (3.33)
\end{aligned}
$$

that can easily be scaled to $[0, 1]$

$$\ell(\tau, \hat{\tau}) = \frac{|U|}{\frac{k \cdot (k-1)}{2}} \tag{3.34}$$

by dividing $|U|$ by the amount of considers pairs (i, j).

The downside of these loss functions is that they disregard the entire runtime information given in the context of algorithm selection. This problem has already been explained in chapter 1 and motivates the usage of ranking loss functions that consider the difference in runtime.

Squared Hinge Ranking Loss The following distance-based loss we consider is the squared hinge ranking loss [Han+20]. This loss function combines the comparison of ranking with runtime information. To that end, a new function v_i

$$v_i : \mathcal{I} \to \mathbb{R} \tag{3.35}$$

is introduced with $v_i(I_n)$ resembling the performance $m_i'(I_n)$ given by the training dataset of A_i on an instance I_n. For this v_i we use the negative runtime:

$$v_i(I_n) = -m_i'(I_n) \tag{3.36}$$

The loss is then defined as

$$
\begin{aligned}
&\ell : P \times P \to \mathbb{R} && (3.37)\\
&\ell(\tau, \hat{\tau}) = \frac{1}{\Omega} \sum_{(i,j):\tau(i)<\tau(j)} l(\epsilon - (v_i(I_n) + v_j(I_n))) && (3.38)\\
&l(x) = (\max(0, x))^2 && (3.39)
\end{aligned}
$$

where Ω is the number of pairs (i,j) over which the sum is calculated.
The intuition of this distance metric is to iterate over all pairs (i,j) where A_i should perform better than A_j according to τ. If this is the case, the function l will yield 0. However, if this is not the case, it would yield a positive value. The value ϵ is introduced as without it equal performance of $v_i(I_n)$ and $v_j(I_n)$ would not be penalized even though it should be as $\tau(i) < \tau(j)$.

Position Error Loss Function The position error is based on the idea of measuring the difference between two rankings $\tau, \hat{\tau}$ solely based on the best ranked algorithms in τ and $\hat{\tau}$. To that end, let ω be the index of the best-ranked algorithm A_ω in τ_c (note τ is the ground truth ranking)

$$\omega := \tau^{-1}(1) \tag{3.40}$$

then the Position error is

$$\ell(\tau, \hat{\tau}) = \hat{\tau}(\omega) - 1 \tag{3.41}$$

which can be rewritten as

$$\ell(\tau, \hat{\tau}) = \hat{\tau}(\omega) - \tau(\omega) \tag{3.42}$$

This error can be modified so that it also takes the performance of algorithms into account. This results in the loss

$$\ell : \mathbb{N} \times \mathbb{N} \to [0, 1] \tag{3.43}$$

$$\ell(\tau, \hat{\tau}) = Y_{N_{\hat{\tau}(\omega)}} - Y_{N_{\tau(\omega)}} \tag{3.44}$$

where Y_{N_i} is the average performance of A_i on on the instances that are in the dataset that results of the split.

3.6.4 Utilizing Losses to Quantify the Spread of Labels

Since our instance labels are more complicated than a real number, and losses are used to measure the homogeneity of a dataset that results from a split, loss functions as simple as the squared error (Equation 3.17) or spearman rank correlation (Equation 3.25) are not directly applicable to the previously introduced binary decision

trees.
Instead, the given losses are not used to evaluate the homogeneity of a dataset. One can view this as fitting a local model on a resulting dataset and then using the loss functions $\mathcal{L}$ to quantify the quality of a prediction made by the local model.

Application of Regression Losses

In the case of regression, one uses the node label from section 3.4 as the model we fit on and then calculates the average squared error (Equation 3.17)

$$\mathcal{L}_{MSE_i} : \mathbb{R}^k \times \mathbb{R}^k \to [0,1] \tag{3.45}$$

$$\mathcal{L}_{MSE_i}(Y, \hat{Y}) = \frac{1}{k} \sum_{i \in \{1,\ldots,k\}} (Y_i - \hat{Y}_i)^2 \tag{3.46}$$

over the set of algorithms $\{A_1, ..., A_k\}$. $\hat{Y}$ denotes the node label (section 3.4) and Y one instance's label.

Lastly, we average over all of these errors

$$\mathcal{L}_{regr}(Y_{\mathcal{D}_N}, \hat{Y}_N) = \frac{1}{|\mathcal{D}_N|} \sum_{Y \in Y_{\mathcal{D}_N}} \mathcal{L}_{MSE_i}(Y, \hat{Y}) \tag{3.47}$$

with $Y_{\mathcal{D}_N}$ denoting the set of labels in $\mathcal{D}_N$. This results in the regression loss used in this thesis. The same was is used in "Hybrid Ranking and Regression for Algorithm Selection" (section 2.2.4).

Application of Ranking Losses

As mentioned in section 3.6.4, the ranking and regression losses are calculated separately for both datasets that result from a split. One can view quantifying a dataset's homogeneity by first fitting a local model and then calculating the average loss. This is realized by calculating a consensus ranking from the rankings in the resulting dataset and then quantifying the quality of this consensus label as a representative label for the dataset with the ranking loss functions introduced in

section 3.6.3.
In this thesis, we calculate the consensus ranking with Borda's Method.

Bordas's Method

The general idea of Borda's method for rank aggregation [Lin10] is, given ranking permutations $\tau_1, ..., \tau_k$, one calculates a Borda Score B_l for each algorithm $A_l \in \mathcal{A}$ that represents A_l's performance on on the dataset. Then one orders those resulting values to get a new ranking over $B_1, ..., B_k$. We call this ranking consensus ranking. The calculated consensus ranking is referred to as τ_c, and the space of permutations for a node N is called P_N.

The critical part of this process is the calculation of Borda Scores. One can do this in many different ways. The basic idea is that we introduce a function f

$$f : \mathbb{R}^{|\mathcal{D}_N|} \to \mathbb{R} \tag{3.48}$$

that maps the performance of one algorithm on all instances to a single value. The resulting value is then called Borda Score.

$$B_l := f(\tau_1(l), \tau_2(l), ...\tau_{|\mathcal{D}_N|}(l)) \tag{3.49}$$

Possible functions f would be the following:

$$f(\tau_1(l), ..., \tau_{|\mathcal{D}_N|}(l)) = median(\tau_1(l), ...\tau_n(l)) \tag{3.50}$$

$$f(\tau_1(l), ..., \tau_{|\mathcal{D}_N|}(l)) = avg(\tau_1(l), ...\tau_n(l)) \tag{3.51}$$

$$f(\tau_1(l), ..., \tau_{|\mathcal{D}_N|}(l)) = (\prod_{i=1}^{|\mathcal{D}_N|} \tau_i(l))^{\frac{1}{|\mathcal{D}_N|}} \tag{3.52}$$

$$f(\tau_1(l), ..., \tau_{|\mathcal{D}_N|}(l)) = \sum_{i=1}^{|\mathcal{D}_N|} m'_l(I_i) \tag{3.53}$$

In the case of tied Borda Scores, these ties are handles as explained in paragraph 3.6.3.

Calculation of the Ranking Loss The resulting ranking loss

$$\mathcal{L}_{rank} : P^{|\mathcal{D}_N|} \times P \to \mathbb{R} \tag{3.54}$$

with $P^{|\mathcal{D}_N|}$ as the set of all possible rankings over $\mathcal{D}_N$'s instances. The loss is then calculated averaging over the instance-wise losses (with one loss function from section 3.6.3) of all instances from the dataset.

$$\mathcal{L}_{rank}(P_N, \tau_c) = \frac{1}{|P_N|} \sum_{\tau \in P_N} \ell(\tau, \tau_c) \tag{3.55}$$

as the average distance of rankings τ to τ_c.

3.6.5 Combined Ranking and Regression Split Loss

An intuitive approach to quantify the homogeneity of a node's dataset is combining the given ranking and regression losses (section 3.6.4). To to that the ranking and regression losses are calculated separately. That results in:

$$\mathcal{L}(D_N) = \mathcal{L}_{rank}(Y_{\mathcal{D}_N}, \hat{Y}_N) + \mathcal{L}_{regr}(P_N, \tau_c) \tag{3.56}$$

However, that would implicitly follow the assumption that we want ranking and regression to have the same impact on the loss. Since this might not be the case, we introduce a parameter λ and build a convex combination

$$\mathcal{L}(D_N) = \lambda \cdot \mathcal{L}_{rank}(Y_{\mathcal{D}_N}, \hat{Y}_N) + (1 - \lambda) \cdot \mathcal{L}_{regr}(P_N, \tau_c) \tag{3.57}$$

with a $\lambda \in [0, 1]$. Then bigger (smaller) values of λ lead to a bigger (smaller) impact of the ranking error making our system more flexible [Han+20].

However, one needs to be careful in choosing the loss functions as if one were to choose an arbitrary popular loss function, their behavior may impact $\mathcal{L}(D_N)$ in an unintended way [Han+20] since losses that reach different co-domains might outperform one another (cf. section 5.3). Due to this, all loss functions utilized in this thesis are scaled to the interval [0,1].

3.7 Stopping Criterion

There exists a wide variety of stopping criteria one can choose. Some focus on the entire tree (like stopping when a certain depth is reached), and some are defined node-wise. The most trivial of such stopping criteria is stopping once a node N whose dataset has only one element is reached. However, this would lead to needlessly large decision trees and overfitting. As a result, one aims to have more than one instance in a leaf node. Therefore, we will evaluate the following stopping criteria:

1. Given the performance y_i, y_j of two algorithms $A_i, A_j \in \mathcal{A}$, all instances rank the performance either $y_i \succeq y_j$ or $y_j \succeq y_i$.

2. Given the performance y_i, y_j of two algorithms $A_i, A_j \in \mathcal{A}$, at least $x\%$ of the instances rank the performance either $y_i \succeq y_j$ or $y_j \succeq y_i$.

3. The maximal depth of the tree is n for some $n \in \mathbb{N}$. However, a node that is at depth n will not be split further. Note that two hybrid decision trees one with max depth n and one with max depth $n + 1$ with otherwise equal configuration have the same structure up to depth n.

4. The dataset reaches a certain homogeneity. To that end, one calculates the loss of $\mathcal{D}_N$ as explained in Equation 3.57 and if it is under a threshold X, the stopping criterion is reached.

Additionally, a new parameter min_sample_split is introduced. Nodes that hold less than min_sample_split instances are not split further to avoid unreasonably small datasets.

4 Goals and Research Questions

This thesis explores how well the approach of hybrid decision trees performs compared to other algorithm selection approaches. To that end build hybrid decision trees on top of the evaluation framework used in the evaluation of survival forests [Tor+20] and then discuss the performance of hybrid decision trees. Additionally we compare the performance of hybrid decision trees to other hybrid models from “Hybrid Ranking and Regression for Algorithm Selection” [Han+20].

4.1 Mandatory Goals

The first goal of the thesis was the implementation of hybrid decision trees. These are implemented hybrid decision trees as a versatile model that is shown in Figure 4.1.

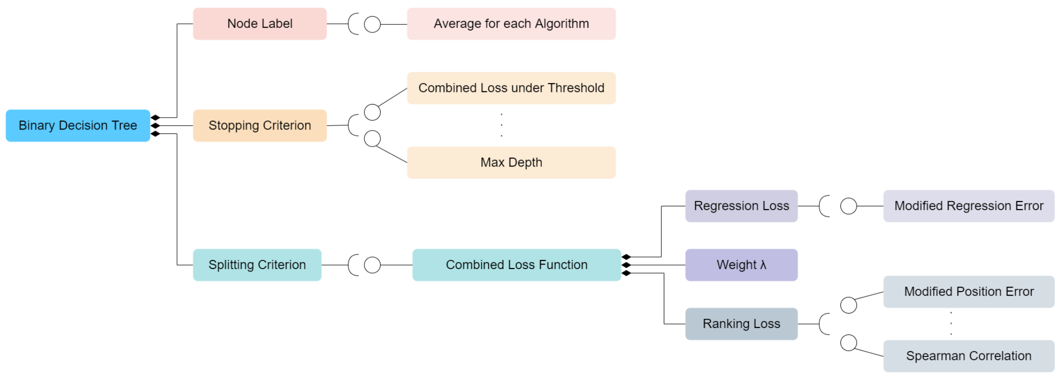


Fig. 4.1.: Structure of the hybrid decision trees, whose implementation is a goal of this thesis. On the left side of the figure is the hybrid decision tree. With each step to the right different components are introduced. These components are always part of the component to their left and can be split into different subcomponents themselves.

On the left side one can see that a hybrid decision tree needs three components:

1. a node label
2. a stopping criterion
3. a splitting criterion

While all of these components can be implemented in various ways we fixed one node label (section 3.4). The other components (stopping criterion and splitting criterion) offer different configraution options. The stopping criterion is not further decomposed but the splitting criterion is build from three components

- Regression Loss
- Ranking Loss
- λ

that determine the loss of a split.

In addition to the binary decision tree, a data preprocessing was implemented that deals with incomplete labels as described in Equation 3.4. Here it was made possible to use different values for μ.

4.1.1 Evaluation of Hybrid Decision Trees

After completing the implementation as described above, there are various possible configurations of decision trees. To find well-performing configurations the next goal was an evaluation of different component configurations. To that end the data is split into a training dataset $\mathcal{D}_{train}$, and a testing dataset $\mathcal{D}_{test}$ (The process of training and testing a model is is further explained in section 5.1.). For evaluation, one uses the trained model to predict labels for all instances in $\mathcal{D}_{test}$. Then the quality of these predictions is quantified with the metrics below.

Kendall's Tau - b

The idea of Kendall's Tau has already been introduced in 3.6.3. However, this explanation was very informal and reduced to the number of discordant pairs. The Kendall's Tau-b [Agr10] introduced here is a modification of the standard Kendall's Tau that also works well with tied datapoints. The calculation of calculating Kendall's tau - b measure for two rankings (that can be calculated from the label given by the dataset $\mathcal{D}_{test}$ and the predicted label as explained in section 3.6.3) can best be described in 2 steps: One starts by extracting the pairs

$$P = \{(i, j) \in \mathbb{N}^2 : 1 \leq i < j \leq k\} \tag{4.1}$$

that are used to compare the two rankings τ, $\hat{\tau}$.
Then the concordant pairs C and discordant pairs D are collected and counted.
The concordant pairs are all pairs

$$\begin{aligned} C = \{(i, j) \in P : & \hat{\tau}(i) < \hat{\tau}(j) \land \tau_(i) < \tau(j) \lor & (4.2) \\ & \hat{\tau}(i) > \hat{\tau}(j) \land \tau(i) > \tau(j)\} & (4.3) \end{aligned}$$

that are ordered in the same way in $\hat{\tau}$ and τ.
The discordant pairs are all pairs

$$\begin{aligned} D = \{(i, j) \in P : & \hat{\tau}(i) < \hat{\tau}(j) \land \tau(i) > \tau(j) \lor & (4.4) \\ & \hat{\tau}(i) > \hat{\tau}(j) \land \tau(i) < \tau(j)\} & (4.5) \end{aligned}$$

that are ordered differently in $\hat{\tau}$ and τ.

The Kendall's Tau - b is calculated as

$$\tau_b(\hat{\tau}, \tau) = \frac{|C| - |D|}{\sqrt{(|P| - t_{\hat{\tau}}) + (|P| - t_{\tau})}} \tag{4.6}$$

whereas $t_{\hat{\tau}}$ is the amount of algorithms tied in $\hat{\tau}$ and t_{τ} is the amount of algorithms tied in τ.
The values calculated are between -1 and 1. In more detail:

$\tau_b(\hat{\tau}, \tau) = -1$	rankings are inversely correlated
$\tau_b(\hat{\tau}, \tau) \in (-1, 0)$	the rankings have a tendency to order algorithms inversely
$\tau_b(\hat{\tau}, \tau) = 0$	the rankings are not correlated
$\tau_b(\hat{\tau}, \tau) \in (0, 1)$	the rankings have a tendency to order algorithms the same way
$\tau_b(\hat{\tau}, \tau) = 1$	the rankings are strongly correlated

For this metric we utilized the Kendall's Tau implementation of the well established scipy libary: "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python" [Vir+20]. However there are circumstances in which this metric does not return a valid result. To avoid compromised evaluation results we disregard the Kendall's Tau metric from further discussion. This decision is required since there was no time for another round of evaluation with another Kendall's Tau implementation that does not give the same problematic results.

Normalized Discounted Cumulative Gain (NDCG)

NDCG [Val+09] stands for normalized discounted cumulative gain. The idea is based on optimizing the solutions given by a search engine for a query but can also be used for other label ranking use cases. In comparison to Kendall's Tau the NDCG metric is stronger impacted by algorithms that have a high ranking than by algorithms that have a low ranking.
Let $\mathcal{I}_{test}$ be the set of instances used to evaluate the NDCG, which is computed as

$$NCDG(\mathcal{I}_{test}, M) = \frac{1}{|\mathcal{I}_{test}|} \sum_{I \in \mathcal{I}_{test}} \frac{Z(I, M(I))}{Z(I, \tau_I)} \tag{4.7}$$

$$Z(I, \tau) := \sum_{A \in \mathcal{A}} \frac{2^{m'_I(A)} - 1}{log_2(1 + \tau_I(A))} \tag{4.8}$$

with M(I) as the ranking predicted by M and τ_I beeing the correct algorithm ranking for the instance. Note that m' denotes the ground truth runtimes.

Solved Instances

Another critical measure of a model's prediction quality is the percentage of solved instances. An instance is solved if the best-predicted algorithm $A \in \mathcal{A}$ does not reach the timeout threshold in its calculation.

Performance Regret

Regret is a fundamental concept of machine learning. The underlying idea is that one measures the impact of a mistake made by a model $M \in \mathcal{M}$ with $\mathcal{M}$ being the space of all possible models [RN02]. Let $M(I_n)$ be the algorithm with the best-predicted performance according to the model M for an instance I_n and A^* be the best performing algorithm (given by $\mathcal{D}_{test}$) for I. Then the performance regret

$$R : \mathcal{M} \times I \times \mathcal{A} \to \mathbb{R}^+ \tag{4.9}$$

$$R(M, I, A^*) = (m_I^*(A^*) - m_I^*(M(I))) \tag{4.10}$$

is an indicator of the prediction quality. Note that m^* is the performance true performance. Therefore it is

$$m^* : \mathcal{I}, \mathcal{A} \to \mathbb{R}^+ \tag{4.11}$$

$$m^*(I, A) = \begin{cases} m(I, A) & , m(I_n, A_i) < C \\ \mu \cdot m(I, A) & \text{,else} \end{cases} \tag{4.12}$$

Par10

The popular par10 metric

$$PAR : M \times I \to \mathbb{R}^+ \tag{4.13}$$

is very similar to the performance regret. However it disregards the ground-truth best performance. Instead it introduces a parameter $x \in \mathbb{R}$ that quantifies performance decrease that results of feature calculation. This is based on the idea that if we compare the performance of an algorithm predicted by our model (which is already trained) with a randomly chosen algorithm, we should not disregard the effort of calculating the features needed to utilize this model. It is therefore calculated as:

$$PAR(M, I)) = \begin{cases} m(I, A) + x & m(I, A) + x < C \\ 10 \cdot C & \text{else} \end{cases} \tag{4.14}$$

4.1.2 Evaluation in Comparison to other Baseline Algorithms

The evaluation with said metrics is expected to show some configurations performing better than others. The resulting candidate configurations are then evaluated in comparison to other approaches (single best solver, survival forests, per algorithm regressor, linear models/quadratic models/neural networks trained with a convex combination of ranking and regression loss like Equation 3.57) mentioned in section 5.2.

4.2 Mandatory Research Questions

In chapter 6 the answers to the following research questions are given;

1. If one fixates all components but one (e.g., λ from Equation 3.57 or the ranking loss function section 3.6.3), which choice of the corresponding component performs best?

2. If one fixates all parameters/components but λ and the ranking loss, which combination of λ and the ranking loss function performs best?

3. Do those best-performing models outperform the other state-of-the-art approaches (mentioned in section 2.2.3 and section 2.2.4)?

4. Does our approach give a better solution than a standard regression tree [Bre+84]?

5. How high is the runtime of our new approach (both training and testing) compared to the other considered approaches in seconds?

Evaluation 5

The evaluation of hybrid decision trees is divided into five sections:

- In section 5.1 the process of evaluating the quality of a model is explained.
- In section 5.2 an overview of the baselines/state of the art models that hybrid decision trees are compared to is given.
- section 5.3 describes how preliminary results have led to adaptions to the initial Hybrid Binary Decision Trees specification (these adaptations are mentioned throughout the thesis). To that end, changes in the model specification are presented and evaluated.
- In section 5.4 the quality of different hybrid decision tree components is assessed.
- This knowledge is then used to propose hybrid decision tree configurations that perform well on average. These proposed configurations are evaluated in comparison to other approaches in section 5.5.

5.1 Evaluation Process

To answer whether or not hybrid decision trees are a well-suited model for algorithm selection, the quality of the model needs to be quantified. The evaluation is done by assessing the quality of predictions made by hybrid decision trees with an evaluation loss function (cf. in section 4.1.1). We treat the quality of predictions made by a hybrid decision tree and the quality of a hybrid decision tree as synonyms.

To give a comprehensive evaluation process overview, it is split into three sections:

- First, we provide information on the machine(s) used for evaluation (section 5.1.1).
- Then, the scenarios used to determine the quality of different models are introduced (section 5.1.2).
- Lastly, the process of quantifying the quality of a model is explained (section 5.1.3).

5.1.1 Virtual Machine(s) Used for Evaluation

To evaluate whether hybrid decision trees are a promising approach for algorithm selection, we utilized a set of 3 virtual machines that feature the Intel(R) Xeon(R) E5-2695 v3 @ 2.30GHz CPU with 16 cores and 64GB of RAM.

5.1.2 Choice of Utilized Scenarios

Since one model might perform better/worse depending on the problem domain, the quality of models is evaluated on scenarios from different problem domains. For this evaluation, we utilize scenarios from the repository associated with the paper "ASlib: A Benchmark Library for Algorithm Selection" [Bis+16].

Scenarios	# Instances	# Unsolved	# $\mathcal{A}$	# Features	C
ASP-POTASSCO	1294	82	11	138	600
CSP-2010	2024	253	2	86	5000
MAXSAT-15-PMS-INDU	601	46	29	37	2100
QBF-2016	1254	241	14	46	900
SAT12-Hand	767	229	31	115	1200
SAT12-INDU	1167	209	31	115	1200
CPMP-2015	527	0	4	22	3600

Fig. 5.1.: Overview of the properties of the used ASlib scenarios. Unsolved refers to instances where all algorithms are terminated due to exceeding the cutoff limit C. While SAT12-HAND and SAT12-INDU use the same algorithms and features, they use different instances.

.

The sets of scenarios in Figure 5.1 is chosen for two reasons: Firstly the selected scenarios cover a wide variety of problem domains. Secondly, they are selected because the datasets cover various properties such as a scenario's amount of instances, set of candidate algorithms, the set of candidate features, and the cutoff value C. Further scenario properties are disregarded here.

5.1.3 Quantifying a Model's Quality

As mentioned above, the evaluation process is divided into the training and testing of a model.
In addition, to this two-step process, one could utilize feature selection or other preprocessing algorithms that would increase a model's prediction quality [KKP]. These algorithms modify the feature space to evaluate models with well-suited features. An example of a preprocessing algorithm is disregarding features that have little information on the performance of an algorithm (e.g., if all instances have the same feature value).
We ignore these preprocessing algorithms as they introduces another layer of abstraction that increases the difficulty of analyzing a model's quality. Using them, one would need to answer whether the quality difference between the two models is rooted in their functionality or the modified feature space.

10-fold cross validation The data given by a scenario is transformed into training and testing datasets with 10-fold (k-fold) cross-validation [RN02]. Since the goal is to test a model's quality on each instance from the scenario, we need to have several rounds of training and testing (one does not want to test a model on an instance used for training). Therefore, there are are 10 (k) separate rounds of evaluation. In each of these rounds the dataset is split into one dataset for training $\mathcal{D}^i_{train}$ and one dataset for testing $\mathcal{D}^i_{test}$ ($i \in \{1, ..., 10\}$). The resulting trained model of round i is called M_i. It is tested with the instances from $\mathcal{D}^i_{test}$. The testing is done by querying M_i for predictions with every instance from $\mathcal{D}^i_{test}$. The quality of the predictions is then quantified the metrics mentioned in section 4.1.1. After 10(k) training rounds, the quantified quality of all used testing instances is averaged as the model's performance on the scenario.
In detail, this means that the scenario is randomly split into 10 folds $\mathcal{D}^1, ..., \mathcal{D}^{10}$ for separate rounds of evaluation. In the i'th round of evaluation the model M^i is

trained with $\mathcal{D}^i_{train} = \bigcup\limits_{\substack{j\in\{1,\ldots,10\}\\ i\neq j}} \mathcal{D}^j$ and evaluated with $\mathcal{D}^i_{test} = \mathcal{D}^i$. The quality of a model according to an evaluation function $\mathfrak{L}$ is then evaluated as:

$$\mathfrak{L}(M, \mathcal{D}) := \frac{1}{10}\sum_{i=1}^{n} \frac{1}{|\mathcal{D}^i_{test}|} \sum_{(\boldsymbol{I}_n, \mathbf{y}_n)\in\mathcal{D}^i_{test}} \mathfrak{L}(M(\boldsymbol{I}_n), \mathbf{y}_n)) \tag{5.1}$$

Here $\mathfrak{L}$ refers to one of the loss functions mentioned in section 4.1.

5.2 Baseline Overview

In the following sections, hybrid decision trees are compared to other approaches to determine whether or not/on which scenarios hybrid decision trees are a promising approach for the problem of algorithm selection. These baselines are the single best solver (sbs), the per algorithm regressor, survival forests, and preexisting models trained with a hybrid loss function. This section aims to give a basic understanding of their functionality.

5.2.1 Single Best Solver

For each dataset of a scenario, one could choose the algorithm with the best average performance for all instances in $\mathcal{D}_{train}$. This algorithm would be the single best solver (sbs).

$$sbs : \mathcal{I} \rightarrow \mathcal{A} \tag{5.2}$$

$$sbs(I_n) = \arg\max_{A\in\mathcal{A}} \sum_{I_n\in\mathcal{D}_{train}} m'(I_n, A) \tag{5.3}$$

This solver has the downside of not using any correlation between an instance's features and its performance. In fact, in chapter 1 we mentioned the gains that can be made by not using the sbs. Therefore, an approach for algorithm selection should only be considered if the evaluation results in the approach performs better than the single best solver.

To give a baseline for ranking, we use a generalization of the sbs that provides information on all algorithms. This generalized sbs provides a ranking based on the algorithms' average performances.

Note that sbs has the same functionality as hybrid decision trees of depth 0 since the node label of hybrid decision trees (section 3.4) consists of the average algorithm performances. A ranking is constructed from that label by ordering the average performances. One can view hybrid decision trees as an extension of the sbs since every leaf has the same functionality as the sbs.

5.2.2 Per Algorithm Decision Tree Regressor

In section 3.1 it is explained how one can utilize regression for algorithm selection. The per algorithm regressor is an implementation of this idea. One trains a separate regression tree [Bre+84] for each algorithm to make runtime predictions m'. Then one either generates a ranking from these predictions or the algorithm with the best prediction runtime. The model utilized here is a regression tree with the scikit-learn [Ped+11] implementation (which is an implementation of the approach proposed in *Classification and Regression Trees*).

5.2.3 Survival Forests

As mentioned in section 2.2.3, a survival forest consists of various survival trees. These can be configured in various ways. In this thesis, we limit our evaluation to a comparison with the expectation survival forest since all survival forests introduced in "Run2Survive: A Decision-theoretic Approach to Algorithm Selection based on Survival Analysis"[[Tor+20]] show similar performances.

5.2.4 Preexisting Hybrid Ranking and Regression Models

The preexisting hybrid ranking and regression models are either a linear model, quadratic model, or feed-forward neural network. These are trained with a convex combination of a regression loss (like ours) and a ranking loss. The paper that defined this approach is further explained in section 2.2.4.

5.3 Hybrid Binary Decision Tree Specification Adaptations

In the proposal's preliminary hybrid decision tree, neither scaling ranking losses (c,f, section 3.6.3) nor scaling the given data on the algorithm performances (c.f. Equation 3.7) was considered. Due to that, we faced the problem of the regression error and the ranking error reaching vastly different values. The degree of mismatch is dependent on the given scenarios since the regression error is dependent on e.g. the cutoff value C and the ranking error is dependent on e.g. the number of considered algorithms.

If one error reaches much larger values than the other, the loss function

$$\mathcal{L}(D_N) = \lambda \cdot \mathcal{L}_{rank}(Y_{\mathcal{D}_N}, \hat{Y}_N) + (1-\lambda) \cdot \mathcal{L}_{regr}(P_N, \tau_c) \tag{5.4}$$

might mainly be impacted by either the ranking or regression loss. If the loss is then evaluated for $\lambda \in \{0.1, .., 0.9\}$ but

$$x \cdot \mathcal{L}_{rank}(Y_{\mathcal{D}_N}, \hat{Y}_N) < \mathcal{L}_{regr} \quad \text{for some large x} \tag{5.5}$$

it can happen that the ranking loss does not impact the choice of split. In that case, we say that the regression loss overpowers/outperforms the ranking loss.
If a split is chosen based on both the ranking and the regression loss, the loss (quality of a hybrid decision tree is different than for $\lambda = 0$ and $\lambda = 1$) is called a true combined loss.

This mismatch is visualized in Figure 5.2. This figure shows the comparison the quality of two hybrid decision trees trained on 'ASP-POTASSCO' with the NDCG. It compares a hybrid decision tree trained without any scaling and a hybrid decision tree that uses a scaled ranking loss function and scaled performance data. The quality of both models is evaluated for $\lambda \in \{0, 0.1, ..., 0.9, 1\}$. The NDCG performance is on the y-axis, and different λ values are on the x-axis. As explained, different values of λ

should result in the model making different predictions. These different predictions would then result in the models reaching different scores. However, this is not the case for the unscaled spearman correlation. Here, the evaluated model quality is equal for $\lambda \in \{0, 0.1, ..., 0.7\}$. It only changes if $\lambda > 0.7$. This indicates that fitting a tree with the training dataset will result in the same tree for all other values of $\lambda < 0.7$ caused by the regression error overpowering the ranking error.
The hybrid decision tree trained with scaled performance data and a scaled loss function behaves differently. For every other λ, the evaluated quality changes. The general behavior indicates that the regression and ranking error reach comparable large numbers.

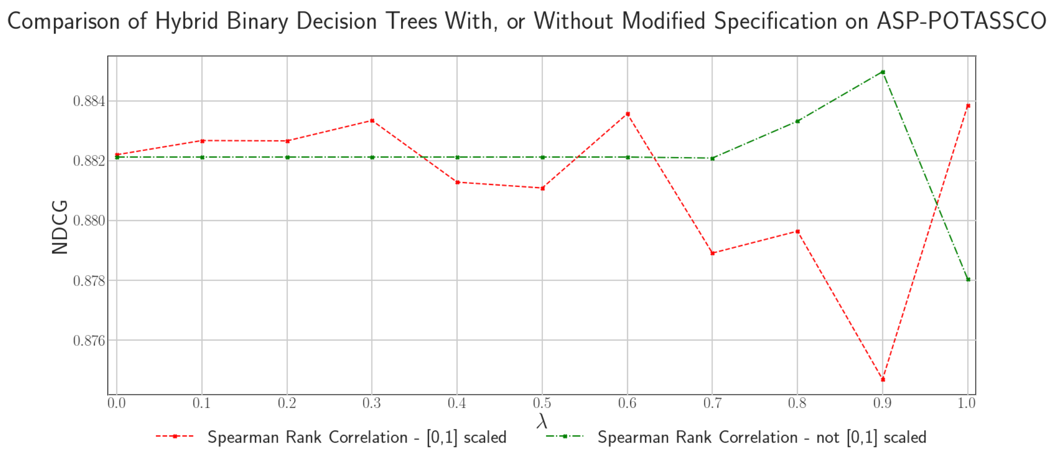


Fig. 5.2.: Comparison of two different hybrid decision trees. One tree is trained according to the preliminary specification. The other tree is trained with [0,1] scaled performance data and a [0,1] scaled loss functions.

The result of the regression error outperforming the ranking error is similar or worse for the following scenarios:

- QBF-2016
- ASP-POTASSCOf
- MAXSAT15-PMS-INDU
- SAT12-INDU
- CPMP 2015

From this observation, one can conclude that an evaluation of hybrid decision trees without the given modifications is not as viable to analyze the performance of hybrid decision trees.

Note that this issue only exists for loss functions that are not based on the performance of algorithms:

- spearman rank correlation
- spearman footrule
- umber of discordant pairs

Therefore, the modified position and squared hinge errors do not have the same problem. However, in this thesis it is not explored how hybrid decision trees with unscaled performance data perform for these losses.

All hybrid decision trees that are trained with the modified position error are not affected by the scaling of performance data. This is because the modified position error is solely based on performance data. Therefore, the regression and modified position errors use the same performance data. If one were to multiply the overall loss with $\mu \cdot C$, this factor can be propagated into the final simple losses introduced in section 3.6.1. Since $\mu \cdot C$ would be multiplied to all losses the best split would not be impacted by this..

5.4 Component-Wise Evaluation

As mentioned in section 4.1, there are various hybrid decision tree configurations possible that utilize the components introduced in chapter 3. To find well-performing configurations, we evaluate the quality of different components in an ablation study. This means that we compare the quality of various hybrid decision tree models that only differ in one component. This does not have to result in finding the best hybrid decision tree configurations since not every combination of components is tested. However, if we proceed this way, we can find viable candidate hybrid decision trees. The component wide discussion is split into the following sections:

1. First, the impact of different split loss configurations (introduced in section 3.6.4) have on the model is discussed. To that end, the quality of hybrid decision trees trained with different ranking losses and λ values is evaluated (section 5.4.1).

2. Then, it is studied which borda score is best to determine a consensus ranking (section 5.4.2).

3. Afterwards, we study how different stopping criteria impact the quality of hybrid decision trees (section 5.4.3).

4. Lastly, we study the impact of different μ values for censored labels in the data design (section 5.4.4).

5.4.1 Loss Evaluation

As mentioned above, there are different configuration options for hybrid decision trees. In this section, it is discussed how trees differ that are trained with different combinations of λ and $\mathcal{L}_{rank}$. The component choice mainly impacts the choice of the best splitting feature and point since the node-wise loss function

$$\mathcal{L}(D_N) = \lambda \cdot \mathcal{L}_{rank}(D_N) + (1 - \lambda) \cdot \mathcal{L}_{regr}(D_N) \tag{5.6}$$

is dependent on just λ, the the regression loss $\mathcal{L}_{regr}$, and $\mathcal{L}_{rank}$. To find the best configuration of $\mathcal{L}$ we evaluate different choices of λ and ranking losses. Note that for $\lambda = 0$ the hybrid decision tree solely uses the regression loss and for $\lambda = 1$ it solely uses the ranking loss.
To evaluate which combination of λ and $\mathcal{L}_{rank}$ performs best, we set all other components to a fixed configuration:

- The borda function is fixed to the arithmetic mean ranking.

- The stopping criterion is fixed to the maximal depth of 3.

- The timeout penalty is fixed to $\mu = 1$.

These components are chosen for their simple behavior. Additionally, we did not want to use the stopping criterion that is based on the node-wise loss since this is dependent on the loss as well.

Since the quality of models is dependent on the underlying scenario, scenarios for which hybrid decision trees are a good/poor fit are discussed separately. For this evaluation, we consider hybrid decision trees to be a good fit for a scenario if they outperform the single best solver for more than one choice of ranking loss and λ according to more than one evaluation metric.
For both good and bad scenarios, we discuss one scenario in detail. In the scenario-wise model discussion, we focus on the quantified performance with the metrics introduced in section 4.1. First, we use the metrics that are solely based on the best performing algorithm:

- Par10
- Performance regret
- Percentage of unsolved instances

Then, we discuss the quality of an entire ranking prediction with the NDCG metric. Lastly, we make conclusions on the overall model performance on the scenarios. This results in the structure shown in Figure 5.3.

performance / **...metric**	**trees perform well**	**trees perform poorly**
best algorithm...	page 48	page 52
entire ranking...	page 50	page 55

Fig. 5.3.: Structure overview over the loss evaluation.

The evaluation's results are visualized in figures of the structure shown in Figure 5.4. Each figure shows the quality of different hybrid decision tree configurations and the sbs based on one metric from section 4.1.1. The λ values are on the x-axis, and the measured quality according to an evaluation metric is on the y-axis. The graphs illustrate the performance of different hybrid decision tree configurations. They are shown in different colors. The lines in between evaluation points are just for visualization and do not imply a model's performance between evaluation points. In addition, to the performance of hybrid decision trees, the sbs baseline is shown as a thick line.

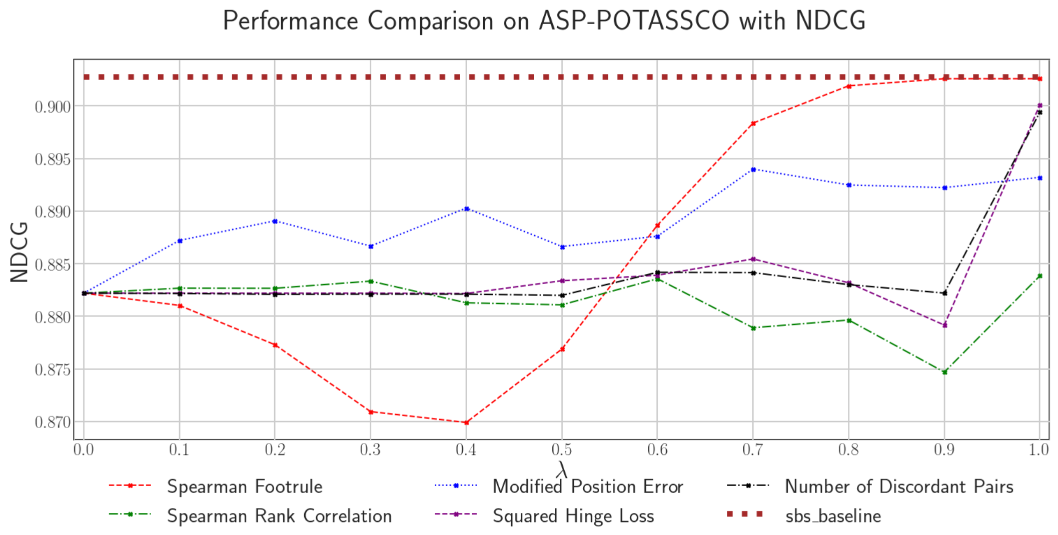


Fig. 5.4.: Example figure that compares the quality of hybrid decision trees trained with different ranking losses to the single best solver

Evaluation Results for Scenarios on which Hybrid Decision Trees Perform Well

As mentioned above, there are scenarios for which some hybrid decision tree configurations of $\mathcal{L}_{rank}$ and λ perform well in comparison to the single best solver (sbs). These scenarios are:

- CPMP-2015
- CSP-2010
- QBF-2016
- SAT12-INDU

The scenario that is chosen to represent the scenarios on which hybrid decision trees perform well is SAT12-INDU. It is chosen for several reasons:

- It is a good representation of models trained with spearman footrule and the number of discordant pairs.
- It shows interesting results for hybrid decision trees trained with the modified position error. On all other scenarios, it does not rival the best performance.

Evaluation Discussion of Results with Metrics that Focus on the Quality of the Best Predicted Algorithm on SAT12-INDU In Figure 5.5, the impact of different choices of λ and $\mathcal{L}_{rank}$ on the prediction quality is shown. Here, it is evaluated with the Par10 metric. The noteworthy results of this evaluation are:

- All hybrid decision trees have the same prediction quality at $\lambda = 0$.
- The best hybrid decision tree trained with the number of discordant pairs loss has $\lambda = 1$. However, the hybrid decision trees trained with this loss have a local minimum for $\lambda = 0.8$.
- he best hybrid decision tree trained with the spearman rank correlation has $\lambda = 0$.
- All other losses perform best for some $\lambda \in \{0.1, ..., 0.9\}$.
- At all evaluation points the measured performance is worse than the single best solver.
- The best performance is given by the modified position error and $\lambda = 0.6$.
- All hybrid decision trees trained with the spearman footrule have the same performance for $\lambda \geq 0.5$. This indicates that the spearman footrule overpowers the regression loss for $\lambda \geq 0.5$.
- The hybrid decision tree trained with the spearman rank correlation performs poorly for all $\lambda \geq 0.1$. However, there is a local best performance for $\lambda = 0.9$.

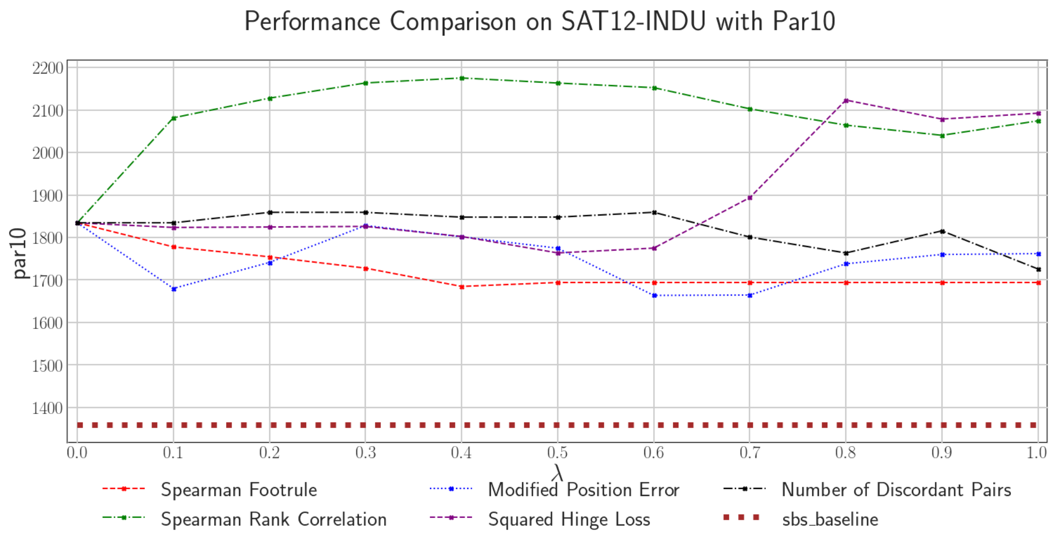


Fig. 5.5.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario SAT12-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric Par10.

The bad performance of all hybrid decision tree configurations according to the Par10 error, indicates that hybrid decision trees perform poorly on SAT12-INDU. This result contrasts the performances evaluated with the performance regret and the percentage of unsolved instances. These are shown in Figure 5.6 and Figure 5.7. The main difference is that the figures indicate that hybrid decision trees are viable for SAT12-INDU for some configurations. In detail one can see the following results:

- If one compares the evaluation resutls of Figure 5.5 with these two graphs the hybrid decision trees that are trained with the same loss functions show similar differences in quality for $\lambda \mapsto \lambda + 0.1$. However, the general quality is much better.
- The hybrid decision tree trained with the modified position error beats to sbs for $\lambda \in \{0.1, 0.6, 0.7, 0.8\}$.
- The hybrid decision tree trained with the number of discordant pairs beats the sbs for $\lambda = 1$.

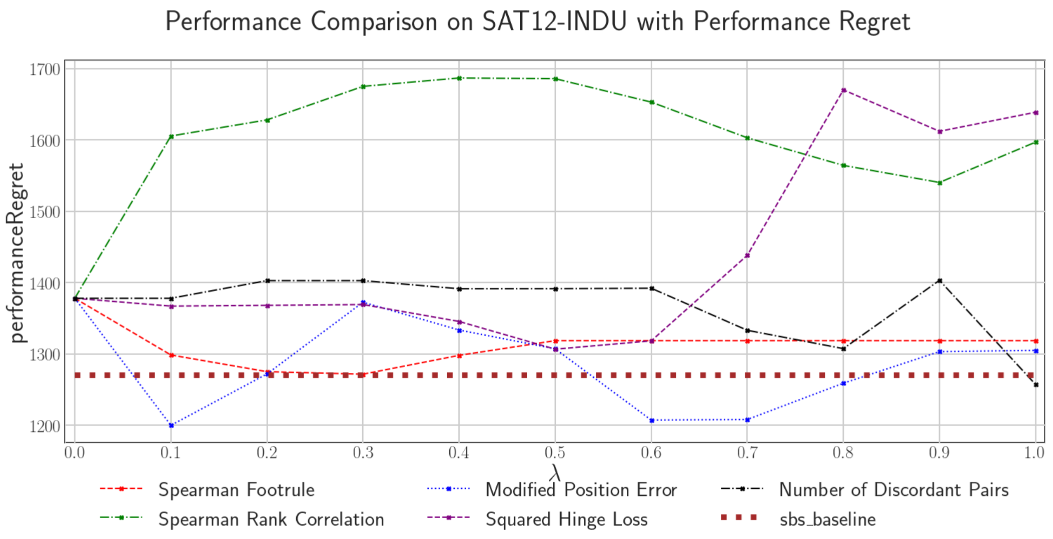


Fig. 5.6.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario SAT12-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the performance regret metric.

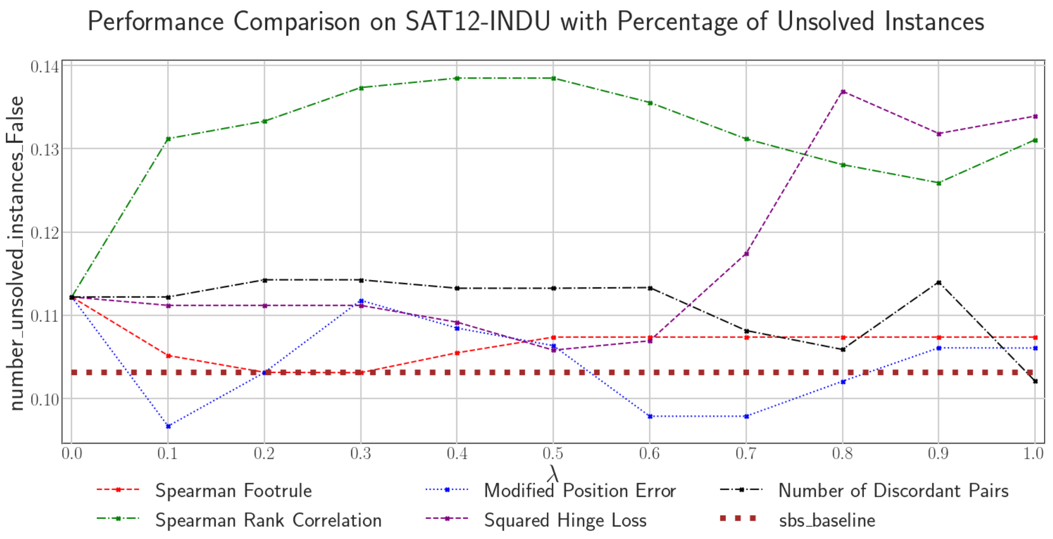


Fig. 5.7.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario SAT12-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric Percentage of Unsolved Instances.

This contrast might be caused by par10 being impacted by the cost of calculating features into account while the performance regret and percentage of unsolved instances are not. For other scenarios, the difference between par10 and the performance regret is not noticeable.

The evaluations on other scenarios are found in the appendix. They show that for most hybrid decision tree configurations, the best performance is given by a hybrid loss and that there are configurations for which they beat the sbs. Additionally, the hybrid decision trees trained with the spearman footrule have the highest quality for some $lambda$ on most scenarios.

Evaluation Discussion of Results that Utilize the Metric NDCG on SAT12-INDU The results of the evaluation with NDCG are shown in Figure 5.8. If one compares this result with Figure 5.6, the evaluated quality of hybrid decision tree models is very similar but mirrored (different values on the y-axis due to different metrics). However, the sbs is not outperformed by any configuration. Almost all hybrid decision trees reach the best performance for some $0 \neq \lambda \neq 1$. The hybrid decision tree that is trained with the number of discordant pairs and $\lambda = 1$ almost beats the single best solver.

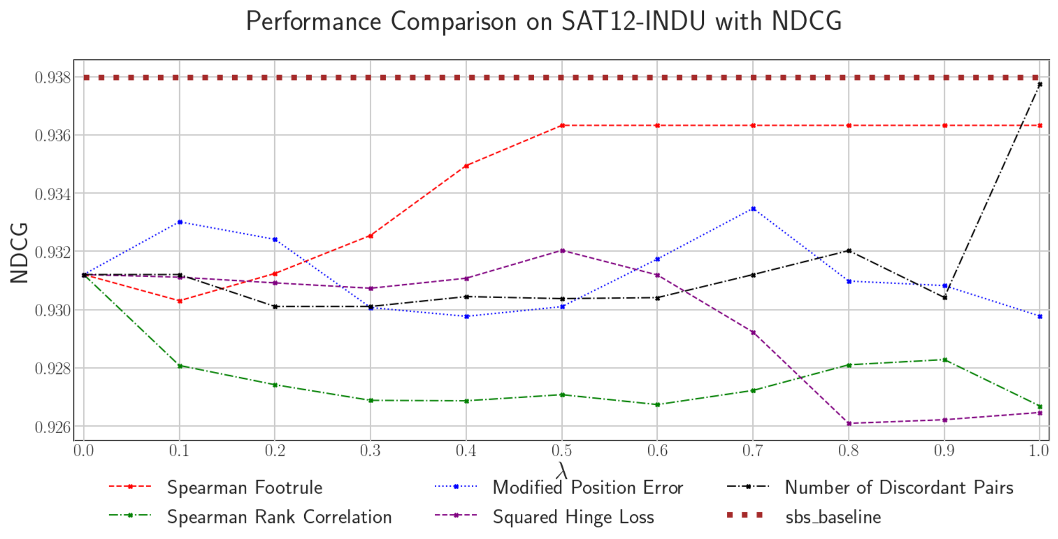


Fig. 5.8.: Comparison of the quality of different hybrid decision tree configurations. They are built with different values for λ and $\mathcal{L}_{rank}$ on the scenario SAT12-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric NDCG.

Interpretation of Results From the evaluation of different hybrid decision trees on this scenario and the other scenarios (figures in appendix), one can conclude that even on the scenarios where some hybrid decision trees rival the sbs, very few hybrid decision tree configurations (with the fixed components) perform well compared to the sbs. Additionally, the best losses are the spearman footrule, the number of discordant pairs, and the modified position error.

Evaluation Results for Scenarios Where Hybrid Decision Trees Perform Poorly

In contrast to the scenarios given above, hybrid decision trees have a relatively poor performance on the following scenarios:

- ASP-POTASSCO
- MAXSAT15-PMS-INDU
- SAT12-HAND

As an example, we discuss the evaluation on MAXSAT15-PMS-INDU.

Evaluation Discussion that Utilizes Metrics that Focus on Quality of the Best Predicted Algorithm on MAXSAT15-PMS-INDU One scenario, on which hybrid decision trees perform relatively poor, is MAXSAT15-PMS-INDU. In Figure 5.9, Figure 5.10, Figure 5.11 the following results can be deduced:

- The figures that show the performance with the par10, performance regret, and percentage of unsolved instances metric, all are very similar.
- All hybrid decision tree configurations make the same predictions for $\lambda = 0$.
- For all ranking losses, the best hybrid decision tree is given for $\lambda > 0$.
- The hybrid decision tree configurations trained with the spearman footrule have a similar prediction for $\lambda \geq 0.1$. This might indicate that the ranking loss outperforms the regression loss for any λ other than 0.
- Only for hybrid decision trees trained with the number of discordant pairs, the best model quality is evaluated for $\lambda = 1$.

- For hybrid decision trees trained with any other loss than the number of discordant pairs, the best hybrid decision tree configuration is given by some $\lambda \in \{0.1, ..., 0.9\}$. The number of discordant pairs also has a local minimum at $\lambda = 0.7$.

- The best solution is given by the tree trained with the number of discordant pairs for $\lambda = 1$.

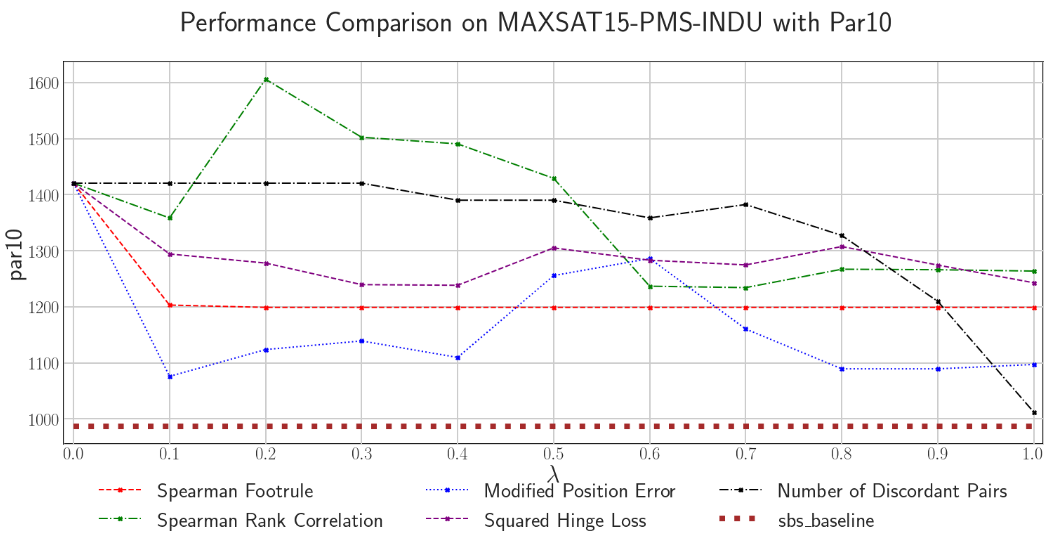


Fig. 5.9.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario MAXSAT15-PMS-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric Par10.

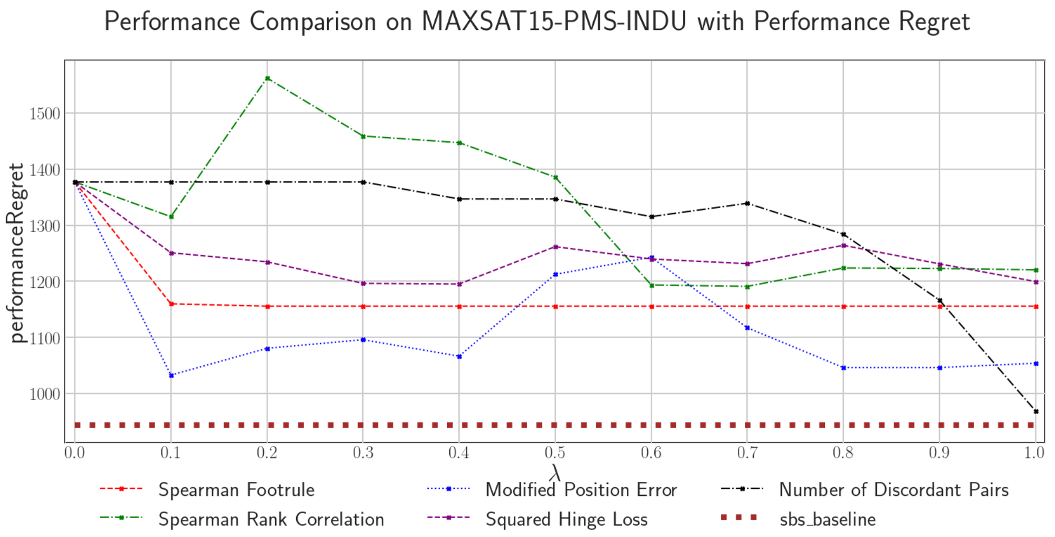


Fig. 5.10.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario MAXSAT15-PMS-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric Performance Regret.

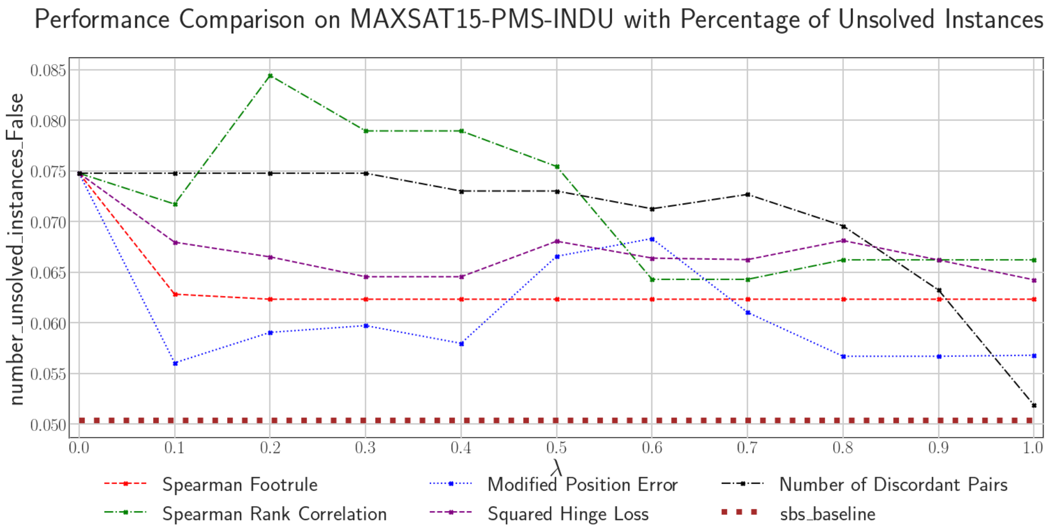


Fig. 5.11.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario MAXSAT15-PMS-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric Percentage of Unsolved Instances.

While these evaluations results differ for other scenarios, they are still very similar. The main difference is that another ranking loss might be the best hybrid decision tree ranking loss. Additionally, the difference in quality between the sbs the best hybrid decision tree configuration is larger on other scenarios.

Evaluation with 'NDCG' on MAXSAT15-PMS-INDU In Figure 5.12 the following results can be found:

- As in the evaluation of good scenarios, the results are similar but mirrored to the evaluation results for metrics that focus on the algorithm with the best-predicted performance.
- All hybrid decision trees have the same performance for $\lambda = 0$.
- The only hybrid decision tree configuration that has the highest prediction quality for $\lambda = 1$, is the Modified Position error. It even outperforms the sbs for $\lambda = 1$. However, for every ranking loss a hybrid decision tree has locally best performance for some $\lambda \in \{0.1, ..., 0.9\}$.
- All other ranking losses have their best performance for some $0 \neq \lambda \neq 1$.
- This is the scenario where hybrid decision trees trained with the number of discordant pairs do not perform best for $\lambda = 1$.
- The hybrid decision tree trained with the spearman footrule does not make predictions of the same quality for $\lambda = 0.1$ and $\lambda = 0.2$. This differs from the above evaluation.

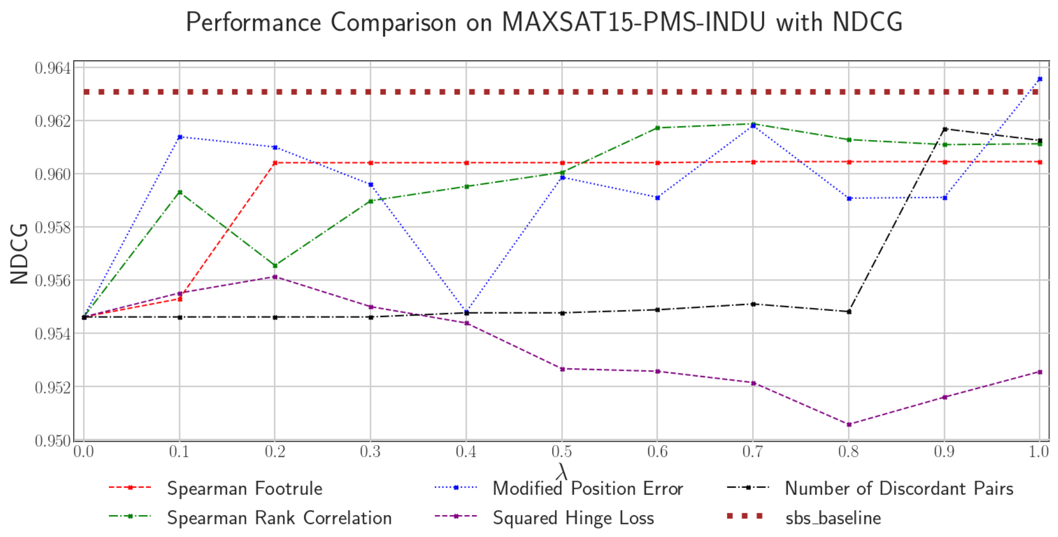


Fig. 5.12.: Comparison of the quality of different hybrid decision tree configurations. They are build with different values for λ and $\mathcal{L}_{rank}$ on the scenario MAXSAT15-PMS-INDU. For reference, the single best solver is also shown in the figure. The evaluation is done with the metric NDCG.

Interpretation of Results From the loss evaluation (including figures in appendix figures in appendix), one can conclude that for scenarios on which hybrid decision trees generally perform poorly, some hybrid decision tree configurations may rival the quality of the single best solver according to some losses.

Conclusion of Loss Evaluation

One can conclude that the hybrid decision tree model (with fixed components), performs either well or poorly compared to the sbs depending on the scenario's properties. In general, hybrid decision trees perform better (compared to the sbs) if scenarios have a lower feature to instances ratio. This can be seen in Figure 5.13. Three of the four scenarios for which hybrid decision trees perform well have a feature instance ratio of less than 4.3%. Two out of three scenarios for which hybrid decision trees perform poorly have a feature to instance ratio of more or equal to 10%.

Scenario	**Feature to Instance ratio**	**Some hybrid decision tree configuration outperforms sbs**
CSP	$86/2024 \approx 4.2\%$	yes
CPMP	$22/527 \approx 4.2\%$	yes
QBF	$46/1254 \approx 3.7\%$	yes
ASP-POTASSCO	$138/1294 \approx 10\%$	no
MAXSAT15-PMS-INDU	$37/601 \approx 6.2\%$	partly
SAT12-HAND	$115/767 \approx 15\%$	no
SAT12-INDU	$115/1167 \approx 9.9\%$	partly

Fig. 5.13.: Overview over the impact of scenario's features to instance ratio and its impact on hybrid decision trees. The label partly is given if only one hybrid decision tree configurations outperforms the sbs on one scenario according to one metric

One explanation of this could be the fact that choosing a good split is harder if more features are considered since each feature increases the number of possible splits by approximately $|\mathcal{D}|$. This might be an issue since a split is chosen locally. If this bad split is chosen more than one level above the leaf that the bad leaf result is not considered. Therefore, this split choice might lead to less homogeneous leafs than another split would.

The following general conclusions can be made:

- For most losses, hybrid decision trees perform best if they work with a combined ranking and regression loss (on most scenarios).

- The spearman footrule overpowers the regression error on most scenarios. This results in some trees of $\lambda \leq 1$ having the same quality as a hybrid decision tree trained solely with the ranking error even though the regression error should also have an impact.

- The spearman ranking correlation can be disregarded since there is no hybrid decision tree configured with it that is of high quality (except CSP-2010 with $\lambda = 1$). Additionally hybrid decision trees trained with only the ranking loss ($\lambda = 1$) generally outperform trees that are trained solely with the regression loss ($\lambda = 0$). This is not the case for the spearman rank correlation.

- The squared hinge loss performs poorly on most scenarios and only performs well on CSP (which has just two algorithms) and can therefore be disregarded as well. The scenario CSP is prone to giving different results than the other scenarios because of its differing properties.

- Since hybrid decision trees trained with the number of discordant pairs perform best for $\lambda = 1$ on all scenarios, we disregard this loss as well. Note that since we did not do a detailed evaluation for $\lambda \in (0.9, 1)$, it is not proven that the loss is a bad choice for a convex loss combination.

- The candidate losses for further investigation are the spearman footrule and modified position error.

5.4.2 Ablation Study on Borda Score

In section 3.6.4, it was mentioned that the split is determined based on the loss of the resulting datasets. These losses quantify the similarity of the instance labels with the consensus label.
For the node-wise ranking loss calculation, the consensus label is calculated with borda's method (section 5.4.2). Then, an average loss is calculated between the instance label Y and the consensus label $\hat{Y}$. Since the chosen borda function can impact this loss, the effect of different functions on hybrid decision trees is worth

exploring.

We evaluated the quality of hybrid decision trees, if they utilize different borda functions to calculate a consensus label. For this evaluation, we fix all other hybrid decision tree components:

- The chosen ranking loss is the spearman footrule.
- The spearman footrule is evaluated with $\lambda = 0.7$.
- The splitting is stopped if the depth three is reached.
- The performance data is preprocessed with $\mu = 1$.

In Figure 5.14, Figure 5.15, Figure 5.16, and Figure 5.17, the performance of the mentioned hybrid decision trees is evaluated. In each figure, the performance of binary decision trees is compared according to one metric. The evaluation results are shown in a bar diagram with one bar for each borda function. On the y-axis is the score according to the given metric.

- Firstly, all the Borda Scores give the same evaluation results on the scenario CSP-2010. This is due to the set of candidate algorithms being of size two resulting in the likelihood of all trees having the same structure.
- There is only one scenario on which mean ranking performs worst (CPMP-2015).
- There is only one scenario on which median ranking performs worst (SAT12-HAND).
- Mean performance performs worst on two scenarios (SAT12-INDU and ASP-POTASSCO).
- Geometric mean performs worst on two scenarios (QBF-2016 and MAXSAT15-PMS-INDU).

This evaluation indicates that the mean ranking or the mean performance is the best borda function for hybrid decision trees. However, this can differ based on the scenario.

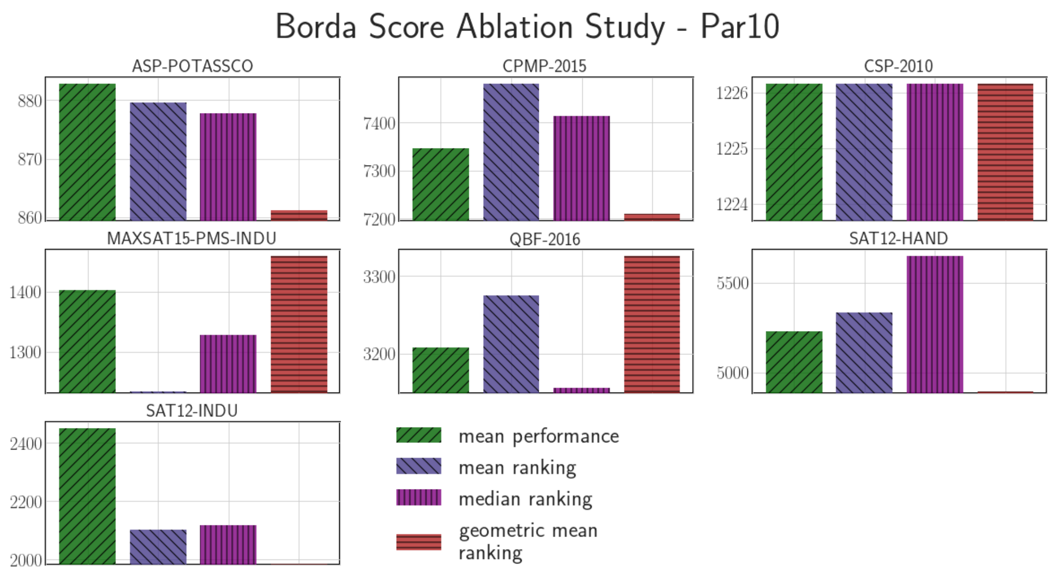


Fig. 5.14.: Comparison of the par10 score reached by hybrid decision trees configured with the spearman rank correlation, $\lambda = 0.7$, $\mu = 1$, a maximum depth of 3, and different borda functions.

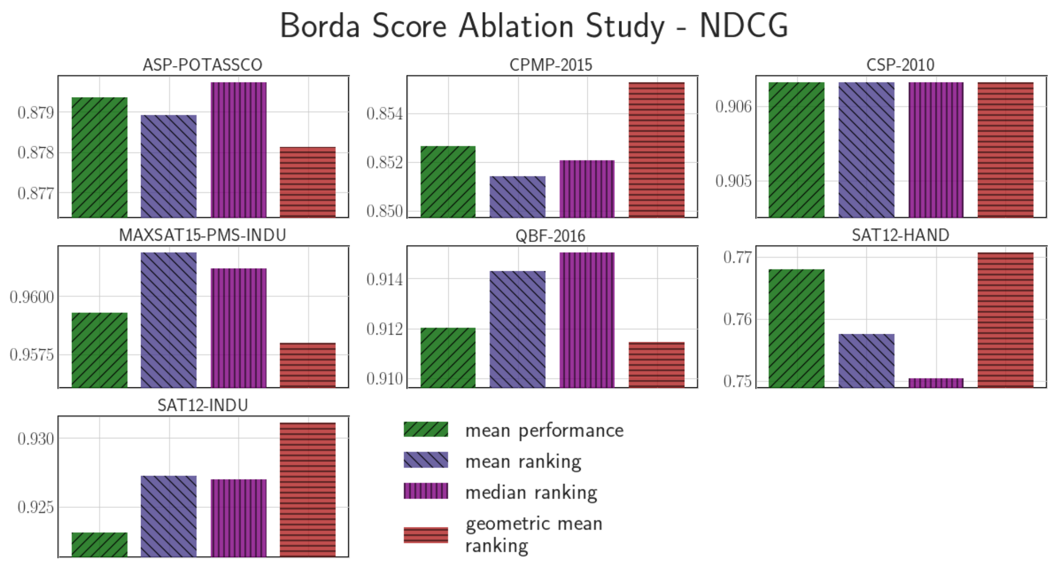


Fig. 5.15.: Comparison of the NDCG score reached by hybrid decision trees configured with the spearman rank correlation, $\lambda = 0.7$, $\mu = 1$, a maximum depth of 3, and different borda functions.

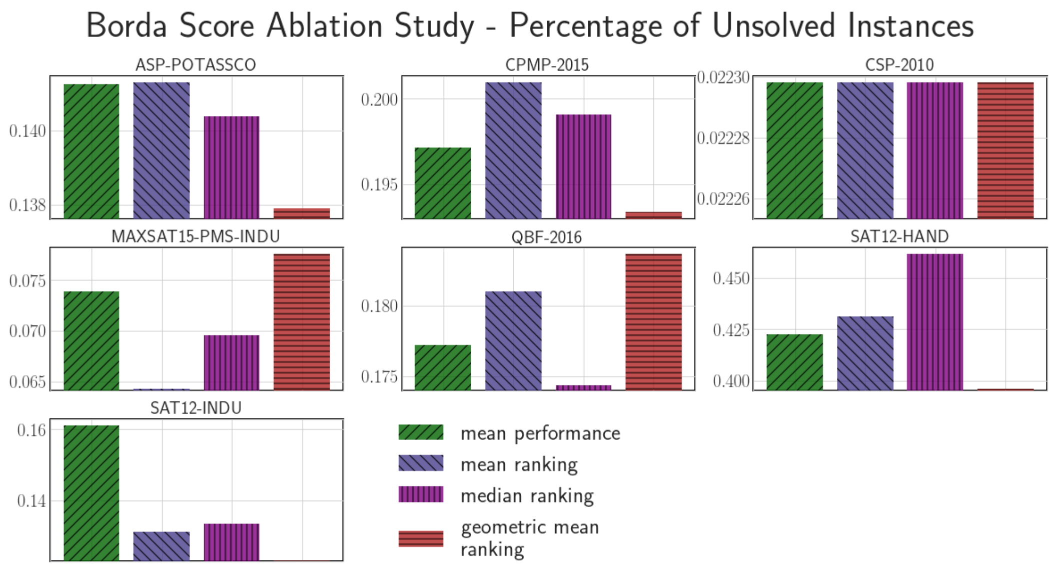


Fig. 5.16.: Comparison of the percentage of unsolved instances reached by hybrid decision trees configured with the spearman rank correlation, $\lambda = 0.7$, $\mu = 1$, a maximum depth of 3, and different borda functions.

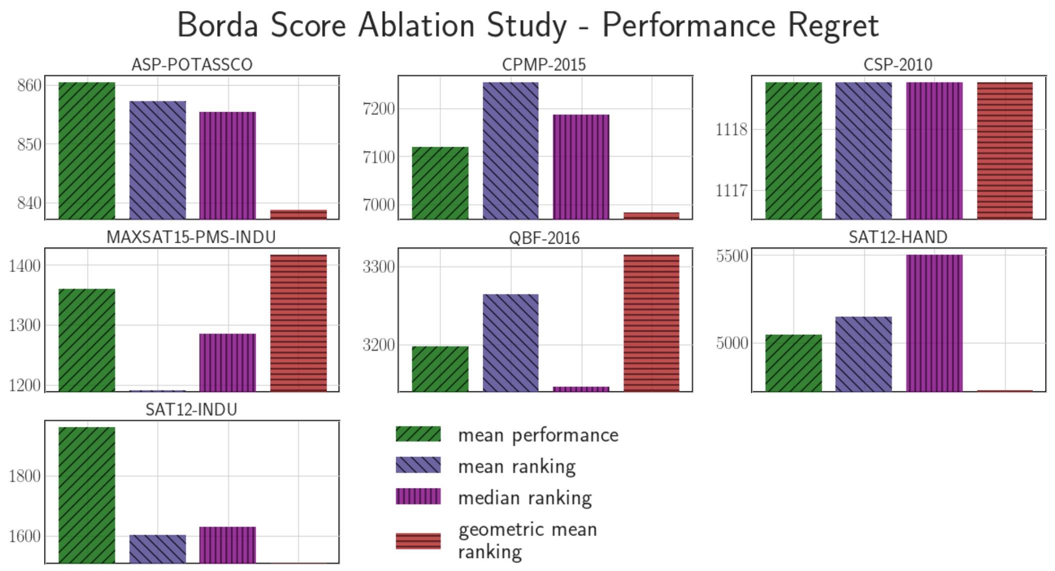


Fig. 5.17.: Comparison of the performance regret score reached by hybrid decision trees configured with the spearman rank correlation, $\lambda = 0.7$, $\mu = 1$, a maximum depth of 3, and different borda functions.

5.4.3 Ablation Study on Stopping Criteria

In section 3.7 different stopping criteria were introduced. To evaluate the quality of stopping criteria, all other components are fixed. The fixed components are:

- The chosen ranking loss is the spearman footrule.
- The spearman footrule is evaluated with $\lambda = 0.6$.
- The consensus ranking is calculated with the mean ranking.
- The performance data is preprocessed with $\mu = 1$.

These components are chosen since they perform well on average according to the loss evaluation (section 5.4.1) and borda evaluation (section 5.4.2) above.

The results of the evaluation are very similar according to all evaluation metrics. Similar to the evaluation of different borda scores, the bar diagram in Figure 5.18 shows the evaluation of hybrid decision trees that utilize different stopping criteria. On the y-axis are the scores according to performance regret. The bars are grouped into three types of hybrid decision trees and the single best solver.
In section 3.7 we introduced four stopping criteria for hybrid decision trees. However, two of them are essentially the same:

1. Given the performance y_i, y_j if two algorithms $A_i, A_j \in \mathcal{A}$ all of the instances rank the performances either $y_i \succeq y_j$ or $y_j \succeq y_i$.
2. Given the performance y_i, y_j if two algorithms $A_i, A_j \in \mathcal{A}$ at least x% of the instances rank the performances either $y_i \succeq y_j$ or $y_j \succeq y_i$.

If the algorithms are ranked equally for all instances, they are ordered equally for 100% of them. Therefore, the four mentioned stopping criteria can be reduced to three. In addition, hybrid decision trees are compared to the single best solver (sbs). This comparison is interesting because the single best solver makes the same predictions as does a hybrid decision tree of depth 0 (cf. section 3.7).

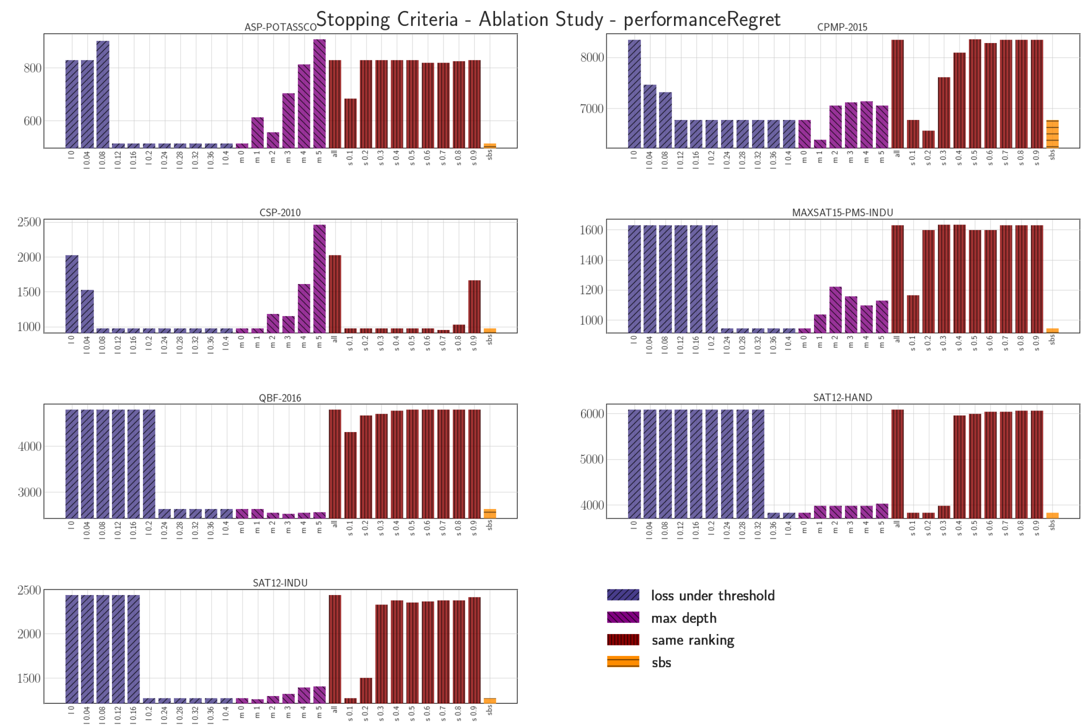


Fig. 5.18.: Evaluation of hybrid decision trees that are build with different stopping criteria according to the metric performance regret. In addition, the hybrid decision trees are compared to the sbs to explore whether some hybrid decision tree configurations imitate the sbs.

From the evaluation, one can make the following conclusions:

- There exists a limit $\alpha_s \in (0, 1]$ for each scenario s in the given scenarios. The property that if the loss under threshold method is applied hybrid decision trees are of depth 0 since the calculated loss is smaller than the threshold. This holds true for all proposed configurations of the losses.
- Loss under threshold is an inadequate criterion for hybrid decision trees because it either performs poorly or builds a tree of depth 0.
- Trees of depth 0 give the same predictions as the sbs. This was already mentioned in section 5.2.1.
- The best stopping criterion on the given scenarios is the maximal hybrid decision tree depth. It works best with a depth chosen from $\{1, 2, 3\}$ based on the scenario. The best depth might be correlated with the number of instances

in the scenario. This can be seen in Figure 5.19 where the best depth for each scenario is given. If a dip differs from that, it is given in braces. If one does not consider the scenarios CSP-2010 and CPMP-2015 (different behavior due to very few algorithms), there is a tendency for a higher optimal maximum depth for scenarios with more features. However, this evaluation does not give enough information for a precise result.

- For most stopping points, the best performance has a local minimum for a depth in $\{2, 3, 4\}$ as shown in Figure 5.19.

- For scenarios with very few algorithms (CSP-2010 and CPMP-2015), the best hybrid decision tree configuration has a maximum depth of 1. However, the CPMP-2015 scenario has a local minimum of the maximum depth criterion at depth 5. This result might imply that hybrid decision trees perform better on scenarios with few algorithms if they have a small depth. However, this is not a definitive result as we only have two scenarios with few algorithms.

Scenario	Instances	Best Maximum Depth
ASP-POTASSCO	1294	2
CSP-2010	2024	1 (3)
MAXSAT-15-PMS-INDU	601	1 (4)
QBF-2016	1254	3
SAT12-Hand	767	2
SAT12-INDU	1167	1
CPMP-2015	527	1 (5)

Fig. 5.19.: In this table every scenario is given with the best maximum depth configuration for hybrid decision trees. If there is a local minimum in the maximum depth for some other depth it is given in braces.

The main results of this evaluation can be reduced to the fact that hybrid decision trees are of the highest quality for the stopping criterion maximum depth with the depth in $\{1, 2, 3, 4\}$. This is the best choice since the stopping criteria 'loss under threshold', and 'same ranking percentage' either perform poorly or result in a tree of depth 0, which is essentially an sbs. For increasing instances, one should also explore whether increasing the maximal depth results in better performing hybrid decision trees. One should also explore whether a depth of 1 is generally good for scenarios with few candidate algorithms.

The other evaluation metrics support these evaluation results (cf. appendix). Note that in the par10 score, there are scenarios for which the sbs and hybrid decision trees of depth 0 do not indicate the same hybrid decision tree quality. This does not mean that the prediction of the hybrid decision trees and sbs are not the same. This difference is caused by the fact that the par10 loss is impacted by the time used for feature generation that hybrid decision trees need (cf. Equation 4.13). However, since the sbs does not need any features, the effort of calculating those features is disregarded. This impacts the scenarios SAT-12 HAND and SAT12-INDU. The scenarios QBF-2016 and ASP-POTASSCO are also impacted, but this is not visible in the figure.

5.4.4 Ablation Study on Handling of Censored Labels

In section 3.2.1, the problem of incomplete performance data was introduced. These incomplete labels are handled by setting them to $\mu \cdot C$.
To evaluate the impact of different μ on the behavior of hybrid decision trees, this evaluation is done with the following configuration:

- The chosen ranking loss is the spearman footrule.
- The consensus ranking is calculated with the mean ranking.
- The stopping criterion is a maximum tree depth of 3.

In this evaluation, we do not fix λ as μ impacts the regression loss (and some ranking loss functions). Therefore, the impact of λ values on hybrid decision trees trained with different μ values, is interesting as well.

In Figure 5.20, the impact of μ on the quality of hybrid decision trees is visualized (on SAT12-INDU). Even though the evaluated behavior of hybrid decision trees changes on different scenarios/with different metrics for evaluation, we only show one evaluation result here because it is a good representation of all possible evaluations and there is no clear best μ. Almost all μ, λ configurations have scenarios for which they give good results and scenarios for which they give bad results. However, the tendency of $\mu = 1$ to give good performance can be seen in this graph.

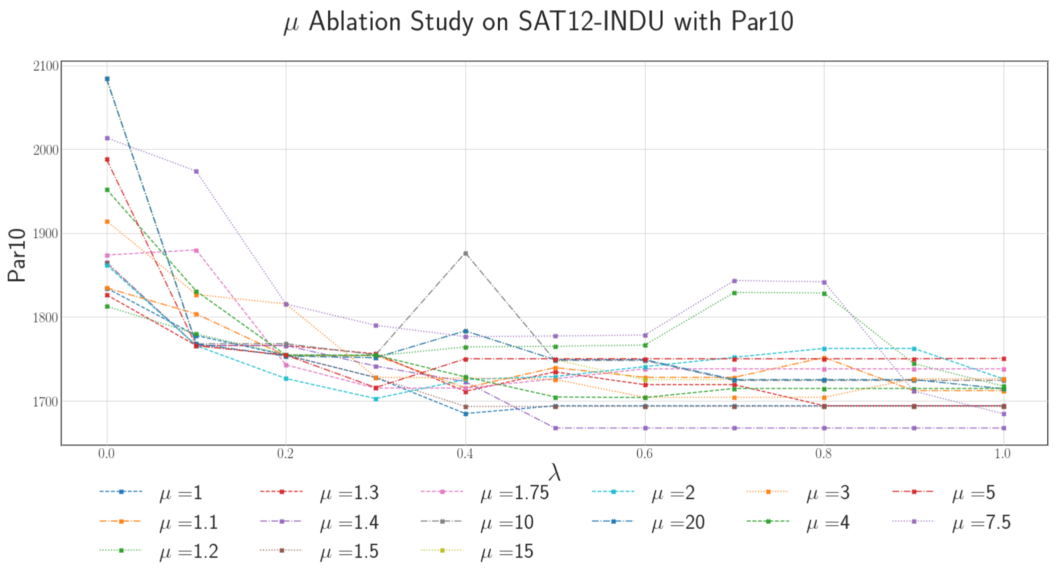


Fig. 5.20.: Overview over the performance of different hybrid decision tree configurations with different values for μ. Since we expect both λ and μ to influence each other this evaluation is done for $\lambda \in \{0, 0.1, ..., 1\}$.

In addition, to the aforementioned result one should take note of the following results:

- In difference to Figure 5.20 there is a difference in hybrid decision tree performances for $\lambda = 0$.
- For most hybrid decision tree configurations, the curve is similar. This means that the performance has a similar change in value for each step. However, there are configurations ($\mu = 7.5$ and $\mu = 1.2$) that behave different to the rest of the graphs (for $\lambda = 0.7$ and $\lambda = 0.8$). Interestingly, these outliers happen for μ values that are not similar.
- While $\mu = 1$ does not give the best quality hybrid decision trees on every scenario, there is no scenario on which it is the worst configuration- These are not chosen because all other evaluations above are done with $\mu = 1$.
- There is no apparent correlation between the similarity of μ values and their curves.

All other graphs are presented in the appendix. As mentioned before, there is no clear best μ configuration. Nevertheless, $\mu = 1$ it is still chosen as a hybrid decision tree configuration for two reasons. Firstly it seems to be an excellent general configuration. Secondly, all other components were evaluated with $\mu = 1$. If there is an unforeseen strong impact of other μ values on the quality of other components, this is not an issue here.

5.5 Comparison of Hybrid Decision Trees with Other State of the Art Approaches

To answer whether hybrid decision trees are a good model for algorithm selection, we need to compare them to other algorithm selection models. To that end, we first introduce candidate hybrid decision tree configurations that are used for the final evaluation (section 5.5.1). Then, we compare the candidate hybrid decision trees to per algorithm decision tree regressor (section 5.5.2), a basic survival tree configuration (section 5.5.3), and preexisting hybrid models that are also trained with a a convex combination of a ranking and regression loss Equation 3.57 (section 5.5.4).

5.5.1 Component Selection for Final Evaluation

The component-wise evaluations of hybrid decision trees show that depending on the scenarios different hybrid decision tree components perform best. It follows that the comparison of hybrid decision trees and other models should include different candidate hybrid decision tree configurations. We use all combinations of the following components:

1. For ranking loss functions we evaluate hybrid decision trees with the spearman footrule and the modified position error. The spearman footrule is chosen since it performs well on most scenarios (best on some). The modified position error is chosen since it performs adequately on most scenarios and is by far the best loss function SAT12-IND(for some λ). The other loss functions are disregarded.
   However, it is noteworthy that an evaluation with the squared hinge loss would be interesting for further investigation as well since it consistently has its best performance for some $\lambda \in (0, 1)$.

2. We evaluate hybrid decision trees with all $\lambda \in \{0, 0.1, ..., 1\}$, since the hybrid decision tree configurations. This volatile behavior shows the fixing λ might result in us not evaluating some good hybrid decision tree configurations. In addition, a comparison to hybrid models [Han+20] is more interesting, if we explore the impact of λ on all compared models.

3. The chosen borda function is the mean ranking. It is chosen since the results from section 5.4.2 indicated that either the mean ranking or the median ranking has the best performance on most scenarios. Since evaluating both borda functions here would double the number of candidates hybrid decision trees, we decided for one.

4. The only stopping criterion used is the maximum depth stopping criterion. However, section 5.4.3 showed that the best configuration of the maximal depth stopping criterion depends on the scenario. Therefore, we explore hybrid decision trees with a maximal depth of 1, 2, 3, or 4.

The following figures have the same structure introduced in section 5.4.1. For every scenario, we have four separate figures for the evaluation with a different scoring metric (We will only show a subset of figures. The others can be found in the appendix). They always compare different hybrid decision tree configurations with baselines/other approaches. Therefore, each figure shows a comparison of the model qualities that are quantified with either the scoring function par10, the performance regret, the percentage of unsolved instances, or the NDCG.

There are some hybrid decision tree properties that can be seen in every figure:

- All hybrid decision tree configurations of the same depth reach the same score for $\lambda = 0$. This behavior was already noted in section 5.4.1 and is caused by trees configured with $\lambda = 0$ being solely based on regression.

- A change in the stopping criteria can lead to an entirely different result. For example in Figure 5.21 the performances of maximum depth 3 spearman footrule and maximum depth 2 spearman footrule have very different profiles for $\lambda < 0.5$.

- Oftentimes hybrid decision trees that are trained with the same ranking loss function have a very similar change in prediction quality based on a change of λ.

5.5.2 Comparison to Per Algorithm Decision Tree Regressor

As mentioned in section 5.2.2 per algorithm decision trees are based on regression rather than ranking. The best predicted algorithm/the ranking are then extracted from separate regression trees.
For an evaluation of the tree, hybrid decision tree comparison, we divide the scenarios into three groups (based on the comparison result). For each of these groups the comparison to the per algorithm regressor is discussed separately.

1. Scenarios on which the best hybrid decision tree configurations perform better than per algorithm decision tree regressors.

2. Scenarios on which the best hybrid decision tree configurations perform equal to per algorithm decision tree regressors.

3. Scenarios on which the best hybrid decision tree configurations perform worse than per algorithm decision tree regressors.

Evaluation for Scenarios Where Hybrid Decision Trees Perform Well As mentioned above, there are scenarios for which we have hybrid decision tree configurations that perform well. They are:

- QBF-2016

- CPMP-2015

- MAXSAT15-PMS-INDU

While the evaluation results differ from scenario to scenario, the evaluation results on SAT12-INDU represent the overall results well. This is the case since all of the common evaluation properties can be found here. The evaluation is only done with the performance regret metric, since the other results with other scoring functions do not give further information on the compared quality of hybrid decision trees.

Results Specific to the Evaluation of Hybrid Decision Trees on SAT12-INDU

- The best hybrid decision tree is trained with a node-wise loss function impacted by the ranking and the regression error (maximum depth 2 - modified position error). Here, the modified position error and the regression error do not overpower each other. This is seen at the evaluation point $\lambda = 0.4$.

- The best performance score of a hybrid decision tree trained with the modified position error, is reached for a true combined loss.

- The best performance score of a hybrid decision tree trained with the spearman footrule is reached for a maximum depth of 2 and $\lambda = 0.1$. There is also overpowering of errors for hybrid decision trees trained with the spearman footrule here.

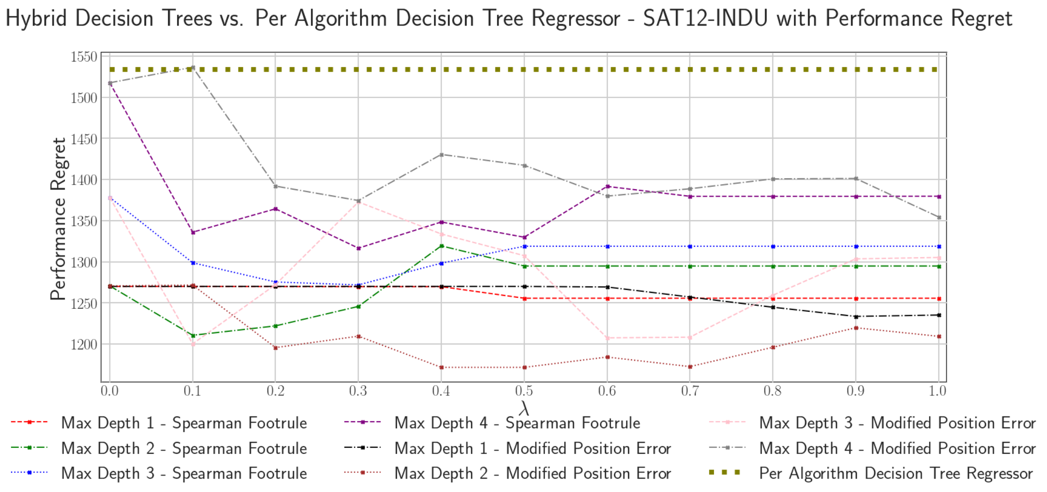


Fig. 5.21.: Quality Comparison of different hybrid decision tree configurations with the per algorithm decision tree regressor. This evaluation is done on the scenario SAT12-INDU with the metric performance regret.

General Results for the Quality of Hybrid Decision Trees on Scenarios where They Perform Well

- For all hybrid decision tree configurations trained with the spearman footrule, there is a point $x \in (0, 1)$, and for all $x > \lambda$, the quality of the predictions is the same. This indicates that the spearman footrule overpowers the regression error. However, x is dependent on the maximum depth. This result can be seen on all scenarios where hybrid decision trees perform well or poorly alike.

- As in the evaluation on SAT12-INDU, the highest quality hybrid decision trees are always trained with a true combined loss function.

- The best-evaluated configuration (maximum depth 2 - modified position error) outperforms the per algorithm regressor considerably. However on other scenarios hybrid decision trees trained with the modified position error perform worse than those trained with the spearman footrule.

- On SAT12-INDU, all hybrid decision tree configurations outperform the per algorithm regressor. This is different for other scenarios on which hybrid decision trees perform well.

Evaluation for Scenarios Where Hybrid Decision Trees Perform Poorly

- SAT12-HAND

- ASP-POTASSCO

There are scenarios for which hybrid decision trees perform poorly compared to the per algorithm regressor. The example discussed here is ASP-POTASSCO.

Results of the Evaluation of Hybrid Decision Trees on ASP-POTASSCO

- The best hybrid decision tree configurations is given by a tree that utilizes the spearman footrule and a maximum depth of 3 for $\lambda \geq 0.9$, whereas the trees trained with $\lambda = 0.9$ and $\lambda = 1$ have the same performance. This might imply that the ranking error outperforms the regression error. However, since hybrid decision trees with $\lambda < 0.9$ make poorer predictions, the errors are somewhat similar.

- The best quality of a hybrid decision tree that utilizes the modified position error, is evaluated with a maximum depth of 1 and $\lambda = 1$. However, hybrid decision trees with $\lambda \in \{0.3, 0.4\}$ have almost the same prediction quality.

- There always exists $\lambda \in \{0.1, ..., 0.9\}$ for which the hybrid decision tree outperforms $\lambda = 0$. This means that hybrid decision trees trained true hybrid losses perform better than trees trained solely with a regression loss.

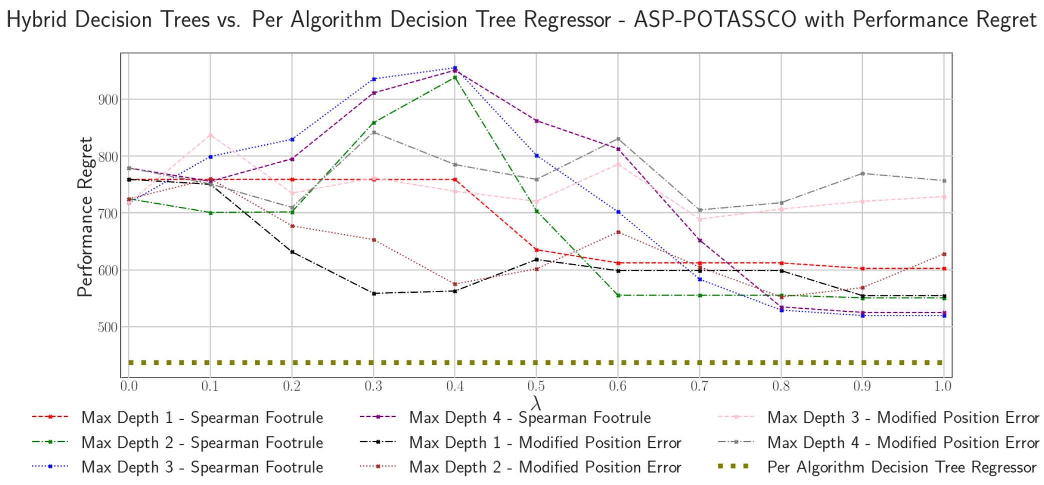


Fig. 5.22.: Quality Comparison of different hybrid decision tree configurations with the per algorithm decision tree regressor. This evaluation is done on the scenario ASP-POTASSCO with the metric performance regret.

General results for the performance of hybrid decision trees on scenarios where they perform poorly

- The hybrid decision trees trained with a modified position error are more likely to have the best quality for $\lambda \neq 1$. They are more likely to have a better performance than for $0 \neq \lambda = 1$ for some other λ.

- Most hybrid decision trees trained with the spearman footrule have a constant quality for $\lambda > x$ (for some $x < 1$) have equal quality. This indicates that The spearman footrule overpowers the ranking error.

Evaluation on CSP-2010 In evaluating different components, was mentioned that the evaluation results on CSP-2010 often differ from the other results. Since this is the case for the comparison to per algorithm decision tree regressors we discuss this scenario separately. In Figure 5.23 one can see the following results.

- The hybrid decision tree configurations of depth 1 (with both evaluated errors) have the same prediction quality for all λ. This is the best evaluated hybrid decision tree and beats the per algorithm decision tree regressor by a small margin.

- Some other configurations beat the sbs for $\lambda = 1$. This might imply that a prediction that is mostly/only based on the ranking is best for two algorithms. However, this might be different for other scenarios with two algorithms.

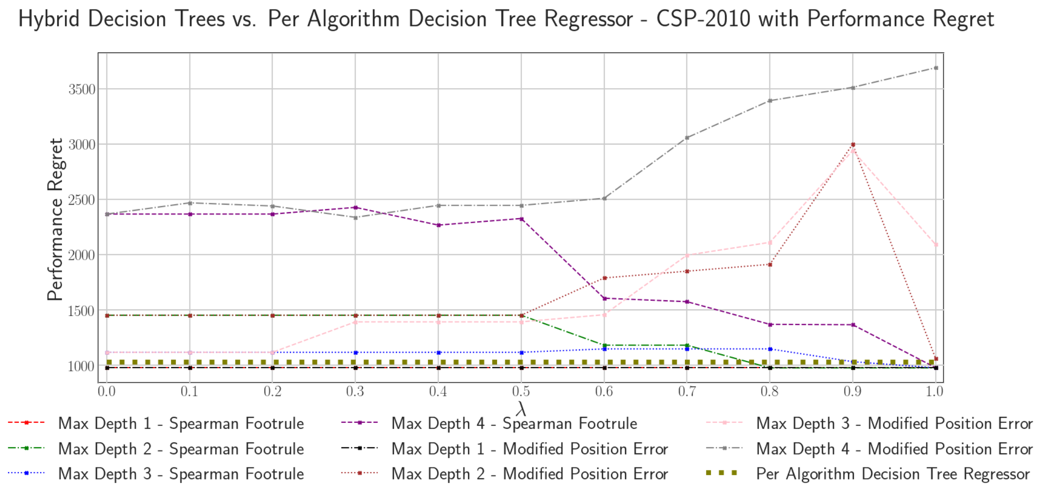


Fig. 5.23.: Quality Comparison of different hybrid decision tree configurations with the per algorithm decision tree regressor. This evaluation is done on the scenario CSP-2010 with the metric performance regret.

Conclusion on the Quality Comparison between Hybrid Decision Trees and Per Algorithm Decision Tree Regressors The main conclusion from this evaluation is that hybrid decision trees outperform the regressors for some scenarios. However there is no simple reason for this since the scenarios properties differ and this is worth further exploration.

One thing to note about the change in quality of hybrid decision trees that is caused by a change in λ, is that hybrid decision trees trained with the same loss function often show similar characteristics.

5.5.3 Comparison to Survival Forests

In comparison to all survival forests [Tor+20] hybrid decision trees perform poorly. This can be attributed to a variety of factors like the comparison of trees and forests being unequal. Since the other survival forest approaches give largely the same results, we only compare hybrid decision trees to the expectation survival forest here.
Since hybrid decision trees perform poorly in comparison to survival forests on every scenario, there is no need to split the scenarios into groups. Instead, we first discuss the results (with par10, performance regret, percentage of unsolved instances) on CPMP-2015 (the scenario on which hybrid decision trees perform best compared to survival forests) in detail and then give some general context on other scenarios. We then discuss the results evaluated with NDCG on the outlier scenario CSP-2010 and then give context on the performance of hybrid decision trees on other scenarios by evaluating the aforementioned compared approaches on CPMP-2015 . Note that since we already discussed the performance of different hybrid decision trees in detail in section 5.5.2, we do not discuss them here again.

Quality Comparison of Hybrid Decision Trees and Survival Forests Based on the Best Predicted Algorithm As mentioned above, hybrid decision trees perform poorly against survival forests on the scenario CPMP-2015. This can be seen in Figure 5.24, Figure 5.25, and Figure 5.26. From these figures, one can make the following conclusions:

- In each of those metrics, no hybrid decision tree configurations has a quality close to the quality of the expectation algorithm survival forest.

- The performance evaluations with par10 and the performance regret indicate that the best hybrid decision tree configurations perform about 7% worse than the survival tree.

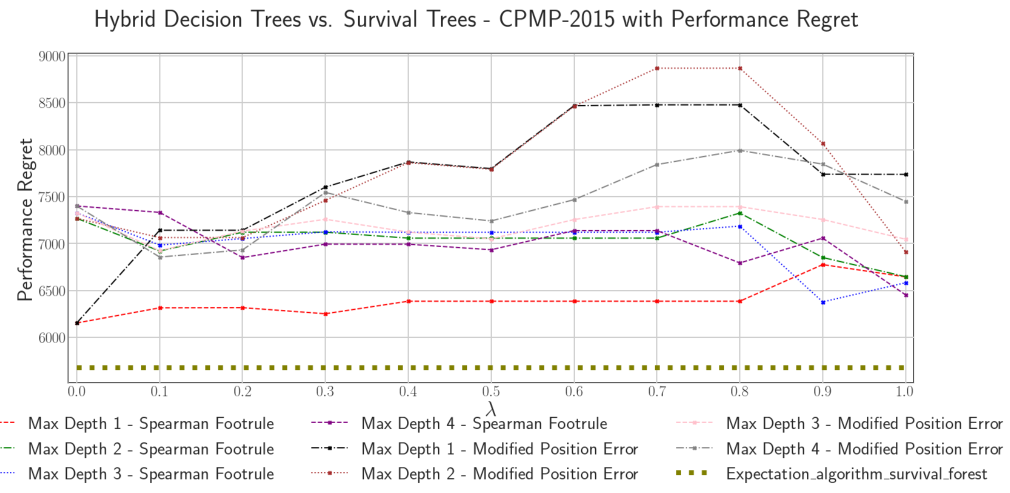


Fig. 5.24.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario CPMP-2015 with the metric performance regret.

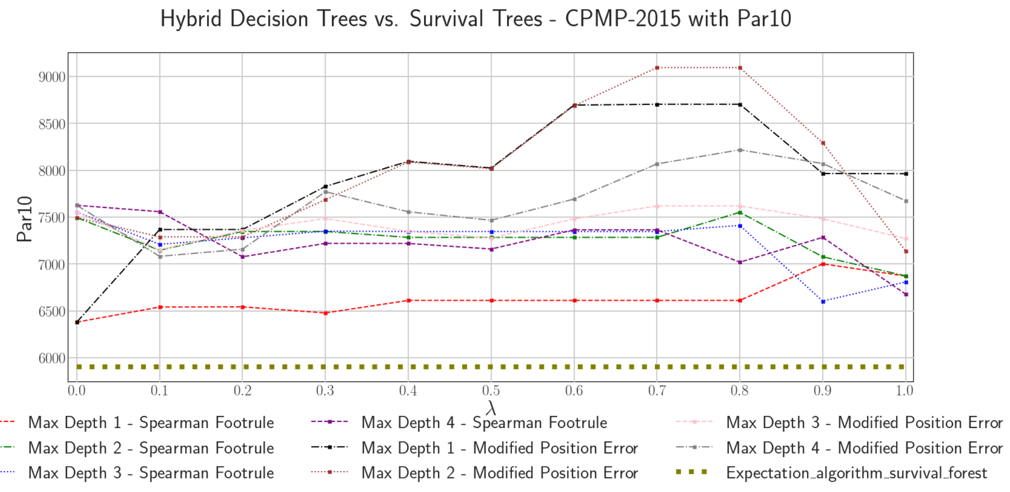


Fig. 5.25.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario CPMP-2015 with the metric par10.

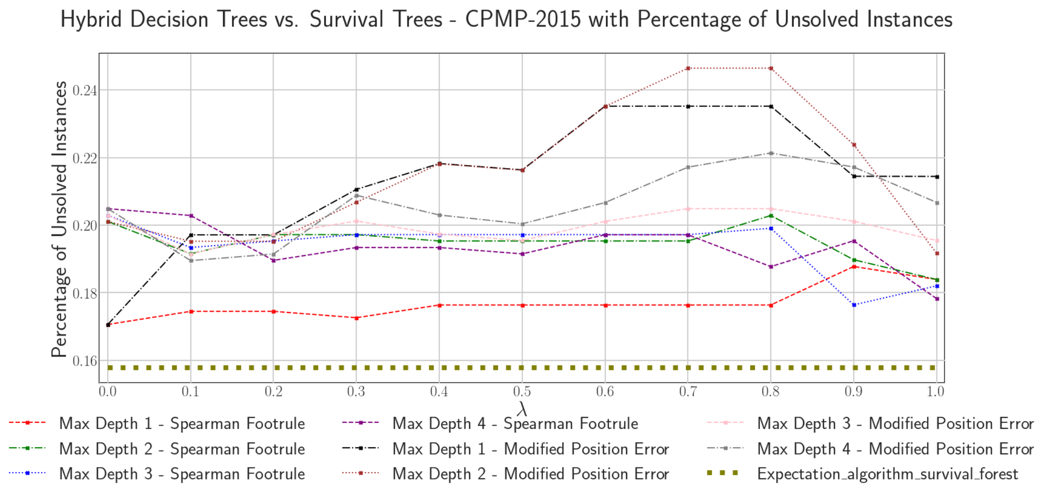


Fig. 5.26.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario CPMP-2015 with the metric percentage of unsolved instances.

On all other scenarios, this difference in performance is much higher than the difference on CPMP-2015. The best hybrid decision tree configurations commonly make predictions that are evaluated to be twice as bad by par10 and the performance regret. On the scenario, with the smallest difference in quality of hybrid decision trees and expectation survival trees is the lowest, SAT12-INDU, hybrid decision trees still perform 60% worse than the expectation survival froestaccording to par10 (shown in Figure 5.27). In other metrics, the percentual difference in performance is higher.

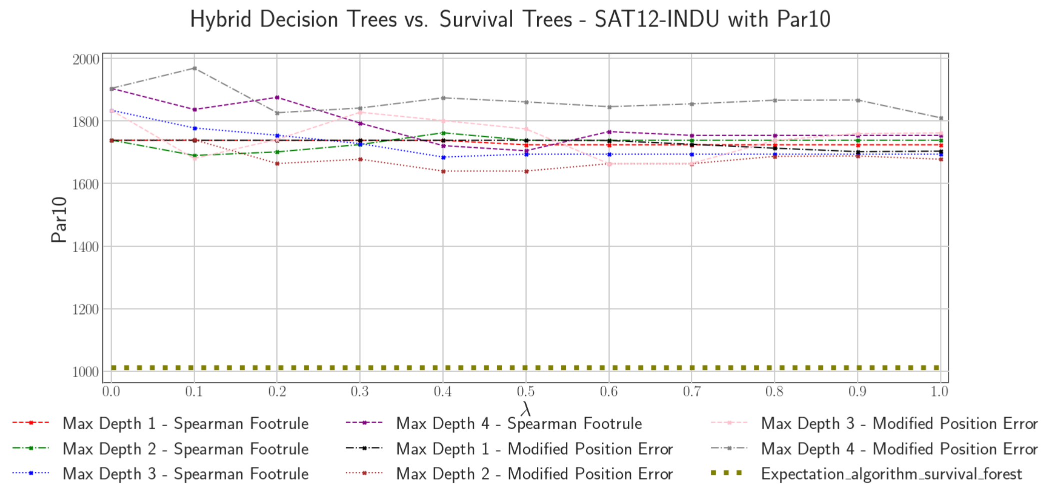


Fig. 5.27.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario SAT12-INDU with the metric par10.

Quality Comparison of Hybrid Decision Trees and Survival Forests Based on NDCG

Survival trees commonly beat hybrid decision trees on all (evaluated) scenarios on the metric NDCG. The only exception is the scenario CSP-2010, on which the results of the ablations studies frequently differed already. In Figure 5.28 the following results of the evaluation can be seen:

- As shown in the comparison of hybrid decision trees and per algorithm decision tree regressors (Figure 5.23), hybrid decision trees of depth 1 perform best and are not impacted by different values of λ. They even beat expectation algorithm survival forests.

- The hybrid decision trees trained with maximum depth 2, the spearman footrule, and $\lambda \geq 0.8$ have similar quality as the hybrid decision trees discussed above.

- The hybrid decision trees trained with maximum depth 2 or 4, the spearman footrule, and $\lambda = 1$ also similar quality to the hybrid decision trees discussed above.

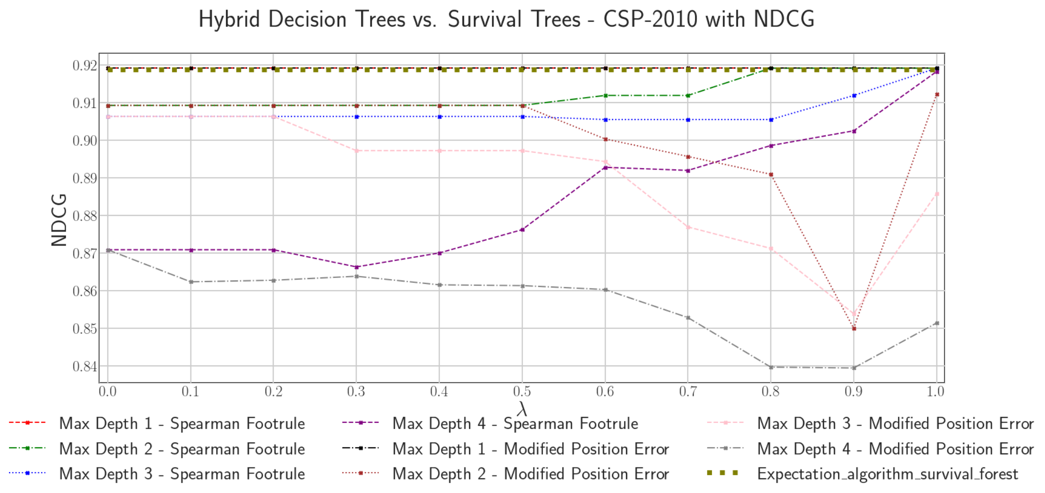


Fig. 5.28.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario CSP-2010 with the metric NDCG.

However, the results of an evaluation on CSP-2010 differ from the evaluations on all other considered here scenarios. On these scenarios hybrid decision trees perform poor in comparison to survival trees. An example of this is given in Figure 5.29

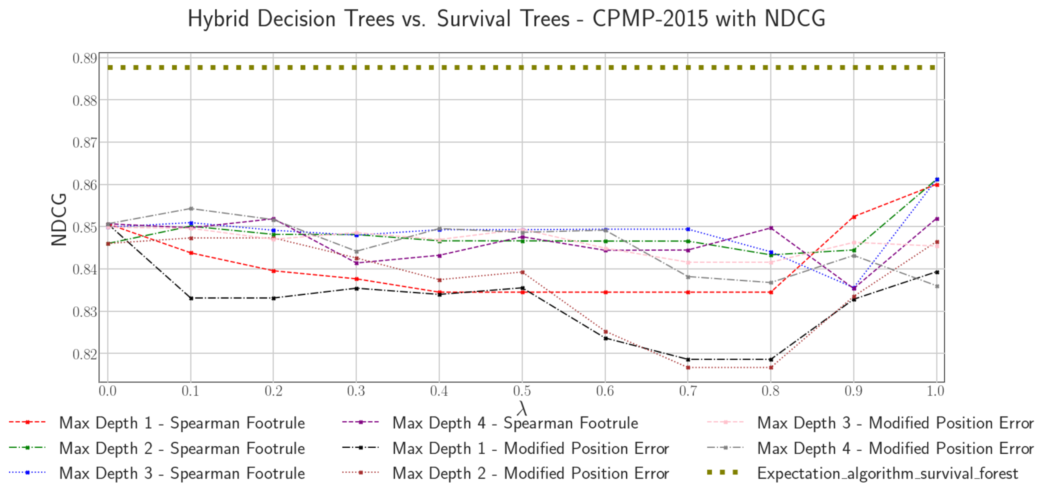


Fig. 5.29.: Quality comparison of different hybrid decision tree configuration and expectation algorithm survival forests. This is evaluated on the scenario CPMP with the metric NDCG.

A general result of the detailed NDCG evaluation here is that most candidate hybrid decision tree configurations (section 5.5.1) perform best with a split that is only based on ranking according to the NDCG.

5.5.4 Comparison to Preexisting Hybrid Models

Due to time concerns, we could not evaluate the quality of preexisting ranking and regression models ourselves. Instead, we build our evaluation on top of the predictions used in the paper that originally proposed the models ([Han+20]). To evaluate these scenarios with the metrics, described in section 4.1, we slightly altered the code used to evaluate the preexisting hybrid models to include our evaluation metrics. To compare the preexisting models with hybrid decision trees, we evaluated them on the scenarios mentioned in the paper. An overview of the properties of these scenarios is given in Figure 5.30. Note that the scenarios CPMP-2015 and CSP-2010 are also part of the original evaluation.

Scenarios	# Instances	# Unsolved	# $\mathcal{A}$	# Features	C
CPMP-2015	527	0	4	22	3600
CSP-2010	2024	253	2	86	5000
MIP-2016	218	0	5	143	7200
SAT11-HAND	296	77	15	115	5000
SAT11-INDU	300	47	17	115	1200
SAT11-RAND	600	108	9	115	5000

Fig. 5.30.: Overview of the properties of the ASlib scenarios and their properties that are used to compare hybrid decision trees to preexisting hybrid approaches. Unsolved refers to instances all algorithms are terminated.

.

The following figures used to compare different approaches with combined loss functions show the evaluation results in the same structure as other evaluations. The only difference is that the quality of all models is subject to change for different λ values.

The results of the comparison are discussed in three sections:

1. First, we discuss the results evaluated with the scoring functions performance regret and percentage of unsolved instances.

2. Then, we discuss the results evaluated with par10 and NDCG. This evaluation is divided into two parts. The discussion of scenarios on which hybrid decision trees perform well and the discussion of scenarios on which hybrid decision trees perform in comparison.

3. Lastly, the results of the comparison are summarized.

Discussion of Performance Regret and Percentage of Unsolved Instances From the results of the quality comparison of hybrid decision trees and other hybrid models according to either the performance regret in Figure 5.31 or the percentage of unsolved instances in Figure 5.32, one can deduce the following results::

- The other hybrid models outperform hybrid decision trees by a large margin in both metrics.
- All hybrid decision tree configurations perform best for $\lambda \geq 0$.
- All hybrid decision tree configurations of depth 1 have the same quality for all $\lambda \geq 0.1$. This might imply that the ranking error overpowers the regression error.
- All hybrid decision trees of depth greater than 1 (except depth two modified position error) have the highest quality for a true combined loss.
- The neural network trained with the hinge loss appears perform optimal.
- All other preexisting hybrid models have the best quality according to the performance regret for some $\lambda \in \{0.1, 0.2, ..., 0.9\}$.
- The preexisting models appear to solve each instance. This means that for each instance, the predicted algorithm does not need to be terminated due to exceeding the threshold C.

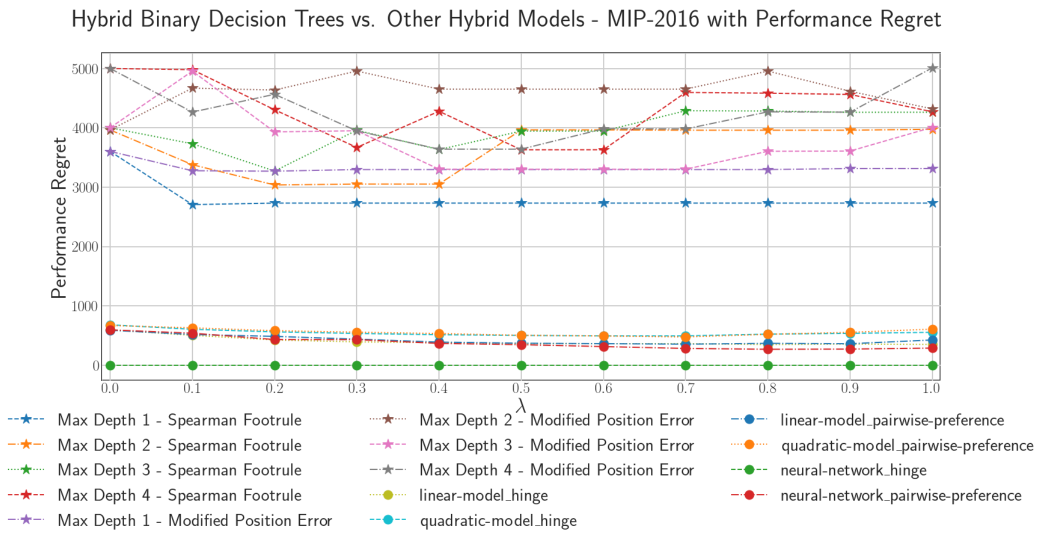


Fig. 5.31.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario MIP-2016 according to the performance regret.

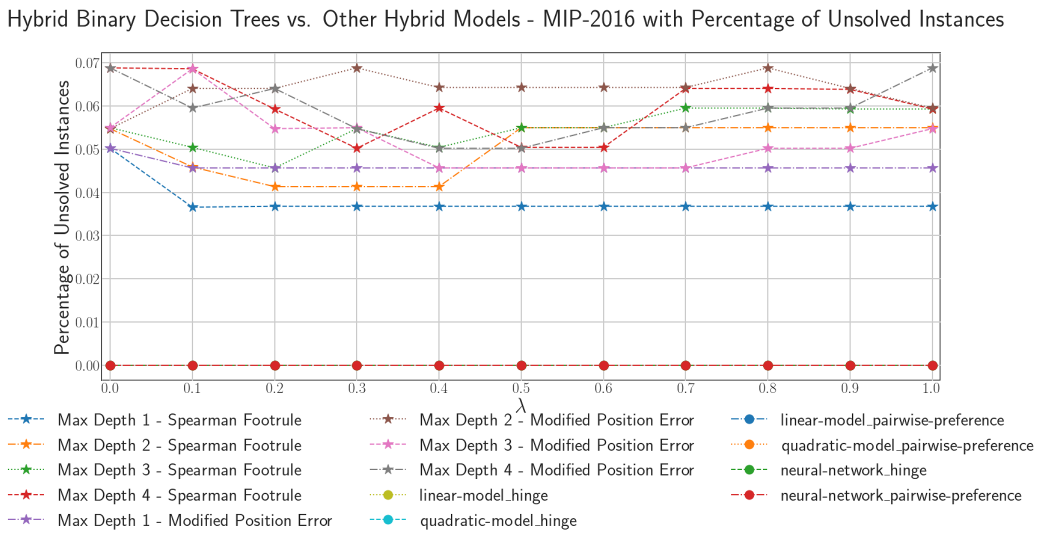


Fig. 5.32.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario MIP-2016 according to the percentage of unsolved instances.

Similar results are seen on all other scenarios.

Discussion of Par10 and NDCG The results of an evaluation with par10 and the NDCG differ from the results above. They indicate that hybrid decision trees perform adequate or better than the preexisting models on MIP-2015 and SAT11-HAND.

Evaluation of Scenarios where Hybrid Decision Trees Perform Well As mentioned above, the results on MIP-2015 and SAT-HAND differ from the other results. This means that there are hybrid decision tree configurations that outperform some other models according to the par10 metric. On SAT11-INDU, the four best performing models are hybrid decision trees. This can be seen in Figure 5.33 and Figure 5.34.

In addition, the following results can be concluded from the figures:

- The preexisting models perform worst for $\lambda = 0$ and best for $\lambda = 1$ on SAT11-INDU. While the quality of the predictions does not improve with every $\lambda \mapsto \lambda + 0.1$, there is still a general trend that the quality of the model improves for bigger λ values.

- On SAT11-INDU all hybrid decision tree configurations, but the hybrid decision tree trained with spearman footrule and maximum depth 4, have the highest quality for some $\lambda \in \{0.1, ..., 0.9\}$. However, the hybrid decision tree trained with spearman footrule and maximum depth four also has a quality that rivals its best quality at $\lambda = 0.5$.

- Some hybrid decision trees that are of higher quality than the preexisting models.

- On MIP-2016, the preexisting models perform best for a combined ranking and regression loss, with a tendency for the ranking loss to have a larger impact on the choice of split than the regression loss.

- On MIP-2016, hybrid decision trees still perform best for $0 \neq \lambda \neq 1$ except for the tree with maximal depth 1 and the spearman footrule (which is the best hybrid decision tree configuration). However, for all trees trained with a spearman footrule and maximal depth less than 4, the ranking error overpowers the regression error for high λ.

- The λ values for which a hybrid decision tree configurations perform best are distributed over almost all evaluation points. There does not seem to be an underlying structure.

- Some hybrid decision tree configurations have higher quality than some preexisting models. However the best configurations of the preexisting models are of higher quality than the best hybrid decision tree configurations

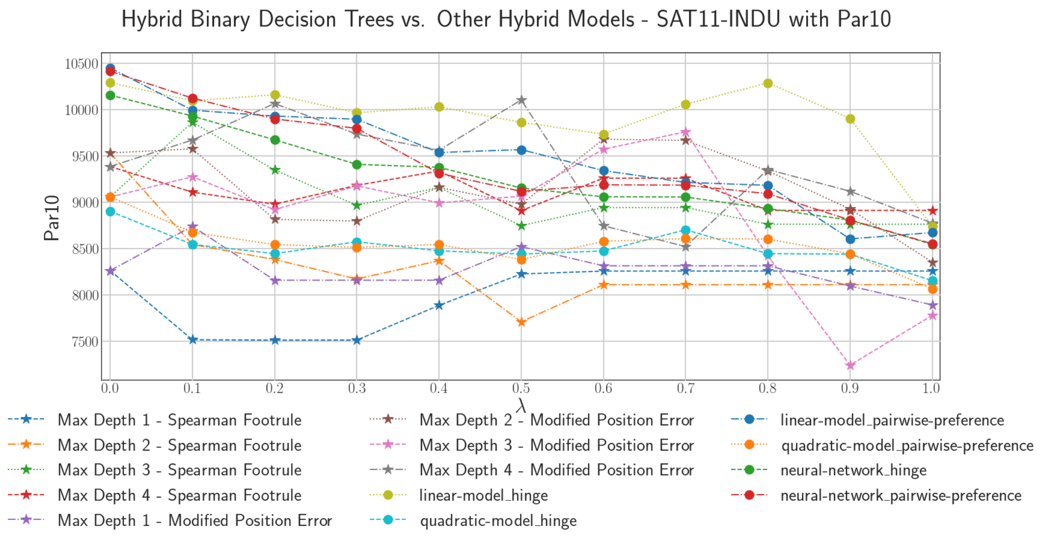


Fig. 5.33.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario SAT11-INDU according to par10.

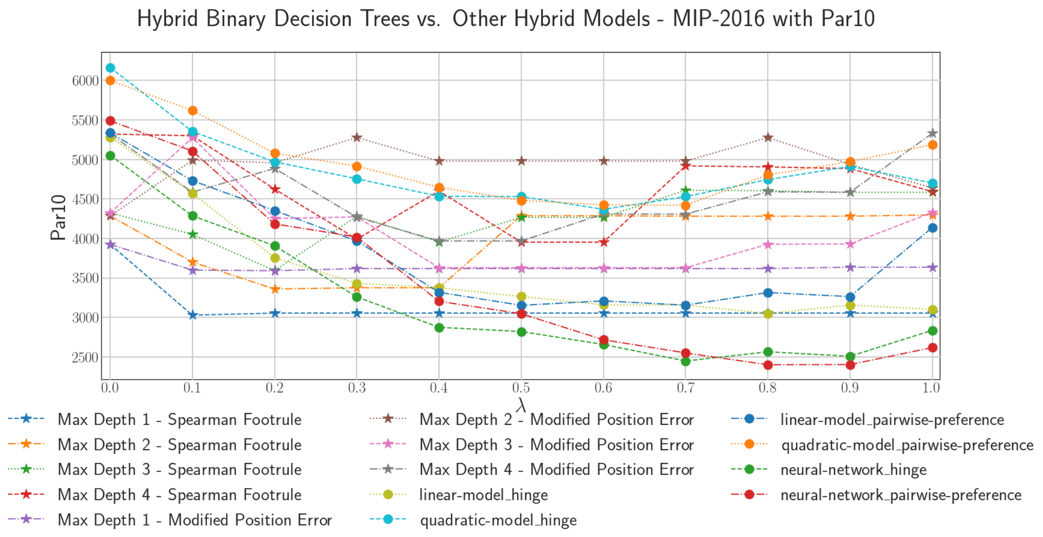


Fig. 5.34.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario MIP-2016 according to par10.

Comparing the two different approaches with NDCG, the result is that the best performing models are different hybrid decision tree configurations. This can be seen in Figure 5.35 and Figure 5.36. According to NDCG, one can find the following results:

- On SAT11-INDU, trees trained with the spearman footrule (either maximum depth 1 or maximum depth of 2) perform best for a true combined loss with $\lambda = 0.3$.

- On MIP-2016 hybrid decision trees perform best if the split is solely based on the ranking loss.

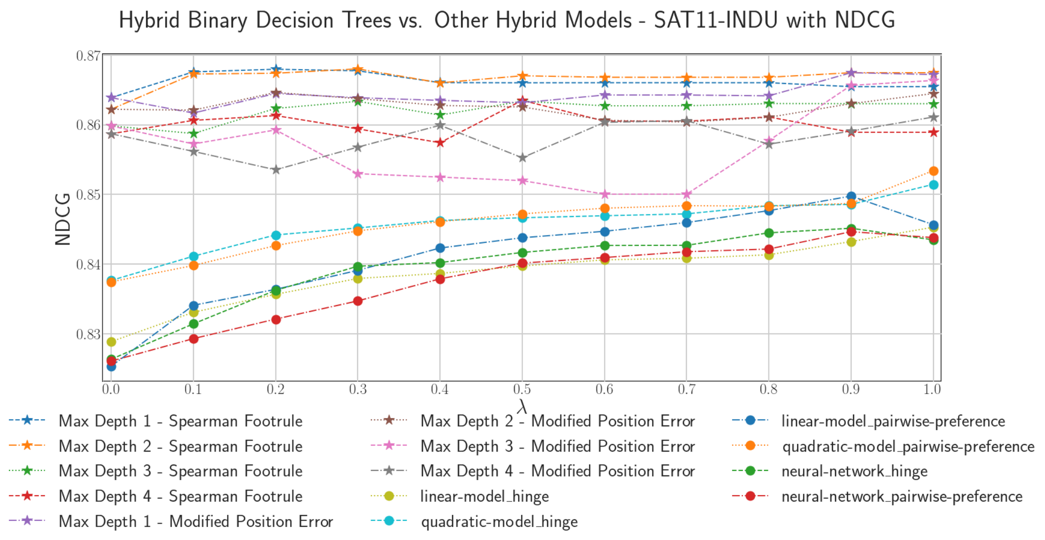


Fig. 5.35.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario SAT11-INDU according to NDCG.

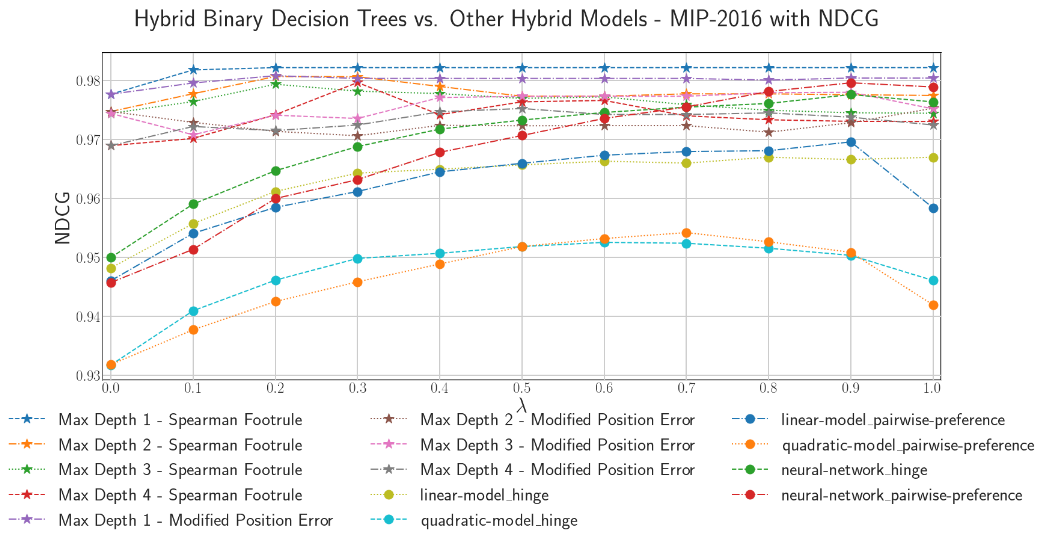


Fig. 5.36.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario MIP-2016 according to NDCG.

It can be seen that hybrid decision trees of depth 1 can rival the quality of preexisting models. Since the number of instances of MIP-2016 and SAT11-INDU is lower than average this further supports the idea that hybrid decision trees perform best if the depth is chosen depending on the number of instances. This is further supported by the fact that on SAT11-HAND (which also has few instances) the quality margin also appears to be smaller. However, there is not enough data for this to be more than a hypothesis.

Scenarios Where Hybrid Decision Trees Perform poorly On all other scenarios than SAT11-INDU and MIP-2016, hybrid decision trees get outperformed by all other models. A good example of this is the evaluation on SAT11-HAND given in Figure 5.37 and Figure 5.38. Since the results that focus solely on the hybrid decision trees are largely the same as above, they are not explained in detail. However one should note that hybrid decision trees of the same loss functions show a very similar behavior here.

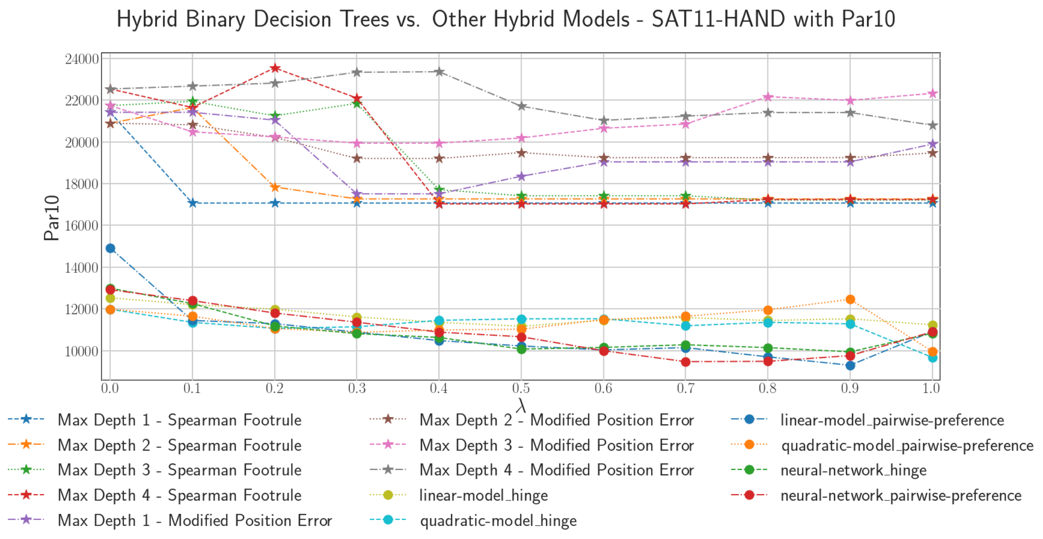


Fig. 5.37.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario SAT11-HAND according to par10.

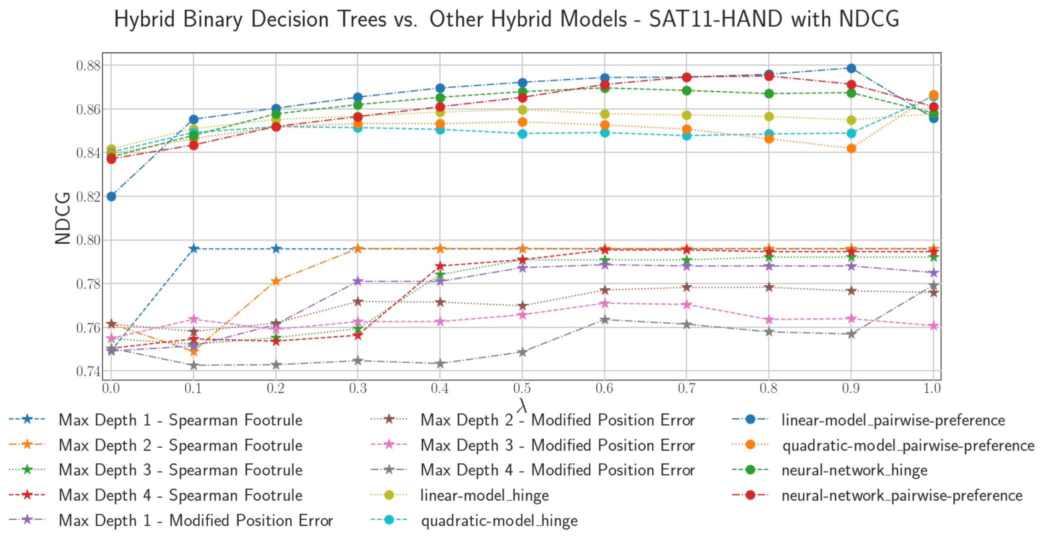


Fig. 5.38.: Comparison of hybrid decision trees to preexisting hybrid models on the scenario SAT11-HAND according to NDCG.

Summary The evaluation results in comparison to other models can be summarized with the fact that hybrid decision trees get generally outperformed by preexisting models. However, there are scenarios for which the performance of hybrid decision trees can rival the performance of these models (according to some metrics).
Another result is that while there is a general tendency for preexisting models to perform best for a higher λ (that does not equal 1) this generally can not be seen for hybrid decision trees to the same extend. However there scenarios for which some hybrid decision trees trained with the same loss function perform very similar (with changing λ) like Figure 5.37 .
Lastly the comparison indicated that the idea that the maximal depth of hybrid decision trees should be configured according to the number of instances is further supported by this evaluation.

5.5.5 Runtime Comparison of Hybrid Decision Trees to other Approaches

While the prediction quality of different models is essential, it is not the only factor determining the best model. Therefore, we also evaluate the time to train hybrid decision trees and other models on the given scenarios. The results of this evaluation can be found in Figure 5.39. This diagram is based on the average training time in seconds on the folds as explained in section 5.1. However, the machine was idle, and no multiprocessing was used.

The results of this training process are:

- The per algorithm decision tree regressor is the model with the by far best training runtime.

- The runtime of training hybrid decision trees is smaller than training an expectation survival forest on all scenarios but CSP-2010 and CPMP-2015. This might indicate that hybrid decision trees seem to perform comparatively worse if there are fewer candidate algorithms. This is further supported by the training time needed for ASP-POTASSCO since it has the third-lowest amount of algorithms, and the training of hybrid decision trees needs the third most resources compared to the survival forest.

- The amount with which the training time needed for hybrid decision trees increases faster than linear based on the depth.

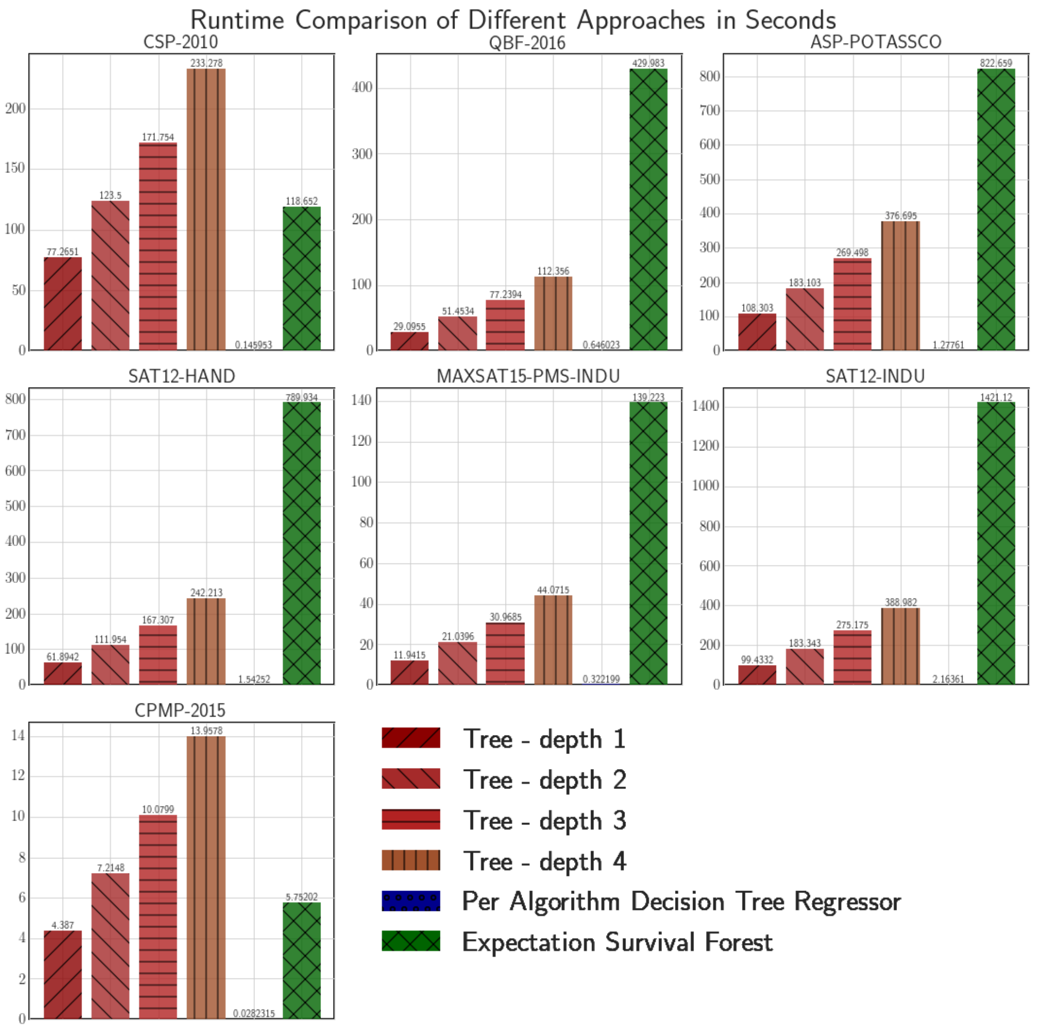


Fig. 5.39.: Comparison of of different models' training runtimes depending on the scenario.

6 Conclusion on Hybrid Decision Trees

The prediction quality of all hybrid decision tree configurations proposed in this thesis is lower than the quality of state-of-the-art approaches. However, this thesis still gives interesting results discussed in section 6.1. Building on these results, possible avenues of future exploration are discussed in section 6.2.

6.1 Evaluation Results

Different results can be concluded from the evaluation in chapter 5. Some of them are general results, some deal with the quality of hybrid decision tree components, and some are are results of the comparison of hybrid decision trees and other algorithm selection models. The discussion of the evaluation results is structured as follows:

- Preceding a discussion of results that answer our research questions, some general evaluation results are explained in section 6.1.1.
- In section 6.1.2 results of our components-wise evaluation are discussed, which answers our first and second research questions.
- Then, the results of a comparison between hybrid decision trees and other state-of-the-art approaches (in both model quality and training runtime) are discussed to answer the third, fourth, and fifth research questions (section 6.1.3).
- Lastly, the overall results are discussed.

6.1.1 General Results

The following general results can be derived from our evaluation: Firstly, the evaluation metrics percentage of unsolved instances and par10 (as defined in section 4.1.1) show some contradicting behavior caused by the definition of unsolved instances. In the percentage of unsolved instances, we do not consider the effort of calculating features to determine whether an algorithm solves an instance. However, in par10, the effort of calculating features is added to the ground truth performance of the predicted algorithm, and if the resulting performance exceeds the cutoff, the prediction is considered unsolved. This mismatch complicates an interpretation of the evaluation results.
The second general result is that the quality of algorithm selection models depends on the scenario. The comparison of hybrid decision trees to other models often shows that the best model is dependent on the scenario. In addition, the scenario CSP-2010 often causes different performances of hybrid decision trees.
The last noteworthy result mentioned here is that some ranking losses outperform the regression loss on some scenarios. This commonly happens for the spearman footrule.

6.1.2 Component-Wise Evaluation Results on Hybrid Decision Trees

The component-wise evaluation is mainly used to find suitable candidate configurations compared to other algorithm selection approaches. The results are best discussed separately for each component.

The evaluation of different split loss configurations (different ranking losses and different λ) showed that on most scenarios, the split is best chosen based on both ranking and regression loss functions. The best ranking loss functions for the choice of split are the modified position error (Equation 3.44) and the spearman footrule (Equation 3.27). The other losses either worked poorly with the hybrid decision tree structure or as a loss function in a convex combination.
In evaluating different borda functions (also used in calculating a split loss), our results indicate that no overall best borda function exists. However, hybrid decision trees trained with mean and median ranking had the highest average quality.

The evaluation of different stopping criteria indicated that the best stopping criterion is the maximal depth of a tree. Therefore a node is only split if the maximum depth is not reached, whereas the choice of maximal depth strongly depends on the given scenarios. The results imply that trees of smaller depth might perform better if either a small set of training instances or a small set of candidate algorithms exist.

6.1.3 Evaluation Results of the Comparison of Hybrid Decision Trees with other models

Evaluation Based on the Quality of Predictions As mentioned above, the quality of different models is dependent on the given scenarios.
This is supported by the comparison of hybrid decision trees with the per algorithm regressors. This comparison shows that hybrid decision trees perform better than regressors on some scenarios and more poorly on others.
However, the comparison with the expectation survival forest (candidate survival forest model chosen in this thesis) shows that hybrid decision trees generally perform worse. The only exception is the evaluation on CSP-2010 with the metric NDCG (a measure of the overall ranking quality, not the prediction of the best algorithm).

The most interesting comparison is of hybrid decision trees and preexisting algorithm selection models. Interestingly there is no general similarity in the prediction quality of the two models. While the preexisting hybrid models mostly perform best for a focus on ranking without disregarding regression, hybrid decision trees have no such general tendency over all loss functions. However, they perform best for one λ that disregards neither ranking nor regression. As in preexisting models, hybrid decision trees trained with the same ranking loss but different stopping criteria often perform similarly on the same scenario.
In prediction quality, hybrid decision trees are worse than the preexisting models on almost every scenario (with almost every metric). The only exceptions are the scenarios SAT11-INDU and MIP-2016 (both have relatively few instances) on which hybrid decision trees perform comparatively well or some hybrid decision tree configurations even beat the best preexisting models.

Implications for the Training Resources Needed As in the quality evaluation of trees, the training resources needed are dependent on the underlying scenario. The

results of this evaluation can be divided into two parts. Results that are only based on hybrid decision trees and results that are based on the comparison to other models.
While it is obvious that the time needed to train a hybrid decision tree increases if the maximal depth is increased, it is noteworthy that the runtime increases faster than linear to the depth.
Compared to the per algorithm decision tree regressor, both hybrid decision trees and expectation forests perform poorly. They need much longer training.
Survival forests need more training time than hybrid decision trees for most scenarios. However, on the scenarios, CSP-2010 and CPMP-2015 expectation forests need less training time than hybrid decision trees. This might be induced by the small number of candidate algorithms in CSP-2010 and CPMP-2015.

6.1.4 Overall Result

Overall the prediction quality of hybrid decision trees is worse than of other state-of-the-art approaches on most scenarios. However, there are a few scenarios on which hybrid decision trees rival per algorithm decision tree regressors or preexisting hybrid approaches.
The result that hybrid ranking and regression loss functions increase the prediction quality and that some hybrid decision tree configurations rival other approaches are by far the most noteworthy results and imply that either further research could improve the quality of hybrid decision trees, other methods of combining ranking and regression should be explored, or the quality of combined ranking and regression models should be explored for approaches outside of the scope of algorithm selection.

6.2 Future Work

The results of this thesis imply possible directions for future research. These can be divided into future work that aims to improve hybrid decision trees as a model for algorithm selection and other research encouraged by this thesis's result.

Future Work on Hybrid Decision Trees There are various ways to extend our research of hybrid decision trees over the scope of this thesis. Firstly one could introduce more hybrid decision tree components. This is especially interesting for components that impact the split loss function since one central issue of combining ranking and regression is finding loss functions that work well together. Additionally, one could extend the evaluation in this thesis with more λ values or datasets, which could lead to more impactfull and concrete results.
Compared to these more minor additions, one could also expand hybrid decision trees to a forest of hybrid decision trees or explore the usage of feature preprocessing algorithms for hybrid decision trees.

6.2.1 Future Work Motivated by Hybrid Decision Trees

Since a combination of ranking and regression is an improvement over plain ranking/regression, one could also try to combine these two approaches for other models. An example would be creating a model consisting of separate ranking and regression forests with a consensus mechanism. Another combined ranking and regression approach would be utilizing a regression model trained with a performance function based on a combination of performances and ranking.

In addition to applying ranking and regression to algorithm selection, one could try to apply hybrid decision trees in other machine learning areas. This is interesting since the quality of hybrid decision trees is dependent on the underlying scenario and there might be scenarios for which hybrid decision trees are a better fit.

Bibliography

[Agr10] Alan Agresti. *Analysis of ordinal categorical data*. Vol. 656. John Wiley & Sons, 2010 (cit. on pp. 25, 33).

[Bis+16] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, et al. "ASlib: A Benchmark Library for Algorithm Selection". In: *Artificial Intelligence Journal (AIJ)* 237 (2016), pp. 41–58 (cit. on pp. 6, 38).

[Bot18] Alexei Botchkarev. "Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology". In: *CoRR* abs/1809.03006 (2018). arXiv: `1809.03006` (cit. on p. 20).

[Bre+84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984 (cit. on pp. 3, 10, 14, 36, 41).

[CHH09] Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier. "Decision Tree and Instance-Based Learning for Label Ranking". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 161–168 (cit. on pp. 3, 8, 14).

[Han+20] Jonas Hanselle, Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. "Hybrid Ranking and Regression for Algorithm Selection". In: *KI 2020: Advances in Artificial Intelligence*. Ed. by Ute Schmid, Franziska Klügl, and Diedrich Wolter. Cham: Springer International Publishing, 2020, pp. 59–72 (cit. on pp. 3, 7, 10, 11, 25, 27, 29, 31, 67, 78).

[HF10] Eyke Hüllermeier and Johannes Fürnkranz. "On loss functions in label ranking and risk minimization by pairwise learning". In: *Journal of Computer and System Sciences* 76.1 (2010), pp. 49–62 (cit. on p. 24).

[Hül+08] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. "Label ranking by learning pairwise preferences". In: *Artificial Intelligence* 172.16-17 (2008), pp. 1897–1916 (cit. on pp. 2, 22).

[Hut+15] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. "Algorithm Runtime Prediction: Methods and Evaluation (Extended Abstract)". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 4197–4201 (cit. on pp. 1–3, 6, 14).

[Ker+18] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. “Automated Algorithm Selection: Survey and Perspectives”. In: *CoRR* abs/1811.11597 (2018). arXiv: `1811.11597` (cit. on pp. 1, 5, 6).

[KKP] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. “Data preprocessing for supervised leaning”. In: () (cit. on p. 39).

[Lin10] Shili Lin. “Rank aggregation methods”. In: *WIREs Computational Statistics* 2.5 (2010), pp. 555–570. eprint: `https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.111` (cit. on p. 28).

[OHL15] Richard Jayadi Oentaryo, Stephanus Daniel Handoko, and Hoong Chuin Lau. “Algorithm Selection via Ranking”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 1826–1832 (cit. on p. 8).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 41).

[Ric76] John R. Rice. “The Algorithm Selection Problem.” In: *Advances in Computers* 15 (1976), pp. 65–118 (cit. on pp. 1, 5).

[RM07] Lior Rokach and Oded Z Maimon. *Data mining with decision trees: theory and applications*. Vol. 69. World scientific, 2007 (cit. on p. 14).

[RN02] Stuart Russell and Peter Norvig. “Artificial intelligence: a modern approach”. In: (2002) (cit. on pp. 35, 39).

[Tor+22] Alexander Tornede, Lukas Gehring, Tanja Tornede, Marcel Wever, and Eyke Hüllermeier. “Algorithm selection on a meta level”. In: *Machine Learning* (2022), pp. 1–34 (cit. on p. 22).

[Tor+20] Alexander Tornede, Marcel Wever, Stefan Werner, Felix Mohr, and Eyke Hüllermeier. “Run2Survive: A Decision-theoretic Approach to Algorithm Selection based on Survival Analysis”. In: *Proceedings of The 12th Asian Conference on Machine Learning*. Ed. by Sinno Jialin Pan and Masashi Sugiyama. Vol. 129. Proceedings of Machine Learning Research. PMLR, Nov. 2020, pp. 737–752 (cit. on pp. 2, 6, 7, 12, 31, 41, 73).

[Val+09] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. “Learning to Rank by Optimizing NDCG Measure”. In: *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*. Ed. by Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta. Curran Associates, Inc., 2009, pp. 1883–1891 (cit. on pp. 8, 34).

[VG10] Shankar Vembu and Thomas Gärtner. “Label Ranking Algorithms: A Survey”. In: *Preference Learning*. Ed. by Johannes Fürnkranz and Eyke Hüllermeier. Springer, 2010, pp. 45–64 (cit. on p. 8).

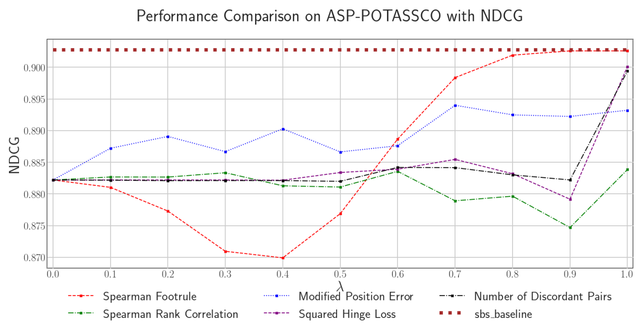
[Vir+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272 (cit. on p. 34).

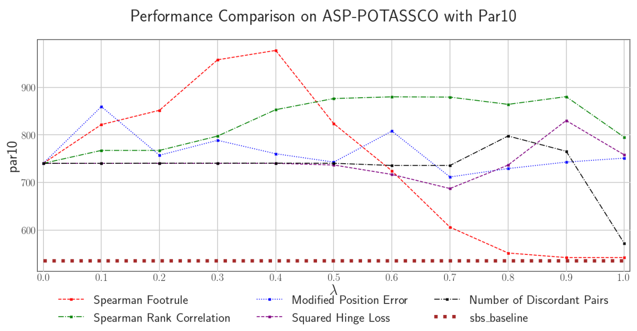
[Xu+] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT”. In: *Principles and Practice of Constraint Programming – CP 2007*. Springer Berlin Heidelberg, pp. 712–727 (cit. on pp. 1, 6, 7, 10).

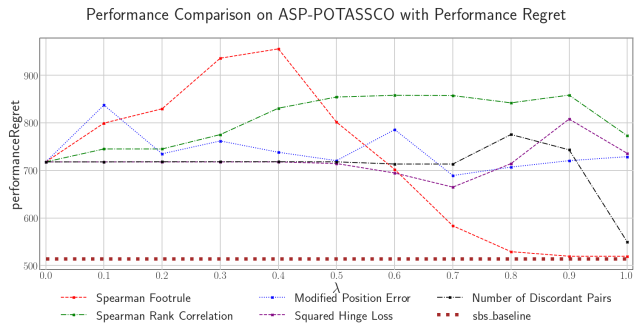
List of Figures

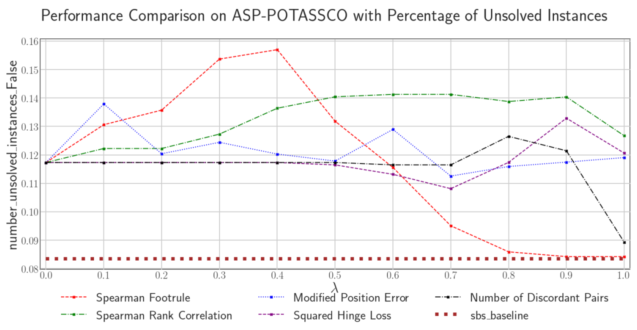
A Further Figures

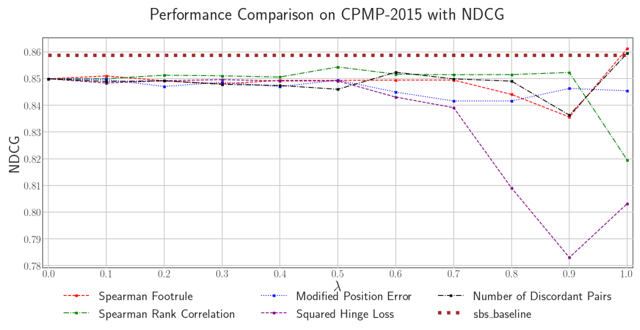
A.1 Loss Evaluation

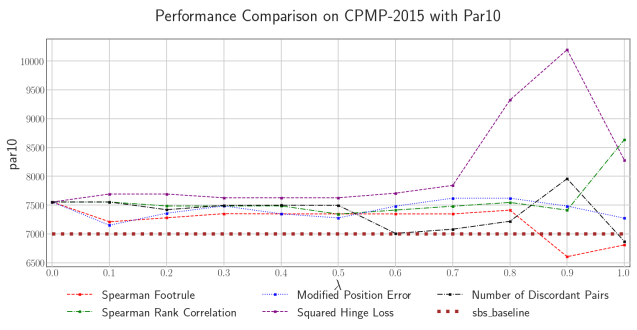


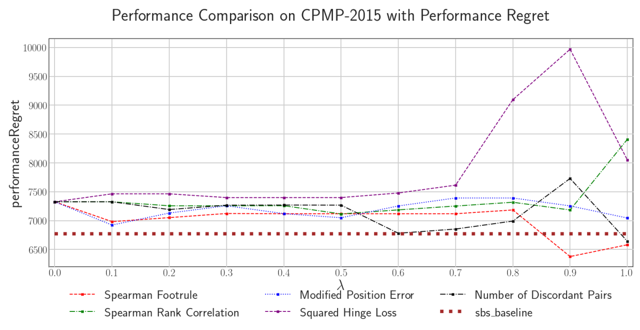


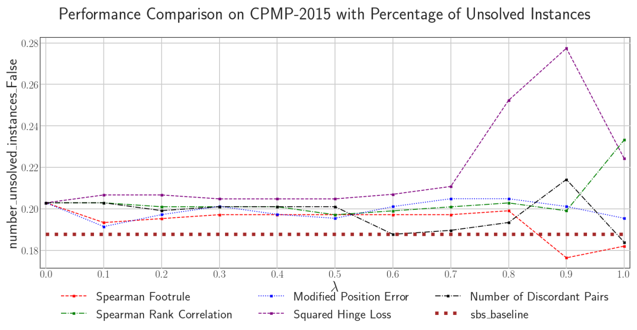


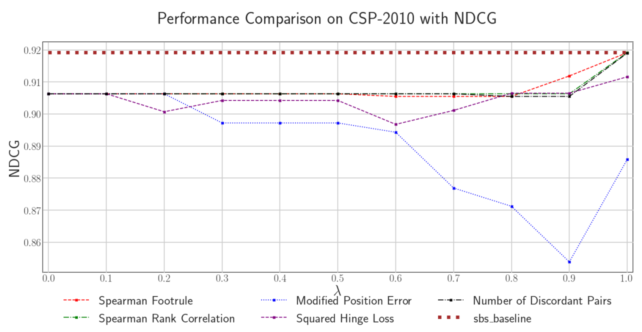


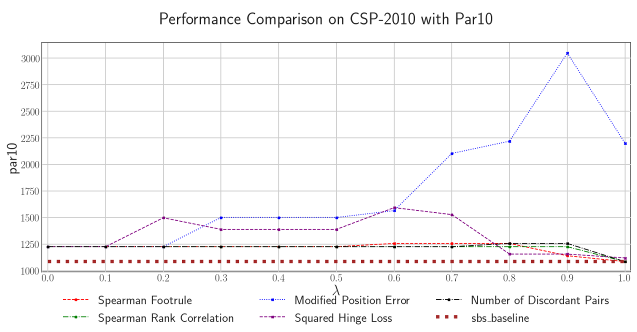


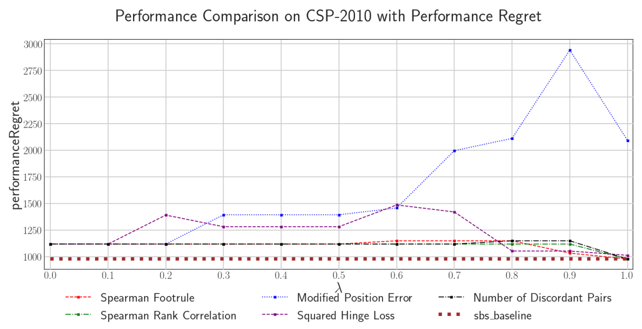


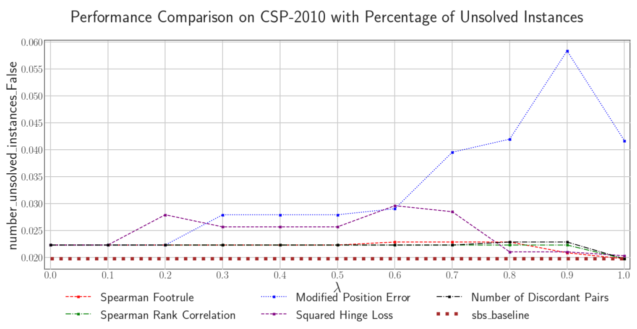


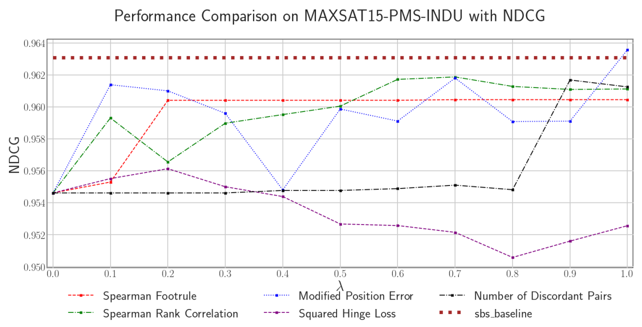


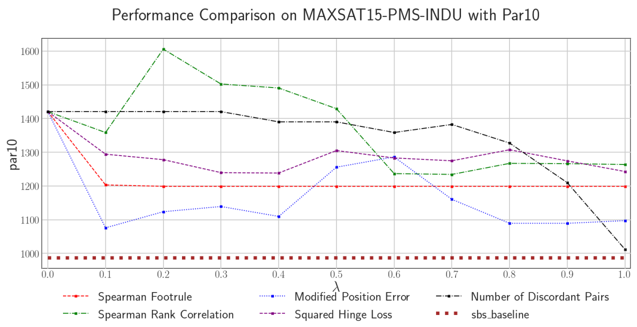

Performance Comparison on CSP-2010 with NDCG
NDCG
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

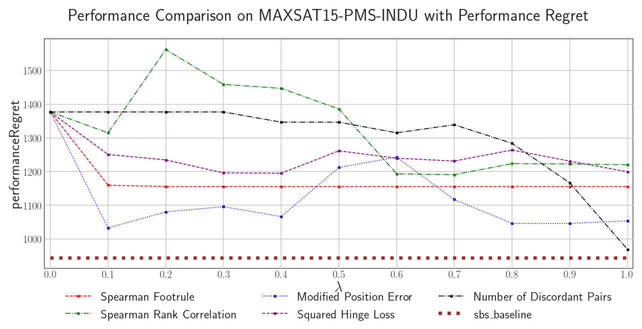

Performance Comparison on CSP-2010 with Par10
par10
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

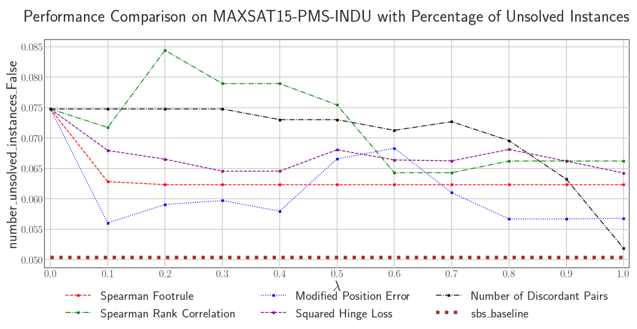

Performance Comparison on CSP-2010 with Performance Regret
performanceRegret
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

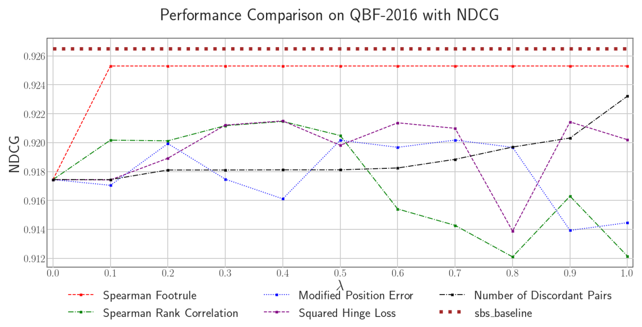

Performance Comparison on CSP-2010 with Percentage of Unsolved Instances
number_unsolved_instances_False
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

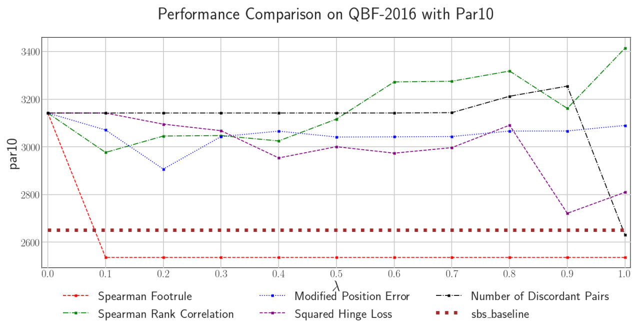

Performance Comparison on MAXSAT15-PMS-INDU with NDCG
NDCG
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

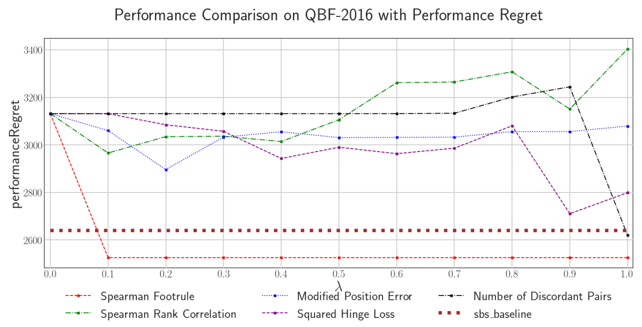

Performance Comparison on MAXSAT15-PMS-INDU with Par10
par10
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

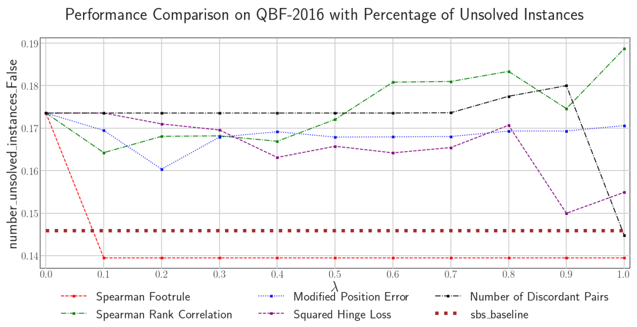

Performance Comparison on MAXSAT15-PMS-INDU with Performance Regret
performanceRegret
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

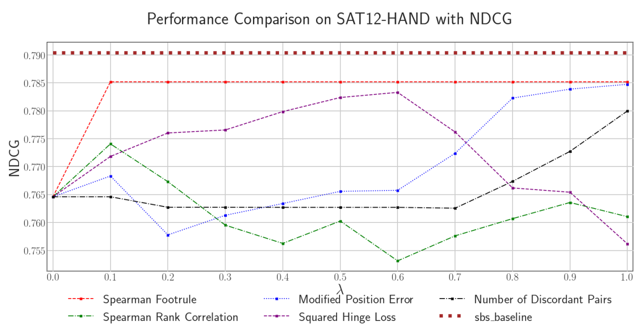

Performance Comparison on MAXSAT15-PMS-INDU with Percentage of Unsolved Instances
number_unsolved_instances_False
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

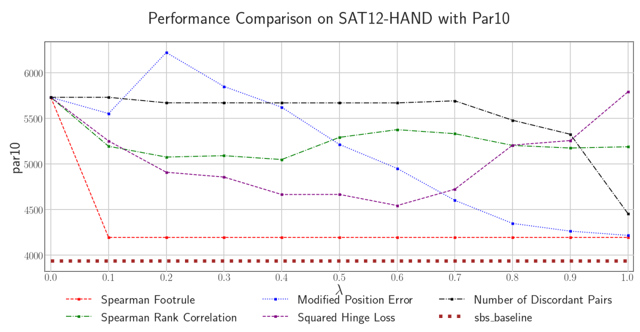

Performance Comparison on QBF-2016 with NDCG
NDCG
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

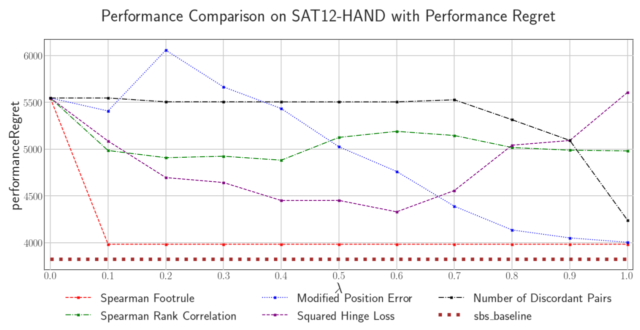

Performance Comparison on QBF-2016 with Par10
par10
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

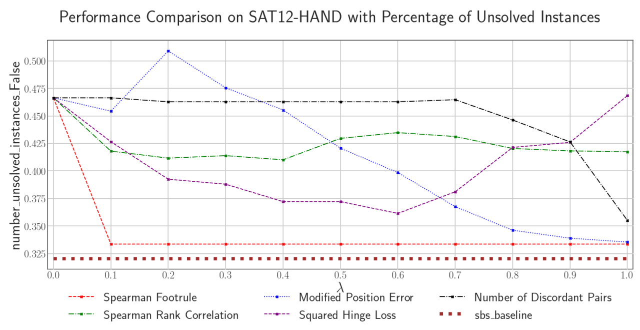

Performance Comparison on QBF-2016 with Performance Regret
performanceRegret
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

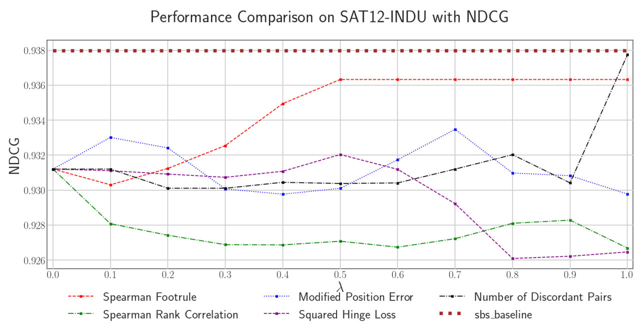

Performance Comparison on QBF-2016 with Percentage of Unsolved Instances
number_unsolved_instances_False
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

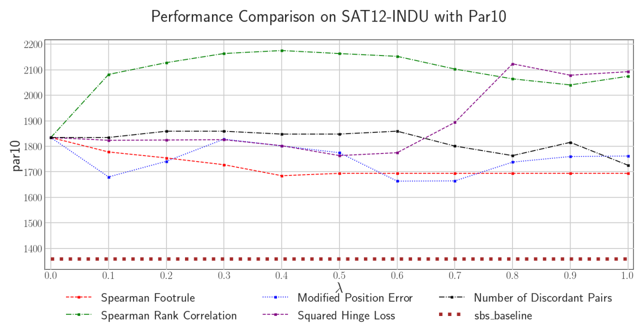

Performance Comparison on SAT12-HAND with NDCG
NDCG
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

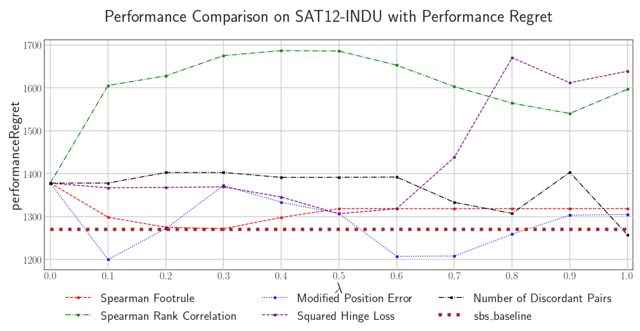

Performance Comparison on SAT12-HAND with Par10
par10
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

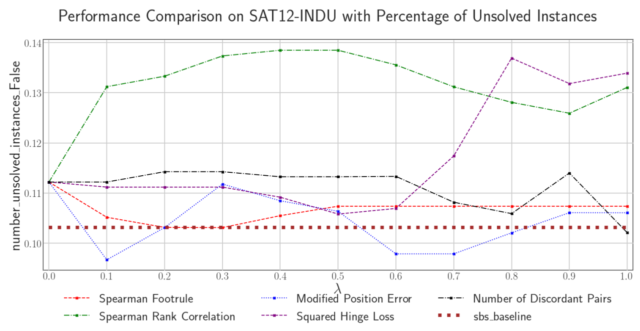

Performance Comparison on SAT12-HAND with Performance Regret
performanceRegret
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline


Performance Comparison on SAT12-HAND with Percentage of Unsolved Instances
number_unsolved_instances_False
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

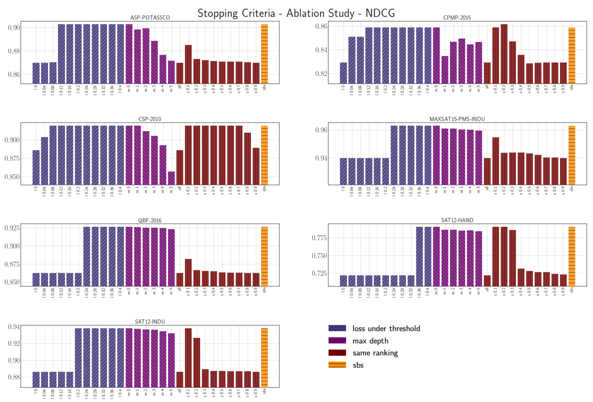

Performance Comparison on SAT12-INDU with NDCG
NDCG
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

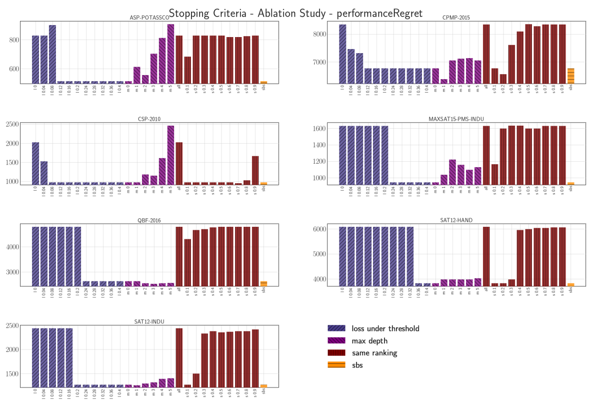

Performance Comparison on SAT12-INDU with Par10
par10
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

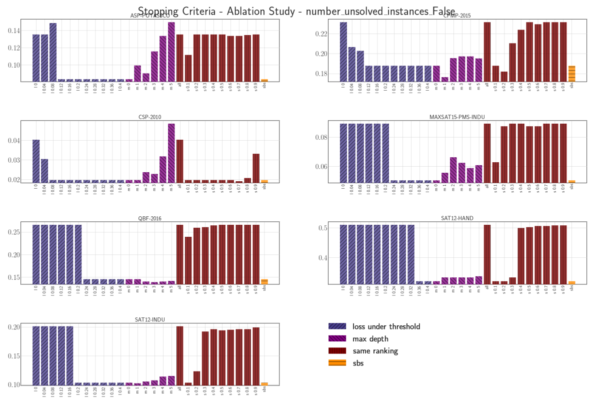

Performance Comparison on SAT12-INDU with Performance Regret
performanceRegret
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

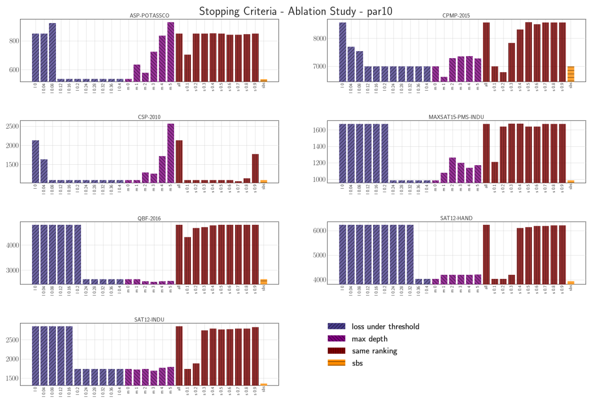

Performance Comparison on SAT12-INDU with Percentage of Unsolved Instances
number_unsolved_instances_False
λ
Spearman Footrule
Spearman Rank Correlation
Modified Position Error
Squared Hinge Loss
Number of Discordant Pairs
sbs_baseline

A.2 Ablation Study on Stopping Criteria



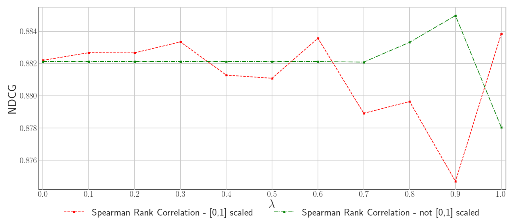




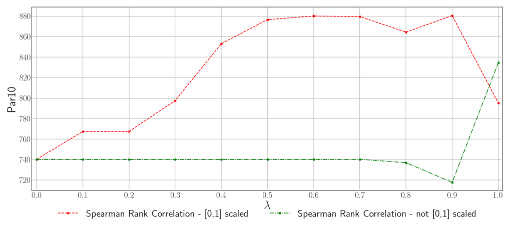


A.3 Hybrid Decision Trees before and after modification

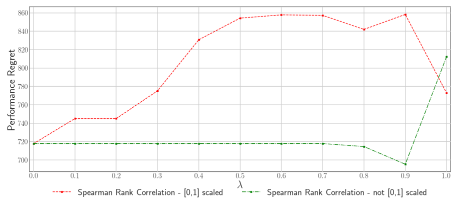
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on ASP-POTASSCO with NDCG



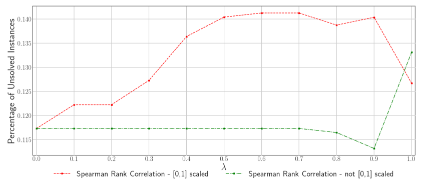
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on ASP-POTASSCO with Par10



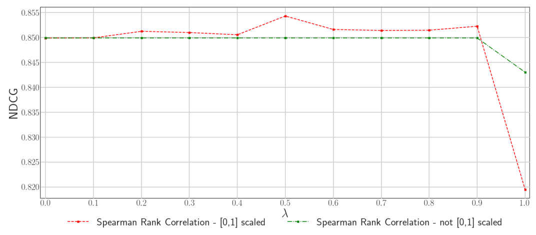
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on ASP-POTASSCO with Performance Regret



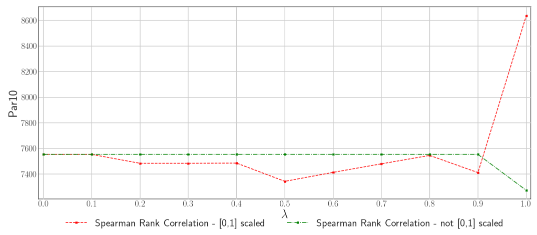
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on ASP-POTASSCO with Percentage of Unsolved Instances



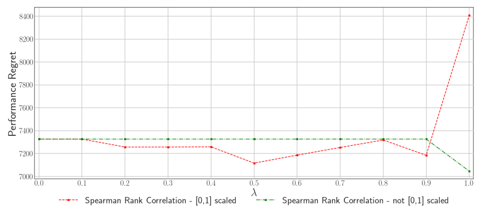
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CPMP-2015 with NDCG



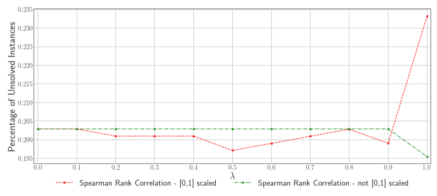
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CPMP-2015 with Par10



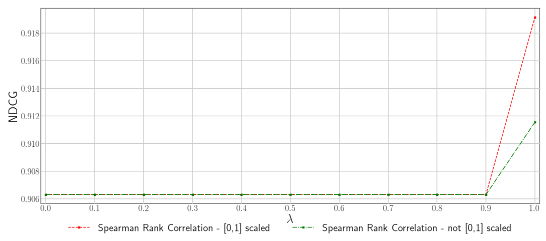
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CPMP-2015 with Performance Regret



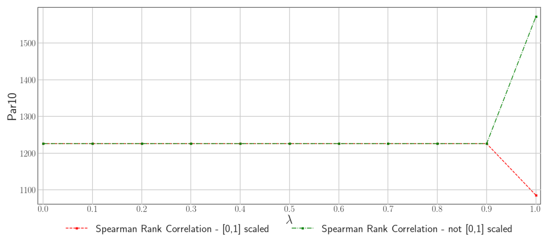
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CPMP-2015 with Percentage of Unsolved Instances



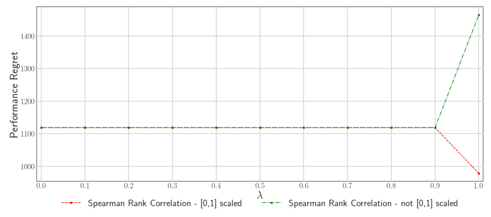
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CSP-2010 with NDCG



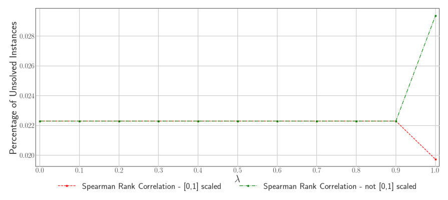
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CSP-2010 with Par10



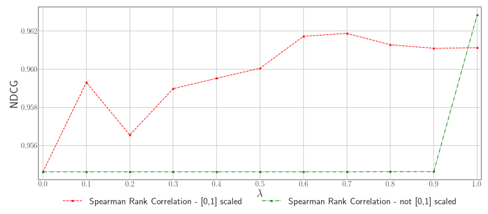
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CSP-2010 with Performance Regret



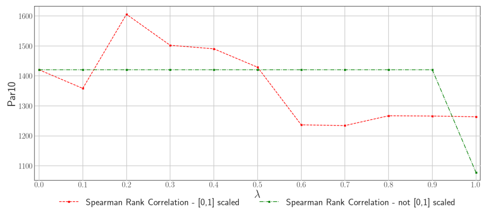
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on CSP-2010 with Percentage of Unsolved Instances



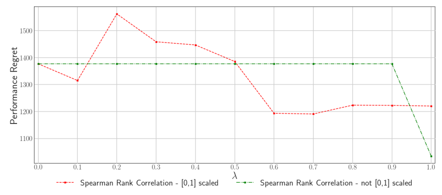
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on MAXSAT15-PMS-INDU with NDCG



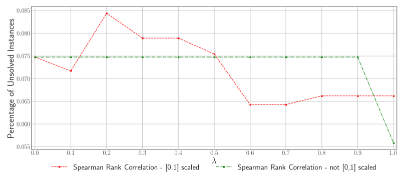
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on MAXSAT15-PMS-INDU with Par10



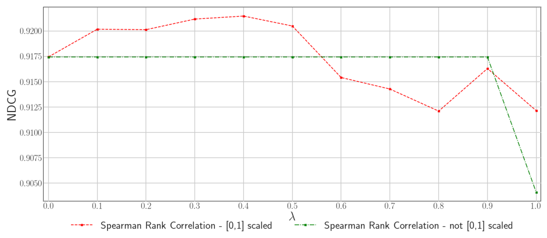
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on MAXSAT15-PMS-INDU with Performance Regret



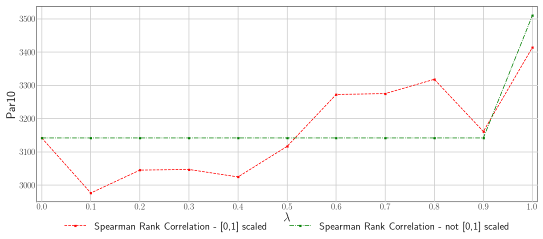
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on MAXSAT15-PMS-INDU with Percentage of Unsolved Instances



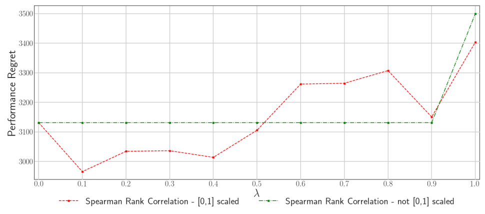
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on QBF-2016 with NDCG



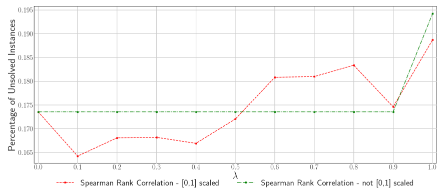
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on QBF-2016 with Par10



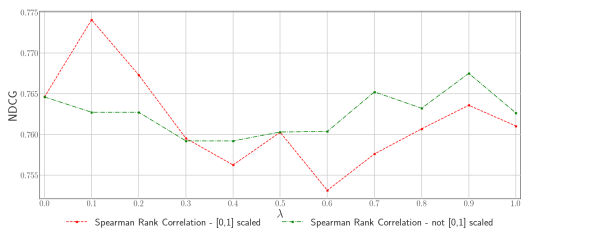
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on QBF-2016 with Performance Regret



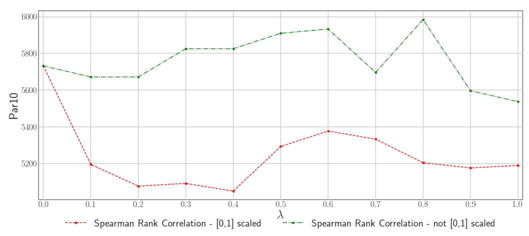
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on QBF-2016 with Percentage of Unsolved Instances



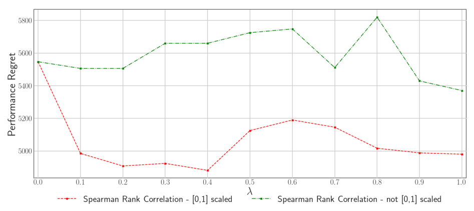
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-HAND with NDCG



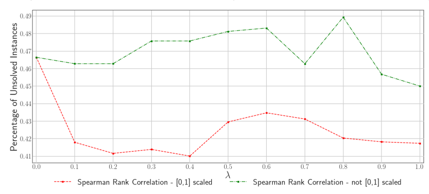
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-HAND with Par10



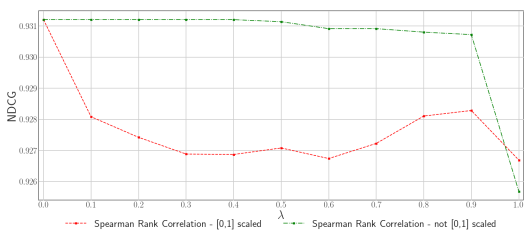
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-HAND with Performance Regret



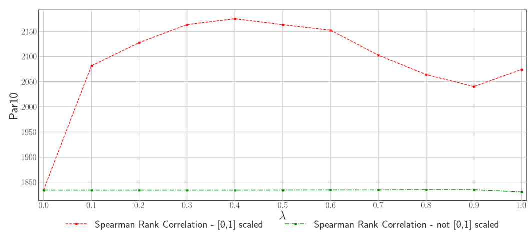
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-HAND with Percentage of Unsolved Instances



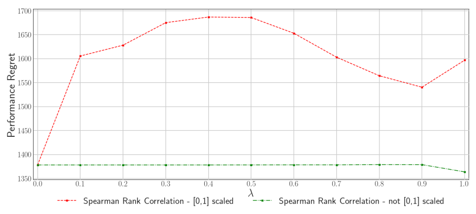
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-INDU with NDCG



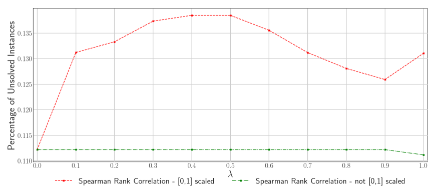
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-INDU with Par10



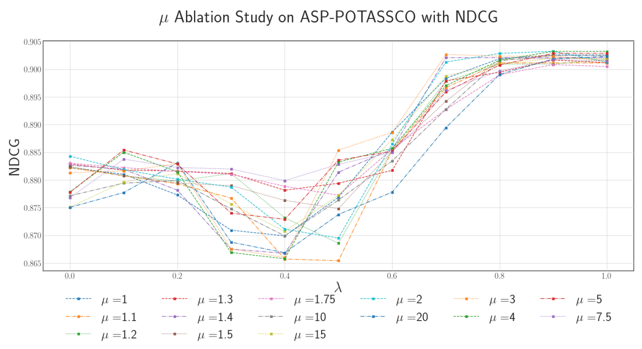
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-INDU with Performance Regret

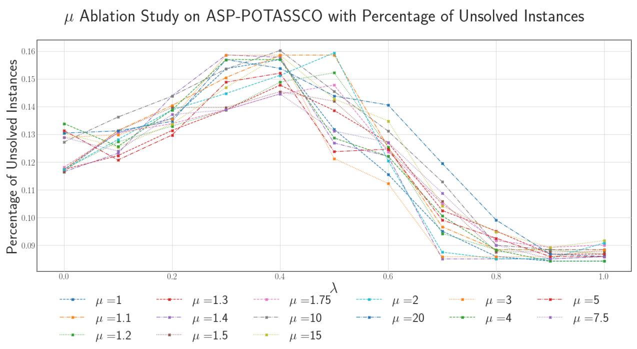


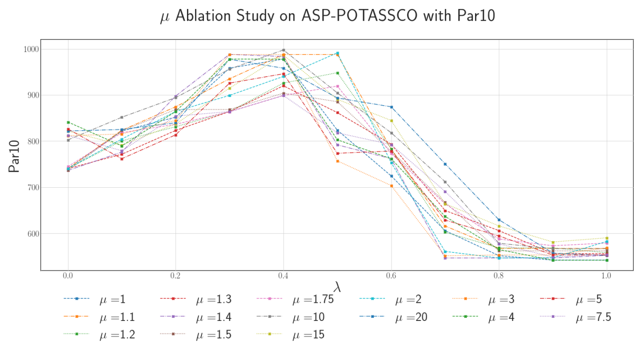
Comparison of Hybrid Binary Decision Trees With, or Without Modified Specification on SAT12-INDU with Percentage of Unsolved Instances

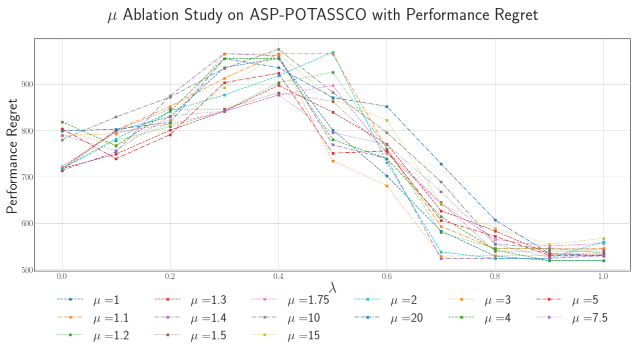


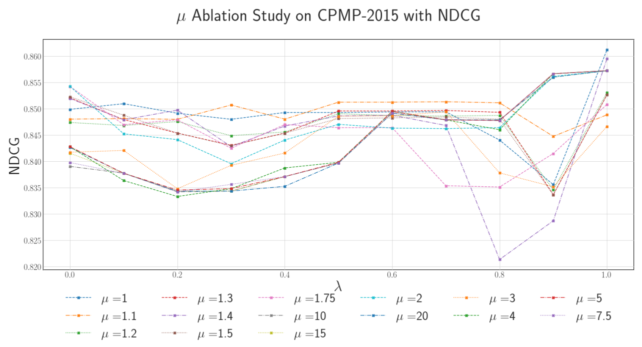
A.4 Ablation Study on Handling of Censored Data

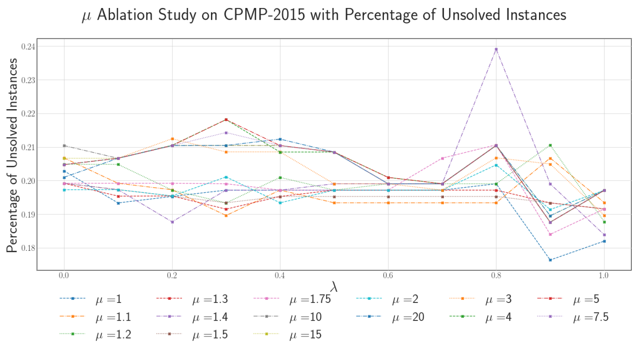


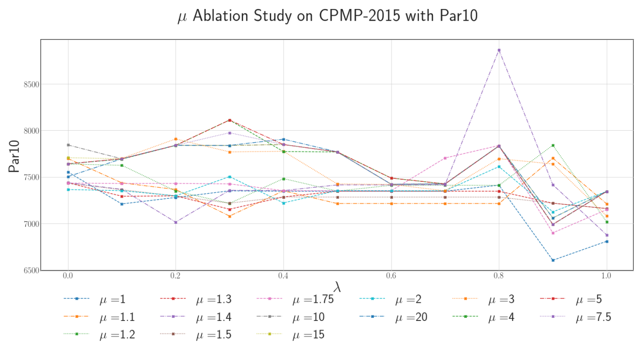


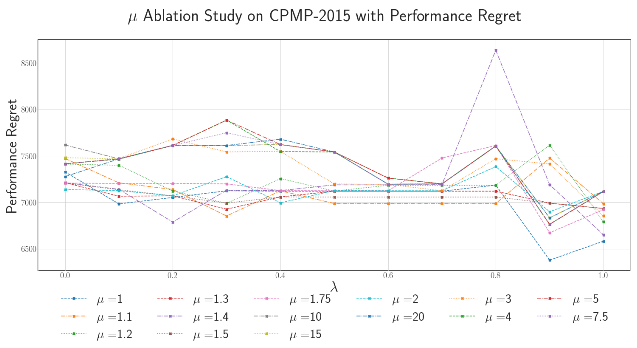


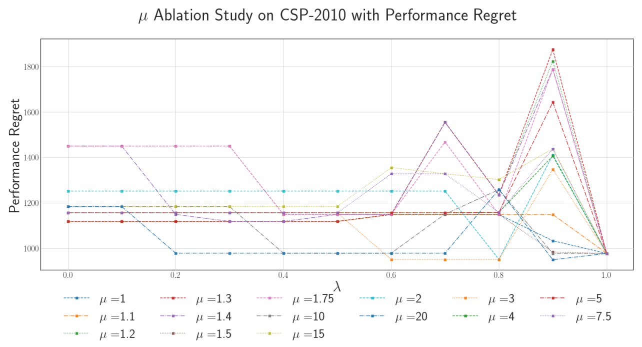


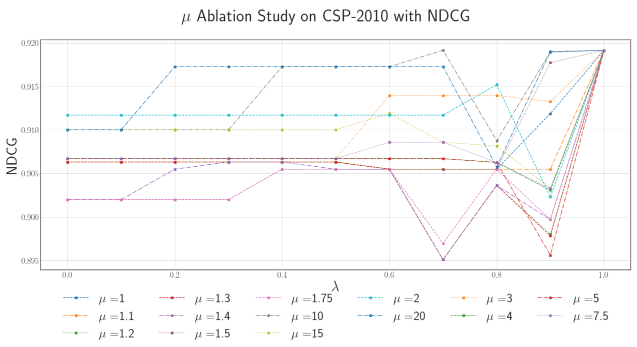


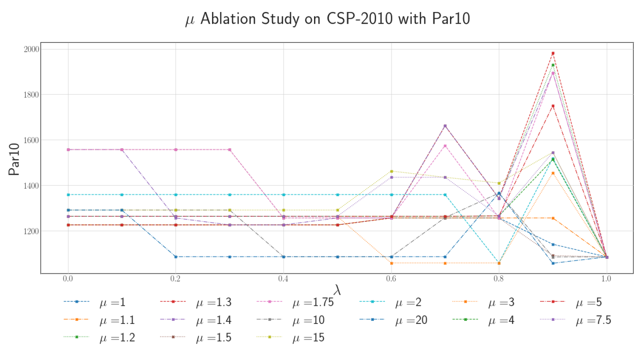


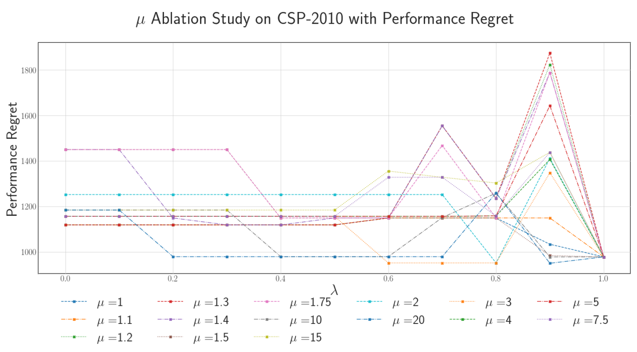


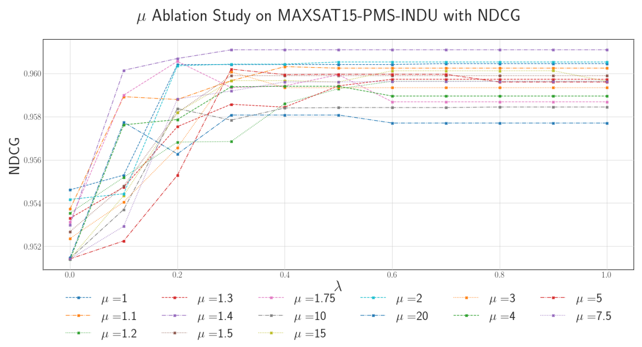


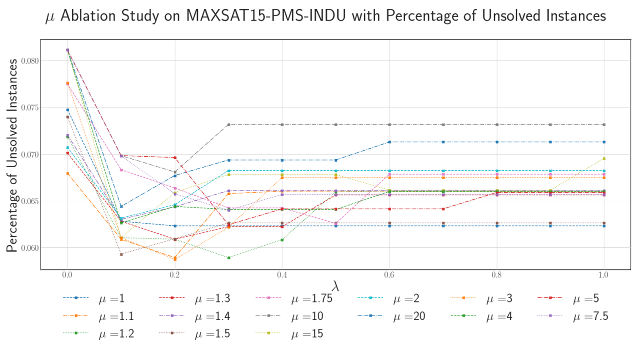


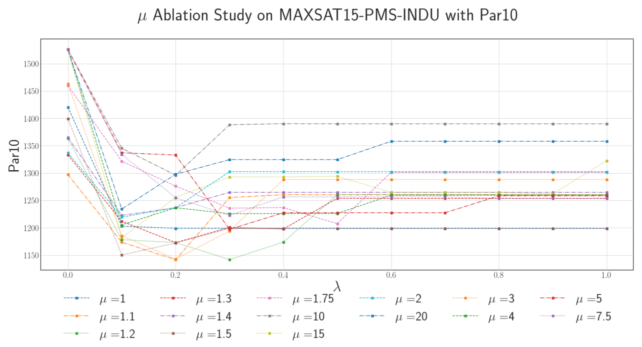


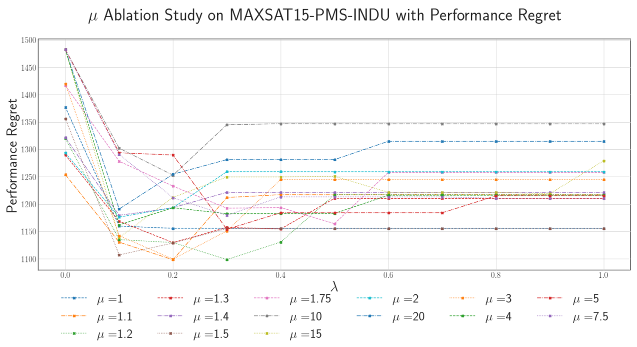

μ Ablation Study on CSP-2010 with Par10
Par10
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

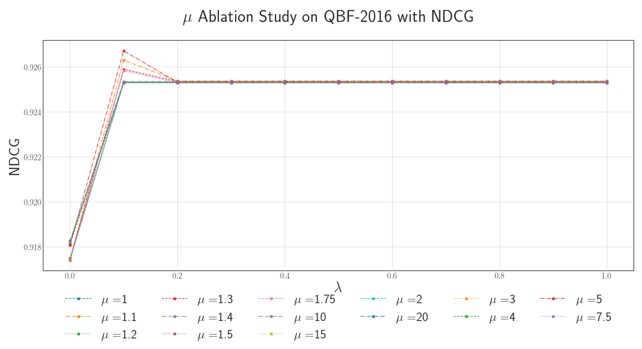

μ Ablation Study on CSP-2010 with Performance Regret
Performance Regret
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

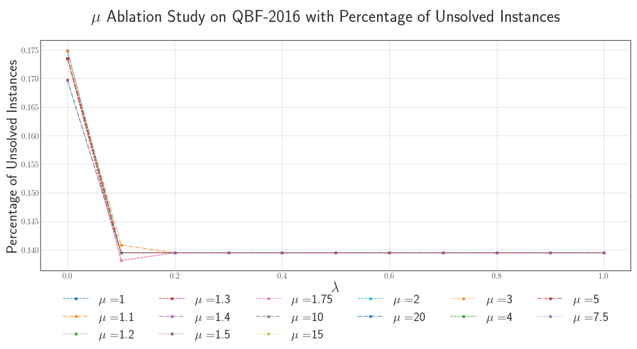

μ Ablation Study on MAXSAT15-PMS-INDU with NDCG
NDCG
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

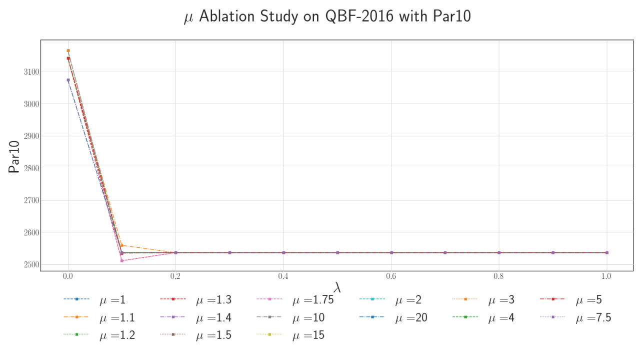

μ Ablation Study on MAXSAT15-PMS-INDU with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

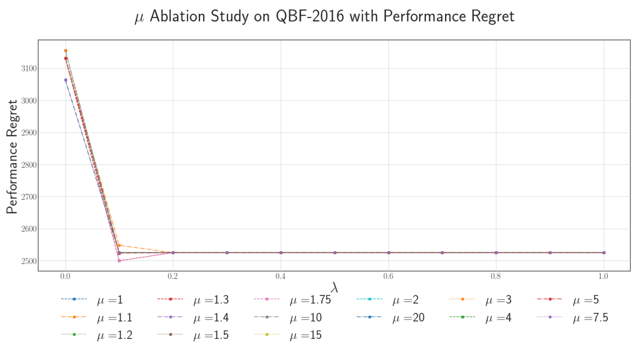

μ Ablation Study on MAXSAT15-PMS-INDU with Par10
Par10
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

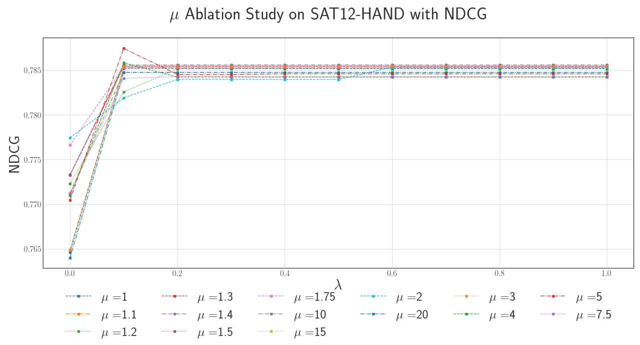

μ Ablation Study on MAXSAT15-PMS-INDU with Performance Regret
Performance Regret
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

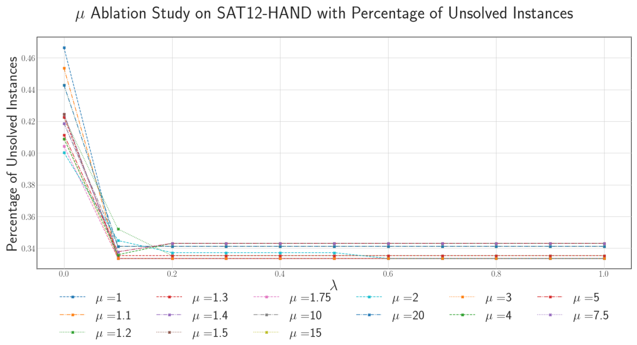

μ Ablation Study on QBF-2016 with NDCG
NDCG
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

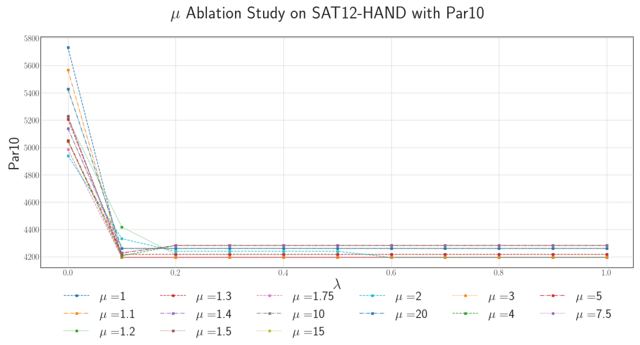

μ Ablation Study on QBF-2016 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

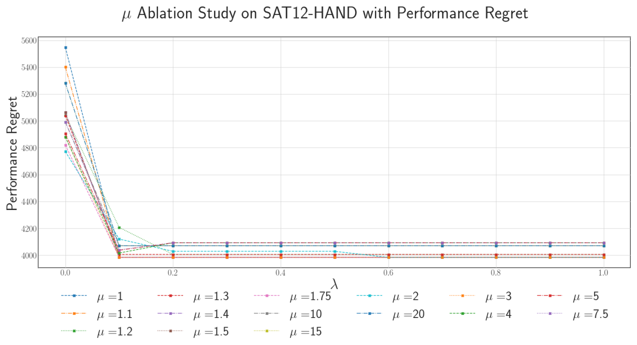

μ Ablation Study on QBF-2016 with Par10
Par10
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

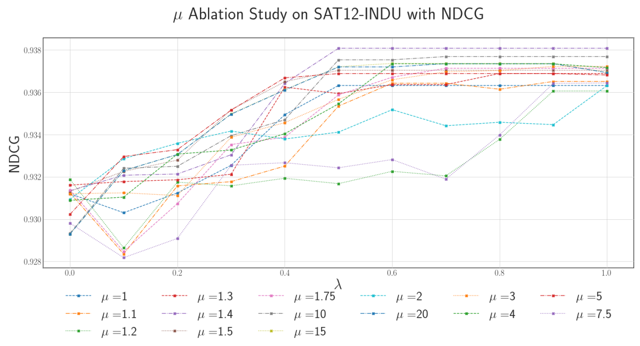

μ Ablation Study on QBF-2016 with Performance Regret
Performance Regret
λ
μ =1, μ =1.3, μ =1.75, μ =2, μ =3, μ =5
μ =1.1, μ =1.4, μ =10, μ =20, μ =4, μ =7.5
μ =1.2, μ =1.5, μ =15

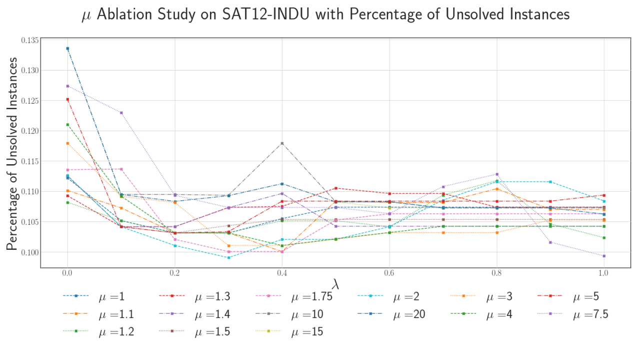

μ Ablation Study on SAT12-HAND with NDCG
NDCG
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5

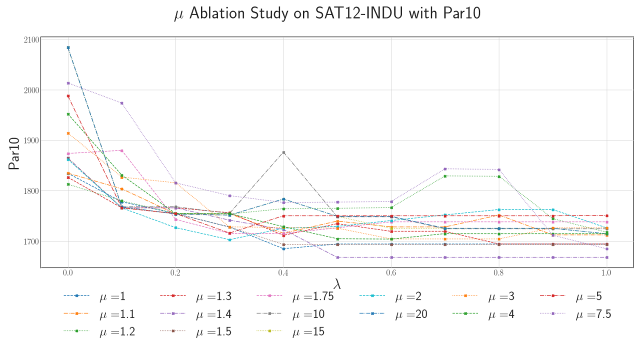

μ Ablation Study on SAT12-HAND with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5

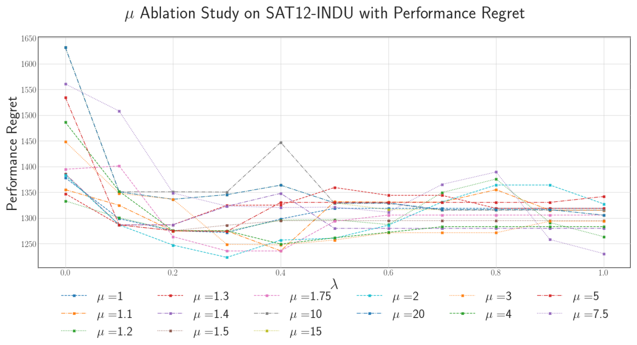

μ Ablation Study on SAT12-HAND with Par10
Par10
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5


μ Ablation Study on SAT12-HAND with Performance Regret
Performance Regret
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5


μ Ablation Study on SAT12-INDU with NDCG
NDCG
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5

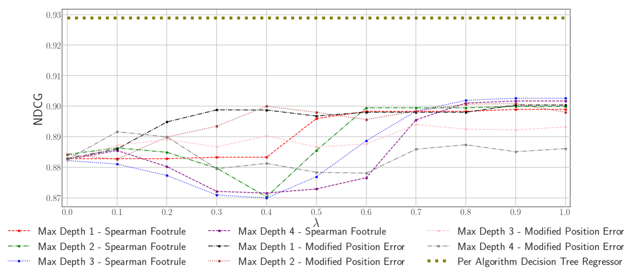

μ Ablation Study on SAT12-INDU with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5


μ Ablation Study on SAT12-INDU with Par10
Par10
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5

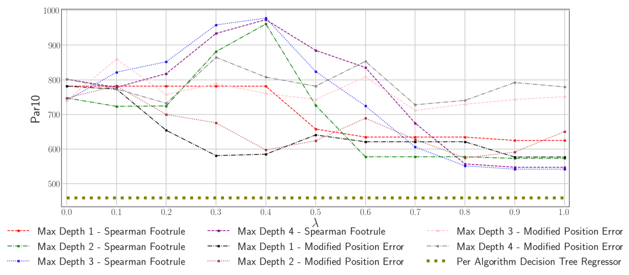

μ Ablation Study on SAT12-INDU with Performance Regret
Performance Regret
λ
μ =1, μ =1.1, μ =1.2, μ =1.3, μ =1.4, μ =1.5, μ =1.75, μ =10, μ =15, μ =2, μ =20, μ =3, μ =4, μ =5, μ =7.5

A.5 Comparison to Per Algorithm Decision Tree Regressor

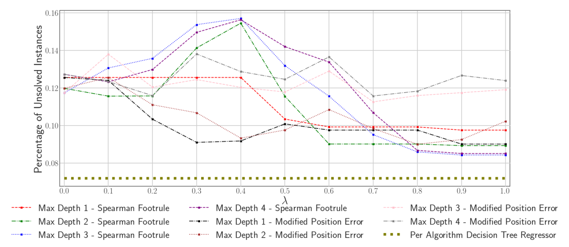
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - ASP-POTASSCO with NDCG



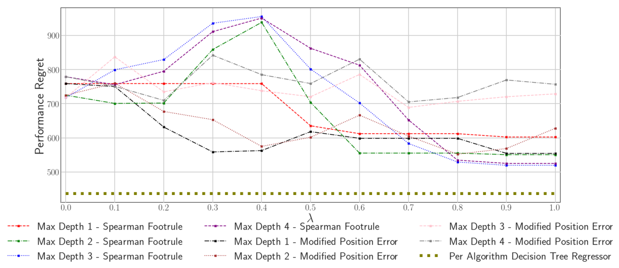
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - ASP-POTASSCO with Par10



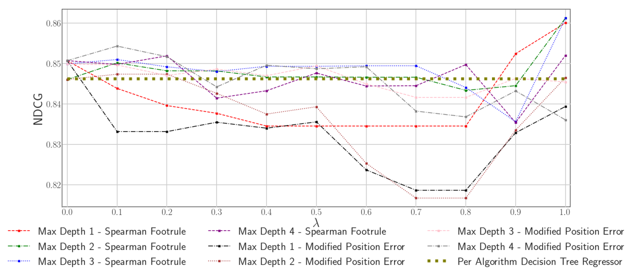
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - ASP-POTASSCO with Percentage of Unsolved Instances



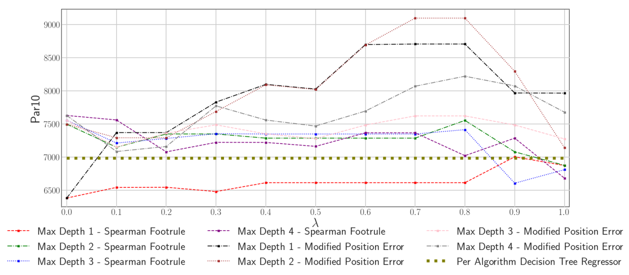
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - ASP-POTASSCO with Performance Regret



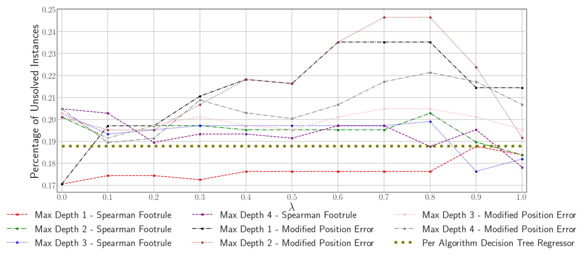
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CPMP-2015 with NDCG



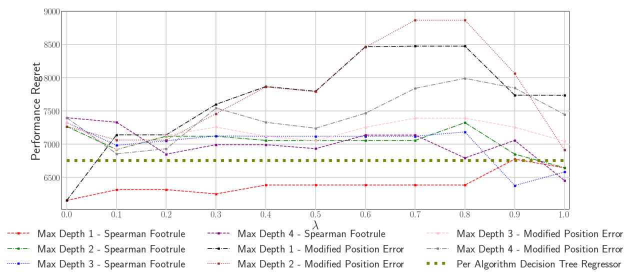
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CPMP-2015 with Par10



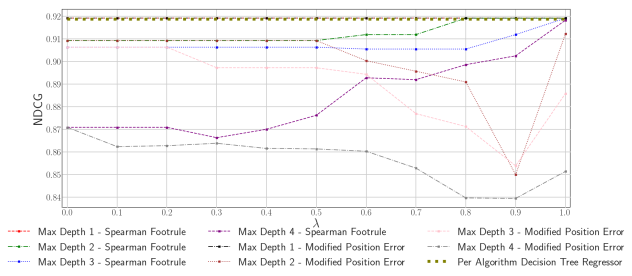
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CPMP-2015 with Percentage of Unsolved Instances



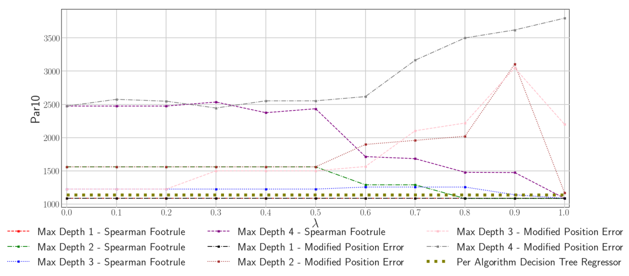
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CPMP-2015 with Performance Regret

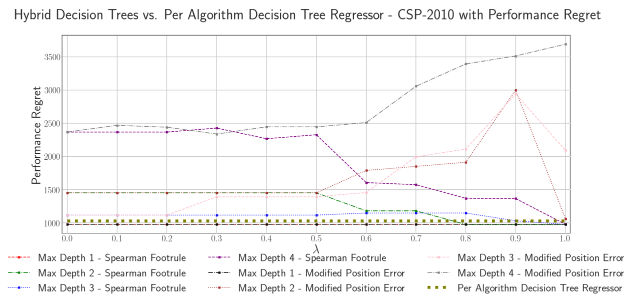


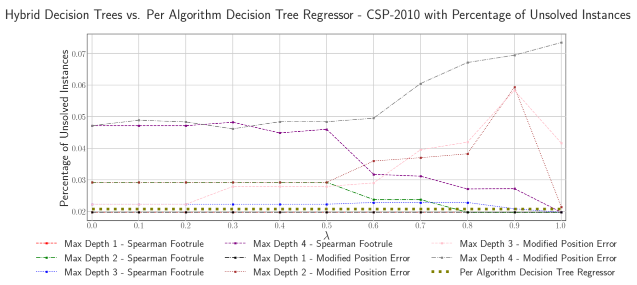
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CSP-2010 with NDCG

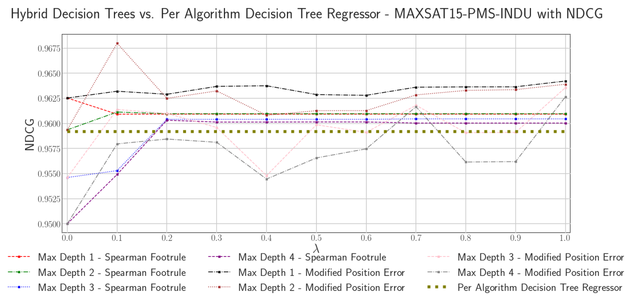


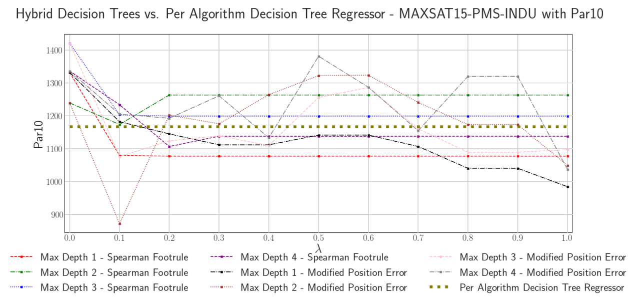
Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CSP-2010 with Par10

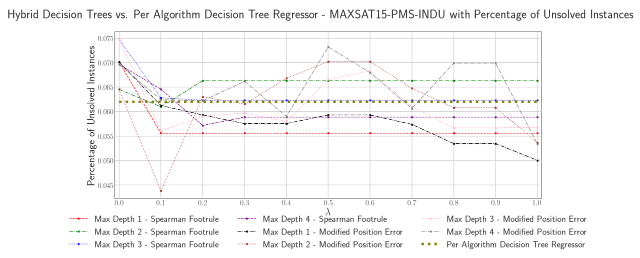


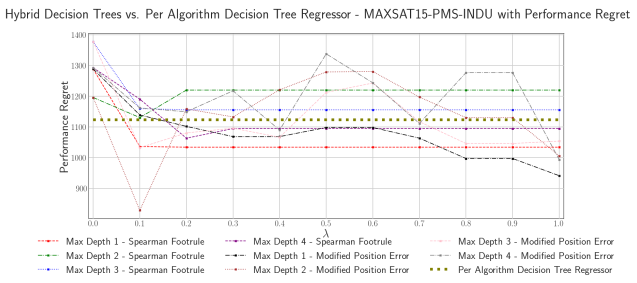

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CSP-2010 with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

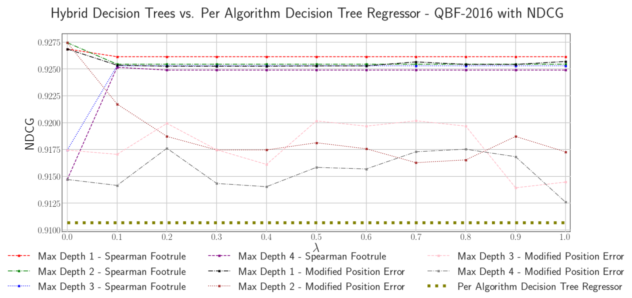

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - CSP-2010 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

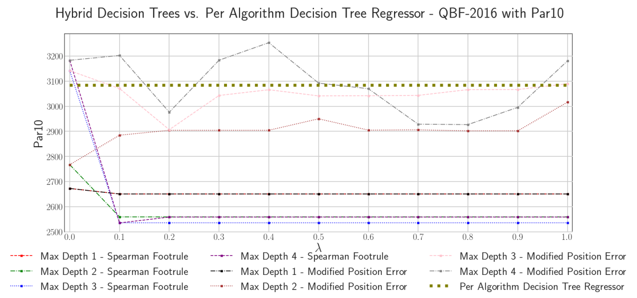

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - MAXSAT15-PMS-INDU with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

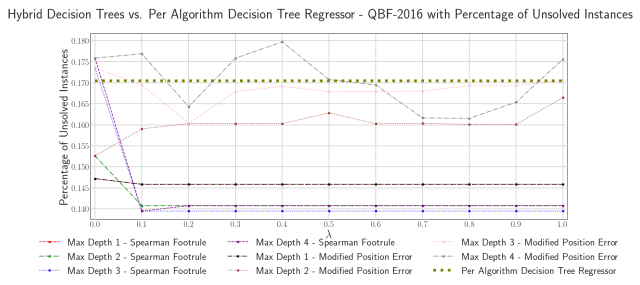

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - MAXSAT15-PMS-INDU with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

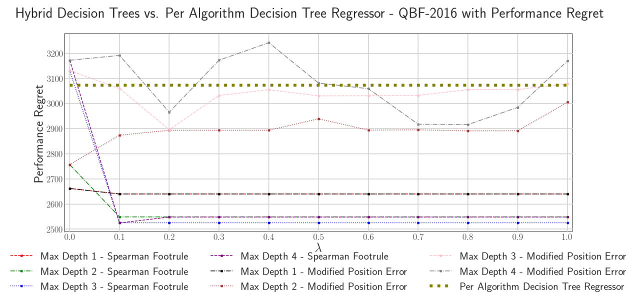

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - MAXSAT15-PMS-INDU with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

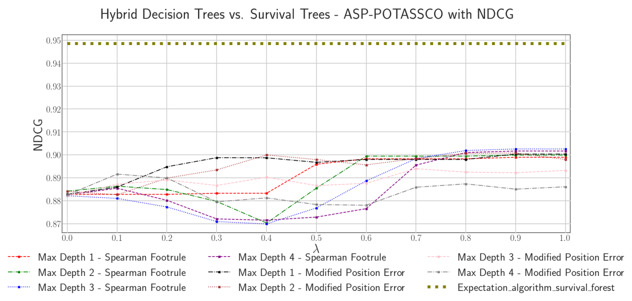

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - MAXSAT15-PMS-INDU with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

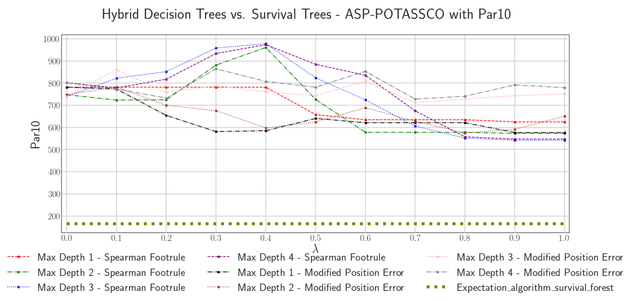

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - QBF-2016 with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

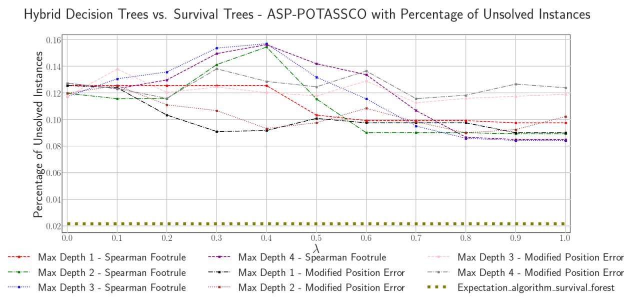

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - QBF-2016 with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

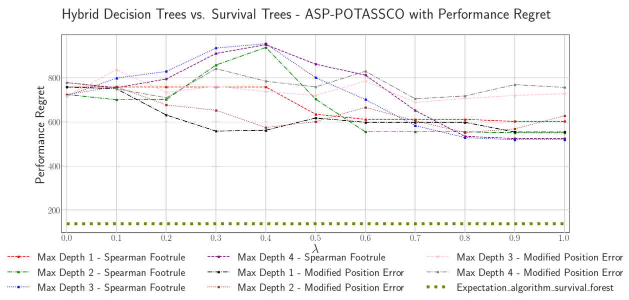

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - QBF-2016 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

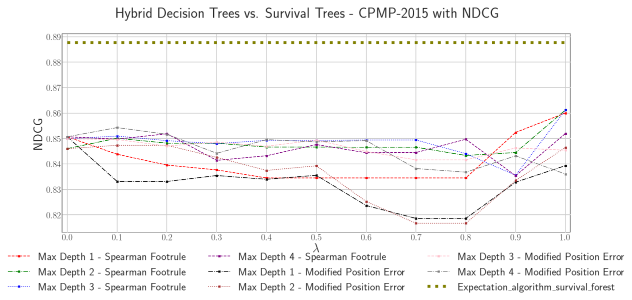

Hybrid Decision Trees vs. Per Algorithm Decision Tree Regressor - QBF-2016 with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Per Algorithm Decision Tree Regressor

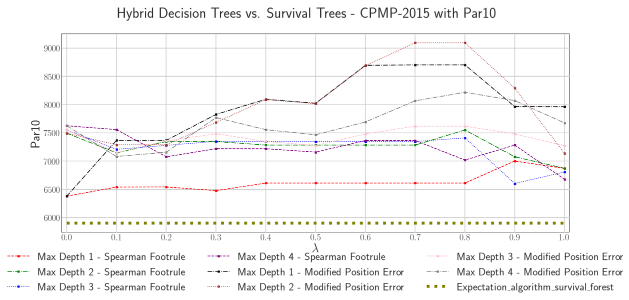

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

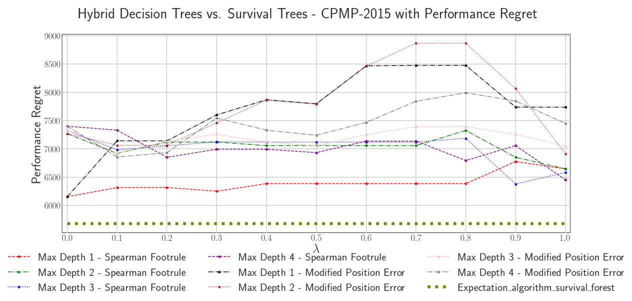

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

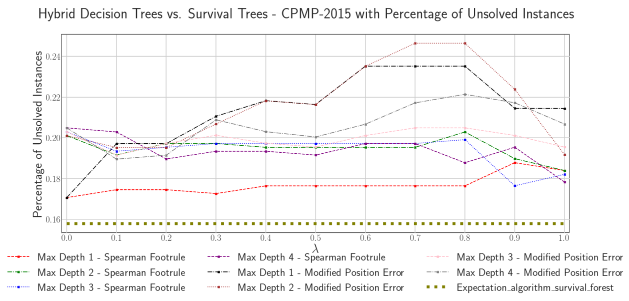

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest


Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

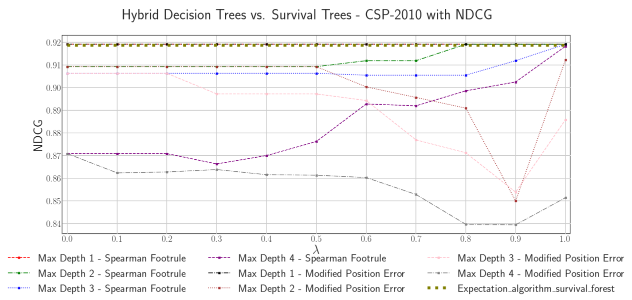

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

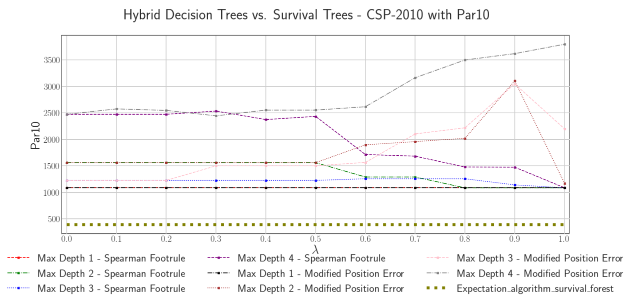

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

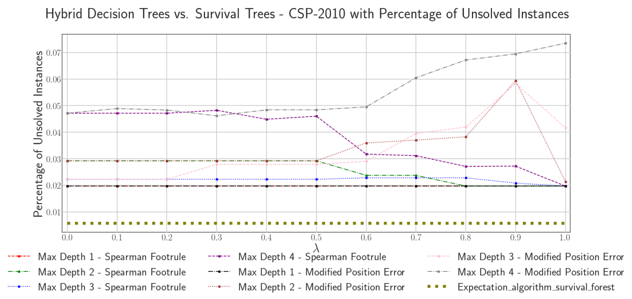

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

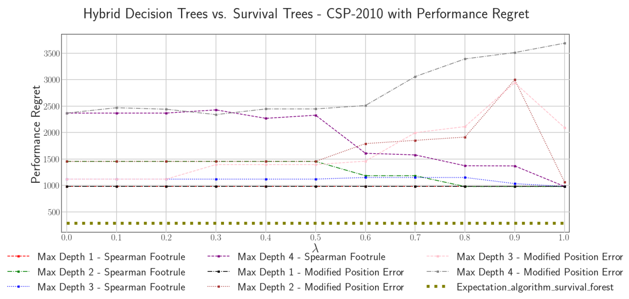

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

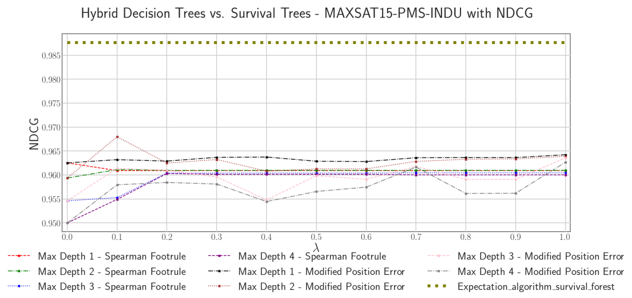
A.6 Comparison to Survival Trees

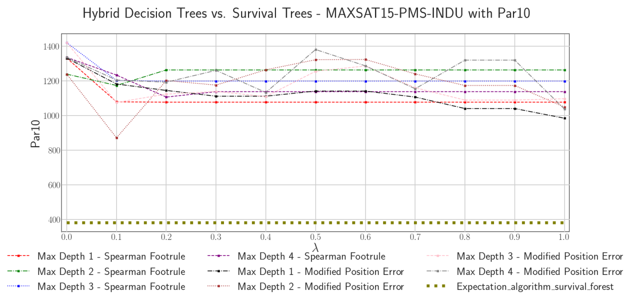


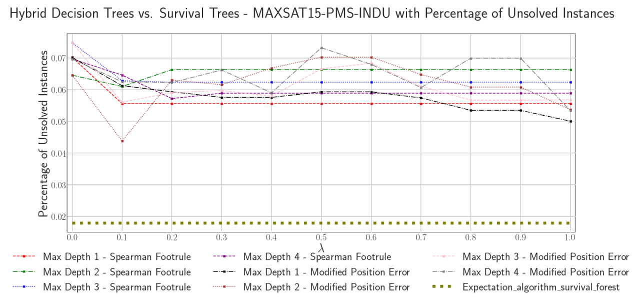


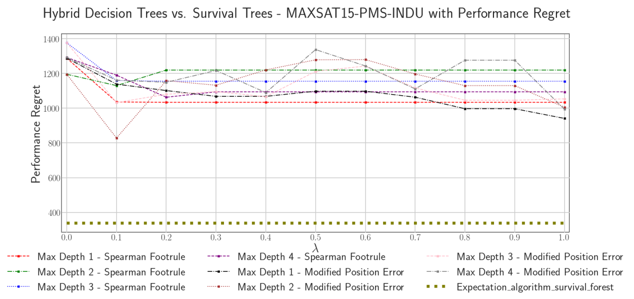


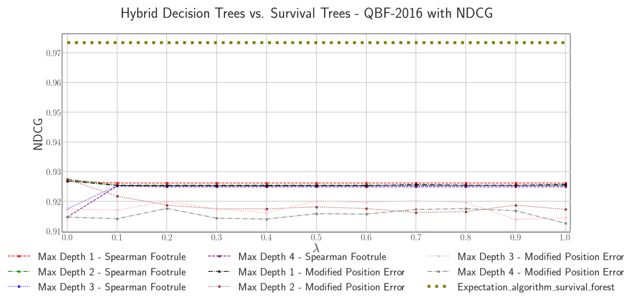


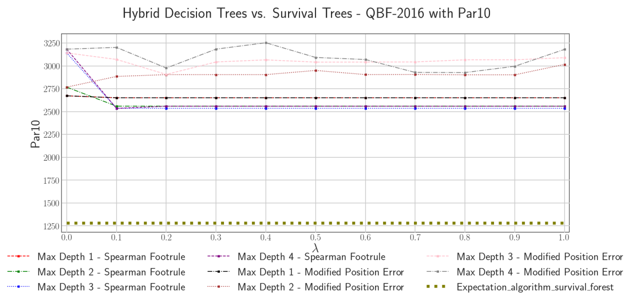


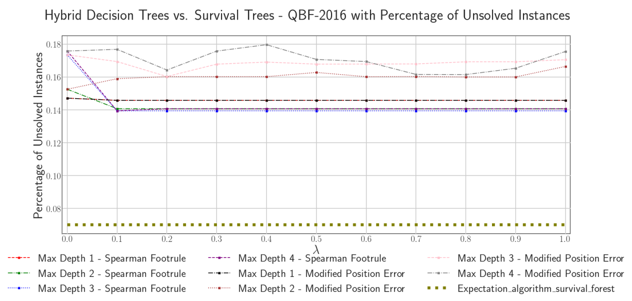


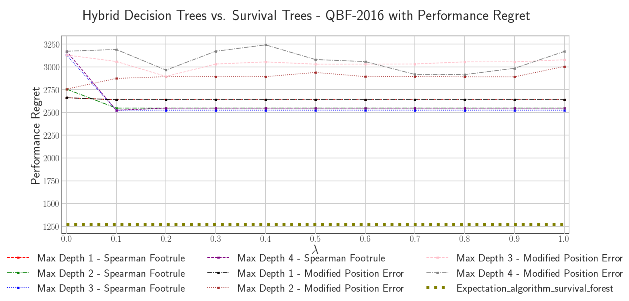


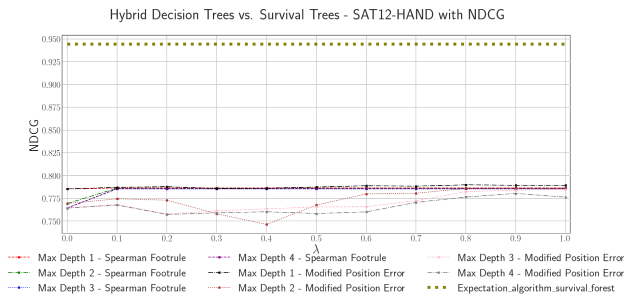


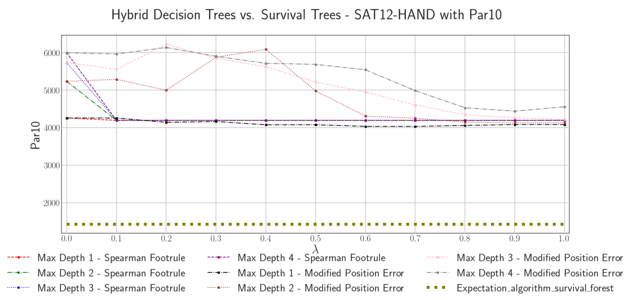


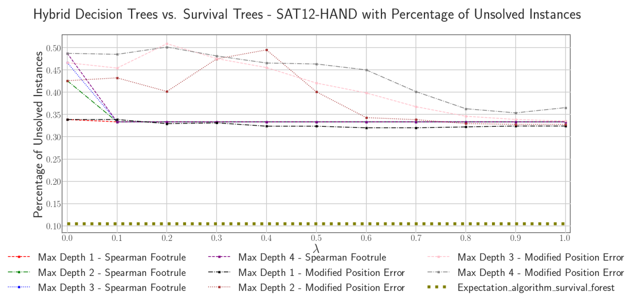


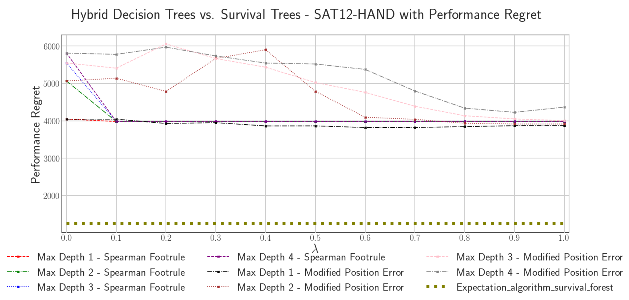

Hybrid Decision Trees vs. Survival Trees - QBF-2016 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

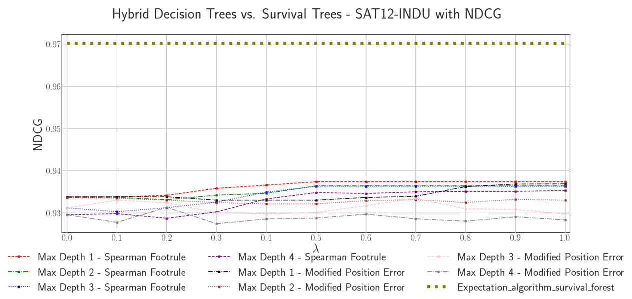

Hybrid Decision Trees vs. Survival Trees - QBF-2016 with Performance Regret
Performance Regret
λ

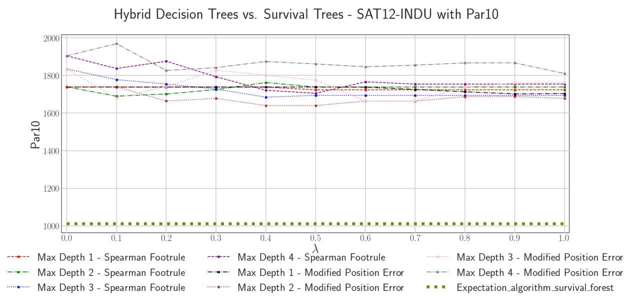

Hybrid Decision Trees vs. Survival Trees - SAT12-HAND with NDCG
NDCG
λ

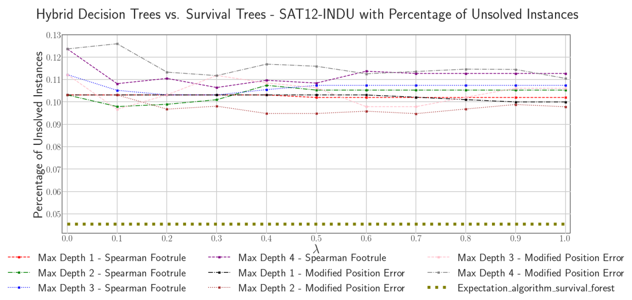

Hybrid Decision Trees vs. Survival Trees - SAT12-HAND with Par10
Par10
λ

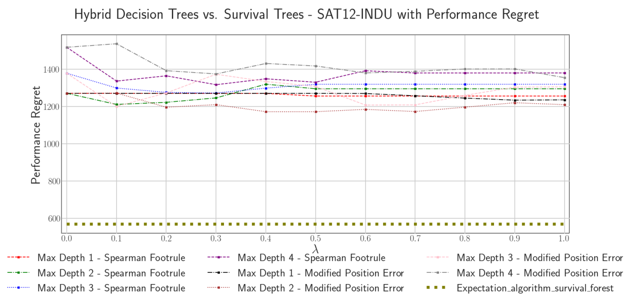

Hybrid Decision Trees vs. Survival Trees - SAT12-HAND with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ

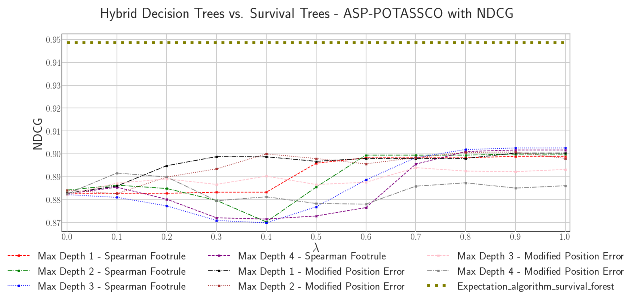

Hybrid Decision Trees vs. Survival Trees - SAT12-HAND with Performance Regret
Performance Regret
λ

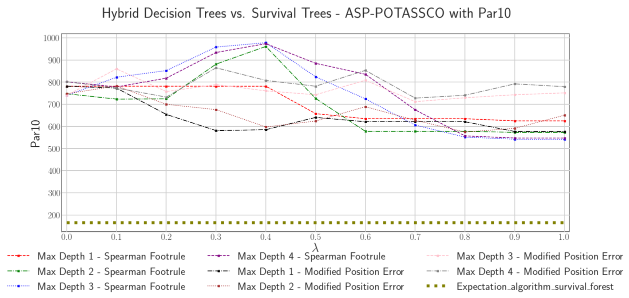

Hybrid Decision Trees vs. Survival Trees - SAT12-INDU with NDCG
NDCG
λ

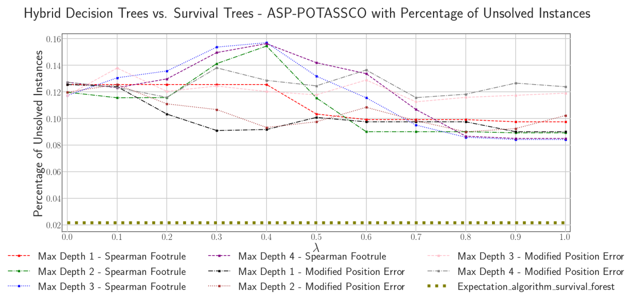

Hybrid Decision Trees vs. Survival Trees - SAT12-INDU with Par10
Par10
λ

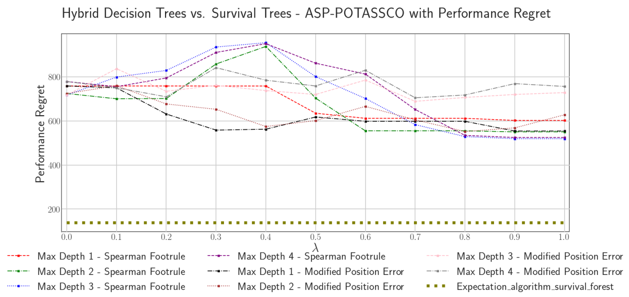

Hybrid Decision Trees vs. Survival Trees - SAT12-INDU with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ

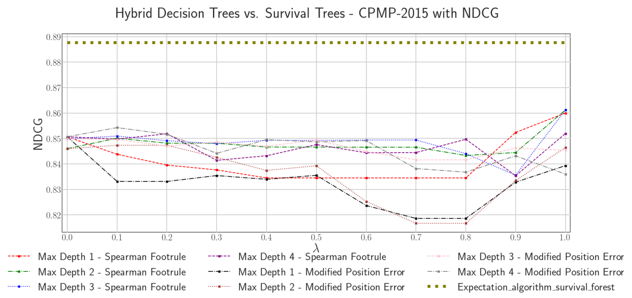

Hybrid Decision Trees vs. Survival Trees - SAT12-INDU with Performance Regret
Performance Regret
λ

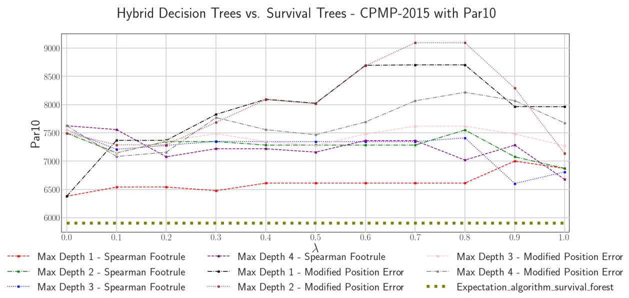

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

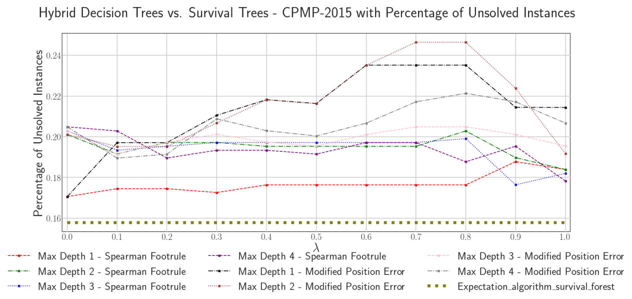

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

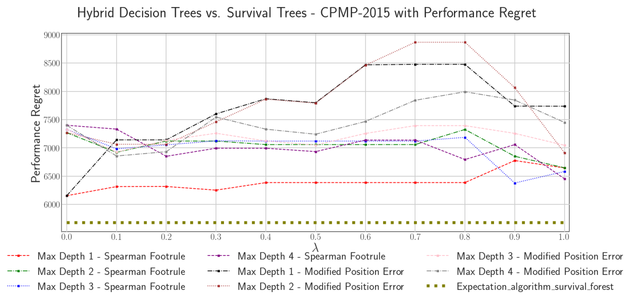

Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest


Hybrid Decision Trees vs. Survival Trees - ASP-POTASSCO with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

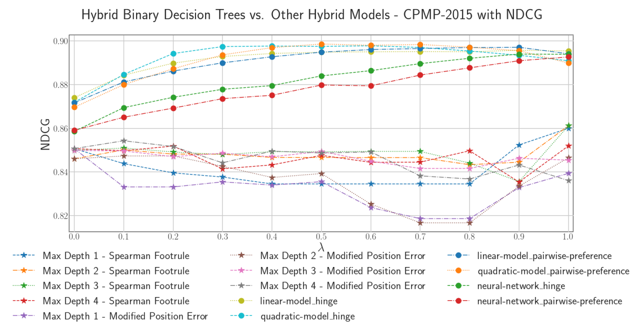

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with NDCG
NDCG
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

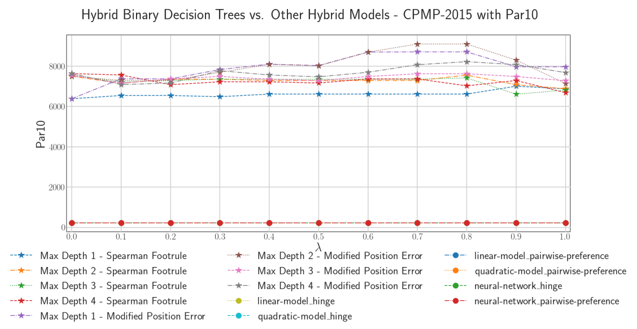

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Par10
Par10
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

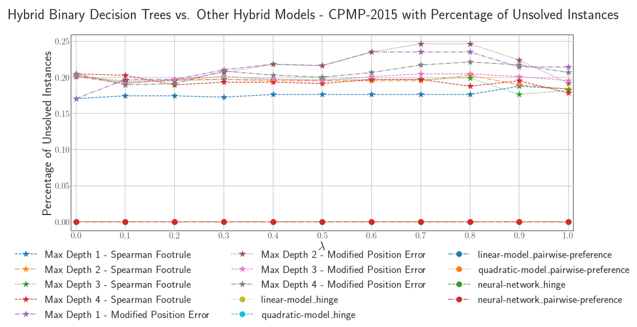

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Percentage of Unsolved Instances
Percentage of Unsolved Instances
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

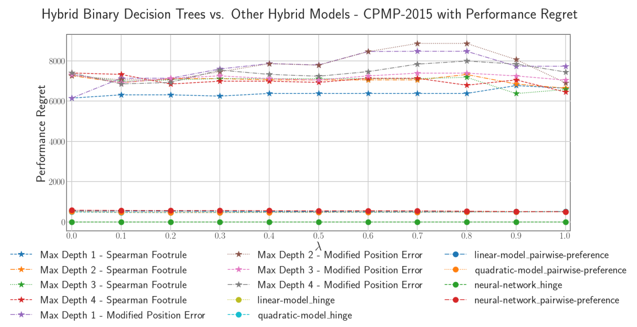

Hybrid Decision Trees vs. Survival Trees - CPMP-2015 with Performance Regret
Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
Expectation_algorithm_survival_forest

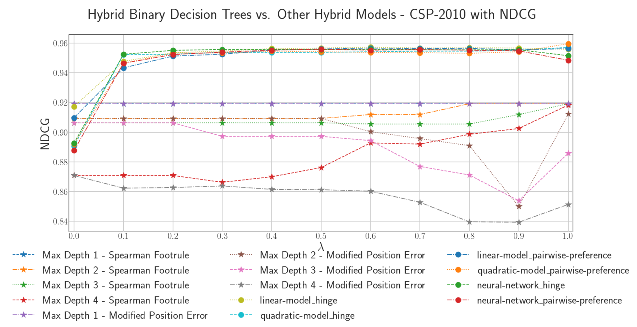
A.7 Comparison to Preexisting Hybrid Models

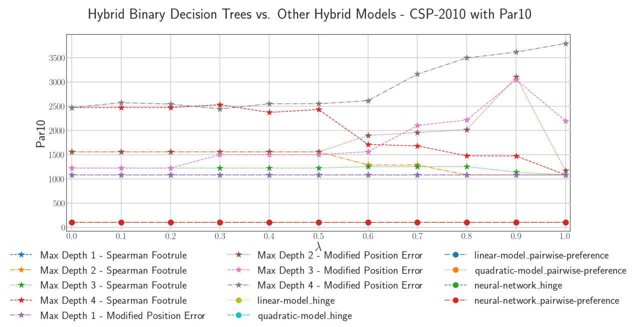


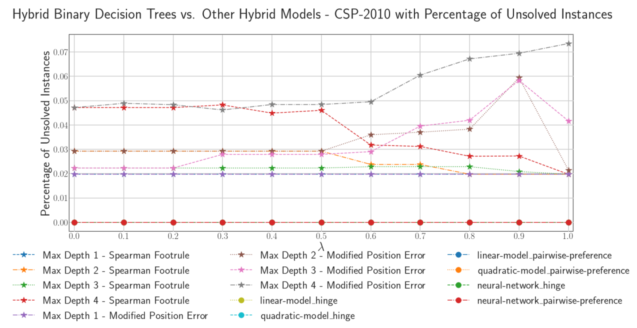


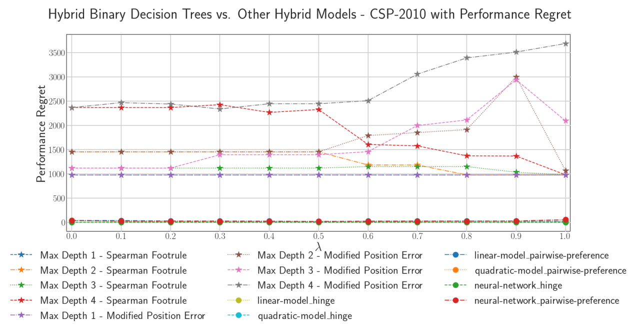


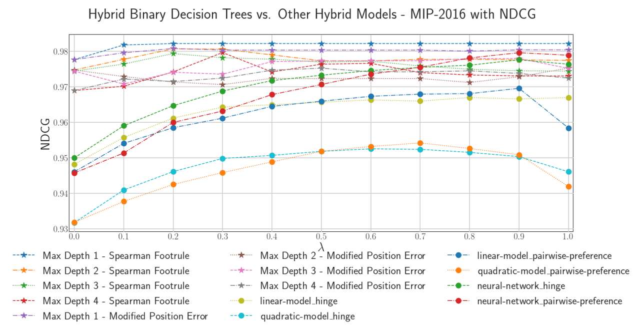


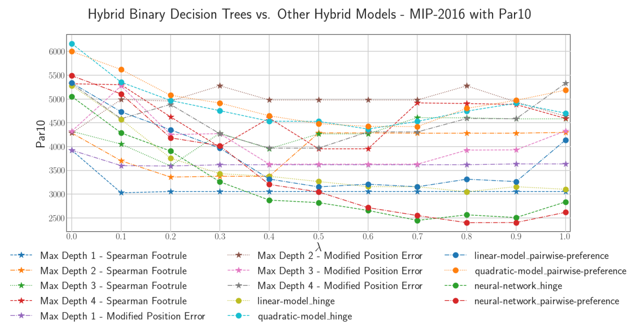




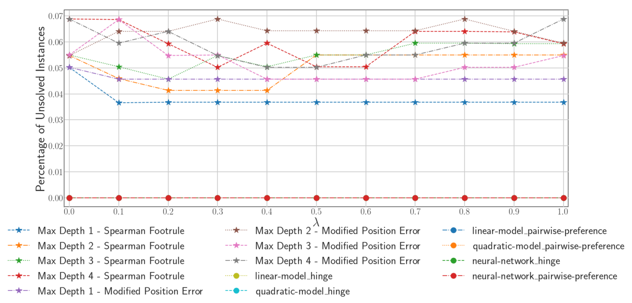




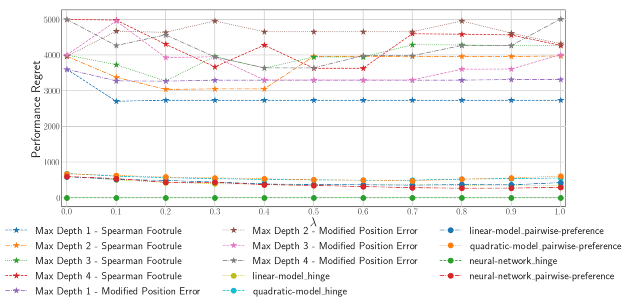




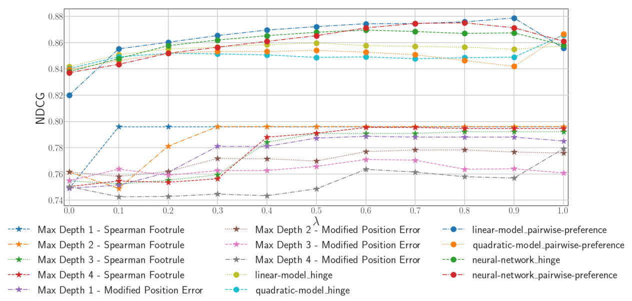
Hybrid Binary Decision Trees vs. Other Hybrid Models - MIP-2016 with Percentage of Unsolved Instances



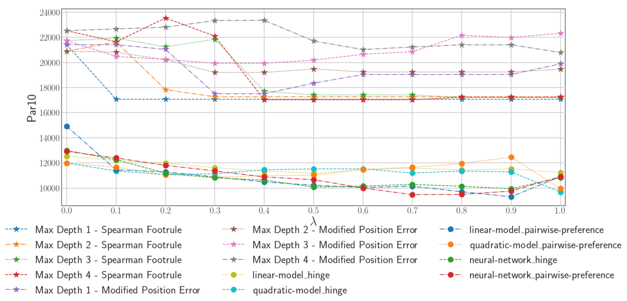
Hybrid Binary Decision Trees vs. Other Hybrid Models - MIP-2016 with Performance Regret



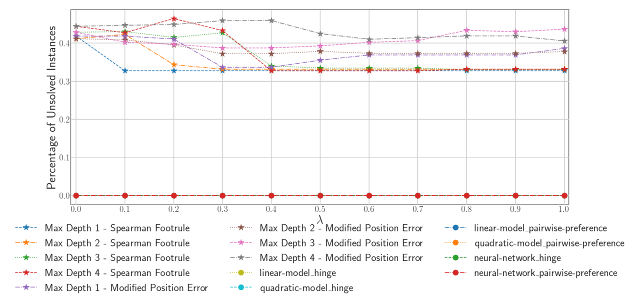
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-HAND with NDCG



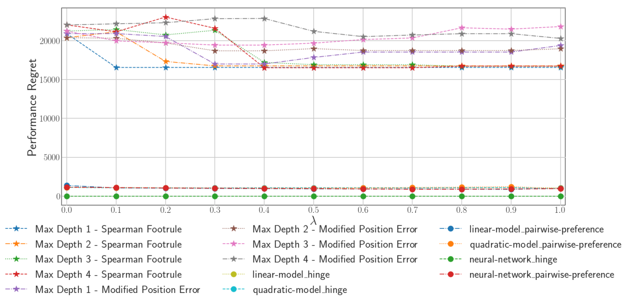
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-HAND with Par10



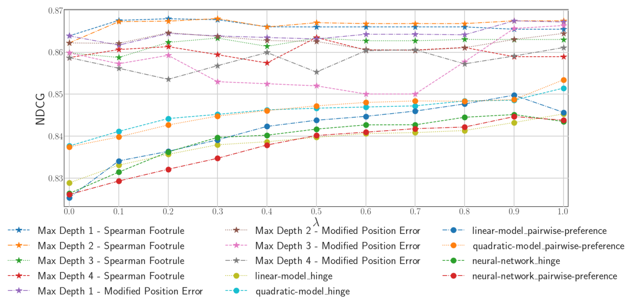
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-HAND with Percentage of Unsolved Instances



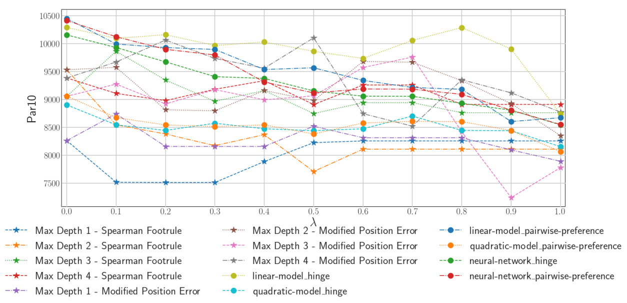
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-HAND with Performance Regret



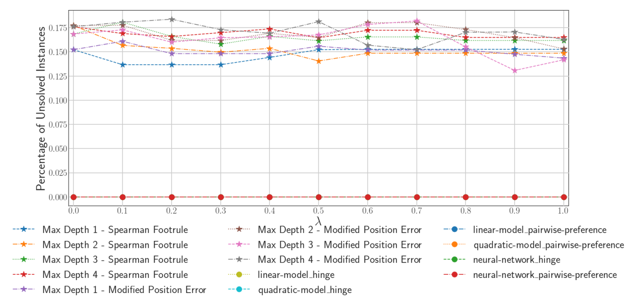
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-INDU with NDCG



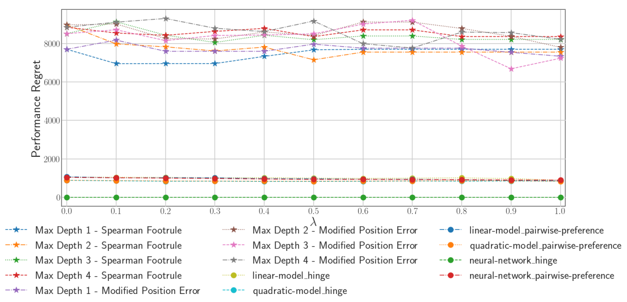
Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-INDU with Par10



Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-INDU with Percentage of Unsolved Instances



Hybrid Binary Decision Trees vs. Other Hybrid Models - SAT11-INDU with Performance Regret


Performance Regret
λ
Max Depth 1 - Spearman Footrule
Max Depth 2 - Spearman Footrule
Max Depth 3 - Spearman Footrule
Max Depth 4 - Spearman Footrule
Max Depth 1 - Modified Position Error
Max Depth 2 - Modified Position Error
Max Depth 3 - Modified Position Error
Max Depth 4 - Modified Position Error
linear-model_hinge
quadratic-model_hinge
linear-model_pairwise-preference
quadratic-model_pairwise-preference
neural-network_hinge
neural-network_pairwise-preference