Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis in Quantitative Marketing

MSc Econometrics and Management Science

The Quest for Optimal Trees on Large Datasets

Author:
Bart Dubbeldam (480675)

Supervisor:
Dr. M. van de Velden
Second assessor:
Dr. W. van den Heuvel

June 13, 2022

Abstract

Classification and Regression Trees (CART) are widely used Machine Learning techniques, due to it's versatility and interpretability. However, the CART algorithm creates every split in isolation of the other splits, this leads to suboptimal decision trees. A second downside is that decision trees are not robust to varying training samples. A different training sample can result in a different tree. Bertsimas and Dunn (2017) used a Mixed Integer Program (MIP) to estimate the tree at once, achieving global optimality. The downside of this method is that it does not perform well on larger datasets. In this thesis I study four MIP formulation: OCT, BinOCT, Benders and Flow. The main goal of this thesis is to apply these methods to larger datasets. I use real world datasets from the UCI data repository and synthetically created datasets. A secondary goal is to study whether optimal tree methods make the trees more robust to different training samples. My main finding is that the biggest influence on the computation time of the MIP is the depth of the tree. If the depth of the tree is higher than two, then only for datasets with less than 150 observations, the MIP is solved to optimality. Since almost every problem requires deeper trees, which is time-wise not feasible, the MIP approach seems to not to be a good method to use at this moment in time. However, it does appear that optimal trees are more robust to different training samples, though further research is necessary to ensure.

Contents

Symbols

Sets

$\mathcal{B}$	Set of branching nodes
$\mathcal{L}$	Set of leave nodes
$\mathcal{T}$	Set of terminal nodes
$\mathcal{F}$	Set of features
$\mathcal{I}$	Set of Observations
$\mathcal{K}$	Set of classes
$\mathcal{A}(n)$	Set of ancestor nodes

Flow Graph Variables

$\mathcal{V}$	Vertices of the flow graph
$\mathcal{A}$	Arcs of the flow graph
$l(n), r(n)$	Left and right child node of n
$a(n)$	Ancestor of node n
s	Source node
t	Sink node

Decision Variables

w_k^n	Binary variable: whether leave node n has label k
b_{nf}	Binary variable: whether feature f is used to branch on node n
$z^i_{a(n),n}$	Binary variable: whether observation i flows from $a(n)$ to n
p_n	Binary variable: whether a prediction is made on node n

f	Feature
k	Class
I	Individual observation
d	Depth of the tree
λ	regularization parameter

Formulations

OCT	Formulation from Bertsimas and Dunn (2017)
binOCT	Formulation from Verwer and Zhang (2019)
FlowOCT	Flow-formulation from Aghaei et al. (2021)
BendersOCT	Benders-formulation from Aghaei et al. (2021)

1 Introduction

In this research I investigate whether the popular decision tree can be improved and solved to optimality for large datasets. A decision tree can be used for classification and regression problems. Although, a single decision tree might not always perform as good as more advanced Machine Learning techniques, in terms of prediction accuracy, it is still a widely-used technique. Because decision trees are versatile, fast and easily interpretable, it is also a popular tool for policy- and decision makers.

The Classification and Regression Tree (CART), proposed by Breiman et al. (1984), is the most common form of a decision tree,it has almost 17.000 citations according to semanticscholar in March 2022. However, there are two main disadvantages with this approach.

Firstly, CART trees are highly variable. The tree can have a completely different form depending on which part of the data is used to train the tree. It is possible that when a different set of observations is used to train CART, the resulting decision tree has a different form. Needless to say, this is not desirable from a policy and decision making perspective. Because with a highly variable tree, the decisions or policies that are created might be contingent upon the subset of the data that is used to train the tree.

Secondly, the final tree can be suboptimal due to the way the tree is constructed. CART is constructed in a top-down approach. Starting from the root node, new splits are created until a stopping condition is met. Stopping conditions can be: the maximum depth of the tree, a minimum number of datapoints in every node, a minimum increase per split, etc. Every split is created by optimizing an impurity measure, for example the gini- or entropy-coefficient. These impurity measures ensure that the data is split optimally into the child nodes. However, the impurity measure only optimizes the distribution of the data for the next split. It does not take into account the splits that are yet to come, resulting in the fact that CART might be suboptimal.

In this thesis, I investigate a method that formulates the problem of finding a classification tree as a Mixed Integer Program (MIP). This formulation makes it possible to estimate the tree at once while optimizing an objective function, like the Mean Squared Error (MSE) or the accuracy. Bertsimas and Dunn (2017) were

the first to successfully create decision trees through solving a MIP. There are three main factors that influence the size of the MIP, *1)* the number of observations in the dataset, *2)* the number of features in the dataset and *3)* the depth of the tree. This MIP problem is NP-hard, which means that there is no known algorithm that can solve the problem in polynomial time. Therefore, the approach of Bertsimas and Dunn takes a long time to find optimal trees, especially for larger datasets and deeper trees.

The goal of this thesis is to study whether optimal tree methods can be applied to larger datasets, and if not, whether I can create a method that is able to handle larger datasets. To answer this question, I study the effect of the number of observations, number of features and the depth of the tree on the computation time. By discovering which factor influences the computation time the most, it becomes possible to create a targeted solution.

Secondly, I aim to study whether optimal tree methods reduce the variability that we observe in CART trees. The form and variables in a CART tree can vary depending on the training sample used to train the tree. I want to study whether the shape and the variables in an optimal tree, are more robust than CART to different training samples. I study this by comparing the true discovery rate[1] for the synthetic datasets and the standard deviation of the accuracy for different training samples.

To study the influence of different factors in the complexity of the MIP, I use two kind of datasets. On the one hand, synthetically created datasets, of which the data-generating process (DGP) is known. On the other hand, a variety of real-world datasets from the UCI Machine Learning Repository. The synthetically created datasets are used to study the effect of different parameters (number of observations, number of features, depth of the tree) on the computation time. The real-world datasets are used to study how the methods fare when applied to real datasets. In this paper I study the performance of CART and four optimal tree methods: OCT (Bertsimas and Dunn, 2017), BinOCT (Verwer and Zhang, 2019), Benders and Flow (Aghaei et al., 2021) on these datasets.

From analyzing the four MIP methods and CART on the datasets, I discovered that the biggest influence on the running time of the method is the depth of the

[1]the percentage of features that is both in the decision tree and the features used to create the dataset

tree. For trees that have a higher depth than two, the optimal tree methods are only able to find an optimal tree for small datasets with less than 150 observations. Therefore, I believe that it is currently not possible to apply these methods on larger and deeper trees, since the trees will start being too small to have good predictive power.

Besides the optimality, I also studied the discovery rate for the synthetic datasets and the variance of the mean accuracy, in order to find out how variable the trees are between different training samples.

I found that the true discovery rate of the MIP approaches is higher than of CART and that the standard deviation of every method is low. The standard deviation of the mean accuracy turns out to be very low for every method (also for CART). This might indicate that the optimal tree methods result in more robust trees. However, I discovered that the trees can be different, in spite of the same true discovery rate and the same accuracy. Therefore, the proxies used might not be very suitable to research the robustness of the trees.

In the remainder of this paper I first give an overview of the existing literature (Section 2). Then I discuss the theory that is necessary for a good understanding of the methodology (Section 3). After this, I discuss the methods that are used (Section 4), followed by the result section (Section 5). I end with a conclusion and some ideas for future research (Section 6).

2 Literature Review

The idea to form the tree at once, was already proposed in the original work on CART. The authors write (Breiman et al., 1984, p.42):

> Finally, another problem frequently mentioned (by others, not by us) is that the tree procedure is only one-step optimal and not overall optimal. (...) If one could search all possible partitions (...) the two results might be quite different. (...) At this stage of computer technology, an overall optimal tree growing procedure does not appear feasible for any reasonably sized dataset.

So, the top-down approach with which CART is created is not because Breiman et al. deemed this the best approach. However, their approach was taken due to computational limitations in finding an optimal tree. Because it was known that the problem of constructing optimal binary decision trees is NP-hard (Hyafil and Rivest, 1976).

Since, the publication of the work on CART, several approaches have been proposed to create optimal trees. A few of the different approaches are: linear optimization (Bennett, 1992), continuous optimization (Bennett and Blue, 1996) and genetic algorithms (Son, 1998). However, none of these approaches provide certifiable optimal trees and there is no way to prove that the tree constructed through these methods is optimal. Something which is possible when using MIP.

Even though it was known that many statistical problems can be formulated in a MIO way (Arthanari, 1981), it took a long time before someone applied the MIO approach to practical problems. According to Bertsimas and Dunn (2017) the reason that this approach has not been tried before is the common perception that MIO problems are intractable, i.e. not computationally feasible, for small to medium instances.

There has been an astonishing increase in the computational power of MIO solvers and the computation power of computers. Bertsimas and Dunn note that in the last 25 years the increase in the computational power of mixed-integer linear-programming(MILP) solvers increased by approximately 800 billion times. Due to changes and improvements in the solvers and an increase in computational hardware. This large increase has led Bertsimas to believe that the view that MIO

problems are intractable is outdated. He tackled multiple statistical problems by applying MIO methods. He succesfully applied MIO to: least quantile regression(Bertsimas and Mazumder, 2014), linear regression(Bertsimas and King, 2016), logistic regression (Bertsimas and King, 2017) and best subset selection (Bertsimas et al., 2015).

After being succesful with MIO in the above mentioned statistical problems, Bertsimas and Dunn tackled the problem of suboptimal decision trees in Bertsimas and Dunn (2017). They propose a novel formulation of the decision trees as an MIO, resulting in optimal classification trees (OCT). The method performed well on training and test data. It outperformed CART on all instances, by approximately 1-2%, in an out-of-sample comparison on 53 datasets from the UCI machine learning repository.

However, the method of Bertsimas and Dunn (2017) becomes computationally infeasible for a treedepth larger than 5 and a dataset with more than approximately 5500 observations. This is not suprising when we study the factors that influence the number of decision variables and constraints. The tree depth, the number of observations and the number of variables all play a role.

Datasets of more than 5500 observations are often encountered. Hence, it would be very beneficial to find ways in which the framework of Bertsimas and Dunn (2017) can be applied to applications where the datasets that are encountered are larger. Creating a MIO that can tackle larger datasets has been tried by multiple academics since the publication of Bertsimas and Dunn (2017) paper.

The work of Bertsimas and Dunn (2017) opened the road to more research on the topic of optimally creating classification trees. There are two main ways in which researchers tried to improve upon the original work on optimal classification trees(OCT). Either by changing the settings of the MIO solver, so that the solver solves the problem faster, or by creating a better formulation. In this case 'better' is either a formulation that uses less decision variables, or a formulation that is tighter and hence, solves the problem faster.

The work of Feijen (2018) falls mainly in the first category. He continues with the framework of Bertsimas and Dunn (2017). Feijen focuses on finding smart branching strategies, his adaption to the solver reduces the running times significantly. However, he tests his methods only on the Iris dataset (Fisher, 1936), a small dataset with only 150 observations, four features and three classes. Next

to that, the trees he constructs have a maximum depth of two. These two factors make it questionable whether this approach is sufficient for larger datasets.

Verwer and Zhang (2019) use the second approach and formulate the decision tree as a binary linear program, meaning that the formulation only uses binary variables. This formulation has the benefit that the number of decision variables in the formulation is largely independent of the number of rows in the dataset. However, in the process of reducing the number of decision variables the Linear Optimization (LO) relaxation, compared to the formulation of Bertsimas and Dunn (2017) is less *strong*. The result is that neither method is consistently outperforming the other method.

Aghaei et al. (2021) also follows the second approach, they propose a flow formulation to obtain OCT. In this formulation, correctly identified datapoints flow from the root node through the tree, while incorrectly classified datapoints are not allowed to flow through the tree. The number of decision variables is the same as in the formulation of Bertsimas and Dunn (2017). However, the linear programming relaxation is *stronger*. Next to the *stronger* formulation they are able to decompose the problem into multiple smaller problems that are easily solvable. Using the Benders' decomposition algorithm (Benders, 1962), they are able to speed-up computation and reduce computer memory consumption, allowing the solution of larger MIO problems. They find that this *strong* formulation leads to solving the problem 31 times as fast and obtaining an out-of-sample performance increase by up to 8% compared to existing OCT-approaches.

3 Theory

The main goal of this thesis is to study whether optimal tree methods can be applied to larger datasets. CART is the main method to estimate decision trees. However, there are a few limitations to CART which are resolved by the optimal tree methods. In this section I look at the basic idea of CART, and take a closer look at the limitations of CART. Dunn (2018) is taken as a guide to writing the section on CART, the figures are also taken from Dunn (2018). Secondly, I discuss the theory on how to solve a Mixed-Integer Program(MIP). Followed by a discussion on how Gurobi, the solver used in this thesis, solves an MIP. Finally, I give some basic theory on Benders' decomposition, a method that is used in this thesis to solve MIP faster.

3.1 CART Decision Tree

CART takes a top-down approach to determine the partitions in the tree. Starting from the root node, a measure is optimized to find the best next split. The points are divided according to this split. Then, the same measure is optimized at the new partition. Common impurity measures for classification problems are the *gini-* or *entropy-coefficient*. The impurity measure quantifies the similarity of the labels among points in a group. The lower the value, the more similar the points are in that group.

This top-down approach continues at each new node until one of the following stopping criteria is met:

1. The impurity of the node cannot be reduced further (all the points in the node have the same label).

2. The node being partitioned has fewer points than the minimum allowed leaf size, N_{min}.

3. The maximum depth, d_{max}, is reached

When the stopping criteria is met for each new node (i.e. the partitioning ends), then labels are assigned to the leaf nodes. The label of leaf nodes is used to predict the label of the new data points. The label that is assigned to the leaf node, is the mode of the labels of the training points that fall into that leaf node.

3.1.1 Example

In this section I illustrate the workings of CART applied to the Iris dataset (Anderson, 1936), a dataset from the UCI Machine Learning Repository (Dua and Graff, 2017). The Iris Dataset contains four features (length and width of sepals and petals) of 150 samples of three different species of Iris (Iris setosa, Iris virginica and Iris versicolor). The goal is to predict the species of Iris using only the four physical measurements.

In Figure 1a the data is plotted with the Length and Width on the x- and y-axis, respectively. Figure 1b-e show the consecutive splits of the CART algorithm on the dataset. The first split is on the petal length. Since this partitions *Setosa* perfectly from the other two species, no further split is needed on the left side of the partitioning. The second split, splits on petal width. Since the split doesn't completely seperate the species *Versicolor* and *Virginica* from each other, next splits can be made. The third and fourth split make the separation between the species *Versicolor* and *Virginica* pure. Figure 1f shows the final labels assigned to each partitioning. The final partitioning is also shown in tree form in Figure 2.

One of the problems with CART, is the risk of overfitting the tree on the training data. With a very large tree, it is easy to achieve a high prediction accuracy on the training data, but probable that the accuracy on the test data is low. The chance of overfitting can be reduced by controlling for the trade-off between the training accuracy and the number of splits in the tree. The more splits in a tree, the higher the complexity of the tree.

In CART, this trade-off is regulated by introducing a cost-complexity parameter(CCP), α. The CCP ensures that a new split improves the accuracy by at least a certain percentage. If not, then the split is removed from the decision tree. For an α of 0.01, each split needs to improve the accuracy with more than 0.01. So through this parameter, the complexity of the final tree can be controlled. Hence, the complexity of the tree and the chances of overfitting can be reduced. The optimal value of the CCP is often determined through cross-validation.

After the tree has been fitted on the training data, the accuracy improvement is determined for each split in the tree. If the accuracy improvement due to a split is lower than α, then this split is replaced with a single leaf node. This process is known as *pruning*.

Figure 1: An example of the CART algorithm applied to the Iris dataset that shows the recursive partitioning step-by-step.

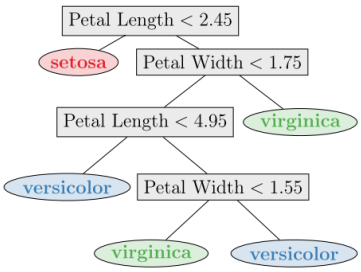

Figure 2: The tree learned by CART for the Iris dataset.

In the above example, the fourth split, applied in Figure 1e, creates a partition that increases the accuracy by 1 point. Without this split, the misclassification would be 2 points, while with the split, the misclassification is 1 point. The split thus only increased the accuracy by 1 point. The other splits in the tree are much more significant in terms of accuracy improvement. Through pruning, the fourth split can be removed (depending on the value of the CCP). This would result in the tree in Figure 3.

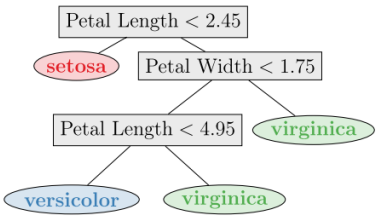

Figure 3: The CART tree learned for the Iris dataset after pruning.

Two of CARTs main problems are, *1)* the high variability and *2)* the greedy top-down approach. Another problem might be the use of a gini- and entropy coefficient instead of the accuracy to build the tree. In this section, I touch upon these problems of CART and give a graphical example.

The main problem with CART is the greedy nature. In each branching node, a new split is determined in isolation of future splits. This can lead to a split that is initially 'strong', i.e. a split that has a low impurity measure, but the splits after this split are all 'weak', i.e. there are only high impurity measures. The danger is that the CART tree doesn't capture the true characteristics of the dataset well, which might lead to a poor performance on the test set. Figure 4 shows an example

where a tree grown by the top-down approach is very different from the tree that generated the data.

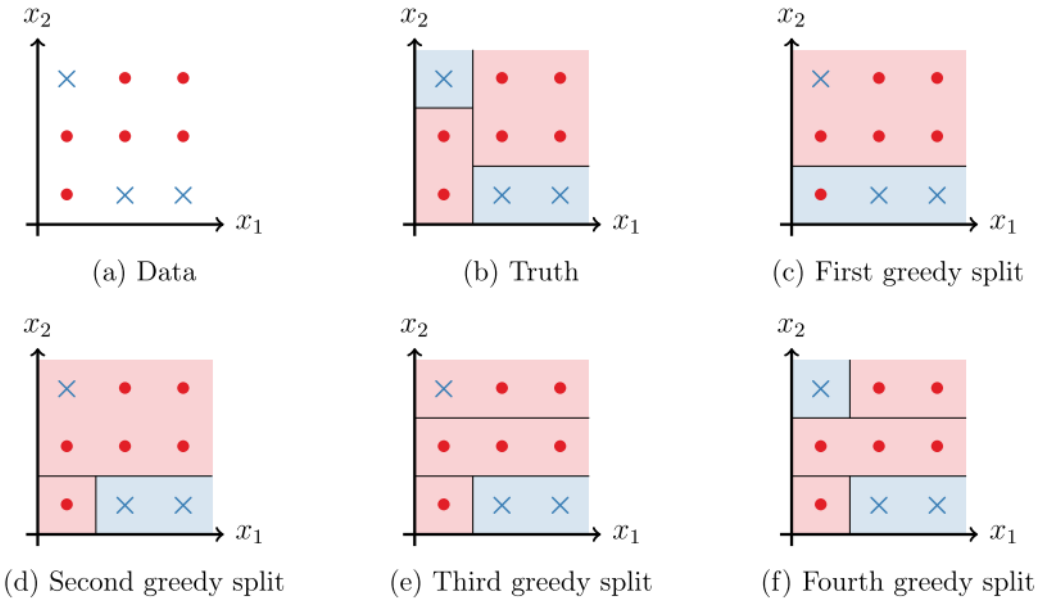

Figure 4: An example of CART on a synthetic dataset. Figure 4a shows the data, and 4b shows the splits that are used to generate the data. Figure 4c-f show the splits that are made in a greedy approach. The final tree is totally different from the tree used to generate the data.

A second problem with CART is the high variability in the tree. Depending on the training set used, the splits and the shape of the tree can vary. One of the ways that decision trees are used is to make policies. Because the rules given by a decision tree are easy to interpret. However, if the trees differ, then it is hard to create consistent policies.

Another limitation of CART is the use of an the gini- and entropy coefficient, rather than the misclassification rate when selecting the next split. Given that the misclassification rate is the final objective of the decision tree, it is peculiar that an impurity measure is used to build the tree. Breiman et al. (1984, p. 97) explains why CART uses an impurity measure instead of the misclassification rate:

> ... the [misclassification] criterion does not seem to appropriately reward splits that are more desirable in the context of the continued

> growth of the tree. ... This problem is largely caused by the fact that our tree growing structure is based on a one-step optimization procedure.

This also becomes clear from Figure 4. From Figure 4d to 4e, a split is added that improves the impurity measure, but does not improve the misclassification error. So if the misclassification error would be optimized, we could not continue growing the tree. It's sensible to believe that growing the decision tree by optimizing the final objective leads to better splits. However, because of top-down induction, this is not possible.

By estimating the entire tree at once, CARTs problems might be resolved. Each split is determined with knowledge of all the other splits. On top of that, the CCP can be applied during the growing, instead of after the tree has been grown, as is the case with CART. This results in the optimal decision tree. A tree can be estimated at once by writing it as a MIP.

Every decision tree has to satisfy a few rules. For example: at each node just one feature can be used to branch upon; a datapoint cannot end up in multiple leave nodes; an observation can only go left or right on a branching node, etc. By writing all the rules down as constraints in a MIP, the decision tree can be estimated at once by solving the MIP.

3.2 Solving a MIP

There are different ways to formulate a MIP. Some formulations are *stronger* (i.e. have a smaller array of possible solutions, and hence are easier to solve), than others. The different MIPs can be solved by mathematical programming solvers. There are a wide range of commercial and non-commercial solvers available. A few of the most common solvers are: Gurobi(Gurobi Optimization, LLC, 2022), CPLEX(Cplex, 2009), Xpress(FICO, 2022). In Mittelmann (2018), recent versions of the different solvers are compared to each other on a benchmarking dataset. From this comparision it becomes clear that Gurobi is the best solver, in terms of speed and number of solved problems.[2] Since, Gurobi is the fastest solver in this

[2]Mittelmann (2018) is the last comparison I can find where GUROBI, CPLEX and Xpress are compared together. Apparently this is due to a conflict between IBM, FICO and GUROBI. Gurobi misused the results of the 2018 benchmark results to make their solver appear better

benchmark we use this solver to solve the MIP problems.

Mathematical programming solvers use branch-and-bound to solve the MIP. In this section I briefly discuss branch-and-bound following the Gurobi manual(Gurobi Optimization, LLC, 2022). In a basic branch-and-bound we begin with the original MIP. Due to the integrality restrictions it is hard to solve the MIP directly. Hence, the integrality restrictions are dropped. The resulting linear program(LP) is the relaxation of the original MIP. Solving an LP problem is computationally easier than solving a MIP. It can happen that the solution to the LP problem satisfies all the integrality constraints, in that case this solution is also the solution to the MIP. However, it is more likely that the variables that are supposed to be integers are fractional in the LP-solution. In that case we branch on one of these fractional variables.

As an example suppose that we have the MIP denoted by P_0. In the LP relaxation one of the fractional variable is x and it has a value of 7.7. We can then exclude this value by solving one problem with the added constraint $x \leq 7$ and one problem with the added constraint $x \geq 8$. The variable x is then called the branching variable, and by branching on x we produce two sub-MIPs P_1(P_0 with the added constraint $x \leq 7$) and P_2 (P_0 with the added constraint $x \geq 8$). If we compute the optimal solution to P_1 and P_2 then the best of these two optimal solutions is the optimal solution to P_0. So P_0 is replaced by two sub-MIPs P_1 and P_2. To solve P_1 and P_2 we might again need to relax the MIP and choose a branching variable. In doing so we generate what is called a search tree. Every MIP generated is a node of the tree, with P_0 being the root node. The leaves of the tree are all the nodes that we haven't yet branched from. If a point is reached in which we can solve, or otherwise dispose of all leaf nodes, then we solved the original MIP.

A solved or disposed node is called a fathomed node. There are three ways in which a node can be fathomed. The first way is to find a solution that satisfies all the integrality restrictions. Then this is a feasible solution to the original MIP. The current node is fathomed, since there is no need to branch on this node anymore. If the integer solution is the best integer solution found up to this point (i.e. the solution with the best objective value), then this solution is the incumbent. The

compared to the CPLEX and Xpress solver. IBM and FICO reacted by requesting the results of their solvers to be removed from the benchmarks(Mittelmann, 2020)

incumbent is only updated when the new integer solution is better than a previous integer solution, or if there was no incumbent solution (at the start of the search there is no incumbent solution).

The second way in which a node can be fathomed is when we branch and the added restriction makes the LP-relaxation infeasible. If there is no feasible solution to the LP relaxation there is no feasible solution to the MIP problem.

The third way in which a node is fathomed is, when the solution to the LP-relaxation is worse than the incumbent solution. Since the solution to the LP relaxation is a lower bound (in case of a minimization problem), to the optimal MIP solution. Thus, the optimal solution of the search from that node is never better than the incumbent.

Another important concept in the branch-and-bound method is the optimality gap. The optimality gap measures the difference between the best known upper bound and the best known lower bound. I discuss the bounds in the case of a minimization problem (for a maximization problem the upper bound becomes a lower bound and vice versa). The upper bound is the incumbent solution. The lower bound is the minimum of all the objective values of the LP-relaxations. Because, the optimal solution to the MIP is never higher than the optimal solution of its LP-relaxation. Hence, the optimal solution to the original problem is never better than the minimum value of all the LP-relaxations in the leaf nodes. The difference between the lower and upper bound is called the optimality gap.

The optimality gap becomes smaller throughout the process of branch-and-bound. When branching is applied constraints are added to the LP relaxation and hence the optimal value to the LP relaxation increases. Ensuring a higher lower bound. When a new incumbent solution is found the upper bound is updated. When the upper and lower bound meet, i.e. the optimality gap is zero, optimality is reached and we found the solution to the original problem.

3.3 Gurobi and Branch-and-Bound

The Gurobi solver does not just use a textbook branch-and-bound algorithm. They improved their solver with a variety of methods that increase the solving time significantly. The four biggest contributors are *presolve, cutting planes, heuristics* and *parallelism*. In the coming section I elaborate on these methods.

3.3.1 Presolve

Presolve is often applied in advance of the start of the branch-and-bound procedure. It reduces the problem, with the intend to reduce the size and tighten the formulation. A simple example is the following. Suppose that a MIP problem and the following constraints are part of the formulation:

$$\begin{aligned} 2x_1 + 2x_2 &\leq 1 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Dividing both sides by 2 leads to:

$$x_1 + x_2 \leq \frac{1}{2}$$

Since x_1 and x_2 are both required to be non-negative integers this inequality clearly implies that $x_1 = x_2 = 0$. Hence both of these variables and the constraint can be removed from the formulation. By removing these variables the set of feasible integer solutions did not change, but the set of feasible solutions to the LP relaxation did. The set of feasible solutions to the LP relaxation became smaller. Tightening the MIP formulation can be very beneficial to the speed with which a MIP problem is solved.

3.3.2 Cutting Planes

The second idea that improves the performance of a solver is to add cutting planes. The idea of a cutting plane is similar to presolve. Namely, tighten the formulation by removing undesirable fractional solutions. Instead of tightening the formulation before starting the branch-and-bound, as in presolve, this is done during the solution process. During the solution process constraints are added to the LP relaxation, the constraint is chosen in such a way that it cuts off a fractional solution. This is similar to branching where, to remove a fractional solution, a constraint is added to the LP relaxation. However, adding a cutting plane does not have the undesirable effect of creating sub-problems.

Figure 5 shows an example of a cutting plane. The dots in the figure are integer points and the grey area is the set of feasible solutions. The purple line shows the objective function, each point on the purple line has the same value. The closer

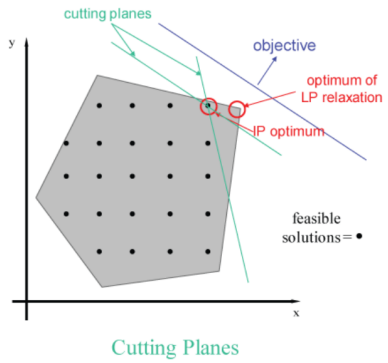

Figure 5: From Klotz and Newman (2013): An example of cutting planes.

to the origin the worse the optimal solution. Hence, the optimal solution to the LP relaxation is at the corner that has a circle around it. At this point the the objective function has the maximum distance to the origin without the solution being infeasible. The best IP solution is the circled dot.

The solution to an LP relaxation can be found relatively quickly, since existing methods(like simplex, or the interior point method) are able to find the vertice at which the solution is optimal. However, finding the MIP solution is a lot harder. By adding a cutting plane (the green line in Figure 5) a set of the fractional solutions is removed. Next to that, it might happen that an IP solution becomes the vertice of the set of feasible solutions (as is the case in Figure 5) . This is an ideal scenario, since the solution to the LP relaxation (which is easier to find) is also the solution to the IP.

3.3.3 Heuristics

Gurobi also implements heuristics to quickly find feasible solutions (even though it is unlikely to be optimal), this may or may not improve the incumbent. If it doesn't improve the incumbent the effort is wasted. However, having good incumbent, and finding them quickly, can speed up the branch-and-bound algorithm. Thus, they argue, it is worth to invest a bit of computation time at each node to find a feasible solution. Having a good incumbent tightens the bound, which makes it more likely that the value of an LP relaxation exceeds the bound (in a minimization problem), which in turn leads to a node being fathomed.

One example of such a heuristic is to take the solution to the LP relaxation

and analyze which variables are close to an integer value. Then, they round these fractional solutions to the nearby value, and solve the LP relaxation again. This is done iteratively in the hope that all variables fall in line and the feasible solution has a better objective value than the current incumbent.

3.3.4 Parallelism

The last big contributor to the speed-up of searching the search tree is parallelism. The concept is easy. Instead of processing one node at a time the nodes are processed in parallel. Next to these above mentioned techniques Gurobi's MIP solver has more techniques. On their website they mention; sophisticated branch variable selection, node presolve, symmetry detection and disjoint subtree detection. The goal in most cases is to limit the size of the branch-and-bound tree that must be explored.

3.4 Introduction to Benders' Decomposition

With Benders' decomposition (Benders, 1962) it becomes possible to obtain solutions for large problems that can be split into a master problem(MP) and subproblems(SP). The MP only contains a subset of the original variables and a subset of the constraints. A schematic representation of Benders' Decomposition is given in Figure 6. The MP and SP are in a continuous feedback loop. The MP is solved to optimality, resulting in a solution (and hence values for the subset of variables). The solution of the MP is used in the SP, then the SP are solved. If there is a constraint that is not added to the MP, but violated by the current solution, then this constraint is added to the MP and the MP is solved again. In each iteration three scenarios can happen.

The first scenario is that a subproblem is unbounded. A problem is unbounded when the solution is feasible, but the objective function can be improved arbitrarily. In that case a feasibility constraint is added to the MP. This feasibility constraint ensures that when the MP is solved again, it is not possible to get the solution that resulted in an unbounded subproblem.

The second scenario is that a subproblem is bounded, but the optimal solution is better than the optimal solution of the MP. This cannot be a feasible solution, since the MP is a relaxation of the full problem (remember that only a subset of

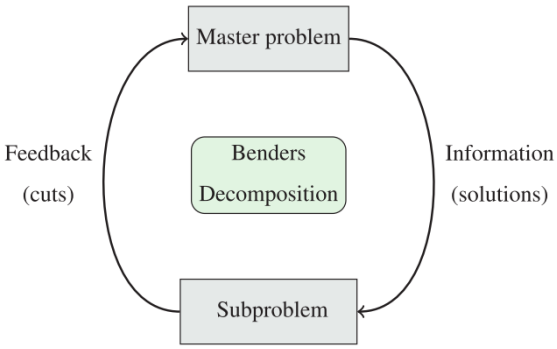

Figure 6: From Rahmaniani et al. (2017): A schematic representation of Benders' Decomposition

constraints are used), hence the solution to the MP is a lower bound(LB) to the optimal solution. If this is the case, then an optimality cut is added to the MP and the problem is solved again.

This procedure of adding constraints is repeated until no new constraints have to be added to the MP. When no new constraints have to be added, the solution to the MP is feasible, unbounded, and not lower than the LB.

4 Methodology

In this section I discuss the methods that are proposed in Aghaei et al. (2021). Due to the strong Linear Optimization relaxation and the high speed-up compared to BinOCT (Verwer and Zhang, 2019) and OCT (Bertsimas and Dunn, 2017), the framework of Aghaei et al. (2021) seems to be the most applicable for larger datasets. I compare this method to CART, OCT and BinOCT, but I don't discuss these methods here. The formulation used for OCT and BinOCT can be found in Aghaei et al. (2021, EC.2.) and Verwer and Zhang (2019) respectively.

4.1 Notation of Decision Trees and Acyclic Flow Graphs

For every decision tree a maximum number of consecutive splits is chosen *a priori*, this is the depth, d, of the tree. In a decision tree there is a distinction between *branch* and *leave* nodes. On the branching nodes, the dataset is split on a certain feature, a leave node is the final node of the tree. To each leave, a label is assigned depending on the majority class in the leaf.

Another distinction that can be made is between balanced and imbalanced trees. In a balanced decision tree, the number of splits from the *root node* to the *leave node* is the same for every possible decision path. In an imbalanced tree the number of consecutive splits before reaching a leaf node is not the same for every decision path. This is a *pruned* tree, through *pruning* the the chance of overfitting is reduced.

In a balanced tree, every non-leaf node has exactly two children nodes. The nodes are numbered in the order they appear in a breadth-first search, the bottom right node has number $2^{d+1}-1$. The set of branching nodes, $\mathcal{B}$, are the first 2^d-1 nodes. The remaining 2^d nodes are the set of leave nodes, $\mathcal{L} = \{2^d, 2^d+1, ..., 2^{d+1}-1\}$. The left side of Figure 7 shows an example of a balanced decision tree of depth two.

Bertsimas and Dunn (2017) and Verwer and Zhang (2019) use a Mixed Integer Program (MIP) to model a tree as seen in the left side of Figure 7. Aghaei et al. (2021) slightly changes this tree, to obtain a *stronger* formulation. They add a source node, s, and a sink node, t, as can be seen in the right side of Figure 7. Adding these two nodes makes it possible to model the decision tree as an acyclic flow graph. All the datapoints flow into the graph from source node s. If

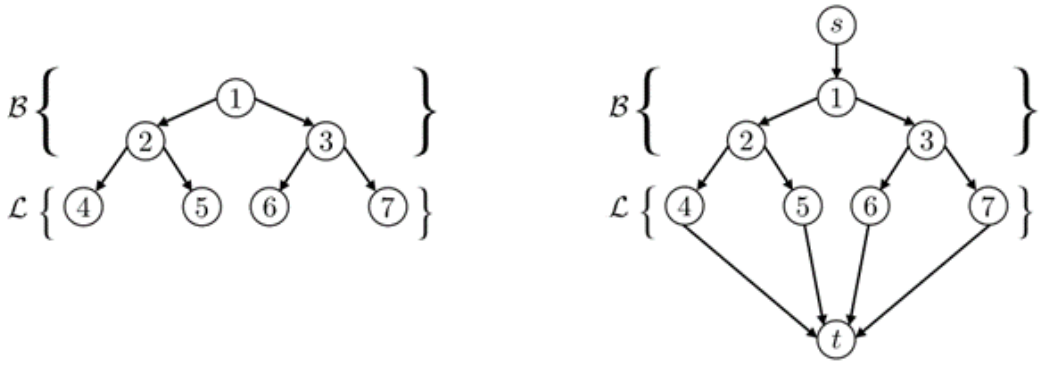

Figure 7: From Aghaei et al. (2021): A balanced decision tree of depth 2(left) and the associated flow graph(right). Here, $\mathcal{V} = \{s, 1, 2, ..., 6, 7, t\}, \mathcal{B} = \{1, 2, 3\}, \mathcal{L} = \{4, 5, 6, 7\}$ and $\mathcal{A} = \{(s, 1), (1, 2), ..., (6, t), (7, t)\}$

an observation is correctly predicted, it flows through the whole graph, from the source node to the sink node. However, if an observation is not predicted correctly then the observation stops in a leave node, and does not flow through to the sink node. In the optimal tree, the flow from the source node to the sink node is the highest. In the MIP of Aghaei et al. (2021) the flow through the tree is optimized.

In this paragraph I give the formal definition of a *flow* graph corresponding to a balanced decision tree of depth d. The vertices, $\mathcal{V}$, of the flow graph are $\mathcal{V} := \{s, t\} \cup \mathcal{B} \cup \mathcal{L}$. The arcs, $\mathcal{A}$, are

$$\mathcal{A} := \{(n, l(n)) : n \in \mathcal{B}\} \cup \{(n, r(n)) : n \in \mathcal{B}\} \cup \{(s, 1)\} \cup \{(n, t) : n \in \mathcal{L}\}.$$

Where $l(n)$, the left child node of node n, is mathematically defined as $l(n) := 2n$. For $r(n)$, the right child node of node n, $r(n) := 2n + 1$. The ancestor of node $n \in \mathcal{B} \cup \mathcal{L}$ is $a(n) := \lfloor \frac{n}{2} \rfloor$ when $n \neq 1$ and $a(1) = s$.

The notation for an imbalanced trees is almost the same as for a balanced tree. There is a set of branching nodes, $\mathcal{B} := \{1, ..., 2^d - 1\}$ and a set of terminal nodes, $\mathcal{T} = \{2^d, ..., 2^{d+1} - 1\}$. Nodes $\{2^d, ..., 2^{d+1} - 1\}$ are not called leave nodes, as in balanced trees. In an imbalanced tree node $n \in \mathcal{B} \cup \mathcal{T}$ is only called a leaf node when no branching occurs in this node.

In the formulation for balanced tree there is no flow possible from node $n \in \mathcal{B}$ to t. However, for an imbalanced tree this is possible. So the set of arcs, $\mathcal{A}$, is supplemented with arcs that go from any branching node to the sink node, t. This

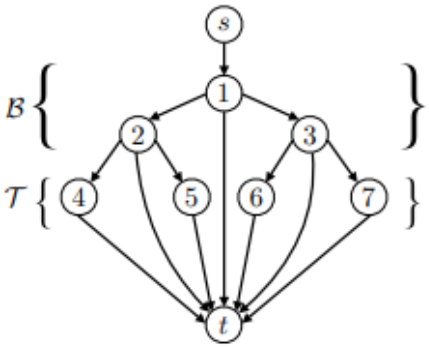

Figure 8: From Aghaei et al. (2021): The flow graph to model an imbalanced decision tree of maximum depth 2(right). Here, $\mathcal{V} = \{s, 1, 2, ..., 6, 7, t\}, \mathcal{B} = \{1, 2, 3\}, \mathcal{T} = \{4, 5, 6, 7\}$ and $\mathcal{A} = \{(s, 1), (1, 2), (1, t), ..., (6, t), (7, t)\}$

results in the set of arcs

$$\mathcal{A} := \{(n, l(n)) : n \in \mathcal{B}\} \cup \{(n, r(n)) : n \in \mathcal{B}\} \cup \{(s, 1)\} \cup \{(n, t) : n \in \mathcal{T} \cup \mathcal{B}\}.$$

An example of this can be found in Figure 8.

4.2 MIO Flow Formulation for Balanced Trees

In this section I discuss the strong formulation of Aghaei et al. (2021) to optimize a decision tree. Besides the LP-relaxation of their formulation being stronger than other formulations, it also perfectly lends itself for Benders' Decomposition (which is discussed in Section 3.4). The strong formulation and Benders' Decomposition are two advantages that reduce the running time significantly. To model the acyclic flow graph as a binary decision tree, three set of variables are used: w, b and z. The variable w is used to indicate the label of the leaf nodes. The variable b indicates the features that are used to branch. Variable z dictates the flow through the graph. In the next paragraph I discuss these three types of decision variables more elaborately.

The variable $b_{nf}, n \in \mathcal{B}$ and $f \in \mathcal{F}$, is used to indicate whether feature f is used to branch on node n. This variable is one, if and only if feature f is used to branch at node n.

The binary variable $w_k^n \in \{0, 1\}, k \in \mathcal{K}$ and $n \in \mathcal{L}$, is used to indicate what the

label of leaf node n is. For example, if the majority of datapoints in the first leaf node belong to the second class, then $w_2^1 = 1$.

The z variables are routing variables, dictating whether an observation passes through a certain arc. The variable $z^i_{a(n),n} \in \{0,1\}$, for $n \in \mathcal{B} \cup \mathcal{L}$ and $i \in \mathcal{I}$, is one if and only if observation i is correctly classified and passes through arc $(a(n), n)$(which is the arc between node n and it's ancestor) and zero otherwise. The variable that dictates the flow from the leaf nodes to the sink node is $z^i_{n,t} \in \{0,1\}$, for $n \in \mathcal{L}$ and $i \in \mathcal{I}$. The variable $z^i_{n,t}$ is one, if and only if datapoint i traverses leaf node n and the datapoint is correctly classified, if the datapoint is correctly classified it holds that $w^n_{y^i} = 1$.

A decision tree of depth d, fitted on the dataset $\mathcal{D} := \{\mathbf{x^i}, y^i\}_{i\in\mathcal{I}}$, with $|\mathcal{I}|$ observations, $\mathbf{x^i}$ consisting of F binary features, $\mathbf{x^i} \in \{0,1\}^F$, and $\mathcal{K}$ different classes where y^i is the class that observation i belongs to can be modelled as follows:

$$\begin{aligned}
\text{max} \quad & \sum_{i\in\mathcal{I}} \sum_{n\in\mathcal{L}} z^i_{n,t} && && \text{(3a)}\\
\text{subject to} \quad & \sum_{f\in\mathcal{F}} b_{nf} = 1 && \forall n \in \mathcal{B}, && \text{(3b)}\\
& z^i_{a(n),n} = z^i_{n,l(n)} + z^i_{n,r(n)} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(3c)}\\
& z^i_{a(n),n} = z^i_{n,t} && \forall n \in \mathcal{L}, i \in \mathcal{I} && \text{(3d)}\\
& z^i_{s,1} \leq 1 && \forall i \in \mathcal{I} && \text{(3e)}\\
& z^i_{n,l(n)} \leq \sum_{f\in\mathcal{F}: x^i_f = 0} b_{nf} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(3f)}\\
& z^i_{n,r(n)} \leq \sum_{f\in\mathcal{F}: x^i_f = 1} b_{nf} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(3g)}\\
& z^i_{n,t} \leq w^n_{y^i} && \forall n \in \mathcal{L}, i \in \mathcal{I} && \text{(3h)}\\
& \sum_{k\in\mathcal{K}} w^n_k = 1 && \forall n \in \mathcal{L} && \text{(3i)}\\
& w^n_k \in \{0,1\} && \forall n \in \mathcal{L}, k \in \mathcal{K} && \text{(3j)}\\
& b_{nf} \in \{0,1\} && \forall n \in \mathcal{B}, f \in \mathcal{F} && \text{(3k)}\\
& z^i_{a(n),n} \in \{0,1\} && \forall n \in \mathcal{B} \cup \mathcal{L}, i \in \mathcal{I} && \text{(3l)}\\
& z^i_{n,t} \in \{0,1\} && \forall n \in \mathcal{L}, i \in \mathcal{I}. && \text{(3m)}
\end{aligned}$$

The objective (3a) optimizes the flow through the graph, or in other words the total number of correctly classified observations. The first constraint (3b) ensures that at every branching node exactly one feature is used to branch on. The second constraint (3c) is a flow conservation constraint. Every datapoint i that enters branching node n needs to leave that node either to the left or to the right. In a similar fashion, the third constraint (3d) makes sure that every datapoint i that enters leaf node n goes to the sink node. The fourth constraint (3e) limits the units of flow that can enter the graph, a maximum of one flow unit per observation can enter the graph. The fifth (3f) and sixth (3g) constraints, ensure that an observation goes left when $x_f^i = 0$ and right when $x_f^i = 0$, with f being the feature used to branch on at node n. Because of the flow conservation constraint (3c), an observation that arrives at node n, has to leave it to either the left or the right. The seventh constraint (3h), only allows correctly classified observations to flow to the sink node. Finally, constraint 3i ensures that each leaf node is assigned to one predicted class $k \in \mathcal{K}$.

Even though the formulation of Aghaei et al. (2021) uses binary features, it can be applied to non-binary features as well. There are two kind of features, quantitative and categorical features. Quantitative features can be discrete (i.e. the feature can only take certain values) and continuous (i.e. the feature can take any value). Categorical features can be categorized. These categories can be ordinal (the order in the categories matter) or nominal (the order in the categories doesn't matter). All these different kind of variables can be preprocessed and changed into binary variables.

Categorical nominal variables can be changed into binary variables using one-hot encoding. For every level of the feature, a binary column of size $1 \times |\mathcal{I}|$ is created. Entry i in this binary column is one, if and only if the categorical value of this observation corresponds to the level of the binary column. For categorical ordinal variables and quantitative features you can follow a similar approach. For these kind of variables, the binary column has value one, if and only if the main column has the corresponding value, or any value smaller than it.

4.3 MIO Flow Formulation for Imbalanced Trees

Formulation 3 models balanced trees, meaning that the number of splits between the root node to each leaf node is the same length. However, this might result in overfitting and hence it is desirable to prune the branches. When the branches are pruned the tree is imbalanced, meaning that the path from the root node to the leaf node is not always of the same length. In CART pruning is done after the tree has been created, with OCT pruning is incorporated in the MIP. In this section I extend Formulation 3 to imbalanced trees.

Next to the decision variables that are used for formulation 3, the binary variable p_n is introduced for every node $n \in \mathcal{B} \cup \mathcal{T}$. This variable is one, if and only if we make a prediction at node n. Furthermore, $\mathcal{A}(n)$ and λ are introduced. $\mathcal{A}(n)$ is the set of all the ancestor nodes of node n. The parameter $\lambda \in [0, 1]$ is a regularization parameter. The formulation for an imbalanced decision tree is as follows

$$\max \quad (1-\lambda)\sum_{i\in\mathcal{I}}\sum_{n\in\mathcal{T}\cup\mathcal{B}} z^i_{n,t} - \lambda\sum_{n\in\mathcal{B}}\sum_{f\in\mathcal{F}} b_{nf} \tag{4a}$$

$$\text{subject to} \quad \sum_{f\in\mathcal{F}} b_{nf} + p_n + \sum_{m\in\mathcal{A}(n)} p_m = 1 \qquad \forall n\in\mathcal{B}, \tag{4b}$$

$$p_n + \sum_{m\in\mathcal{A}(n)} p_m = 1 \qquad \forall n\in\mathcal{T} \tag{4c}$$

$$z^i_{a(n),n} = z^i_{n,l(n)} + z^i_{n,r(n)} + z^i_{n,t} \qquad \forall n\in\mathcal{B}, i\in\mathcal{I} \tag{4d}$$

$$z^i_{a(n),n} = z^i_{n,t} \qquad \forall n\in\mathcal{T}, i\in\mathcal{I} \tag{4e}$$

$$z^i_{s,1} \leq 1 \qquad \forall i\in\mathcal{I} \tag{4f}$$

$$z^i_{n,l(n)} \leq \sum_{f\in\mathcal{F}:x^i_f=0} b_{nf} \qquad \forall n\in\mathcal{B}, i\in\mathcal{I} \tag{4g}$$

$$z^i_{n,r(n)} \leq \sum_{f\in\mathcal{F}:x^i_f=1} b_{nf} \qquad \forall n\in\mathcal{B}, i\in\mathcal{I} \tag{4h}$$

$$z^i_{n,t} \leq w^n_{y^i} \qquad \forall n\in\mathcal{T}\cup\mathcal{B}, i\in\mathcal{I} \tag{4i}$$

$$\sum_{k\in\mathcal{K}} w^n_k = p_n \qquad \forall n\in\mathcal{B}\cup\mathcal{T} \tag{4j}$$

$$w^n_k \in \{0,1\} \qquad \forall n\in\mathcal{T}, k\in\mathcal{K} \tag{4k}$$

$$b_{nf} \in \{0,1\} \qquad \forall n\in\mathcal{B}, f\in\mathcal{F} \tag{4l}$$

$$z^i_{a(n),n} \in \{0,1\} \qquad \forall n\in\mathcal{B}\cup\mathcal{T}, i\in\mathcal{I} \tag{4m}$$

$$z^i_{n,t} \in \{0,1\} \qquad \forall n\in\mathcal{T}, i\in\mathcal{I} \tag{4n}$$

$$p_n \in \{0,1\} \qquad \forall n\in\mathcal{B}\cup\mathcal{T}. \tag{4o}$$

The objective function (4a) consists of two terms. The first term optimizes the flow through the tree. Contrary to formulation 3 the summation is over all the nodes, instead of just the set of leave nodes. The second term is a regularization term, where the number of total branching nodes is counted. The second term is subtracted from the first term, hence, more splits results in a lower objective function. The regularization parameter, λ, determines the weight put on the regularization term. With higher values of λ, the second term in the objective function is bigger. Hence, for bigger values of λ, less splits are made.

The first modified constraint is constraint 4b, this constraint implies that node $n \in \mathcal{B}$ is either a leaf node ($p_n = 1$), a branching node ($\sum_{f\in\mathcal{F}} b_{nf} = 1$), or a pruned

node (a prediction is made at one of the ancestor nodes, $\sum_{m \in \mathcal{A}(n)} p_m = 1$).

Constraint 4c forces node $n \in \mathcal{T}$ to be either a leaf node, or one of the nodes' ancestors is a leaf node. In formulation 3, the flow conservation constraint (3c) ensured that an observation flows either left or right. In the current model, observations can also flow to the sink node. Hence, in this formulation, the flow conservation constraint (4d) is extended with the term $z^i_{n,t}$, which is one when observation i flows from node n to the sink node t and zero otherwise.

Constraint 4e to 4j are the same as in the first formulation. The last modification is in constraint 4k. In formulation 3 this constraint equals $\sum_{k \in \mathcal{K}} w^n_k = 1$. While, in the current formulation this constraint is $\sum_{k \in \mathcal{K}} w^n_k = p_n$. This constraint ensures that a class can only be assigned to a node when a prediction is made at this node.

4.4 Reformulating the Flow Formulation

In this section formulation 4 is reformulated in such a way that Benders' Decomposition can be applied. Formulation 4 can be split in two MIP's. In one problem, the shape of the tree is determined, while in the other, the optimal flow path for each observation is determined.

The shape of the tree is dependent upon three variables. 1) The variable that models the splits on each node, b, 2) the variable that determines whether a prediction is made on a node, p, and 3) the variable that determines the label assigned to a leave node, w. This results in the following decomposition:

$$\text{max} \quad (1-\lambda) \sum_{i \in \mathcal{I}} g(b, w, p)^i - \lambda \sum_{n \in \mathcal{B}} \sum_{f \in \mathcal{F}} b_{nf} \tag{5a}$$

$$\text{subject to} \quad \sum_{f \in \mathcal{F}} b_{nf} + p_n + \sum_{m \in \mathcal{A}(n)} p_m = 1 \qquad \forall n \in \mathcal{B}, \tag{5b}$$

$$p_n + \sum_{m \in \mathcal{A}(n)} p_m = 1 \qquad \forall n \in \mathcal{T} \tag{5c}$$

$$\sum_{k \in \mathcal{K}} w^n_k = p_n \qquad \forall n \in \mathcal{T} \cup \mathcal{B} \tag{5d}$$

$$w^n_k \in \{0, 1\} \qquad \forall n \in \mathcal{T} \cup \mathcal{B}, k \in \mathcal{K} \tag{5e}$$

$$b_{nf} \in \{0, 1\} \qquad \forall n \in \mathcal{B}, f \in \mathcal{F} \tag{5f}$$

$$p_n \in \{0, 1\} \qquad \forall n \in \mathcal{B} \cup \mathcal{T}. \tag{5g}$$

Where $g^i(b, w, p)$ is the optimal value of the problem:

$$
\begin{aligned}
g^i(b,w,p) = \max \quad & \sum_{n \in \mathcal{B} \cup \mathcal{T}} z^i_{n,t} && && \text{(6a)}\\
\text{subject to} \quad & z^i_{a(n),n} = z^i_{n,l(n)} + z^i_{n,r(n)} + z^i_{n,t} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(6b)}\\
& z^i_{a(n),n} = z^i_{n,t} && \forall n \in \mathcal{T}, i \in \mathcal{I} && \text{(6c)}\\
& z^i_{s,1} \leq 1 && \forall i \in \mathcal{I} && \text{(6d)}\\
& z^i_{n,l(n)} \leq \sum_{f \in \mathcal{F}: x^i_f = 0} b_{nf} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(6e)}\\
& z^i_{n,r(n)} \leq \sum_{f \in \mathcal{F}: x^i_f = 1} b_{nf} && \forall n \in \mathcal{B}, i \in \mathcal{I} && \text{(6f)}\\
& z^i_{n,t} \leq w^n_{y^i} && \forall n \in \mathcal{B} \cup \mathcal{T}, i \in \mathcal{I} && \text{(6g)}\\
& z^i_{a(n),n} \in \{0,1\} && \forall n \in \mathcal{B} \cup \mathcal{T}, i \in \mathcal{I} && \text{(6h)}\\
& z^i_{n,t} \in \{0,1\} && \forall n \in \mathcal{B} \cup \mathcal{T}, i \in \mathcal{I}. && \text{(6i)}
\end{aligned}
$$

By substituting the objective function of the second problem (6a) in the objective function of the first problem (5a) we arrive at formulation 4 again.

4.4.1 Capacitated Graphs

In formulation 6, the tree is fixed (b, w and p are given), the objective is to maximize the flow of node $n \in \mathcal{B} \cup \mathcal{T}$ to the sink node t ($\max \sum_{n \in \mathcal{B} \cup \mathcal{T}} z^i_{n,t}$). This is the same as optimizing a maximum flow problem on a capacitated graph, i.e. a graph where each arc has a capacity.

In the next paragraph I give the formal mathematical definition of a capacitated graph. For a given tree, defined by the variables b, w and p, the flow graph is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. An example of a flow graph can be found on the left side of Figure 9. The capacitated flow graph for observation i is the flow graph

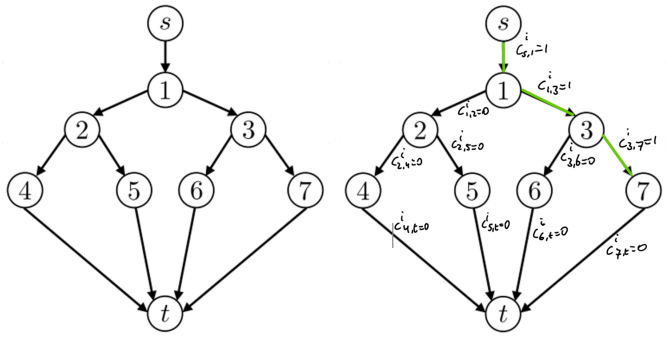

Figure 9: On the left side the flow graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. $\mathcal{V} = \{s, 1, 2, ..., 6, 7, t\}$ and $\mathcal{A} = \{(s,1), (1,2), (1,t), ..., (6,t), (7,t)\}$. On the right side the capacitated flow graph for observation i, $\mathcal{G}^i = (\mathcal{V}, \mathcal{A})$. With the capacities $c^i_{a(n),n}$. The capacity is one when an observation flows over the arc and zero otherwise.

augmented by binary arc-capacities, $c^i_{a(n),n}$. The capacities are defined as follows:

$$
\begin{aligned}
c^i_{s,1} &:= 1, \\
c^i_{n,l(n)}(b,w) &:= \sum_{f \in \mathcal{F}: x^i_f = 0} b_{nf}, \\
c^i_{n,r(n)}(b,w) &:= \sum_{f \in \mathcal{F}: x^i_f = 1} b_{nf}, \\
c^i_{n,t}(b,w) &:= w^n_{y^i} \qquad\qquad \forall n \in \mathcal{B} \cup \mathcal{T}.
\end{aligned}
$$

4.4.2 Max-Flow Min-Cut Theorem

The objective in each formulation for OCT is to correctly classify as many observations as possible. We have seen that for formulation 6 this is the same as optimizing a maximum flow problem on a capacitated graph. To solve this we can apply the max-flow min-cut theorem.

The max-flow min-cut theorem, states that the maximum flow passing through a graph is equal to the total weight of edges in a minimum cut. A cut divides the nodes in two sets, in such a way that the nodes s and t are not in the same set. This means that the weight of a maximum flow cannot be larger than the weight of the edges in a minimum cut.

An example of this theory can be seen in Figure 10. On the left, we see the

capacitated graph for observation i, with three possible minimal cuts. The first cut creates two sets: the source set, $\mathcal{V}_1 = \{s, 1, 3, 7\}$, and the sink set, $\mathcal{V}_2 = \{t, 2, 4, 5, 6\}$. This cut goes through the following arcs: $\mathcal{A} = \{(1, 2), (3, 5), (7, t)\}$. Since the capacity on each of these arcs is zero, the value of this cut is zero. The maximum flow through the graph is also zero, as there is no connected path between source node s and sink node t. On the right side of Figure 10 we see another capacitated flow graph. The observation in this flowgraph is correctly predicted, and hence the maximum flow is one. The minimum cut that can be made in this graph is also zero.

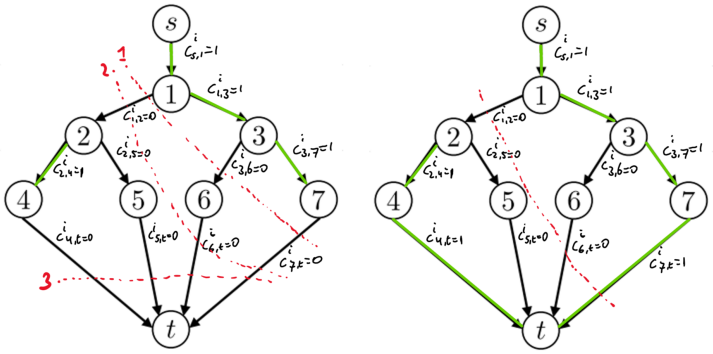

Figure 10: On the left side the capacitated flow graph for a non-correctly classified observation, and three possible cuts. On the right side the capacitated flow graph for a correctly classified observation and one minimum cut.

Due to the max-flow/min-cut duality, it follows that $g^i(b, w, p)$ (the objective of formulation 6), is the minimum *(s,t)* cut of $\mathcal{G}^i(b, w)$. So $g^i(b, w, p)$, is the greatest value that is smaller than or equal to the value of all cuts in graph $\mathcal{G}^i(b, w)$. To put this into a constraint we define the cut-set. The cut-set is defined as follows: given $\mathcal{S} \subseteq \mathcal{V}$, the cut-set is $C(\mathcal{S}) := \{(n_1, n_2) \in \mathcal{A} : n_1 \in \mathcal{S}, n_2 \notin \mathcal{S})\}$ where $\mathcal{S}$ is the set of all vertices that are on the source side of the cut.

4.4.3 Flow Formulation With the Max-Flow Min-Cut Theorem

Incorporating the max-flow min-cut constraint results in the following formulation:

$$\begin{aligned}
\text{max} \quad & (1-\lambda)\sum_{i\in\mathcal{I}} g^i - \lambda \sum_{n\in\mathcal{B}}\sum_{f\in\mathcal{F}} b_{nf} && && \text{(8a)}\\
\text{subject to} \quad & g^i \leq \sum_{(n_1,n_2)\in C(\mathcal{S})} c^i_{n_1,n_2}(b,w) && \forall i\in\mathcal{I}, \mathcal{S}\subseteq\mathcal{V}\backslash\{t\} : s\in\mathcal{S} && \text{(8b)}\\
& \sum_{f\in\mathcal{F}} b_{nf} + p_n + \sum_{m\in\mathcal{A}(n)} p_m = 1 && \forall n\in\mathcal{B}, && \text{(8c)}\\
& p_n + \sum_{m\in\mathcal{A}(n)} p_m = 1 && \forall n\in\mathcal{T}, && \text{(8d)}\\
& \sum_{k\in\mathcal{K}} w^n_k = p_n && \forall n\in\mathcal{B}\cup\mathcal{T} && \text{(8e)}\\
& w^n_k \in \{0,1\} && \forall n\in\mathcal{L}, k\in\mathcal{K} && \text{(8f)}\\
& b_{nf} \in \{0,1\} && \forall n\in\mathcal{B}, f\in\mathcal{F} && \text{(8g)}\\
& p_n \in \{0,1\} && \forall n\in\mathcal{B}\cup\mathcal{T} && \text{(8h)}\\
& g^i \leq 1 && \forall i\in\mathcal{I}. && \text{(8i)}
\end{aligned}$$

The objective of this reformulation is to maximize the flow through the graph, while making the least amount of splits. Maximizing the flow through the graph is the equivalent of maximizing the number of correctly classified datapoints as the value of the minimum cut, g^i, is either one (a correctly classified datapoint) or zero (an incorrectly classified datapoint). The value of g^i cannot be larger than the sum of edges in a minimum cut, this is ensured by constraint 8b. This constraint enforces the minimum cut for all observations and all possible cut sets. This results in an exponential number of constraints. Constraint 8c to constraint 8e are similar to constraints in previous formulations.

4.5 Benders' Decomposition on the Flow Formulation

Flow formulation 8 is stronger than already existing formulations, as proved by Aghaei et al. (2021). Next to the strong formulation an added benefit is the decomposable structure. Decomposing the formulation into a master- and a subproblem

speeds-up the formulation. Due to constraint 8b the above formulation consists of an exponential number of inequalities.

The first constraint (8b) is initially dropped, each dropped constraint is the sub-problem(SP) in Benders' Decomposition. The master problem(MP) can be solved relatively quickly. However, it is likely that for a corresponding solution the max-flow min-cut constraint is not satisfied for some datapoint i and some cut *(s,t)*. The violated SP-constraints are added, then the MP is solved again, until none of the SP-constraints (8b) are violated.

The constraints that are violated and need to be added can be found by solving the minimum cut problem. Well-known general algorithms to solve the minimum cut problem are: Goldberg and Tarjan (1988), Karger (2000) and Hochbaum (2008). However, Aghaei et al. propose an algorithm that creates a minimum cut set that is tailor-made for this application. The algorithm they propose creates *stronger*, i.e. tighter constraints.

The constraints that are proposed by the algorithm are known as lazy constraints. Normally constraints are defined upfront, however, lazy constraints are added during optimizing the MIP. At every node of the branch-and-bound tree Algorithm 1 is called, using the callback function. If a max-flow min-cut constraint is violated then the violated constraint is added to the formulation. In this way it is not necessary to add an exponential number of constraints upfront, only the constraints that are needed to create a feasible optimal solution are needed.

Algorithm 1 either returns -1 if all constraints for observation i corresponding to 8b are satisfied, or the source set $\mathcal{S}$ of the minimal cut otherwise. The first time the algorithm returns -1 is on line 2. In line 2 the flow capacity, g^i, is 0, thus the min-cut constraints will always be satisfied.From line 4 to 14 the source set $\mathcal{S}$ is defined. This is done by recursively adding the node n to which observation i flows. The way an observation flows is determined by the if/else-if statements on lines 8 and 10. On line 14 we ended up on the leaf node that observation i flows to. This leaf node is added the the source set. If the observation is correctly classified then $g^i \leq c^i_{n,t}(b, w)$, in that case the min-flow max-cut constraints (8b) are satisfied and the algorithm returns -1 (line 18). When $g^i = 1$, but the observation is missclassified we have more flow than the max-cut allows. Hence, the min-flow max-cut constraints need to be added to the MIP and the source set $\mathcal{S}$ is returned.

Algorithm 1 Cut Generation Procedure

Input: $(b,\ w,\ g) \in \{0,1\}^{\mathcal{B}\times\mathcal{F}} \times \{0,1\}^{\mathcal{L}\times\mathcal{K}} \times R^{\mathcal{I}}$ satisfying 8c - 8i
$i \in \mathcal{I}$: datapoint used to generate the cut.

Output: -1 if all constraints 8b corresponding to i are satisfied;
source set $\mathcal{S}$ of min-cut otherwise

if $g^i = 0$ **then**
    return -1
end if
Initialize: $n \leftarrow 1$ ▷ Current node = root
Initialize: $\mathcal{S} \leftarrow \{s\}$ ▷ $\mathcal{S}$ is in the source set of the cut
while $n \in \mathcal{B}$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{n\}$
    if $c_{n,l(n)}(b,w) = 1$ **then** ▷ Datapoint i is routed left
        $n \leftarrow l(n)$
    else if $c_{n,r(n)}(b,w) = 1$ **then** ▷ Datapoint i is routed right
        $n \leftarrow r(n)$
    end if
end while ▷ At this point, $n \in \mathcal{L}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{n\}$
if $g^i > c^i_{n,t}(b,w)$ **then** ▷ Minimum cut of $\mathcal{S}$ with capacity 0 found
    return $\mathcal{S}$
else ▷ Minimum cut of $\mathcal{S}$ has capacity 1, constraints 8b satisfied
    return -1
end if

5 Results

To answer the research questions I apply the methods on datasets of varying sizes and varying tree depths. In the first part of the results section I apply the methods (CART, OCT, BinOCT, FlowOCT and BendersOCT) on synthetic datasets. Table 2 shows the different synthetic datasets. In the second part of the result section the methods are applied on six publicly available datasets from the UCI data repository(Dua and Graff, 2017), the datasets and their characteristics can be found in Table 1.

Table 1: UCI repository datasets including the number of rows ($|\mathcal{I}|$), number of features($|\mathcal{F}|$), encoded features ($|\mathcal{F}_{enc}|$) and the number of classes (($|\mathcal{K}|$)

Name	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{F}_{enc}\rvert$	$\lvert\mathcal{K}\rvert$
kr-vs-kp	3196	37	39	2
car-evaluation	1728	7	20	4
balance-scale	625	5	21	3
breast-cancer	277	10	39	2
monk-1	124	7	16	2
soybean-small	47	36	46	4

Table 2: Synthetic datasets including the number of rows ($|\mathcal{I}|$), number of features($|\mathcal{F}|$), encoded features ($|\mathcal{F}_{enc}|$) and the number of classes (($|\mathcal{K}|$)

$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{F}_{enc}\rvert$	$\lvert\mathcal{K}\rvert$
100	10	18	2
100	30	50	2
100	100	168	2
500	10	18	2
500	30	50	2
500	100	168	2
2500	10	18	2
2500	30	50	2
2500	100	168	2

5.1 Setup

All the approaches are implemented in Python and solved using Gurobi 9.5 (Gurobi Optimization, LLC, 2022). The problems are solved on a single core of an Intel i5 processor running at 2.6GHz using 8GB of RAM, running Windows 10 version 21H1 with a 10 minutes time limit for each problem. My code is available on: `https://github.com/holodorum/StrongTreesThesis`.

For each dataset, I create 4 random splits of the data, each consisting of a

training set of 75%, and a test set of 25%. My code allows for calibrating and finetuning the parameters as well. However, the emphasis in my research is not on getting the highest accuracy, but decreasing the running times for larger datasets. Finetuning the parameters would be infeasible time-wise. Hence, I set λ to zero and use each OCT-method to fit trees of depths 2 to 5 on the synthetic datasets and datasets from the UCI data repository.

To run CART on the various datasets I use the *Decision Tree Classifier* class from the Python scikit-learn library(Pedregosa et al., 2011). For CART the regularization parameter is also set to zero, in order to have a similar degree of pruning as the optimal tree methods. The maximum depth of the trees range from 2 to 5 and the trees are trained on the same trainingset as is used for the optimal tree methods.

In the remainder of this section I first discuss the in-sample performance, computation times and the ground truth discovery rate of the MIO methods on the synthetic datasets. Then I discuss the in-sample performance and computation times of the real datasets.

5.2 Synthetic Dataset

The synthetic datasets are created in the same way as the datasets in Murthy and Salzberg (1995). For each dataset I first set the number of observations, n, and the number of features. A percentage of the features is binary the other percentage is categorical (with three categories). The distribution of the binary and categorical features is as follows:

$$\begin{aligned} \mathbf{x}^{bin} &\sim Ber(\theta_{bin}, n), \quad \theta_{bin} \sim Unif(0,1) \\ \mathbf{x}^{cat} &\sim Cat(n, C, \theta_{cat}), \quad C = [0,1,2], \theta_{cat} = [0.5, 0.3, 0.2]. \end{aligned}$$

Feature $\mathbf{x}^{bin}$ is bernoulli distributed, with a uniform distribution of the θ-parameter and with n trials. The categorical variable follows a multinomial distribution with three categories. The probability that category 1, 2 or 3 is drawn for observation i is respectively 0.5, 0.3 and 0.2. The categorical variables are one-hot encoded. This results in independent variable $\mathbf{X}$. A temporary bernoulli distributed dependent variable, $\mathbf{y}_{temp}$, is created. The variable is temporary, because after estimating the decision tree it is discarded. A CART decision tree with a pre-specified depth is

fit on the synthetic dataset with independent variables $\mathbf{X}$ and dependent variable $\mathbf{y}_{temp}$.

To create dependent variable y the leaves of the fitted decision tree are labeled in such a way, that no two leaves sharing a parent have the same label. If the leave nodes would have the same label, then they could be replaced by the parent node bearing that label. Theoretically speaking we could use any number of classes, but for the sake of simplicity I set the dependent variable to be binary.

The fitted tree is used to generate the dependent variable. The branching features used in the tree are called the ground truth features, which are fundamental in constructing the dependent variable. The fitted tree is the underlying structure of the synthetic dataset, or the data-generating process(DGP).

When fitting a tree on the synthetic dataset, it is possible to compute how many of the branching features that are used in the estimated decision tree are ground truth features. If a used feature is also a ground truth feature, then it is a true discovery (TD). If, on the other hand, the feature is not part of the ground truth features, then it is a false discovery (FD). The true discovery rate(TDR) and the false discovery rate(FDR) are computed as follows:

$$TDR = \frac{\text{No. of TD}}{\text{No. groundset features}} \qquad FDR = \frac{\text{No. of FD}}{\text{No. features in fitted tree}}.$$

The TDR indicates to what extent the decision tree is able to recover the true DGP. However, a high TDR alone is not satisfactory. It might be the case that the fitted tree uses so many features that it by chance includes features from the ground truth set of features. To see whether this happens we compute the FDR. This measure indicates how many of the features in the fitted tree are not part of the set of ground truth features. A combination of a high TDR and a low FDR is desirable.

5.2.1 In-Sample Accuracy

Table 3 shows the in-sample accuracy of the different methods on synthetic datasets. Since the datasets are synthetically created without any noise the optimal tree methods should be able to construct trees that give an accuracy of 1. However, in multiple instances this is not the case, for these instances the maximum running time (see Table A1) is reached.

Table 3: In-Sample Accuracy on Synthetic Datasets

depth	nrow	approach features	BendersOCT	FlowOCT	binOCT	OCT	CART
2	75	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.97(0.01)
		30	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.86(0.01)
		100	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.91(0.02)
	375	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.87(0.01)
		30	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.92(0.01)
		100	**1.00(0.00)**	0.93(0.00)	**1.00(0.00)**	0.99(0.01)	0.93(0.01)
	1875	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.92(0.00)
		30	**1.00(0.00)**	0.97(0.03)	0.99(0.03)	0.96(0.03)	0.94(0.00)
		100	**0.95(0.06)**	0.90(0.00)	0.90(0.00)	0.90(0.00)	0.90(0.00)
3	75	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.94(0.01)
		30	**1.00(0.00)**	0.94(0.03)	0.94(0.04)	0.98(0.02)	0.88(0.03)
		100	0.93(0.02)	**0.97(0.04)**	0.96(0.03)	**0.97(0.04)**	0.87(0.08)
	375	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.98(0.03)	0.95(0.01)
		30	**1.00(0.00)**	0.94(0.01)	0.96(0.03)	0.98(0.03)	0.94(0.01)
		100	0.83(0.04)	**0.91(0.08)**	0.85(0.01)	0.85(0.00)	0.85(0.00)
	1875	10	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.96(0.00)	0.96(0.00)
		30	**0.90(0.00)**	**0.90(0.00)**	**0.90(0.00)**	**0.90(0.00)**	**0.90(0.00)**
		100	**0.81(0.00)**	**0.81(0.00)**	**0.81(0.00)**	**0.81(0.00)**	**0.81(0.00)**
4	75	10	**0.99(0.01)**	**0.99(0.01)**	0.97(0.02)	0.98(0.01)	0.93(0.02)
		30	**0.98(0.02)**	0.96(0.02)	0.94(0.03)	0.96(0.02)	0.88(0.04)
		100	0.99(0.02)	**1.00(0.01)**	1.00(0.00)	0.99(0.00)	0.97(0.02)
	375	10	**0.99(0.01)**	**0.99(0.01)**	**0.99(0.01)**	0.92(0.05)	0.95(0.00)
		30	0.91(0.03)	0.94(0.01)	0.94(0.00)	**0.95(0.00)**	0.94(0.01)
		100	0.84(0.01)	0.92(0.02)	0.92(0.02)	**0.93(nan)**	0.92(0.02)
	1875	10	0.87(0.09)	0.81(0.03)	**0.89(0.05)**	0.81(0.03)	0.81(0.03)
		30	**0.85(0.01)**	**0.85(0.01)**	**0.85(0.01)**	**0.85(0.01)**	**0.85(0.01)**
		100	0.70(0.11)	**0.90(0.02)**	**0.90(0.02)**	0.89(0.02)	**0.90(0.02)**
5	75	10	**1.00(0.00)**	0.99(0.01)	0.99(0.01)	0.98(0.01)	0.91(0.03)
		30	0.99(0.01)	1.00(0.00)	**1.00(0.01)**	0.98(0.01)	0.96(0.01)
		100	**1.00(0.01)**	1.00(0.00)	1.00(0.00)	0.94(0.05)	0.99(0.01)
	375	10	0.94(0.02)	**0.96(0.00)**	**0.96(0.00)**	0.88(0.05)	0.95(0.01)
		30	0.77(0.06)	0.87(0.03)	**0.88(0.03)**	0.78(0.02)	0.87(0.03)
		100	0.73(0.05)	**0.91(0.02)**	**0.91(0.02)**	0.62(0.01)	**0.91(0.02)**
	1875	10	0.93(0.01)	0.94(0.01)	**0.95(0.01)**	0.94(0.01)	0.93(0.01)
		30	0.89(0.00)	0.89(0.00)	0.90(0.01)	**0.90(nan)**	0.89(0.00)
		100	0.66(0.05)	**0.86(0.01)**	**0.86(0.01)**	**0.86(0.01)**	**0.86(0.01)**
Best	Performance	(with ties)	22	21	22	15	6
Best	Performance	(without ties)	6	2	2	3	0

Take for example BendersOCT. Until depth 3, 1875 observations and 10 features it constructs the optimal tree in each instance, except for depth 3, 75 observations and 100 features. Table A1 shows that BendersOCT reaches the maximum running time for depth 3, 75 observations and 100 features. Even though this is not very important for the research, it does provide an implementation sanity check. The methods appear to be properly implemented, since it finds the optimal tree except when the time-limit is reached.

Another interesting observation is that the methods always perform equally well or better than CART. In only two cases CART performs equally good as the optimal methods. The optimal approaches have a 4% higher mean accuracy than the benchmark. For many instances the MIP-approach have a higher accuracy than CART when the MIP is not solved to optimality. However, the performance increase in these scenarios is often only one or two percent. BendersOCT has the overall best performance in the most instances, although the differences between all the methods are small.

The standard deviation of the mean accuracy is low for each method. There are a few instances where the standard deviation is higher than 0.05, but these instances correspond to the maximum time-limit being reached. This is not suprising, since the trees at the moment that the time-limit is reached might differ greatly.

A low standard deviation could be an argument for low variability in the trees. It makes sense to assume that the trees are similar when the prediction accuracy is the same. However, as I show in section 5.3.1 this is not always the case.

A final note is that for some instances the standard deviation is not a number (nan). In these cases, the optimal tree method finds at most one solution to the four different training samples. The standard deviation in this case is non-existing, hence we observe nan values.

5.2.2 Discovery Rate

In Figure 11 the difference in the discovery rates(TDR) between an optimal tree method and CART are shown, Figure 12 shows the TDR for various methods. The table of which these figures are derived can be found in Table B2.

From Figure 11 it becomes clear that every method follows a similar pattern.

Figure 11: A heatmap of the difference in the TDR between CART and the optimal tree methods. A positive value means that the TDR of the optimal tree method is higher than that of CART.

For small trees the TDR of the optimal method is higher. But for a larger tree the TDR difference is either small or higher for CART.

Take for example BendersOCT, this method has a higher or equal discovery rate to CART, for a tree depth lower than four and 10 or 50 encoded features. This means that for these settings the trees created with the optimal approach are not only better in predicting, but also in discovering the right features, or the true data generating process. However, for deeper trees and trees with more features CART often outperforms BendersOCT. This is due to the fact that in these instances BendersOCT does not get close to finding an optimal solution within the time limit.

From Figure 12 it becomes clear that a deeper tree and more features are negatively correlated with the true discovery rate. For a tree of low depth and a small amount of features the optimal tree methods all show a high TDR. The opposite is also true: for a higher tree depth and more features CART has a higher or equal TDR.

The intuition goes as follows. The depth of the tree has an influence on the

Figure 12: A heatmap of the TDR of the different methods.

number of ground truth features. For a deeper tree the number of features used is higher. The more features there are in the synthetic dataset, the less likely it is for the feature used in the decision tree to be a ground truth feature.

The mean TDR is not all that matters. It is also important that the standard deviation of the TDR is low, because in that case I can assume that the variability between the trees of different training samples is low. Since if for different trees the discovery rate is more or less the same, and hence the standard deviation low, then it is likely that the same features are selected for trees generated on different training samples.

We see that the standard deviation is quite low for the different methods, but only BendersOCT has a lower mean standard deviation than CART (0.05 and 0.07 resp.). BinOCT, FlowOCT and OCT have standard deviations of 0.08, 0.09 and 0.08 respectively. The standard deviation for some instances is inflated because for some training samples it was not possible to find any (good) solution within the time limit. Even though the standard deviations are not high, it does not perform significantly better than CART.

5.2.3 Running Times

Table A1(see Appendix A) shows the running times of the methods on the different synthetic datasets. From this table it appears that the biggest influence on the running time is depth. With the synthetic datasets it is possible to see the effect of a single factor on the computation time holding the other factors equal.

However, there are still two uncontrolled factors in this experiment. First of all, no second computer was at my disposal. Therefore, the computer was used for various purposes, while the program was running for a long time. It can be that the use of the computer for these purposes, affected the performance of Gurobi. Secondly, for each depth a new dataset is created, thus the DGP in some dataset might be easier to find for MIP-methods than in other datasets.

These two reasons might explain some unexpected datapoints. For example, the running time of BendersOCT decreases between 75 and 375 observations for a dataset generated with a tree of depth 3 and 10 features. The methods also perform unexpectedly well on the dataset with depth 4, 75 observations and 100 features. Instead of an increase in the running-time compared to the datasets with depth 4, 75 observation and 10 or 30 features there is a decrease in running time.

In Appendix A, I analyze the effect of an increase in the features, depth and number of observations on the running time for different methods. However, this analysis is lengthy and not does not cover all the possible combination of factor increases. Still, this analysis might give useful insights regarding the most important contributor is to an increase in the running times. In Table 4, I provide an alternative way to show the biggest contributor to an increase in the running time.

To create Table 4, I compute the factor increase of the running time when either the features, the observations or the depth increase, while the remaining two stay constant. For example, the depth is increased by one (from 2 to 3, 3 to 4 and 4 to 5) while the features and number of observations stay 10 and 75. This is done for all possible combinations of features and number of observations. In case of the depth, this results in a 27 factor increases (the product of: 3 depth increases, 3 different feature settings and 3 different observation settings). Over this array, the average is taken.

This is another way to study the most important contributor to an increase in running time. However, there are a few limitations to take into consideration.

For example, if an instance already reached the maximum running time, then an increase in depth, feature or observation is likely to give a factor increase of 1. Table 4 does not take this into account. Nevertheless, it gives a good overview of the most important contributor, the depth, to the running time. The average factor increase for depth is around 3.75 times as high as the average factor increase for an increase in features and an increase in observations.

Table 4: The average factor increase in the running time per method for an increment of the number of Features, Observations and the Depth

	Feature	Observations	Depth
BendersOCT	4.7	3.0	11.8
FlowOCT	5.9	4.2	21.1
binOCT	4.0	2.7	12.4
OCT	2.4	6.6	16.0
Mean	4.2	4.1	15.3

From this analysis we learned that. The datasets easily become too large, or the tree becomes too deep. The methods are often not able to solve for datasets with many features, nor are they able to construct trees larger than depth 3 (except when the dataset is relatively small).

5.3 UCI Repository Datasets

5.3.1 In-Sample Accuracy

Table 5 displays the in-sample accuracies for varying depths on the datasets from the UCI data repository. All the methods are close to each other in terms of prediction accuracy. Including ties BendersOCT and binOCT perform the best in 22 instances, closely followed by FlowOCT. CART and OCT are only among the best performing methods in 10 and 15 instances respectively. However, when removing ties the differences are even smaller. In the case that one method performs better another method is just performing 1% worse. Or, the methods have a similar accuracy but standard deviation is a little bit higher. For every method

Table 5: In-sample accuracy UCI Repository Datasets

dataset	depth	BendersOCT	FlowOCT	binOCT	OCT	CART
balance-scale	2	**0.68(0.01)**	**0.68(0.01)**	**0.68(0.01)**	**0.68(0.01)**	**0.68(0.01)**
	3	**0.74(0.01)**	**0.74(0.01)**	0.73(0.01)	0.73(0.01)	0.70(0.02)
	4	0.76(0.01)	0.75(0.01)	**0.77(0.01)**	0.76(0.01)	0.72(0.03)
	5	**0.80(0.02)**	0.76(0.03)	0.80(0.01)	0.74(0.02)	0.79(0.01)
breast-cancer	2	**0.79(0.01)**	**0.79(0.01)**	**0.79(0.01)**	**0.79(0.01)**	0.78(0.01)
	3	**0.82(0.01)**	0.82(0.00)	**0.82(0.01)**	**0.82(0.01)**	0.79(0.01)
	4	0.85(0.01)	**0.86(0.01)**	0.86(0.00)	0.84(0.02)	0.82(0.00)
	5	0.88(0.01)	**0.90(0.01)**	0.89(0.01)	0.85(0.01)	0.85(0.01)
car-evaluation	2	**0.78(0.00)**	**0.78(0.00)**	**0.78(0.00)**	**0.78(0.00)**	**0.78(0.00)**
	3	0.81(0.00)	0.81(0.00)	0.81(0.00)	0.78(0.03)	**0.81(0.01)**
	4	**0.83(0.01)**	0.83(0.00)	**0.83(0.01)**	0.80(nan)	0.82(0.01)
	5	**0.87(0.01)**	**0.87(0.01)**	**0.87(0.01)**	0.70(0.00)	**0.87(0.01)**
kr-vs-kp	2	0.85(0.04)	0.84(0.02)	**0.86(0.01)**	0.78(0.04)	0.77(0.01)
	3	**0.90(0.00)**	**0.90(0.00)**	**0.90(0.00)**	0.63(0.07)	**0.90(0.00)**
	4	**0.94(0.00)**	**0.94(0.00)**	**0.94(0.00)**	0.57(0.00)	**0.94(0.00)**
	5	**0.94(0.00)**	**0.94(0.00)**	**0.94(0.00)**	0.78(nan)	**0.94(0.00)**
monk-1	2	**0.84(0.03)**	**0.84(0.03)**	**0.84(0.03)**	**0.84(0.03)**	0.77(0.01)
	3	**0.93(0.02)**	**0.93(0.02)**	**0.93(0.02)**	**0.93(0.02)**	0.88(0.05)
	4	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.88(0.05)
	5	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.98(0.01)	0.89(0.04)
soybean-small	2	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.84(0.03)
	3	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**
	4	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**
	5	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	0.98(0.01)	**1.00(0.00)**
Best	Performance(with ties)	19	18	19	10	10
Best	Performance(without ties)	1	2	0	0	0

the variance is quite low, also for the CART trees, this might be an argument for a low variability.

In my experiment, the CART trees are restricted to having a depth between 2 to 5. In this way, the results for CART can be compared with the results of the optimal tree methods for the same depth. However, it might be the case that finetuning of the parameters gives different, and better results than the optimal trees. Therefore, I run a randomized grid search over a parameter grid (see Table B5 in the Appendix) to finetune the parameters. The depth and the accuracy of the finetuned CART tree are given in 6.

The running time to finetune the parameters for every dataset was less than 0.005 seconds. For the balance-scale, car-evaluation, kr-vs-kp and soybean-small,

the in-sample accuracy is higher than the other methods. Only for monk-1 and breast-cancer there are optimal tree methods that outperform the finetuned CART. This it not surprising, as maybe the structure of some datasets, especially larger datasets, is only found in a tree with a higher depth. Which is what can be seen as well in Table 6, where only for the smallest datasets, monk-1 and soybean-small, the depth is less than or equal to 5.

Table 6: In-sample accuracy for fine-tuned CART tree

dataset	depth	Accuracy
balance-scale	10	0.87
breast-cancer	7	0.82
car-evaluation	9	0.89
kr-vs-kp	13	1.00
monk-1	5	0.88
soybean-small	3	1.00

This raises the question: why use a slow, optimal tree method when a simple CART with finetuned parameters gives better results? Another problem is that, it appears that larger datasets require deeper trees for a good accuracy. However, depth is the biggest factor in an increase in running times. Hence we run into the problem that on the one hand, deep trees are necessary to get good decision trees, but on the other hand, it is time-wise infeasible to estimate deep trees.

5.3.2 Running Times

Figure 14 shows the average running times for the different approaches, on different datasets and depths. The Table of the running times can be found in Table B3 in the Appendix.

A few things stand out. First of all, there are instances where the method exceeds the maximum running time. For the car-evaluation dataset there are even two methods that run for way more than 600 seconds. I could not find out why this happens. Apparently, from gurobi discussion forums it appear that more people

experience a violation of the maximum running-time. But there is no clear reasons why it happens in some cases and does not in others.

From Figure 14 it also becomes clear that for a depth of two almost every method is able to find an optimal solution within the time limit, and often even in less than 5 seconds. The only dataset for which not every method is able to find an optimal tree within 10 minutes is the kr-vs-kp dataset, the largest dataset on which I test the methods.

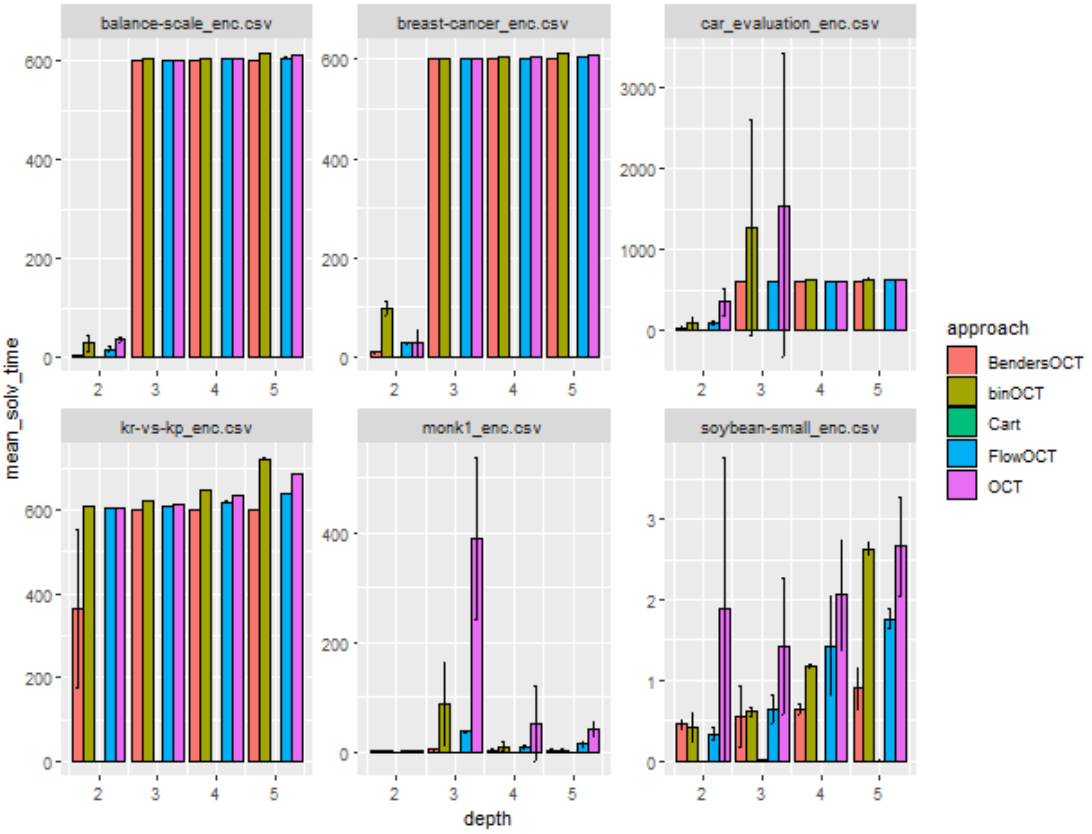

Figure 13: Running times of methods on different datasizes and different depths

For depths higher than two almost every instance reaches the maximum running time, implying that most trees with a depth higher than two are suboptimal. Only for the small datasets with less than 150 observations, monk1 and soybean-small this is not the case. Breast-cancer is another dataset with a small amount of observations, 277, but 39 features. For the breast-cancer dataset the maximum running time is already reached in every instance with a depth larger than two.

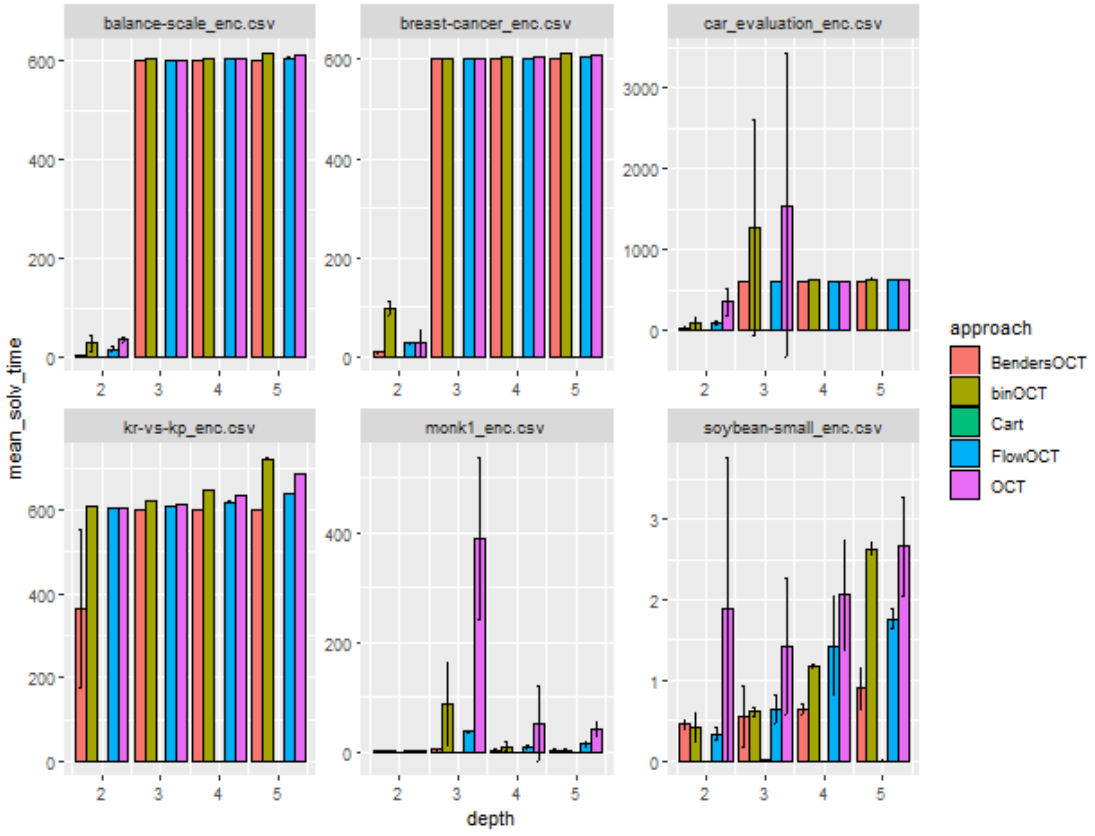

Figure 14: Running times of methods on different datasizes and different depths

In contrast to the experiment with synthetic data it is hard to see the effect of the number of features and observations in this experiment. Mainly because the time limit is reached before the methods are solved. But these results are telling enough. It will be hard to apply optimal tree methods to real life scenarios.

A tree of a depth larger than two is often necessary for good accuracy. However, for a depth of two the methods can not even find an optimal tree for relatively small datasets. Therefore I assume that solutions that reduce the number of features or observations only have a limited effect. Because, even if the dataset is reduced in size the method won't be able to solve a tree of a depth higher than two.

I chose a relatively short time limit right now. The time limit I used is ten minutes, in Aghaei et al. (2021) the time limit is 60 minutes and in Bertsimas and Dunn (2017) 30 minutes (and 2 hours in some instances). Due to time constraints it is infeasible for me to run all the instances on a 30 or 60 minutes time limit.

However, when studying the results of Aghaei et al. I find that even for a 60 minute time-limit only a small percentage of the instances is solved to optimality (see Table B4 in the Appendix).

6 Conclusion

In this research, I studied whether the methods to generate optimal trees are applicable to larger datasets, and if not, whether there are modifications to the methods that make it possible to apply them to larger datasets. Secondly, I studied whether optimal trees are less variable upon the training dataset used. To answer the research question, I applied the methods on synthetic datasets and six publicly available datasets from the UCI data repository. By applying the methods on datasets of varying sizes, I got a better insight into the influences on the computation time of the methods.

From studying the running times of varying methods on the synthetic datasets, it became clear that the depth has the most influence on the running times. The average factor with which the mean running time increases when the depth increases, is almost 4 times as high compared to when the number of observations and the number of feature increases.

By studying the running times of varying methods on the UCI repository datasets I can conclude that for larger depths the optimal tree methods are not able to solve the problem. For a tree depth larger than two, the methods are only able to get an optimal tree for two small datasets, monk-1 and soybean-small.

This finding answers the question, 'are the methods to generate optimal trees applicable to larger datasets?', negatively. Since the depth of the tree is the main cause of large running times, it is hard to find a good solution to the problem. If the cause of slow running times is the number of observations then a method could have been used that selects a subset of the data without throwing away valuable datapoints as in Zhu et al. (2020). If the bottleneck would have been the number of features, then the least important features could be excluded from the dataset. However, even for relatively small datasets, like breast-cancer (277 observations and 39 features), the methods are not able to find a solution for a depth higher than two. Therefore, it seems implausible that these solutions would work in practice.

Diving deeper into the results of Aghaei et al. (2021) and Feijen (2018), and examining their papers more thoroughly, it appears that the methods do not perform well on large datasets and deeper trees. At first glance, the papers gave the wrong impression by presenting the results better than they actually are.

For Feijen (2018) every instance is solved within the time-limit. However, there are two catches. Firstly, he uses the iris dataset, a dataset with 150 observations, 4 features and 3 classes. Secondly, the trees he computes have a maximum depth of 2. And as becomes clear from my results, the problem of not being able to solve a tree to optimality, does not occur at a tree depth of 2, but at a tree depth higher than two.

For Aghaei et al. (2021) we see something similar happening. They present their methods as methods that are way faster than BinOCT and OCT. However, the method is not applicable to larger datasets, which is not immediately clear from reading the paper. They finetuned the λ paramter, so for every dataset and every depth they did 50 runs (10 different λ parameters and 5 different subsets). The results correspond to my findings. For the small datasets, monk1 and soybean-small, every method is able to find an optimal tree within the time limit. However, for the other four datasets (kr-vs-kp, car-evaluation, breast-cancer and balance-scale) this changes. For kr-vs-kp and car-evaluation there are only 4 instances of a tree depth higher than 2 solved within the time-limit. For breast-cancer and balance-scale the solved instances are a bit higher, but still small (for the exact results see Table B4 in the Appendix). The total running time for finetuning the different approaches for different depths on these 6 datasets is almost 80 days.

For now I think it is not feasible in practice to create optimal trees. First of all, it takes a lot of time to run and even then does find an optimal solution. Secondly, using the CART algorithm with finetuned parameters gives a higher accuracy in less than 0.005 seconds.

Even though the optimal tree methods are not solved to optimality, it still contributes to the existing literature as it outperforms CART trees of the same debt in almost every instance. An idea, to make use of the optimal trees use CART as a warmstart for the optimal method. Then, depending on how quickly a model is needed, we can let the optimal method run. In this way we always have at least the same accuracy as CART, and might even improve upon it.[3] Another interesting idea is to investigate local optimal trees. These trees are quick to estimate, even for large datasets and deep trees, and perform better than CART. It does not provide globally optimal trees (i.e. the best tree), but locally optimal

[3]My code already allows for this, but since I for some reason was not able to implement it on BendersOCT(probably because of the lazy callbacks) I did not incorporate it in this thesis.

trees (i.e. it is not possible to improve the tree by changing one of the nodes). Dunn (2018) develops this idea in his phd-thesis.

I also studied the discovery rate and the standard deviation of the accuracy. It might be that optimal trees are less variable and contingent upon the subset of data used to train the tree. Less variable trees are a desirable feature, especially from a policy making perspective. If the decision rules change for every subset that the tree is trained upon, it is harder to make a consistent policy.

On the synthetic datasets, the optimal trees were indeed better in discovering the features of the DGP than CART. However, only for BendersOCT, the mean of the standard deviations was lower than CART. The standard deviation of the mean in-sample accuracy was low for every method.

On the UCI repository datasets, it is not possible to compute the discovery rate. For these datasets, I took the standard deviation of the mean accuracy as a proxy for the variability of the trees. It turns out that this is lower than 0.05 for every instance.

Because the true discovery rate (TDR) of optimal tree methods is higher than the TDR of CART and the standard deviation is low (but only for BendersOCT, it is lower than CART), it might indicate that the trees are less variable. However, I discovered that in some cases, different trees achieve the same accuracy. Hence, it seems that the TDR and standard deviation are not the best proxies for determining the variability of decision trees. To really find an answer to the question of variability, a more extensive analysis is necessary. A good starting point for this research is Bakırlı and Birant (2017) and Sabbaghan et al. (2020), they provide a good framework to measure the similarity between trees.

Even though the research did not deliver the desired results, it still contributes to the existing literature. Mainly in that the MIP methods used in the current literature cannot be applied to large datasets, even though the researchers do not explicitly mention this. In spite of the large increase in computer speed since the first paper on CART, the computer is not fast enough yet. Maybe in a few years, when the speed and power of computers increased manifold again, then solving optimal trees on large datasets will be feasible.

Bibliography

Aghaei, S., Gómez, A., and Vayanos, P. (2021). Strong optimal classification trees.

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509.

Arthanari, T. S. (1981). *Mathematical programming in statistics / T. S. Arthanari, Yadolah Dodge.* Wiley series in probability and mathematical statistics. Wiley.

Bakırlı, G. and Birant, D. (2017). Dtreesim: A new approach to compute decision tree similarity using re-mining. *Turkish Journal of Electrical Engineering Computer Sciences*, 25:108–125.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.*, 4(1):238–252.

Bennett, K. P. (1992). Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Bennett, K. P. and Blue, J. A. (1996). Optimal decision trees. Technical report, R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute.

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

Bertsimas, D. and King, A. (2016). OR Forum—An Algorithmic Approach to Linear Regression. *Operations Research*, 64(1):2–16.

Bertsimas, D. and King, A. (2017). Logistic regression: From art to science. *Statistical Science*, 32:367–384.

Bertsimas, D., King, A., and Mazumder, R. (2015). Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44.

Bertsimas, D. and Mazumder, R. (2014). Least quantile regression via modern optimization. *The Annals of Statistics*, 42(6):2494 – 2525.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees.* CRC press.

Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Dunn, J. (2018). *Optimal trees for prediction and prescription*. PhD thesis.

Feijen, W. (2018). Computing Optimal Classification Trees.

FICO (2022). Xpress-optimizer, reference manual.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940.

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual.

Hochbaum, D. (2008). The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56:992–1009.

Hyafil, L. and Rivest, R. (1976). Constructing optimal binary search trees is np complete. *Information Processing Letters*.

Karger, D. R. (2000). Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76.

Klotz, E. and Newman, A. M. (2013). Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(1-2):18–32.

Mittelmann, H. (2018). Latest benchmark results. pages 4–7.

Mittelmann, H. D. (2020). Benchmarking optimization software-a (hi) story. In *SN Operations Research Forum*, volume 1, pages 1–6. Springer.

Murthy, S. K. and Salzberg, S. L. (1995). Decision tree induction: How effective is the greedy heuristic? In *KDD*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259:801–817.

Sabbaghan, S., Chua, C. E. H., and Gardner, L. A. (2020). Statistical measurement of trees' similarity. *Quality & Quantity*, 54:781–806.

Son, N. H. (1998). From optimal hyperplanes to optimal decision trees. *Fundamenta Informaticae*, 34(1-2):145–174.

Verwer, S. and Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1625–1632.

Zhu, H., Murali, P., Phan, D., Nguyen, L., and Kalagnanam, J. (2020). A scalable mip-based method for learning optimal multivariate decision trees. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1771–1781. Curran Associates, Inc.

A Appendix

First we study the effect of an increase in the number of observations holding the depth and number of features equal, each time the observations increase with factor 5. For BendersOCT the increase in average running time between 75 observations to 375 observations and 375 to 1875 observations is a factor 3.9 and 5.7 respectively for a tree of depth 2 and 10 features. The average running time of BendersOCT between 75 observations and 375 observations decreases and from 375 observations to 1875 observations it increases by factor 6.5 for a tree of depth 3 and 10 features. For depth 4 and 5 the method already reaches it's max computation time (the average running time is close to 600 seconds, so the factor increase does not give us a lot of information). For other methods the factor-increase for the number of observation is higher. For FlowOCT the average running time between 75 observations and 375 observations and 375 to 1875 observations is a factor 27.4 and 11.0 respectively for a tree of depth 2 and 30 features, although for the dataset with 1875 observations and 30 features the method is not always able to find the optimal solution.

For the method binOCT, a method where the complexity should be independent of the number of rows in the dataset, there is a small increase in the running time contingent upon the number of observations in the dataset. For BinOCT the average running time between 75 observations and 375 observations and 375 to 1875 observations is a factor 1.2 and 1.07 respectively for a tree of depth 3 and 10 features. However, for a higher number of features this method already often reaches the maximum computation time.

Secondly we study the effect of an increase in the number of features holding the depth and number of observations equal. This effect is similar in size to the number of observations. For BendersOCT the increase in average running time between 10 and 30 features and 30 to 100 features is a factor 9.7 and 3.0 respectively for a tree of depth 2 and 75 observations. The average running time of BendersOCT between 10 and 30 features and 30 and 100 features increases by factor 8.3 and 3.5 respectively for a tree of depth 3 and 75 observations(although for these settings the time-limit is already reached). For FlowOCT the increase in average running time between 10 and 30 features and 30 to 100 features is a factor 2.8 and 32.7 respectively for a tree of depth 2 and 75 observations. The average running time

Table A1: Running Time on Synthetic Datasets

depth	nrow	approach features	BendersOCT	FlowOCT	binOCT	OCT	CART
2	75	10	0.40(0.22)	0.62(0.21)	0.57(0.14)	0.93(0.60)	0.00(0.00)
		30	3.95(2.62)	1.76(0.97)	8.82(5.98)	4.32(2.97)	0.00(0.00)
		100	11.95(8.86)	57.56(38.26)	38.39(58.73)	7.64(6.91)	0.00(0.00)
	375	10	1.57(0.31)	5.42(2.47)	3.06(2.38)	10.98(5.20)	0.00(0.00)
		30	5.63(3.66)	48.11(39.54)	18.13(26.27)	20.79(11.52)	0.00(0.00)
		100	114.07(88.53)	827.86(437.23)	214.06(129.45)	475.04(266.57)	0.02(0.01)
	1875	10	8.92(3.34)	11.13(4.51)	13.61(8.38)	193.11(25.71)	0.00(0.00)
		30	59.04(91.04)	529.89(89.34)	296.95(221.27)	591.28(28.21)	0.01(0.00)
		100	497.31(172.52)	612.55(0.06)	635.78(0.75)	615.41(0.26)	0.01(0.00)
3	75	10	20.54(26.28)	72.84(55.95)	82.54(99.72)	157.99(110.73)	0.00(0.00)
		30	171.17(237.02)	600.37(0.02)	593.97(13.86)	464.89(222.72)	0.00(0.00)
		100	600.03(0.01)	404.96(236.98)	511.92(182.71)	510.30(183.5)	0.00(0.00)
	375	10	8.89(6.78)	159.05(102.22)	101.75(51.75)	439.00(190.73)	0.00(0.00)
		30	180.41(141.63)	602.24(0.17)	505.54(198.28)	463.60(111.97)	0.00(0.00)
		100	600.05(0.01)	599.48(13.39)	616.87(0.63)	609.84(0.19)	0.01(0.00)
	1875	10	57.44(40.9)	377.64(160.91)	108.52(131.59)	606.01(0.20)	0.00(0.00)
		30	600.10(0.06)	611.82(2.09)	627.96(6.20)	617.20(2.14)	0.01(0.00)
		100	600.19(0.13)	629.82(3.29)	679.28(1.62)	646.12(1.11)	0.02(0.00)
4	75	10	451.50(297.06)	369.24(267.56)	600.83(0.06)	300.61(290.33)	0.00(0.00)
		30	466.64(168.42)	600.78(0.02)	602.34(0.05)	477.69(247.83)	0.00(0.00)
		100	303.67(342.29)	352.24(274.64)	143.32(112.18)	144.59(69.49)	0.00(0.00)
	375	10	566.15(67.76)	491.43(221.4)	457.11(293.42)	477.14(252.47)	0.00(0.00)
		30	600.04(0.01)	603.87(0.12)	610.19(0.38)	606.95(0.41)	0.00(0.00)
		100	600.08(0.04)	612.13(0.24)	634.51(0.41)	624.48(1.24)	0.01(0.00)
	1875	10	600.07(0.04)	610.36(0.15)	618.99(1.09)	615.69(0.17)	0.00(0.00)
		30	600.26(0.18)	628.24(7.46)	668.05(11.76)	643.32(6.86)	0.01(0.00)
		100	600.23(0.15)	2093.28(2879.58)	870.92(204.47)	773.30(124.2)	0.02(0.00)
5	75	10	71.41(42.27)	257.38(238.86)	334.15(310.46)	174.48(68.26)	0.00(0.00)
		30	331.02(312.46)	111.05(65.43)	290.31(237.35)	175.12(101.94)	0.00(0.00)
		100	197.28(282.87)	126.31(125.74)	203.19(277.03)	362.06(287.43)	0.00(0.00)
	375	10	600.04(0.01)	604.35(0.22)	610.09(0.23)	607.98(0.58)	0.00(0.00)
		30	600.06(0.02)	895.01(572.57)	630.90(10.12)	618.21(0.71)	0.00(0.00)
		100	600.14(0.04)	623.25(1.48)	688.14(2.65)	657.39(3.39)	0.01(0.00)
	1875	10	600.12(0.09)	620.77(0.42)	646.92(1.32)	638.10(1.24)	0.00(0.00)
		30	600.11(0.03)	644.42(4.94)	749.32(14.64)	705.32(16.21)	0.01(0.00)
		100	600.34(0.19)	721.60(8.35)	1437.26(429.85)	932.58(32.37)	0.03(0.00)

of FlowOCT between 10 and 30 features is 8.2 and between 30 and 100 features it decreases for a tree of depth 3 and 75 observations(here as well the time-limit is

reached).

Lastly we study the effect of an increase in the depth while holding the number of features and observations equal. The depth of the tree is the biggest factor in the computation time. I first want to note that for depth 4 and 5 the majority of the datasets does not solve the problem within the time-limit. For depth 4 and 5 only the dataset with 75 observations and the dataset with 375 observations and 10 features has a smaller mean computation time than 600, however, even for these datasets the standard deviation is high (meaning that for some samples the time-limit is reached).

For BendersOCT the increase in average running time between depth 2 and 3 and between depth 3 and 4 is 51.4 and 22 respectively (however, for depth 4 the standard deviation is 297 as well) for a tree of depth 75 observations and 10 features. BendersOCT shows the same pattern with a dataset with more observations, for a dataset with 375 observations and 10 features the running time increases with a factor 5.7 between depth 2 and 3, and becomes computationally infeasible for depth 4 and 5. For OCT the increase in average running time between depth 2 and 3 is 22 and between depth 3 and 4 is unknown since the method becomes unsolvable for a tree of depth 375 observations and 30 features.

B Appendix

Table B2: False and True Discovery Rate on the Synthetic Datasets

		approach	BendersOCT		FlowOCT		binOCT		OCT		Cart	
			FDR	TDR	FDR	TDR	FDR	TDR	FDR	TDR	FDR	TDR
depth	nrow	features										
2	75	10	0.00(0.00)	1.00(0.00)	0.08(0.17)	0.92(0.17)	0.08(0.17)	0.92(0.17)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
		30	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
		100	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
	375	10	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
		30	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
		100	0.00(0.00)	1.00(0.00)	0.17(0.19)	0.83(0.19)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.08(0.17)	0.92(0.17)
	1875	10	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.00(0.00)	1.00(0.00)	0.33(0.00)	0.67(0.00)
		30	0.00(0.00)	1.00(0.00)	0.17(0.19)	0.83(0.19)	0.08(0.17)	0.92(0.17)	0.25(0.17)	0.75(0.17)	0.33(0.00)	0.67(0.00)
		100	0.08(0.17)	0.92(0.17)	0.17(0.19)	0.83(0.19)	0.17(0.19)	0.83(0.19)	0.17(0.19)	0.83(0.19)	0.17(0.19)	0.83(0.19)
3	75	10	0.07(0.08)	1.00(0.00)	0.11(0.07)	1.00(0.00)	0.14(0.00)	1.00(0.00)	0.14(0.00)	1.00(0.00)	0.29(0.08)	0.71(0.08)
		30	0.00(0.00)	1.00(0.00)	0.29(0.20)	0.71(0.20)	0.43(0.37)	0.57(0.37)	0.21(0.34)	0.79(0.34)	0.46(0.24)	0.54(0.24)
		100	0.54(0.07)	0.46(0.07)	0.54(0.24)	0.46(0.24)	0.39(0.27)	0.61(0.27)	0.50(0.25)	0.50(0.25)	0.69(0.36)	0.29(0.33)
	375	10	0.00(0.00)	0.96(0.07)	0.04(0.07)	0.96(0.07)	0.04(0.07)	0.96(0.07)	0.29(0.26)	0.71(0.26)	0.33(0.00)	0.57(0.00)
		30	0.26(0.06)	0.83(0.00)	0.33(0.13)	0.75(0.10)	0.36(0.08)	0.75(0.10)	0.32(0.07)	0.79(0.08)	0.21(0.14)	0.83(0.00)
		100	0.43(0.12)	0.57(0.12)	0.21(0.18)	0.79(0.18)	0.36(0.08)	0.64(0.08)	0.32(0.07)	0.68(0.07)	0.32(0.07)	0.68(0.07)
	1875	10	0.00(0.00)	1.00(0.00)	0.07(0.14)	0.93(0.14)	0.00(0.00)	1.00(0.00)	0.29(0.00)	0.71(0.00)	0.36(0.08)	0.64(0.08)
		30	0.32(0.07)	0.68(0.07)	0.32(0.07)	0.68(0.07)	0.32(0.07)	0.68(0.07)	0.32(0.07)	0.68(0.07)	0.32(0.07)	0.68(0.07)
		100	0.29(0.00)	0.71(0.00)	0.29(0.00)	0.71(0.00)	0.29(0.00)	0.71(0.00)	0.29(0.00)	0.71(0.00)	0.29(0.00)	0.71(0.00)
4	75	10	0.34(0.01)	0.70(0.04)	0.34(0.09)	0.66(0.09)	0.38(0.06)	0.66(0.07)	0.40(0.09)	0.64(0.10)	0.29(0.07)	0.52(0.04)
		30	0.57(0.16)	0.52(0.18)	0.54(0.17)	0.54(0.17)	0.67(0.12)	0.42(0.15)	0.55(0.08)	0.56(0.10)	0.60(0.16)	0.33(0.15)
		100	0.82(0.12)	0.15(0.10)	0.83(0.07)	0.15(0.04)	0.87(0.05)	0.17(0.07)	0.88(0.11)	0.15(0.14)	0.83(0.13)	0.10(0.08)
	375	10	0.39(0.08)	0.69(0.11)	0.41(0.10)	0.63(0.12)	0.42(0.08)	0.67(0.10)	0.43(0.07)	0.62(0.00)	0.45(0.08)	0.56(0.07)
		30	0.55(0.08)	0.46(0.09)	0.40(0.05)	0.64(0.06)	0.42(0.03)	0.62(0.04)	0.43(0.04)	0.61(0.04)	0.42(0.03)	0.62(0.04)
		100	0.93(0.00)	0.07(0.00)	0.45(0.04)	0.54(0.09)	0.53(0.08)	0.50(0.08)	0.40(nan)	0.64(nan)	0.45(0.04)	0.54(0.09)
	1875	10	0.37(0.07)	0.63(0.07)	0.38(0.07)	0.60(0.05)	0.33(0.12)	0.67(0.12)	0.40(0.05)	0.60(0.05)	0.40(0.05)	0.60(0.05)
		30	0.48(0.11)	0.52(0.11)	0.48(0.11)	0.52(0.11)	0.47(0.14)	0.53(0.14)	0.47(0.14)	0.53(0.14)	0.48(0.11)	0.52(0.11)
		100	0.92(0.07)	0.07(0.05)	0.44(0.03)	0.55(0.03)	0.45(0.03)	0.55(0.03)	0.44(0.04)	0.56(0.04)	0.44(0.03)	0.55(0.03)
5	75	10	0.50(0.06)	0.79(0.08)	0.43(0.12)	0.74(0.25)	0.55(0.05)	0.82(0.08)	0.54(0.05)	0.75(0.09)	0.28(0.15)	0.47(0.08)
		30	0.70(0.07)	0.55(0.14)	0.65(0.07)	0.56(0.14)	0.69(0.06)	0.59(0.12)	0.71(0.03)	0.50(0.09)	0.35(0.14)	0.47(0.11)
		100	0.78(0.11)	0.20(0.05)	0.71(0.14)	0.28(0.08)	0.90(0.05)	0.22(0.10)	0.92(0.05)	0.13(0.08)	0.75(0.15)	0.18(0.08)
	375	10	0.35(0.05)	0.67(0.05)	0.27(0.08)	0.70(0.09)	0.31(0.04)	0.72(0.04)	0.37(0.08)	0.55(0.16)	0.23(0.05)	0.64(0.04)
		30	0.56(0.07)	0.48(0.07)	0.41(0.07)	0.57(0.08)	0.42(0.08)	0.64(0.10)	0.61(0.11)	0.32(0.17)	0.42(0.06)	0.56(0.08)
		100	0.83(0.05)	0.18(0.05)	0.60(0.05)	0.38(0.05)	0.66(0.03)	0.38(0.03)	0.97(0.04)	0.01(0.02)	0.62(0.06)	0.36(0.05)
	1875	10	0.41(0.04)	0.63(0.04)	0.38(0.06)	0.66(0.07)	0.38(0.05)	0.66(0.05)	0.41(0.02)	0.63(0.02)	0.42(0.08)	0.61(0.09)
		30	0.36(0.05)	0.64(0.04)	0.35(0.03)	0.66(0.02)	0.37(0.02)	0.66(0.02)	0.35(nan)	0.67(nan)	0.36(0.03)	0.64(0.02)
		100	0.92(0.06)	0.08(0.05)	0.54(0.03)	0.46(0.03)	0.55(0.05)	0.45(0.05)	0.54(0.03)	0.46(0.03)	0.54(0.03)	0.46(0.03)

Table B3: Running Time of Methods on UCI Repository Datasets

dataset	depth	BendersOCT	FlowOCT	binOCT	OCT	Cart
balance-scale	2	4.71(0.75)	15.98(5.60)	28.05(15.35)	34.87(6.09)	0.00(0.00)
	3	600.02(0.00)	601.55(0.02)	602.73(0.28)	601.99(0.02)	0.00(0.00)
	4	600.02(0.00)	603.07(0.09)	605.54(0.12)	604.55(0.05)	0.00(0.00)
	5	600.04(0.00)	605.65(0.27)	613.89(0.29)	610.83(0.51)	0.00(0.00)
breast-cancer	2	10.22(2.01)	27.80(2.46)	97.28(13.62)	28.92(25.8)	0.00(0.00)
	3	600.02(0.00)	601.02(0.04)	602.31(0.08)	601.61(0.12)	0.00(0.00)
	4	600.02(0.00)	602.27(0.22)	604.59(0.11)	603.41(0.06)	0.00(0.00)
	5	600.04(0.00)	604.02(0.08)	611.31(0.19)	608.30(0.74)	0.00(0.00)
car-evaluation	2	34.42(5.78)	90.12(12.65)	88.10(62.75)	349.41(168.9)	0.00(0.00)
	3	600.10(0.10)	605.13(1.66)	1276.47(1332.9)	1544.11(1875.68)	0.00(0.00)
	4	600.05(0.03)	607.11(0.11)	614.24(0.23)	611.49(0.28)	0.00(0.00)
	5	600.06(0.03)	614.43(0.08)	635.32(0.55)	627.54(0.40)	0.00(0.00)
kr-vs-kp	2	365.64(188.5)	604.33(0.06)	610.16(0.09)	604.79(0.11)	0.00(0.00)
	3	600.07(0.06)	609.54(0.33)	621.15(0.35)	613.77(0.68)	0.01(0.01)
	4	600.15(0.16)	619.44(0.47)	647.31(0.98)	634.71(0.88)	0.00(0.00)
	5	600.10(0.03)	639.84(0.40)	721.35(2.94)	686.05(0.80)	0.01(0.00)
monk-1	2	1.40(1.45)	2.04(0.95)	1.40(0.39)	2.38(0.51)	0.00(0.00)
	3	4.96(1.59)	37.68(1.60)	87.89(75.3)	389.34(147.61)	0.00(0.00)
	4	2.79(1.04)	8.92(3.42)	9.89(7.97)	50.91(68.27)	0.00(0.00)
	5	2.68(1.31)	14.27(4.03)	3.60(0.64)	42.13(13.22)	0.00(0.00)
soybean-small	2	0.46(0.06)	0.33(0.07)	0.41(0.18)	1.88(1.88)	0.00(0.00)
	3	0.54(0.38)	0.64(0.18)	0.61(0.06)	1.41(0.84)	0.00(0.00)
	4	0.64(0.06)	1.42(0.62)	1.17(0.03)	2.05(0.68)	0.00(0.00)
	5	0.90(0.25)	1.76(0.12)	2.63(0.07)	2.66(0.61)	0.00(0.00)

Table B4: Number of solved instances(max. 5 for BinOCT and 50 for the other methods) on UCI Repository Datasets in Aghaei et al. (2021)

dataset	depth	BendersOCT	BinOCT	FlowOCT	OCT
balance-scale	2	50	5	50	50
	3	50	0	50	0
	4	5	0	0	0
	5	0	0	0	0
breast-cancer	2	50	5	50	50
	3	11	0	9	4
	4	5	0	5	1
	5	5	0	5	0
car-evaluation	2	50	5	50	48
	3	4	0	0	0
	4	0	0	0	0
	5	0	0	0	0
kr-vs-kp	2	50	5	50	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0
monk1	2	50	5	50	50
	3	50	5	50	48
	4	50	5	50	49
	5	50	5	50	31
soybean-small	2	49	5	50	50
	3	50	5	50	50
	4	50	5	50	50
	5	50	5	50	50

Table B5: Parameter Grid For Finetuning Cart

	Grid Options
max features	'auto', 'log2', None
max depth	1, 2, ..., 14, 15
min samples split	2, 6 , 10
min samples leaf	2, 6, 10
max leaf nodes	None, 10, 20, ..., 90