Optimal Decision Trees for The Algorithm Selection Problem

Balancing Performance and Interpretability

Daniël Poolman

Supervisors: Emir Demirović, Jacobus G. M. van der Linden

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Daniël Poolman Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Jacobus G. M. van der Linden, David Tax

Abstract

The Algorithm Selection Problem (ASP) presents a significant challenge in numerous industries, requiring optimal solutions for complex computational problems. Traditional approaches to solving ASP often rely on complex, black-box models like random forests, which are effective but lack transparency, and they often fail to balance performance with interpretability. This paper investigates the performance-interpretability trade-off for the ASP, specifically focused on Optimal Decision Trees (ODTs) as recent innovations have made the use of ODTs more viable. We compare ODTs against 4 other tree-based models, using 11 different datasets. We show there is no apparent trade-off between performance and interpretability for ODTs which have been trained using an instance cost-sensitive approach, as they achieve comparable performance to a Random Forest Regressor while maintaining interpretability through multiple orders of magnitude fewer leaf nodes.

1 Introduction

There exist multiple complexity classes that contain computationally hard problems frequently encountered in various industries, including finance, healthcare, logistics, and telecommunications.

Solving the problems in these complexity classes optimally has been a major area of study in computer science, and most problems have seen many different approaches to solve them over the years. An interesting observation when comparing the performance of different approaches for the same problem is that the performance of different algorithms can vary significantly per instance, and typically no single algorithm outperforms all others [1]. This phenomenon has been observed in various NP-hard problems, such as the constraint satisfaction problem (CSP) [2], the satisfiability problem (SAT) [3], planning and scheduling algorithms [4], and the quantified Boolean formula problem (QBF) [5].

Finding the best-performing algorithm per instance of such a problem is defined as the per-instance Algorithm Selection Problem [1]. The Algorithm Selection Problem has been studied widely, and many different approaches have been suggested, often making use of machine learning models. While these approaches provide good performance, such as random regression forests [6], the studies frequently make use of black-box models, which are not interpretable and consequently do not provide insight into the algorithmic problems.

Interpretability has fundamental importance in contexts where sensitive decisions are made, such as the healthcare industry, criminal justice, public safety, and fairness in a social context. In these contexts, we need machine learning models that can be understood by humans, since interpretable models are crucial for safety, fairness, and reliability [7]. This, in contrast with black-box models, which are hard to understand and difficult to troubleshoot, underlines the need for transparency.

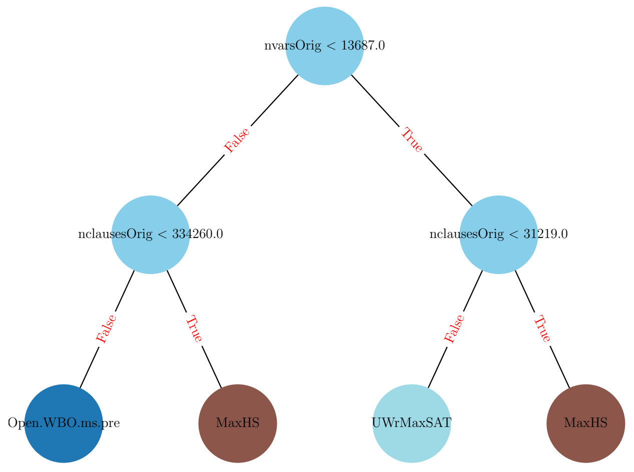


Figure 1: An optimal decision tree of depth 2 for the MAXSAT19-UCMS-ALGO dataset. As the tree is shallow, it is possible to understand the model. A quick analysis of the tree shows, based on the number of variables (nvarsOrig) and clauses (nclausesOrig): Open.WBO.ms.pre is optimal for large problems, MaxHS is preferred for medium-sized problems, and UWrMaxSAT is ideal for smaller problems with many clauses.

An example of an interpretable model is a Decision Tree model, which is widely used for supervised learning [8], and they have been used since the 1960s [9]. An example of such a decision tree is shown in figure 1. Their popularity is partly due to their interpretability, as shallow trees can be easily visualized [10]. The most commonly used decision tree algorithm is the Classification and Regression Trees (CART), which employs heuristics to construct the trees [11]. These heuristics optimize a local objective function to quickly build a tree that fits the training data. When these trees are ensembled, they form a Random Forest [12]. These models have achieved state-of-the-art performance across various classes of machine learning problems [13] and are implemented in popular machine learning frameworks, such as scikit-learn [14].

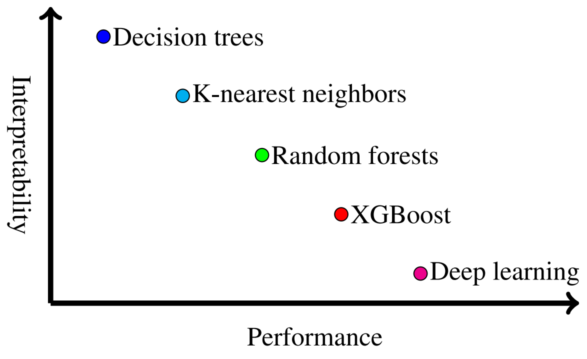


Figure 2: The interpretability - performance trade-off as presented by Sheykhmousa and Mahdianpari [15]

Despite their popularity, heuristic tree-based models have two major drawbacks. First, a downside of heuristic tree-based models is related to their optimization of a local objective function, which does not ensure a globally optimal tree. Consequently, heuristic trees might not represent the data accurately, implying that they do not generalize well to out-of-sample data [16]. Second, a downside of a more interpretable model is the relation between performance and interpretability, which is usually visualized as shown in figure 2, showing that more interpretable models sacrifice performance. While this is the general consensus in the literature, Rudin et al. [7] argue that less complicated models can still achieve high performance.

Optimal decision trees aim to address these drawbacks by providing a relatively small tree, which is interpretable. The provided tree is also globally optimal on the training data, according to a specified objective function and tree depth, which has been shown to perform better on out-of-sample data than heuristics [17], [18].

One downside is that computing optimal decision trees is in itself an NP-hard problem [19]; however, recent advancements in computational power and algorithmic techniques have made their computation more feasible on real-world datasets. Previous approaches include the use of mixed integer programming [18] and constraint programming [20], but these approaches suffer from poor scalability on large datasets. An alternative method involving dynamic programming has been presented by Demirović et al., which computes optimal classification trees. This approach has been demonstrated to outperform other methods by orders of magnitude, enhancing the feasibility of computing these optimal trees [21]. Van Der Linden et al. introduced a framework to extend this approach to all separable optimization tasks [22], referred to as Optimal Decision Trees with Dynamic Programming (ODT with DP).

Multiple studies have used optimal decision trees to solve the Algorithm Selection Problem. One approach used a Mixed Integer Programming approach [23], which did not scale well, and one used the ODT with DP approach [24]. However, none of these studies considered a wide range of datasets to validate the out-of-sample performance and the relation with interpretability.

Therefore, we consider the following research question: What is the trade-off between performance and interpretability when solving the Algorithm Selection Problem (ASP) with optimal decision trees compared to other tree-based methods such as heuristic trees and random forests? In our experiments, we show the possibility of optimal decision trees for the ASP, which are orders of magnitude smaller and thus more interpretable than other tree-based methods while sacrificing little performance or even gaining performance.

Our main contributions are: 1) showcasing the novel uses of Optimal Decision Trees in the Algorithm Selection Problem; 2) extensive experiments with 11 datasets and 6 models to show the trade-off between performance and interpretability.

2 Related Work

The Algorithm Selection Problem has evolved significantly since its formalization by Rice [1], who introduced the idea of selecting the best algorithm for each instance from a set of problem instances, algorithms, and performance measures. Early work in ASP primarily focused on single-domain applications, such as SAT (propositional satisfiability). The international SAT competitions highlighted the variability in solver performance across different instances, underscoring the need for algorithm portfolios [25].

Systems like SATzilla [26], introduced in 2008, pioneered the use of machine learning to predict the best solver for a given instance based on features extracted from the problem instance. SATzilla won several gold medals in SAT competitions and a new version was introduced in 2012 [27], improving the previous version using a weighted Random Forests and winning the 2012 SAT Challenge. Another well-known approach for SAT problems is 3S [28], relying on k-Nearest Neighbours (k-NN), which was amongst the best performing solvers in the 2011 International SAT competition.

Subsequent research expanded ASP techniques to other domains, including Constraint Satisfaction Problems (CSP) [2], [29], planning [4], [30], and Answer Set Programming [31]. Notable systems include SUNNY [29], which solves the ASP for CSP and is based on k-NN, and Claspfolio [32], which solves the ASP for Answer Set Programming and uses several mechanisms such as k-NN, Random Forests and regression. Another interesting system is ME-ASP [33], which uses several classifiers, such as k-NN, Support Vector Machines and Random Forests to solve the ASP for Answer Set Programming.

The introduction of ASlib [6] in 2016 marked a significant advancement by providing a standardized benchmark library for cross-domain evaluation of algorithm selection techniques.

More recent approaches include the use of local optima networks [34], Graph Convolutional Network-Based Generative Adversarial Networks [35], and Graph Neural Networks, which have been applied to the TSP [36]. An interesting trend we identify is the use of more complex machine learning models and the use of multiple black box models in recent years.

A significant approach to solving ASP is through the use of tree-based models such as Random Forests or XGBoost, which have consistently demonstrated outstanding performance for the ASP, rivaling the performance of state-of-the-art neural networks [37]. Random Forests have been used numerous times, such as in the aforementioned solver SATzilla [27], Planzilla [30], MachSMT [38] which solves the ASP for Satisfiability Modulo Theories problems, and ASAP (Algorithm Selector And Prescheduler system) [39] which combines Random Forests and k-NN.

Recently, Vilas et al. demonstrated that optimal decision trees provide accurate results and do not overfit on data, although their study was limited to 500 instances [23]. In another study, Segalini et al. [24] showed that by using the STreeD approach, optimal decision trees can be computed much more scalably (by orders of magnitude). However, this study was limited to MaxSAT data, and no detailed analysis on out-of-sample performance and interpretability was conducted. Visentin et al. [40] proposed a framework for explainabling multi-class classification specifically for the Capacitated Lot Sizing Problem (CLSP); however, they do not study the performance of the model alongside its explainability.

The performance-interpretability tradeoff has been widely studied in machine learning in recent years, as the recognition for the necessity of interpretability is growing. Cayamcela et al. (2019) [41], Luo et al. (2019) [42], Arrieta et al. (2020) [43], Ciatto et al. (2020) [44], Kumar et al. (2021) [45], and Ali et al. (2023) [46] all use some version of figure 2 to depict interpretability as inversely proportional to performance. Lipton (2018) [47] and Rudin (2019) [7] are among the few studies that call this relation into question.

(Optimal) Decision Trees The construction of optimal decision trees is proven to be NP-Complete by Hyafil et al. [19], indicating the complexity and computational challenges involved in finding the best decision tree.

Bennet et al. [48] proposed constructing globally optimal decision trees by first fixing the tree's structure and then solving linear inequalities using existing optimizers. This method introduced a structured approach to manage the complexity of tree construction. Constraint programming and Mixed Integer Programming (MIP) have been employed by Bertsimas et al. [17], [49] and Verwer et al. [20] to refine decision tree optimization.

Recent innovations include the development of methods for building optimal decision trees by Hu et al. [50] and DL8.5 by Aglin et al. [51], which significantly outperforms previous methods by using a Branch-and-Bound search, caching, pruning, and heuristics. Demirović et al. [21] introduced MurTree, employing many specialized techniques tailored to classification trees' unique properties. This approach, along with STreeD [22], which generalizes the methodology to any separable optimization task, shows a trend towards scalable solutions that maintain optimal performance across various tasks.

Summary While the ASP has been addressed with numerous approaches, showing outstanding performance with the use of (multiple) black-box models, such as Random Forests, none have thoroughly studied the trade-off between interpretability and performance across multiple problems and datasets for optimal decision trees. Generally speaking, the consensus about the interpretability-performance trade-off currently leans more toward the idea that less complex models are more interpretable but usually sacrifice performance. However, this may not be the case. As recent innovations have made the use of optimal decision trees more viable, we will study this relation for the ASP in the next sections.

3 Preliminaries

This section details the formalization of the Algorithm Selection Problem, the different Algorithm Selection approaches considered and the evaluation metrics used:

The Algorithm Selection Problem can be formalized as follows [1]:

- **Problem Instance Space ($\mathcal{X}$)**: The set of problem instances $x \in \mathcal{X}$ drawn from a distribution D
- **Algorithm Space ($\mathcal{A}$)**: The set of candidate algorithms $a \in \mathcal{A}$ which are available for solving the problems
- **Performance Metric ($\mathcal{M}$)**: A function $m : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ that measures the performance of algorithm a on instance x. The goal is to minimize (or maximize) this metric.
- **Cost Function (C)**: A function $C : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ representing the cost of selecting algorithm a for instance x.
- **Feature Space ($\mathcal{F}$)**: The set of all features $f \in \mathcal{F}$ that describe the problem instances.
- **Feature Value Function (ϕ)**: A function $\phi : \mathcal{F} \times \mathcal{X} \to \mathbb{R}$ that maps each feature and instance to a real-valued number representing the value of that feature for the given instance.

The goal is to find a mapping $s : \mathcal{X} \to \mathcal{A}$ such that the overall performance is minimized:

$$\arg\min \sum_{x \in \mathcal{X}} \mathcal{M}(x, s(x)))$$

Algorithm Selection Approaches vary based on the kinds of decisions which have to be made in the selection process. To minimize the mapping $s : \mathcal{X} \to \mathcal{A}$, one can simply try to select the best algorithm using a *classification model*. Another widely used approach is *regression*, which first predicts the performance for each algorithm and then selects the best-performing algorithm. The last approach we consider is *instance cost-sensitive classification*, which considers different misclassification costs for each label and each instance. This is particularly relevant in applications where the cost of a misclassification varies significantly across instances, such as in the ASP.

Evaluation metrics in the context of the ASP are atypical. As we are interested in the speed up in runtime with respect to the available algorithms, typical evaluation metrics in machine learning such as accuracy, F1 score and precision cannot be used as they don't provide information about the runtimes. Choosing an appropriate evaluation metric is not straightforward, as a timeout T is defined per Algorithm Selection scenario, because some instances are too difficult to solve in a reasonable amount of time. For this reason, we evaluate the performance of different models using the Penalized Average Runtime with an included penalty factor of ten for timeouts (PAR10), a widely used metric to evaluate performance [6]. PAR10 is defined as follows:

$$\text{PAR}_{10}(s) = \frac{\sum_{x \in \mathcal{X}} \mathcal{M}'(x, s)}{|\mathcal{X}|}$$

$$\mathcal{M}'(x, s) = \begin{cases} \mathcal{M}(x, s(x))) & \text{if } \mathcal{M}(x, s(x))) \leq T \\ 10 \cdot T & \text{else} \end{cases}$$

where

- x is a specific problem instance
- s is a mapping provided by the algorithm selector
- $\mathcal{M}(x, s(x)))$ is the function which calculates the actual runtime based on x and s
- T is the timeout value
- $\mathcal{X}$ is the set of problem instances

To bring the PAR10 score in perspective, we compare the PAR10 scores to the virtual best solver (VBS), and the single best solver (SBS). The VBS selects the best algorithm for each instance and the SBS is the best solver on average. To compare the PAR10 scores over scenarios, we normalize the PAR10 with respect to the VBS and SBS, i.e., the VBS corresponds to a 1 and the SBS corresponds to a 0:

$$\text{Normalized PAR}_{10}(s) = \frac{\text{PAR}_{10}(s) - SBS}{VBS - SBS}$$

We evaluate the interpretability of different models using the number of leaf nodes. This metric is a proxy for the complexity of each model and gives insight into the interpretability of the model as a whole.

4 Methodology

Our methodology consists of the selection and preparation of the datasets we use, the machine learning models we use and how they are trained and selected.

4.1 Data Source

We use 11 datasets from the Algorithm Selection Library (ASlib) [6]. ASlib offers a variety of datasets with distinct problem domains such as Answer Set Programming, Constraint Satisfaction Problems (CSP), and Maximum Satisfiability (MAXSAT). This variety allows for testing the efficacy of the different machine learning models under different feature space conditions. Table 1 gives an overview of the scenarios, number of algorithms, instances, and features we use in our evaluation. Other studies, which make use of ASlib, use a comparable amount of datasets. Liu et al. uses 12 datasets [52], Pulatov et al. uses 9 datasets [53] and Hanselle et al uses 6 datasets [54].

The data in ASlib is provided in a standardized data format which comes with precomputed features and their respective costs to compute the features and predefined cross-validation setups. This cross-validation setup is used in most papers that use ASlib [6], [52]–[54]. We use this setup in this study. This allows us to focus on the analysis without the need for precomputing the features ourselves and it additionally provides the opportunity for other researchers to perform comparative performance assessments with respect to this study.

The selection of the datasets we use in this study is driven by several factors. We only consider datasets that include the data for computing feature costs as this information is needed to assess whether using algorithm selection is more efficient than using the best solver on average. Certain datasets are duplicated, for which we choose the dataset that includes the most data points, e.g., more problem instances or more features, as datasets containing a larger number of features are preferable. For MAXSAT and SAT we include multiple datasets, two and three respectively, since ASlib contains many datasets related to MAXSAT and SAT. The datasets we select for MAXSAT and SAT vary significantly in either the number of instances, algorithms, or instance features.

Scenario	Instances	Algorithms	Instance Features
ASP-POTASSCO	1294	11	138
BNSL-2016	1179	8	86
CSP-Minizinc-Time-2016	100	20	95
MAXSAT-WPMS-2016	630	18	37
MAXSAT19-UCMS-ALGO	572	7	129
MIP-2016	218	5	143
PROTEUS-2014	4021	22	198
QBF-2016	825	24	46
SAT11-HAND-ALGO	296	11	190
SAT12-ALL	1614	31	115
SAT16-MAIN	274	25	55

Table 1: Number of algorithms, instances, and instance features for scenarios used in this study

4.2 Data preparation

We remove instances with missing feature or algorithm values and we remove features that are either constant valued or duplicated since both are irrelevant. To prepare the data for STreeD, we apply a binarization step, which is detailed below:

1. **Feature Selection**: Choose a continuous feature from the feature space.
2. **Sorting**: Arrange all the observed values of this feature in ascending order.
3. **Category Definition**: Divide these sorted values into a specified number of equally sized categories, called bins.
4. **Threshold Determination**: Identify the thresholds that separate these bins. The thresholds are determined based on the position of the values in the sorted list.
5. **Transformation**: Convert each value of the feature into a binary vector. For each threshold, if the value is less than the threshold, it is assigned a 1 in the binary vector; otherwise, it is assigned a 0. This process is repeated for each threshold, resulting in a binary representation for each feature value.

We use the Java implementation of Segalini et al.[1] and adapt it to our specific case.

4.3 Model Training and Model Selection

We employ various tree-based machine learning models with varying complexities and approaches to the ASP in this study. The specific models and parameter ranges are detailed in Table 2. These parameter ranges are broader than earlier works such as the original ASlib paper [6] and a specific study about the ASP for the Travelling Salesman Problem [55]. Since we are interested in the relation between performance and interpretability, we use models with varying degrees of complexity as this relates directly to interpretability [56]. To this end, we use Decision Trees, which is a ”simple” model, and Random

[1]https://github.com/gsegalini/decision-trees-asp

Forests, which is a more complex model. For both models, we use a classification and regression approach. The model complexity of the regression approach scales with the number of algorithms to choose from in each scenario, because each model needs to have a separately trained regression model for each algorithm.

Earlier studies have shown random regression forests to be superior to other machine learning models, such as Decision Trees, XGBoost, and neural networks for the Algorithm Selection Problem [6], [37], however, it would be interesting to analyze the trade-off between performance and interpretability for varying model complexities and compare this to optimal decision trees.

To use Decision Trees and Random Forests for the ASP, we use the Python scikit-learn implementation[2], a widely used machine learning library in both academia and industry. The Decision Tree Classifier and the Decision Tree Regressor are implemented in Scikit-learn using Classification and Regression Trees (CART) [11]. To use ODT with DP for the ASP we use the Python implementation of STreeD, pystreed[3].

We use feature selection to decrease the needed computational resources by training a random regression forest for each scenario and utilizing the resulting feature importance values to select the top $n \in 5, 10, 20, 30$ features for training and selecting the models.

To train the models we tune the hyperparameters of all six models, using the provided parameter ranges, with grid search and a nested cross-validation setup (with ten internal folds as specified by ASlib) for each scenario. This is done to ensure both unbiased performance results and to select the best model within the parameter ranges for each scenario.

All experiments are run on the DelftBlue supercomputer [57], which is equipped with Intel XEON E5-6448Y 32C 2.1GHz CPUs. Each experiment is configured with 8 CPU cores and 16 GB of RAM from such a CPU.

Machine learning model	Parameter ranges
Classification	
ODT with DP (STreeD)	max_depth $\in \{2,3,4,5,6\}$, max_num_nodes $\in \{3,5,10,15,20,25,31,35,40,45,50,55,60,63\}$, bin_size $\in \{2,5,7,10\}$
Decision Tree Classifier	max_depth $\in \{3,4,5,6,10,20,\text{None}\}$, max_features $\in \{\sqrt{\ }, \log_2\}$, min_samples_split $\in \{2,5\}$, min_samples_leaf $\in \{1,2\}$, max_leaf_nodes $\in \{\text{None},5,10,20,50,100\}$
Random Forest Classifier	n_estimators $\in \{100,200,300\}$, max_depth $\in \{\text{None},3,4,5,6,10,20,50\}$, min_samples_split $\in \{2,5\}$, min_samples_leaf $\in \{1,2\}$
Regression	
Decision Tree Regressor	max_depth $\in \{\text{None},3,4,5,6,10,20,30,50\}$, max_features $\in \{\text{None}, \sqrt{\ }\}$, min_samples_split $\in \{2,5,10,15\}$, min_samples_leaf $\in \{1,2,4,6\}$
Random Forest Regressor	n_estimators $\in \{100,200,300\}$, max_depth $\in \{\text{None},3,4,5,6,10,20,50\}$, min_samples_split $\in \{2,5\}$, min_samples_leaf $\in \{1,2\}$
Instance cost sensitive	
ODT with DP (STreeD Instance Cost Sensitive)	max_depth $\in \{2,3,4,5,6\}$, max_num_nodes $\in \{3,5,10,15,20,25,31,35,40,45,50,55,60,63\}$, bin_size $\in \{2,5,7,10\}$

Table 2: Machine learning algorithms and their parameter ranges used

[2]https://scikit-learn.org/

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

5 Experimental Setup and Results

To study the trade-off between performance and interpretability, we carry out multiple experiments. Experiment 5.1 focuses on evaluating the performance of different models across various scenarios. Experiment 5.2 examines the relationship between the aggregates of model performance and the number of leaf nodes to understand the high-level relation between performance and interpretability. Experiment 5.3 investigates performance versus the number of nodes per scenario to down drill in the data to understand the dynamics of the performance-interpretability tradeoff. Experiment 5.4 goes more in-depth about the trade-off between performance and interpretability specifically for the STreeD ICSC (STreeD ICSC).

5.1 Performance of all models and scenarios

To understand the overall performance of models, we select the best model, based on the lowest PAR10 score, for each scenario. The PAR10 score is based on the average of the PAR10 score for each of the ten test folds. We normalize the PAR10 to be able to compare performance against scenarios and models. The results are given in table 3, which outlines the summary statistics per model, and figure 3, which outlines the best normalized PAR10 per scenario for each model.

From the results, we can draw several conclusions. First, we conclude that the Random Forest Regressor and STreeD ICSC models show very similar performance. Both models consistently achieve higher performance scores than other models, and for all scenarios, either the Random Forest Regressor (6/11) or the STreeD ICSC (5/11) performed the best. While the Random Forest Regressor has a higher median, STreeD Instance Cost Sensitive has a higher mean because STreeD Instance Cost Sensitive is more consistent, as shown by the lower σ and the smaller spread between min/max and quartile 1 and quartile 3.

Secondly, when we compare the separate approaches to solving the ASP, the regression approach for the Random Forest outperforms the classification approach by a significant margin, with a μ difference of 0.18. There is no difference in μ and median when comparing the regression and classification for the Decision Tree.

The Instance Cost Sensitive Approach shows an even larger difference between the classification approach of STreeD with a μ difference of 0.43. Interestingly, the Decision Tree and Random Forest with both a classification and regression approach outperform the classification approach of STreeD by a large margin.

5.2 Performance vs. Number of leaf nodes for all models and scenarios

To understand the high-level relation between performance and interpretability, we compute the total number of leaf nodes per model and normalized PAR10 for each scenario and take the average of these metrics.

The results, as presented in table 4, show an interesting pattern. Increasing model complexity increases the performance but reduces interpretability when comparing the Decision Tree Classifier and Decision Tree Regressor to the Ran-

Model	μ	σ	Median	Min	Max	Q1	Q3
Decision Tree Classifier	0.49	0.26	0.58	-0.12	0.74	0.39	0.65
Decision Tree Regressor	0.49	0.20	0.58	0.22	0.82	0.29	0.62
RandomForestClassifier	0.51	0.28	0.55	-0.11	0.82	0.45	0.72
RandomForestRegressor	0.69	0.16	0.74	0.39	0.86	0.58	0.81
STreeD	0.27	0.38	0.41	-0.55	0.66	0.08	0.55
STreeD Instance Cost Sensitive	0.70	0.13	0.72	0.41	0.84	0.62	0.79

Table 3: Summary statistics of the normalized PAR10 for each model, aggregated over all scenarios. The normalized PAR10 indicates how much a model closes the gap between the Single Best Solver (sbs), the best solver on average which is set at 0, and the Virtual Best Solver (vbs), the best possible solver for each instance, which is set at 1. The mean is represented by μ and the standard deviation is represented by σ.

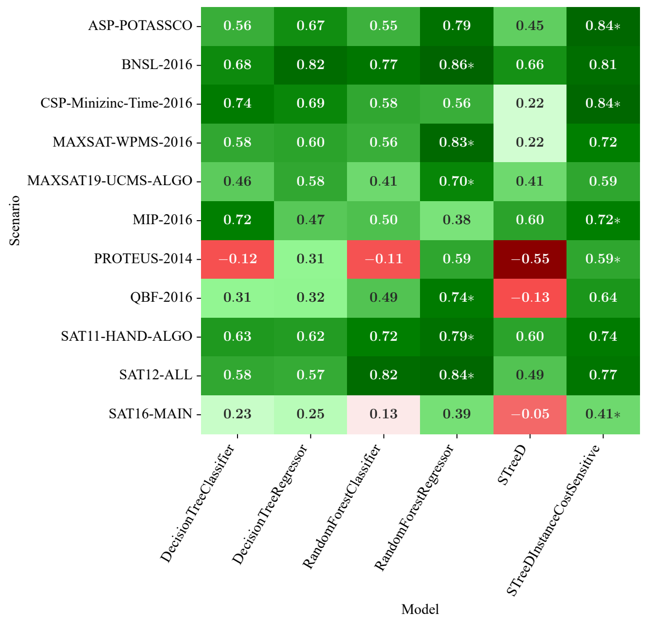


Figure 3: The best achieved normalized PAR10 for all models and for each scenario. The best model for each scenario is marked with an asterisk (*). Values close to 1 indicate a good performance, highlighted in green. Values close to 0 indicate the model did not achieve better results than selecting the sbs, indicated by white and negative values indicate a bad performance, highlighted in red.

Model	μ Normalized PAR10	μ # Leaf Nodes	σ # Leaf Nodes
Decision Tree Classifier	0.49	111.00	123.62
Decision Tree Regressor	0.49	704.36	526.89
RandomForestClassifier	0.51	19168.00	22288.12
RandomForestRegressor	0.69	490705.09	510816.79
STreeD	0.27	14.18	9.85
STreeD Instance Cost Sensitive	0.70	19.64	13.46

Table 4: Summary statistics of normalized PAR10 vs. number of leaf nodes

dom Forest Regressor, as the average normalized PAR10 increases by roughly 40% and the average number of leaf nodes increases by two to three orders of magnitude. However, this is not the case when comparing the Random Forest Regressor to the STreeD ICSC, since the performance for the Random Forest Regressor and the STreeD ICSC is very similar, while the interpretability of the STreeD ICSC is much better, as the average number of leaf nodes is four orders of magnitude larger for the Random Forest Regressor. The STreeD classifier also uses orders of magnitude fewer leaf nodes on average, but the performance is worse than the other models.

From this discussion, we draw one main conclusion. While, on average, the Decision Tree and Random Forest adhere to the general consensus of sacrificing performance for interpretability, the STreeD ICSC does not and instead performs similarly to the Random Forest Regressor while still maintaining interpretability.

5.3 Performance vs. number of leaf nodes per scenario

To better understand the relation between performance and interpretability, we use all data points obtained by the grid search procedure for each scenario and scatter the normalized PAR10 against the number of leaf nodes. Figure 4 illustrates this relationship for ASP-POTASSCO and BNSL-2016.

When analyzing the scatterplot for ASP-POTASSCO, we make a few observations. First, the STreeD ICSC outperforms the Decision Tree Classifier and STreeD Classifier for every number of leaf nodes, e.g., the worst performing STreeD classifier for ten leaf nodes still outperforms the best Decision Tree and STreeD Classifier for the same number of leaf nodes. Second, in the case of fewer leaf nodes, the STreeD ICSC still outperforms all of the other models, e.g., the best performing STreeD Classifier for ten leaf nodes still outperforms all other models.

When analyzing the scatterplot for BNSL-2016, we make some more observations. While the STreeD ICSC performs worse than the Random Forest Regressor, it still shows good performance while using approximately four orders of magnitude fewer leaf nodes.

Other scatterplots show a similar pattern; STreeD Instance Cost Sensitive performs better or sacrifices relatively minimal performance while using significantly fewer leaf nodes.

Based on this analysis, we solidify our conclusion from experiment 5.2.

5.4 Experiment 4: Performance vs. number of leaf nodes for STreeD Instance Cost Sensitive

In this experiment, we explore the relationship between performance and interpretability by analyzing how the performance of the STreeD ICSC changes with varying numbers of leaf nodes. Figure 5 illustrates this for PROTEUS-2014 and ASP-POTASSCO.

For some scenarios, there is potential to significantly reduce the number of leaf nodes while accepting a moderate decrease in performance. For example, for PROTEUS-2014, the number of leaf nodes can be reduced from 32 to 16, resulting in a performance drop from a normalized PAR10 of 59.5 to 53.1. For ASP-POTASSCO, the number of leaf nodes can be reduced from 51 to 11, resulting in a performance drop from a normalized PAR10 of 84.1 to 82.8.

In other cases, the classifier already utilizes a relatively small number of leaf nodes, but there is still room for further reduction. For instance, CSP-Minizinc-Time-2016 shows that the number of leaf nodes can be decreased from 16 to

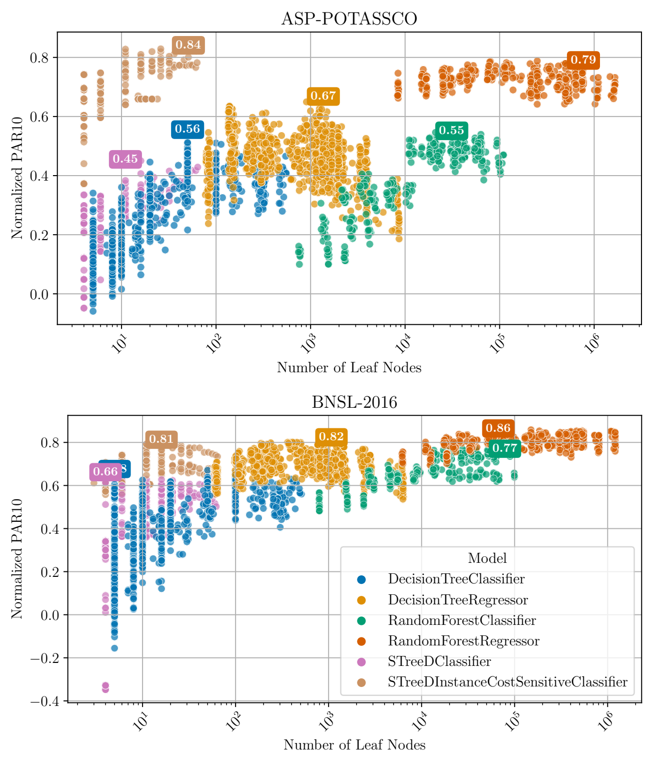


Figure 4: Scatterplot of all normalized PAR10 scores vs. number of leaf nodes, resulting from the grid search procedure, for different scenarios. The best achieved normalized PAR10 is highlighted for each model in white. Note the logarithmic scale on the x-axis.

6, with the performance metric dropping from 84.2 to 71.1. Such scenarios indicate that even models with initially low complexity can be simplified further.

Finally, there are scenarios where the number of leaf nodes is already minimal, leaving little to no room for reduction without significant performance degradation. For instance, datasets such as MIP-2016, MAXSAT-WPMS-2016, and SAT-12 all exhibit a baseline of 11 leaf nodes, indicating simplicity in the model structure. Attempts to reduce the number of leaf nodes further in these cases are likely to result in substantial performance losses.

Based on this analysis, we conclude that the STreeD ICSC uses a relatively small number of nodes and in cases where it uses more than average, usually a simpler model with significantly fewer leaf nodes can be used instead which still offers comparable performance.

6 Conclusions and Future Work

We show that the instance cost-sensitive STreeD classifier performs comparable or better to models requiring orders of magnitude more nodes, which improves interpretability without sacrificing performance. The STreeD ICSC's ability to maintain high performance with fewer nodes shows its potential as a more interpretable alternative to traditional tree-based models and random forests.

The empirical results indicate that the instance cost-sensitive STreeD classifier consistently achieves high performance across various scenarios, often matching or surpassing the performance of more complex models like the Random Forest Regressor. This finding is significant as it suggests that optimal decision trees can be effectively utilized for the Algorithm Selection Problem (ASP) while maintaining a balance between performance and interpretability.

As this research does not define interpretability beyond the number of leaf nodes, future research should delve deeper into the concept of interpretability in the context of decision trees and machine learning models. A comprehensive user study would be beneficial to understand how different stakeholders perceive the interpretability of Optimal Decision Trees, compared to other machine learning models. Additionally, expanding the research to include a broader range of datasets and problem domains will help validate the generalizability of the Optimal Decision Tree approach.

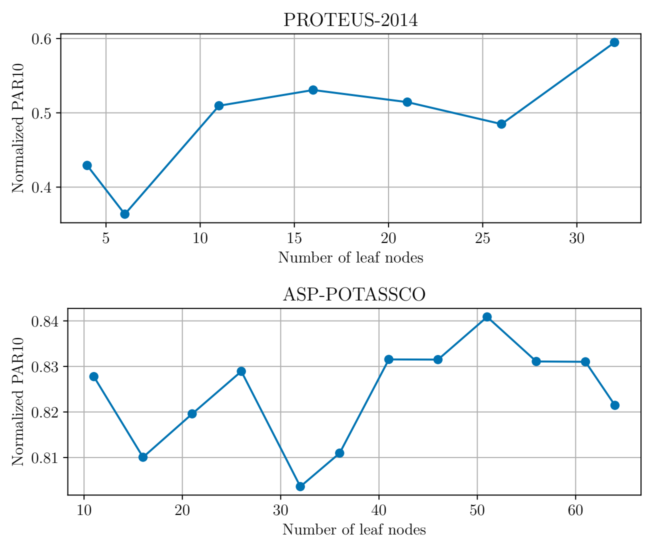


Figure 5: Plot of the relation between normalized PAR10 and Number of leaf nodes for PROTEUS-2014 and ASP-POTASSCO. The data is a result of the grid search procedure.

Responsible Research

The research adheres to the FAIR principles to ensure that the data and code used in the experiments are:

- **Findable:** All metadata and content related to the research are available on the TUD (Technical University of Delft) repository, making it easy for researchers to locate and access.
- **Accessible:** The code will be stored in a public repository. This ensures transparency and allows others to replicate the study. The ASlib datasets are also stored in a public repository [4].
- **Interoperable:** The code is designed to run on both Windows and Unix systems, ensuring compatibility across different operating environments and facilitating broader use by the research community.

[4]https://github.com/coseal/aslib_data/tree/master

- **Reusable:** The repository includes all necessary information to execute and rerun the experiments, including detailed documentation and instructions. This promotes reuse and facilitates further research based on the current study.

References

[1] J. R. Rice, "The algorithm selection problem," in M. Rubinoff and M. C. Yovits, Eds., Elsevier, 1976, pp. 65–118. DOI: https://doi.org/10.1016/S0065-2458(08)60520-3.

[2] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," in *Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[3] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "Evaluating component solver contributions to portfolio-based algorithm selectors," Jun. 2012, pp. 228–241. DOI: 10.1007/978-3-642-31612-8_18.

[4] M. Helmert, G. Röger, and E. Karpas, "Fast downward stone soup: A baseline for building planner portfolios," in *Proceedings of the Workshop on Planning and Learning at the Twenty-First International Conference on Automated Planning and Scheduling*, 2011, pp. 28–35.

[5] L. Pulina and A. Tacchella, "A self-adaptive multi-engine solver for quantified boolean formulas," *Constraints*, pp. 80–116, Mar. 2009. DOI: 10.1007/s10601-008-9051-2.

[6] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, "Aslib: A benchmark library for algorithm selection," *Artificial Intelligence*, pp. 41–58, 2016. DOI: https://doi.org/10.1016/j.artint.2016.04.003.

[7] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, pp. 206–215, May 2019. DOI: 10.1038/s42256-019-0048-x.

[8] A. Agarwal, Y. S. Tan, O. Ronen, C. Singh, and B. Yu, "Hierarchical shrinkage: Improving the accuracy and interpretability of tree-based models.," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., PMLR, 2022, pp. 111–135.

[9] J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *Journal of the American Statistical Association*, pp. 415–434, 1963.

[10] R. Piltaver, M. Luštrek, M. Gams, and S. Martinčić-Ipšić, "What makes classification trees comprehensible?" *Expert Systems with Applications*, pp. 333–346, 2016.

[11] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, *Classification and regression trees*. Chapman and Hall/CRC, 1984. DOI: 10.1201/9781315139470.

[12] L. Breiman, "Random forests," *Machine Learning*, pp. 5–32, Oct. 2001. DOI: 10.1023/A:1010933404324.

[13] G. Hooker and L. Mentch, "Bridging breiman's brook: From algorithmic modeling to statistical learning," *Observational Studies*, pp. 107–125, 2021.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay, "Scikit-learn: Machine learning in python," *Journal of machine Learning research*, pp. 2825–2830, 2011.

[15] M. Sheykhmousa, M. Mahdianpari, H. Ghanbari, F. Mohammadimanesh, P. Ghamisi, and S. Homayouni, "Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, pp. 6308–6325, 2020. DOI: 10.1109/JSTARS.2020.3026724.

[16] H. Blockeel, L. Devos, B. Frénay, G. Nanfack, and S. Nijssen, "Decision trees: From efficient prediction to responsible ai," *Frontiers in Artificial Intelligence*, 2023. DOI: 10.3389/frai.2023.1124553.

[17] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, pp. 1039–1082, 2017.

[18] S. Verwer and Y. Zhang, "Learning decision trees with flexible constraints and objectives using integer optimization," in *Integration of AI and OR Techniques in Constraint Programming*, D. Salvagnin and M. Lombardi, Eds., Cham: Springer International Publishing, 2017, pp. 94–103.

[19] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, pp. 15–17, 1976. DOI: https://doi.org/10.1016/0020-0190(76)90095-8.

[20] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1625–1632, 2019. DOI: 10.1609/aaai.v33i01.33011624.

[21] E. Demirovic, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, "Murtree: Optimal decision trees via dynamic programming and search," in *J. Mach. Learn. Res.*, JMLR.org, 2022, p. 26.

[22] J. van der Linden, M. de Weerdt, and E. Demirović, "Necessary and sufficient conditions for optimal decision trees using dynamic programming," English, in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., Curran Associates, Inc., 2023, pp. 9173–9212.

[23] M. G. Vilas Boas, H. G. Santos, L. H. d. C. Merschmann, and G. Vanden Berghe, "Optimal decision trees for the algorithm selection problem: Integer programming based approaches," *International Transactions in Operational Research*, pp. 2759–2781, 2021.

[24] G. Segalini, "Optimal decision trees for the algorithm selection problem: A dynamic programming approach," Bachelor Thesis, Delft University of Technology, 2023.

[25] A. Biere, "Yet another local search solver and lingeling and friends entering the sat competition 2014," in *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*, 2014, pp. 39–40.

[26] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: Portfolio-based algorithm selection for sat," *Journal of Artificial Intelligence Research*, pp. 565–606, 2008.

[27] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "Evaluating component solver contributions to portfolio-based algorithm selectors," Jun. 2012, pp. 228–241. DOI: 10.1007/978-3-642-31612-8_18.

[28] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in *Principles and Practice of Constraint Programming – CP 2011*, J. Lee, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 454–469.

[29] R. Amadini, M. Gabbrielli, and J. Mauro, "Sunny: A lazy portfolio approach for constraint solving," *Theory and Practice of Logic Programming*, pp. 509–524, 2014.

[30] M. Rizzini, C. Fawcett, M. Vallati, A. E. Gerevini, and H. Hoos, "Static and dynamic portfolio methods for optimal planning: An empirical analysis," *International Journal on Artificial Intelligence Tools*, p. 1 760 006, Feb. 2017. DOI: 10 . 1142 / S0218213017600065.

[31] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. T. Schneider, and S. Ziller, "A portfolio solver for answer set programming: Preliminary report," in *Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning*, Springer, 2011, pp. 352–357.

[32] H. H. Hoos, M. Lindauer, and T. Schaub, "Claspfolio 2: Advances in algorithm selection for answer set programming," *Theory and Practice of Logic Programming*, pp. 569–585, 2014.

[33] M. Maratea, L. Pulina, and F. Ricca, "A multi-engine approach to answer set programming," *CoRR*, 2013.

[34] W. Bożejko, A. Gnatowski, T. Niżyński, M. Affenzeller, and A. Beham, "Local optima networks in solving algorithm selection problem for tsp," *Advances in Intelligent Systems and Computing*, 83 – 93, 2019. DOI: 10.1007/978-3-319-91446-6_9.

[35] G. Drozdov, A. Zabashta, and A. Filchenkov, "Graph convolutional network based generative adversarial networks for the algorithm selection problem in classification," 2020, 88 – 92. DOI: 10 . 1145 / 3437802 . 3437818.

[36] Y. Song, L. Bliek, and Y. Zhang, *Revisit the algorithm selection problem for tsp with spatial information enhanced graph neural networks*, 2023.

[37] A. Kostovska, A. Jankovic, D. Vermetten, S. Džeroski, T. Eftimov, and C. Doerr, "Comparing algorithm selection approaches on black-box optimization problems," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, Lisbon, Portugal: Association for Computing Machinery, 2023, 495–498. DOI: 10.1145/3583133.3590697.

[38] J. Scott, A. Niemetz, M. Preiner, S. Nejati, and V. Ganesh, "Algorithm selection for smt: Machsmt: Machine learning driven algorithm selection for smt solvers," *Int. J. Softw. Tools Technol. Transf.*, 219–239, 2023. DOI: 10.1007/s10009-023-00696-0.

[39] F. Gonard, M. Schoenauer, and M. Sebag, "Algorithm selector and prescheduler in the icon challenge," in Jan. 2019, pp. 203–219. DOI: 10.1007/978-3-319-95104-1_13.

[40] A. Visentin, A. Ó. Gallchóir, J. Kärcher, and H. Meyr, "Explainable algorithm selection for the capacitated lot sizing problem," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, B. Dilkina, Ed., Cham: Springer Nature Switzerland, 2024, pp. 243–252.

[41] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5g/b5g mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, pp. 137 184–137 206, Sep. 2019. DOI: 10.1109/ACCESS.2019.2942390.

[42] Y. Luo, H.-H. Tseng, S. Cui, L. Wei, R. K. Ten Haken, and I. El Naqa, "Balancing accuracy and interpretability of machine learning approaches for radiation treatment outcomes modeling," *BJR Open*, p. 20 190 021, 2019. DOI: 10.1259/bjro.20190021.

[43] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, pp. 82–115, 2020. DOI: https://doi.org/10.1016/j.inffus.2019.12.012.

[44] G. Ciatto, M. Schumacher, A. Omicini, and D. Calvaresi, "Agent-based explanations in ai: Towards an abstract framework," in Jul. 2020, pp. 3–20. DOI: 10.1007/978-3-030-51924-7_1.

[45] A. Kumar, S. Dikshit, V. H. C. Albuquerque, and M. R. Khosravi, "Explainable artificial intelligence for sarcasm detection in dialogues," *Wirel. Commun. Mob. Comput.*, 2021. DOI: 10.1155/2021/2939334.

[46] S. Ali, T. Abuhmed, S. El-Sappagh, K. Muhammad, J. M. Alonso-Moral, R. Confalonieri, R. Guidotti, J. Del Ser, N. Díaz-Rodríguez, and F. Herrera, "Explainable artificial intelligence (xai): What we know and

what is left to attain trustworthy artificial intelligence," *Information Fusion*, p. 101 805, 2023. DOI: https://doi.org/10.1016/j.inffus.2023.101805.

[47] Z. C. Lipton, "The mythos of model interpretability," *Commun. ACM*, 36–43, 2018. DOI: 10.1145/3233231.

[48] K. P. Bennett and J. L. Blue, "Optimal decision trees," *Pattern Recognition*, pp. 1311–1322, 1996. DOI: 10.1016/0031-3203(95)00135-7.

[49] D. Bertsimas and R. Shioda, "Classification and regression via integer optimization," *Operations Research*, pp. 252–271, 2007. DOI: 10.1287/opre.1060.0355.

[50] Y. Hu, C. Rudin, and M. Seltzer, "Optimal sparse decision trees," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2019)*, 2019, pp. 7265–7273.

[51] G. Aglin, S. Nijssen, and P. Schaus, "Learning optimal decision trees using a boosted branch-and-bound search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 3146–3153.

[52] T. Liu, R. Amadini, M. Gabbrielli, and J. Mauro, "Sunny-as2: Enhancing sunny for algorithm selection," English, *Journal of Artificial Intelligence Research*, pp. 329–376, 2021. DOI: 10.1613/JAIR.1.13116.

[53] D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos, "Opening the black box: Automated software analysis for algorithm selection," in *Proceedings of the First International Conference on Automated Machine Learning*, I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, Eds., PMLR, 2022, pp. 6/1–18.

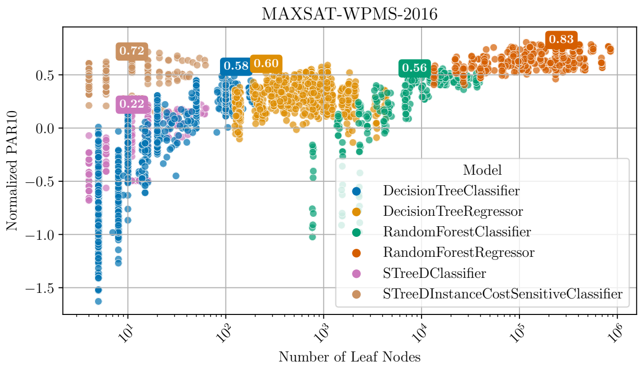
[54] J. Hanselle, A. Tornede, M. Wever, and E. Hüllermeier, "Hybrid ranking and regression for algorithm selection," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 59 – 72, 2020. DOI: 10.1007/978-3-030-58285-2_5.

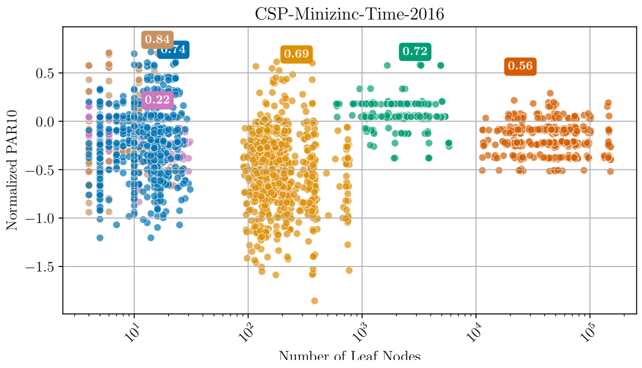
[55] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, and H. Trautmann, "Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 48 – 64, 2020. DOI: 10.1007/978-3-030-58112-1_4.

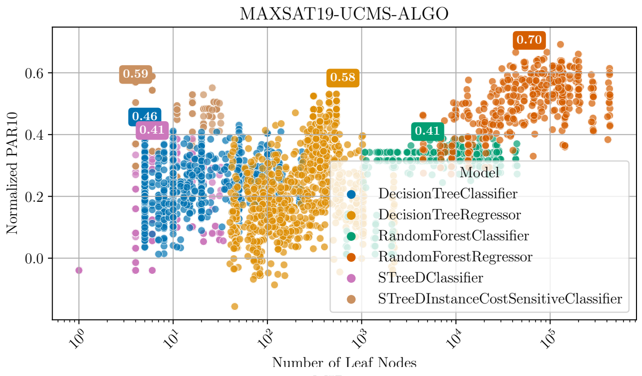
[56] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," eng, *Entropy (Basel)*, p. 18, 2020. DOI: 10.3390/e23010018.

[57] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 2)*, 2024.

A Performance vs. number of leaf nodes (other scenarios)







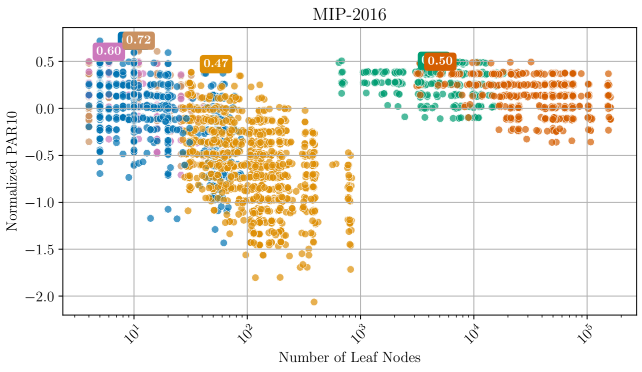
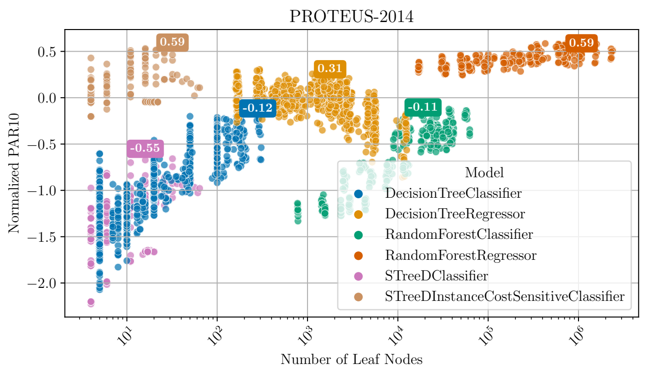
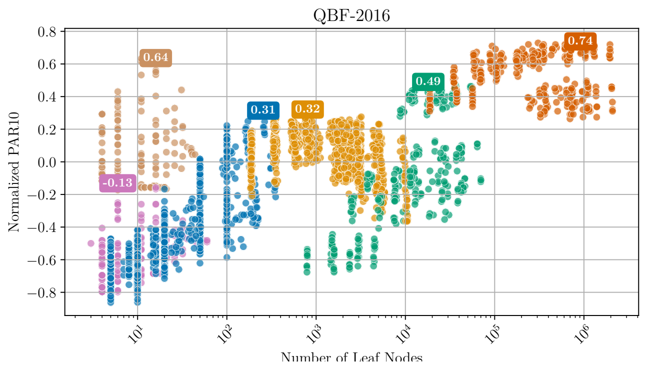
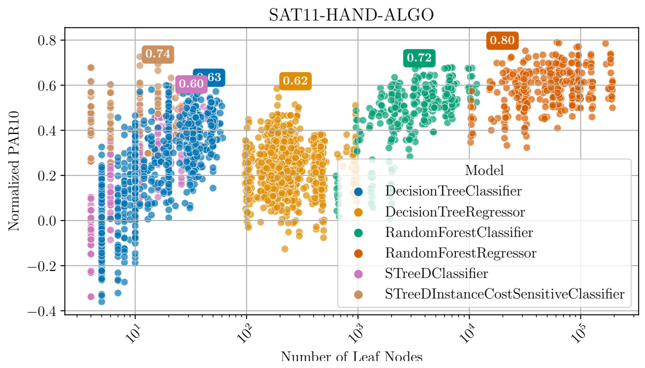
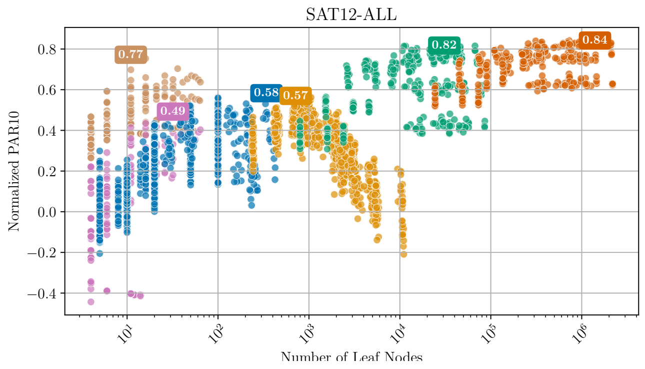


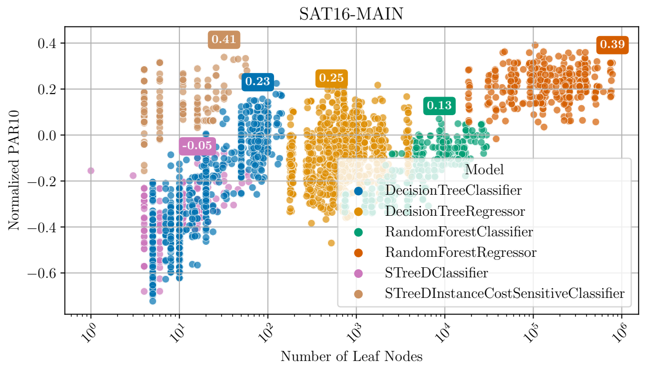
Figure 6: Scatterplot of all normalized PAR10 scores vs. number of leaf nodes, resulting from the grid search procedure, for different scenarios. The best achieved normalized PAR10 is highlighted for each model in white. Note the logarithmic x-axis. TBD: fix plots



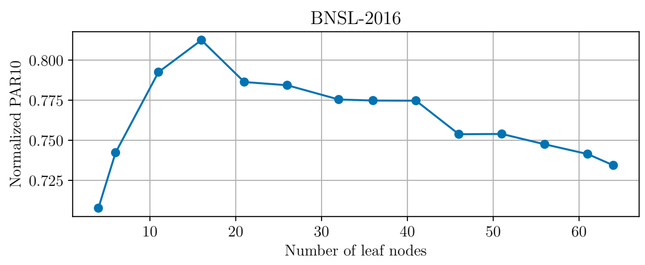


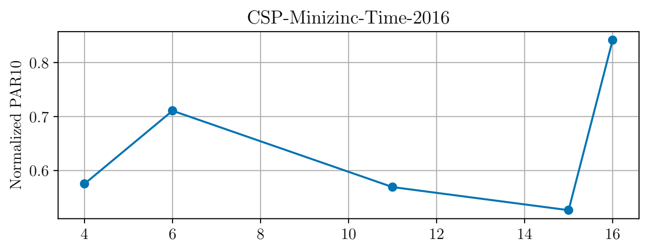


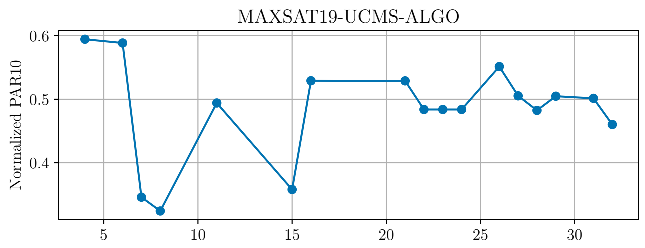


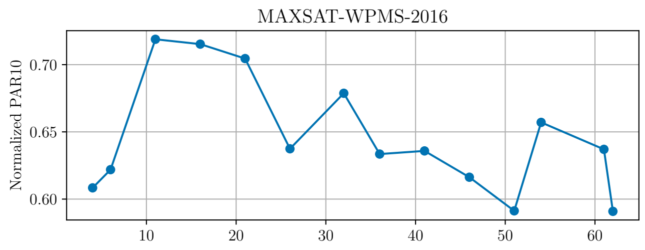


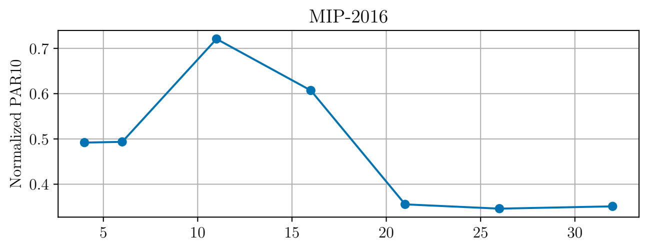
B Performance vs. number of leaf nodes for STreeD Instance Cost Sensitive (other scenarios)

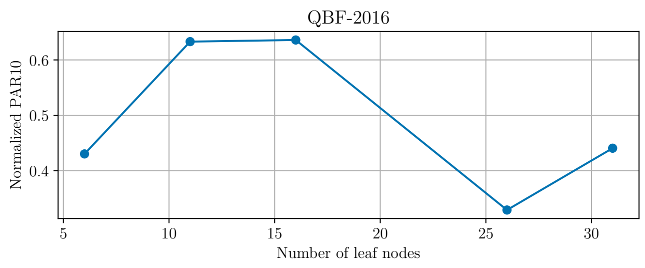

BNSL-2016
Normalized PAR10
Number of leaf nodes

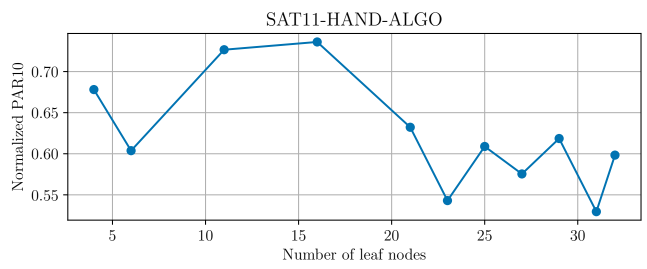

CSP-Minizinc-Time-2016
Normalized PAR10

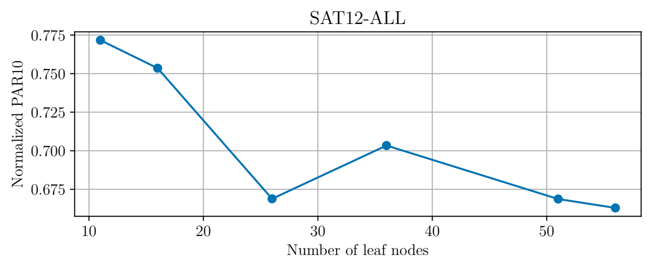

MAXSAT19-UCMS-ALGO
Normalized PAR10

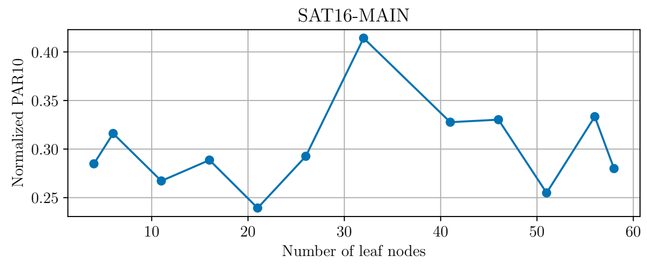

MAXSAT-WPMS-2016
Normalized PAR10


MIP-2016
Normalized PAR10


QBF-2016
Normalized PAR10
Number of leaf nodes


SAT11-HAND-ALGO
Normalized PAR10
Number of leaf nodes


SAT12-ALL
Normalized PAR10
Number of leaf nodes


SAT16-MAIN
Normalized PAR10
Number of leaf nodes