Optimal Robust Decision Trees

A dynamic programming approach

Andreas Benedict Conrad Bien

Supervisor(s): Emir Demirović, Koos van der Linden

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Andreas Benedict Conrad Bien
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Burcu Özkan

Abstract

Decision trees are integral to machine learning, with their robustness being a critical measure of effectiveness against adversarial data manipulations. Despite advancements in algorithms, current solutions are either optimal but lack scalability or scale well, but do not guarrantee optimality. This paper presents a novel adaptation of the Murtree algorithm to address these challenges in the pursuit of optimal robust decision trees. We introduce a new method for modeling an adversary as a network flow problem, and provide a dynamic programming approach to solve optimal robust decision trees beyond a depth of two. The performance of our proposed algorithm is compared with brute-force solutions across varying decision tree depths, feature numbers, and data sizes. This research contributes a significant advancement towards obtaining efficient and effective solutions for optimal robust decision trees, potentially setting a new performance benchmark in this area.

1 Introduction

Decision trees are widely used in machine learning, which can be optimized for many different objectives, such as minimizing the mean squared error, maximizing the f1 score, optimizing survival analysis and so on. They are known for the intuitive visual representation and interpretability, making them a popular choice for classification tasks; however, achieving optimal performance often becomes challenging, especially when factors such as robustness come into play. Robustness in machine learning pertains to the ability of an algorithm to perform accurately despite the presence of adversarial inputs, noise, or other forms of alterations in the data. It is crucial in a world where data is becoming increasingly complex and prone to manipulation.

In the case of this paper, we will focus on optimizing decision trees on robustness: minimizing misclassifications assuming an adversary. An adversary is a mechanism that manipulates samples to maximize misclassifications. In essence, this is a minimax problem, where the model chooses the decision tree with minimum misclassifications, after samples have been altered by the adversary to maximize misclassification. The decision tree that has the least number of misclassifications, assuming an adversary, is known as an optimal robust decision tree. The alternative to finding optimal decision trees are heuristic methods, which can find well performing decision trees with less runtime, but do not guarantee optimal performance. The process of finding optimal decision trees establishes a critical performance baseline. This benchmark acts as a valuable reference for heuristic methods and provides an upper bound on their performance potential. Moreover, the pursuit of optimal decision trees is particularly desirable for small and interpretable models. If the search for an optimal solution is feasible, it is likely the preferred option. This is due to the inherent value in obtaining the most efficient and effective solutions. Consequently, the development of algorithms that broaden the feasibility of searching for these optimal solutions is a particularly desirable and significant advancement. Therefore, the work we present in this paper, is aimed at improving the scalability and efficiency of algorithms to find optimal robust decision trees.

Related work on robust decision trees include ROCT [11] which can find optimal robust decision trees using combinatorial algorithms, such as Mixed Integer Linear Programming (MILP) or Maximum Satisfiability (MaxSAT). Due to the high complexity of standard combinatorial algorithms used in ROCT, the model does not scale beyond decision trees with a max-depth greater than two. Other methods that scale better, include GROOT and the Robust Relabeling algorithm, which can find robust decision trees, however, they do not guarrantee optimality.

Relevant work on improving the scalability of optimal decision trees includes the Murtree algorithm [5] and the STreeD algorithm [9]. These algorithms divide the task of finding an optimal robust decision tree into separable optimization tasks, which are used to solve the optimization task itself, using an efficient combining operator. A separable optimization task refers to dividing a problem into smaller subproblems; in the context of minimizing misclassifications, this would be the subtrees of the root-node, and the combining operator would be the addition of misclassifications of those subtrees.

The Murtree algorithm has shown that optimal decision trees can be generated more efficiently than solutions relying on other combinatorial algorithms, using dynamic programming. However, Murtree has not been generalized to optimize on constraints other than misclassifications. The STreeD algorithm attempts to generalize the Murtree algorithm in order to find decision trees that are optimal with regard to other objectives.

As the Murtree algorithm has not been applied to robust decision trees, and the ROCT method is only feasible for very small decision trees, struggling with decision trees of depth greater than 2. Therefore, the aim of this research is to transform the Murtree algorithm into solving optimal robust decision trees, and answer the research question of this paper: "What are the advantages of an adapted version of the Murtree algorithm when applied to computing optimal robust decision trees, compared to state-of-the-art solutions?"

The contributions of this research is the demonstration of a dynamic programming approach applied to finding optimal robust decision trees, the introduction of a novel method for modeling an adversary which results in an exponential speedup with respect to the number of leaves per tree, and an algorithm that can comfortably solve optimal robust decision trees with a max depth of 3, and feasibly solve trees of depths 4 and 5. The results exhibit linear increase in the runtime gain (defined as the runtime of the brute force method divided by the runtime of our proposed algorithm) with respect to the runtime of the brute force method.

2 Related works

The first prominent paper that examined adversarial attacks, was an early paper by Dalvi et al.[4], emphasizing the rapid degradation of classifier performance after deployment due to adaptive adversaries. The paper incited further research on the topic of adversarial machine learning, such as Lowd et al., who focused their research on black-box adversarial attacks [7] (where the adversary does not know the model completely), and later the notable paper by Ling et al. providing a comprehensive exploration of adversarial machine learning. [6]
Adverserial attacks against decision trees were first mentioned in the paper of Papernot et al.[8] where it was shown that adversarial attacks were generalizable to many different machine learning models, such as SVM, logistic regression, others and decision trees. This prompted research into several algorithms that contribute to the body of work on robust decision trees, which do not guarantee optimality. The first notable research conducted was by Chen et al., which introduced an algorithm which made use of recursive greedy splitting at each node, approximating information gain in the context of an adversary. A later paper introduced TREANT, which based their algorithm on robust splitting and attack invariance[3].

The GROOT algorithm, improved on the work of Chen. et al. and the TREANT algorithm, achieving similar accuracy scores but with a runtime of two magnitudes faster. The GROOT algorithm accomplished this by adopting the same greedy recursive splitting method as Chen. et al., and making use of its main contribution: computing the adversarial GINI coefficient in constant time. [10]

Then there is the Robust Relabeling algorithm [12], which further improves GROOT, by introducing a post-learning procedure to optimally change the leaf labels of the decision tree, in the context of an adversary. The algorithm has higher accuracy scores when combined with GROOT, than GROOT itself, however, the runtime is orders of magnitude slower.

The first notable study on optimal robust decision trees is the ROCT algorithm, which presented a solution to identify optimal robust decision trees through the use of combinatorial optimization, using MILP and MaxSAT. However, this method is limited to decision trees with a depth less than or equal to two, as finding decision trees with larger max depths is computationally infeasible. The approach assumes continuous data and an adversary that can perturb data for each feature within defined limits for increasing and decreasing a feature value. [11]

Finally, there are two papers on computing optimal decision trees, which serve as the basis of our research. The first paper is the Murtree algorithm, a novel method for calculating optimal decision trees using dynamic programming, leveraging the recursive nature of decision trees and incorporating caching and pruning. Murtree also introduced a fast depth-2 solver, which is optimized for finding optimal decision trees with max depth 2. [5]

The second paper is the STreeD paper [9], which builds on the Murtree paper's approach, by proposing a more flexible and general recurrence relation to allow for computing optimal decision trees based on other metrics. The recurrence relation allows for optimal solutions to also be presented as a pareto front, as well as a set of non-dominated solutions for situations where no single solution to a subproblem contributes to the primary solution's optimality. In such cases, a set of non-dominated solutions emerges, ensuring that at least one solution from the subproblem is part of the optimal solution. The STreeD algorithm does this by introducing a notation to allow for a more general formulation, such as a cost function, transition function, and a combining operator, which is discussed in further detail in the preliminaries section.

3 Formal Problem Description

The task at hand involves finding an optimal decision tree, given constraints on the depth of the tree. This problem is classified as NP-hard. There are several metrics that can be optimized. such as the mean squared error, the F1 score, or simply minimizing misclassifications. However, we propose an adversarial optimization method.

An optimal robust decision tree can be described as follows: consider the situation where we generate every possible decision tree architecture constrained by the features of the data and the desired maximum depth. Now, let us hypothetically allow an adversary to manipulate the sample data differently for each of these possible decision tree instances. As the adversary alters the samples to maximize misclassifications optimally, we record the misclassification counts for each decision tree.

Subsequently, the decision tree with the least amount of misclassifications out of all potential trees is considered the optimal robust decision tree. In essence, the optimal robust decision tree is where the combined effects of the adversary's manipulations and the resulting misclassifications are minimized.

This problem can be formally expressed using mathematical notation. A binary decision tree, denoted by τ, where each internal node corresponds to a feature f, each branch denotes a decision rule, and each leaf node represents the prediction label. In terms of a function, a binary decision tree τ is defined as $\tau : 0, 1^n \rightarrow 0, 1$, where n signifies the number of features.

The set T contains all possible decision tree architectures restricted by the number of features and the maximum depth.

The adversarial action, denoted by A, acts on a given dataset $d = (x_1, y_1), ..., (x_N, y_N)$, which consists of N samples where each x_i is a binary feature vector and each y_i is a binary label. The function $A : d \rightarrow D$ transforms this data set into a set of all possible perturbations of the data set.

Finally, we define a function $M : T \times D \rightarrow 0, ..., N$ which represents the number of misclassifications made by the decision tree in the data set.

Our problem essentially boils down to a minimax problem. We are seeking a decision tree that minimizes the maximum

possible misclassification count that any adversarial action could induce. This can be represented as:

$$\min_{\tau \in T} \max_{d \in A(D)} M(\tau, d)$$

The objective is to decompose the problem of finding this adversary-resilient optimal decision tree into separable optimization tasks with an efficient combining operator. This algorithm is then evaluated based on runtime to see how well it scales when compared to a brute force method depending on the amount of features, depth, and amount of data samples.

4 Preliminaries

In this section, we introduce notation that is relevant in other papers, as well as notation used throughout the rest of this paper. The section includes some relevant equations, definitions, and theorems. The notation can be formally described as:

- D: Set of instances
- $D^{+/-}$: Set of instances labeled true/false
- $D_{f/\overline{f}}$: Set of instances where feature is true/false
- D_a: Set of instances that the adversary can move outside of the tree
- D_{f_a}: Set of instances where feature value can be flipped by the adversary
- d: Maximum depth
- $\mathcal{F}$: The feature space
- f: A single feature
- F: Set of features
- $\mathcal{K}$: The label space
- $\hat{k}$: A label
- α: Lower bound on cost
- β: Upper bound on cost
- $M(\tau, D)$: Returns total misclassifications given a tree τ and a dataset D
- $C(D, d)$: Minimum cost
- $\tau = (f, \tau_l, \tau_r, \alpha, \beta)$: A tree represented by its feature split, left and right subtrees, and the upper and lower bounds of cost that the tree contributes to the overall decision tree.
- $T = \{\tau_1, \tau_2, \tau_3, ...\tau_n\}$: a set of trees
- $candidates(T) \rightarrow \{\tau_i \in T | \alpha_i < min\{\beta_j | \tau_j \in T\}\}$: Returns all possible trees that might be part of the optimal solution, given a set of trees
- $\tau_i \oplus \tau_j = (f, \tau_i, \tau_j, \alpha, \beta)$: The combining operator which combines two subtrees into a tree.
- $merge(T_l, T_r) \rightarrow \{\tau_i \oplus \tau_j | \tau_i \in T_l, \tau_j \in T_r\}$: Combines the 2 sets of subtrees for the feature split into one set of trees
- $g(D, F, \hat{k})$: Cost function that returns the cost of a leaf node given the label

In this paper, we introduce an algorithm that builds on the "STreeD" algorithm, a method first described in the "STreeD" paper. This foundational work provides a more general method for calculating the optimal decision trees. The algorithm in the paper is described by the following recurrence relation:

$$C(D,F,d) = \begin{cases} \text{opt}\left(\bigcup_{\hat{k} \in \mathcal{K}} g(D,F,\hat{k})\right), & \text{if } d = 0, \\ \text{opt}\left(\bigcup_{f \in \mathcal{F}} \text{merge}(C(D_f, f \cup F, d-1), \right. & \\ \left. C(D_{\overline{f}}, \overline{f} \cup F, d-1)\right)\right), & \text{if } d > 0. \end{cases} \quad (1)$$

The equation above uses other notation not used in this paper, notably *opt* which returns the optimal solution, a pareto front, or a set of non-dominated solutions given a set of solutions. This is replaced by $candidates$, which is a narrowed-down definition of *opt*, where the return of $candidates$ only refers to a set of non-dominated solutions.
The relation above changes based on the depth of the tree: when the depth d = 0, the non-dominated leaf nodes are given; when $d > 0$, a set of non-dominated subtrees is given. The optimal tree (or pareto front) is determined by combining two sets of non-dominated solutions for all feature splits and returning the solutions where the feature split has a non-dominated score.

The work in our paper narrows this algorithm down, to apply for finding optimal robust decision trees, by changing opt to candidates and only considering data with binary labels, unlike the STreeD paper which also considers discrete labels of arbitrary amounts; hence the cost function $g(D, F, \hat{k})$ is replaced by a lower and upper bound. The lower bound is $min(|D^+|, |D^-|)$, where D contains all instances including those that the adversary can move in and out of the leaf node. The upper bound does the same, except it assumes all the instances are moved out of the leaf node and fully contribute to misclassifications outside the leaf node, which can be formulated as $min(|D^+ \setminus D_a|, |D^- \setminus D_a|) + |D_a|$, where D_a refers to the instances that can be moved in and out of the leaf node.

We also pay attention to Theorems 4.7 and 4.8 in STreeD, which are relevant to proving optimality of the algorithm. Theorem 4.7 states *"An optimization task $o = <g, \succ, \oplus, c>$ is separable if and only if its leaf node cost function g is independent, its combining operator $\oplus$ preserves order over its comparison operator $\succ$ and the constraint c is anti-monotonic."*, and theorem 4.8 states *"STreeD, as defined in Eq. (1), finds the Pareto front for any optimization task that is separable according to Def. 4.2."*. We also refer to definitions 4.3, 4.5, and 4.6 in the STreeD paper, which define the three conditions in theorem 4.7. The definitions are explored in further detail later in the paper.

Finally, this paper uses the Push Relabel algorithm [2], a max-flow algorithm, which in practice has a low runtime. While the algorithm is not intrinsically related to decision trees, this paper employs the max-flow algorithm to efficiently calculate the misclassifications in a tree given an ad-

versary. This is accomplished by converting the adversary into a max-flow problem. The algorithm implementation was copied from the GeeksforGeeks website, with an unknown author.[1]

5 Contribution

In this section, we will discuss a novel way of modeling the adversary, as a network flow problem, and describe the proposed algorithm to solve for optimal robust decision trees, in the form of a recurrence relation. The proposed algorithm finds the solution to the minimax problem:

$$\min_{\tau \in T} \max_{d \in A(D)} M(\tau, d)$$

. The problem is discussed in greater detail under the formal problem description section.

5.1 Adversary Model

The adversary can be modeled as a maximum flow problem, where the leaves are depicted as edges emanating from the source and edges leading into the sink. In a binary classification tree, each leaf node indicates the number of true- and false-labeled samples.

The intuition behind the max-flow formulation can best be explained when considering a regular decision tree, where the total amount of misclassifications is simply the sum of the minority labels in each leaf node. This is because the label of a leaf node is determined by the majority label of its instances (e.g., if there are more 'true' instances than 'false', the node is labeled 'true'), and every minority instance is thus a misclassification.

Similarly, in a network representation, the max flow is determined by the smallest capacity of the edges between any source-sink pair. If we construct the network so that these edge capacities correspond to the counts of each label in our decision tree, then the max flow will represent the sum of the minority labels, just as in the decision tree. Thus, the max flow in the network will equal the total misclassifications in the decision tree.

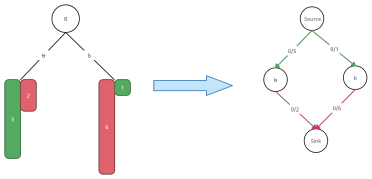

Figure 1: The misclassifications of a decision tree without an adversary transformed into a network flow problem

The max flow of the network described above, visible in Figure 1, is equal to the total amount of misclassifications in the corresponding non-robust decision tree. The total number of misclassifications in a regular binary decision tree is the minimum number of total true and false labels for each leaf added together. The max flow also calculates this same value, since each pair of edges connected by a leaf node can only have a max flow of the edge with the smaller capacity. This shows that Figure 1 is a valid transformation for regular binary decision trees.

For each leaf, two edges are added to the network: an edge that connects from the source node and to a new node (which is representative of the leaf), while another edge connects from the leaf node to the sink node. The capacity of these edges is determined on the basis of the number of true- and false-labeled samples in the leaf node. This means that if a leaf node contains six true samples and three false samples, the capacity of the edge from the source would be six, and the edge entering the sink would have a capacity of three (the other way around is also fine, as long as the network is consistent). This procedure is applied to all leaf nodes, resulting in a network with edges connecting the source and new nodes and new nodes and the sink.

This intuition can be further extended in the context of an adversary, except here the flow through a leaf node can be redirected through edges that represent the different ways the adversary can move the samples around.

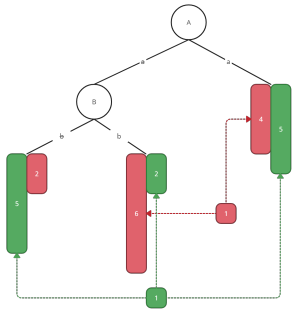

Figure 2: A depth 2 decision tree with three leaf nodes. The green rectangles represent the number of instances labeled true, and the red rectangles represent the number of instances labeled false. The green and red boxes represent a true and false label, respectively, that can be moved to multiple leaf nodes by the adversary (represented by the dotted lines)

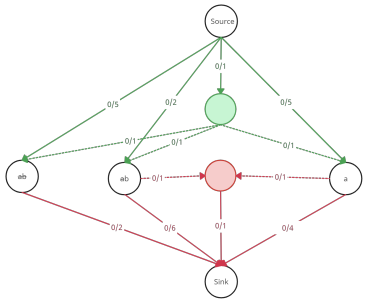

Figure 3: Transformation of the decision tree in figure 2 to a max flow problem, where the red and green nodes represent the instances the adversary can move.

In order to extend a network similar to Figure 1, to account

for an adversary, we consider samples that have the potential to be assigned to multiple leaf nodes. For each sample, an additional node is introduced. In the case of a true sample, an edge of capacity one connects from the source to the sample node, and outgoing edges of capacity one are connected to each leaf node that the adversary can direct the sample to. Conversely, for false samples, the new sample node has one outgoing edge to the sink node, and ingoing edges from all the leaf nodes where the sample could potentially be moved to; all edges also have capacity one.

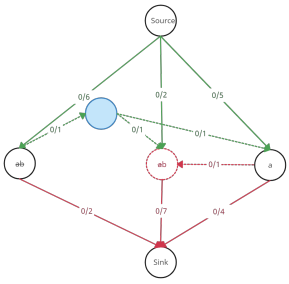

Figure 4: Reduced network flow graph from figure 3, where the red node is combined with the node middle leaf node, and the edge directing flow from the source to the green node is combined with the edge directed towards the left most node.

When a sample can only be directed to two nodes, the new node can be merged with one of the leaf nodes to improve the efficiency of the max-flow algorithm, increasing the capacity of the shared edge by 1, as shown in Figure 4. However, if a sample can be directed to more than two leaf nodes, an additional node is necessary to accommodate the complexity of the network.

Advantages of the adversary model

The advantage we offer by transforming the adversary model into a max flow problem, is that the label of the leaf can remain ambigouos. The conventional way of modelling the adversary is by sending all the adversarial samples to leaf nodes with the opposite label if possible. This increases the search space for each tree, to include labels of the leaf nodes, doubling with each leaf added per tree (assuming binary labels. For example a tree with two leaves could have four possible combinations of labels, a tree with three leaves would have twice that, with eight possible combinations of label. Hence there is a exponential speedup with respect to the number of leaves per tree.

5.2 Algorithm

The primary challenge of developing a STreeD-like algorithm tailored to optimal robust trees lies in the seemingly nonseparable nature of the problem at first glance. Specifically, the optimal and robust subtrees derived from each child node are not necessarily constituents of the overall optimal robust tree. This outcome arises because these optimal robust subtrees fail to consider potential misclassifications engendered by samples that can be moved outside of their respective tree by the adversary.

As a consequence, a degree of uncertainty underlies the number of misclassifications that a subtree might contribute if it is part of the primary solution. We represent this uncertainty by defining an upper and lower bound, wherein the lower bound signifies the minimum conceivable misclassifications and the upper bound indicates the maximum possible misclassifications a subtree could contribute to the overall tree.

This inherent uncertainty implies that there does not exist a single solution within each subtree that is guaranteed to be a part of the complete solution for the optimal robust tree.

Recurrence Relation

$$C(D,F,d) = \begin{cases} \{(F,\emptyset,\emptyset,min(|D^+|,|D^-|) \\ min(|D^+ \setminus D_a|,|D^- \setminus D_a|)+|D_a|\}, & \text{if } d=0, \\ \text{candidates}\Big(\bigcup_{f\in F} \text{merge}\Big(& \\ C(D_f, f\cup F, d-1), & \text{if } d>0. \\ C(D_{\overline{f}}, \overline{f}\cup F, d-1)\Big)\Big), & \end{cases} \tag{2}$$

In examining the given recurrence relation when d = 0, the algorithm returns a set containing a single leaf node, with a lower and upper bound. The leaf node is not assigned a label; instead, the label is decided at every point where it is combined with another tree (either directly, or indirectly through a parent node), when computing the lower and upper bounds of the larger tree.

For cases where $d > 0$, the algorithm returns a list of all potential subtrees that could be part of the optimal robust decision tree. The need for a set of subtrees instead of a single subtree arises because a robust subtree deemed optimal does not ensure its inclusion in the larger, optimal robust decision tree. The basis for this lies in the program's inability to quantify the misclassifications caused by samples that can be moved in and out of the subtree by the adversary. As a result, all returned subtrees contain an upper and a lower bound of misclassifications. Only the final set of potentially optimal decision trees returned is guaranteed to have a single tree, as the tree encompasses all samples, hence no samples can be moved outside of the tree.

The lower bound is dictated by the number of possible misclassifications within a given subtree, which is calculated by formulating the subtree as a max flow problem, and assuming that all samples that can be moved in and out of the subtree are only moved within the subtree.

$$\alpha = M(\tau, D)$$

The upper bound assumes that all samples that can be moved outside will cause misclassifications outside the subtree, hence it follows that the upper bound can be calculated formulating the subtree as a max flow problem, only considering samples that cannot be moved outside of the subtree, and adding the amount of samples that can be moved outside the subtree and the max flow together.

$$\beta = M(\tau, D \setminus D_a) + |D_a|$$

The function merge combines all possible pairs of subtrees and returns that set. The combining operator $\oplus$, used in $merge$, can be described as follows: $\tau_i \oplus \tau_j = (F, \tau_i, \tau_j, M(\tau_i \oplus \tau_j, D \cup D_a), M(\tau_i \oplus \tau_j, D) + |D_a|)$, where $M(\tau_i \oplus \tau_j, D)$ returns the misclassifications of the constructed decision tree. The function $candidates$, further refines the process by retaining all subtrees with a strictly lower lower-bound than the lowest upper bound of all possible subtrees, where the comparison operator $\succ$ is defined as $\alpha_i < \beta_j$ where α_i is the lower bound of a tree and β_j is the upper bound of the other tree.

Optimality

In order to prove optimality, we will first prove that the task is separable and then conclude that it is optimal, as we refer to Theorem 4.8 in the STreeD paper, which states that if the optimization task is separable, it can be solved using Eq. (1). To prove separability, we refer to Theorem 4.7, which asserts that an optimization task is separable based on three conditions. First condition: The cost function g is independent. Second condition: the constraint c is anti-monotonic. Third condition: the combining operator $\oplus$ preserves order. We will also refer to three definitions from the STreeD paper, relevant to the three conditions. The first condition, relies on definition 4.3, stating that a cost function g is independent if it relies purely on the label assignment $\hat{k}$ and the parents' branching decisions F. Definition 4.6 defines a constraint as anti-monotonic, if the constraint is violated in a subtree, it should also be violated in the parent tree. Finally, there is definition 4.5, which asserts that a combining operator preserves order, if an arbitrary subtree s_1 dominates another subtree s_1', then it follows that if both subtrees are merged with another subtree s_2, $s_1 \oplus s_2$ should dominate $s_1' \oplus s_2$.

We can show that the first condition is satisfied, by observing that the cost function for leaf nodes in our recurrence equation is $min(|D^+|, |D^-|)$ for the lower bound and $min(|D^+ \setminus D_a|, |D^- \setminus D_a|) + |D_a|$ for the upper bound. The only inputs for the cost function are D and D_a (a subset of D), which is influenced purely by the parent's branching decisions and hence is independent according to definition 4.3. The second condition is also trivially satisfied according to definition 4.6, as the constraint c allows all trees, which means there cannot be any violations in any tree, and therefore it follows that no constraint is violated in a subtree that is not violated in a parent tree, as no constraints are violated at all.

Finally, to prove that the combining operator preserves order, we will describe definition 4.5 with some mathematical notation. Note that domination is described by the comparison operator $\succ$, which in the context of $\tau_i \succ \tau_j$ is defined as $\alpha_i < \beta_j$. Here, α_i refers to the lower bound of τ_i and β_j refers to the upper bound of τ_j. Given three arbitrary trees: $\tau_1 \in candidates(T_1)$, $\tau_1' \in T_1$, and $\tau_2 \in T_2$, we need to prove $\tau_1 \oplus \tau_2 \succ \tau_1' \oplus \tau_2$, assuming that $\tau_1 \succ \tau_1'$.

In order to prove this we first recall that

$$\alpha = M(\tau, D)$$

and

$$\beta = M(\tau, D \setminus D_a) + |D_a|$$

. We then consider the scenario where the adversary moves all the samples it is allowed to move, into the left subtree, in this case the misclassifications in the tree would be equal to: $M(\tau_l, D_f \cup D_{f_a}) + M(\tau_r, D_{\bar{f}} \setminus D_a) = \alpha_r + \beta_r - |D_a|$. Since we know that the adversary is capable of causing at least that many misclassifications in the tree, we can affirm that the lower bound of the tree must be at least that amount. Consequently: $\alpha \geq \alpha_l + \beta_r - |D_a|$. Now consider the scenario in which all samples that the adversary can move in between the left and right subtrees, as well as outside the tree, are counted as misclassifications, formulated as $M(\tau_r, D_f \setminus D_a) + M(\tau_r, D_{\bar{f}} \setminus D_a) + |D_a| = \beta_l + \beta_r - |D_a|$. In this scenario, we can guarantee that the adversary cannot cause anymore misclassifications, as all samples that could be misclassifications are counted as misclassifications. Hence, it follows that $\beta < \beta_l + \beta_r - |D_a|$.

Now let us substitute α with $\alpha(\tau_1' \oplus \tau_2)$ and β with $\beta(\tau_1 \oplus \tau_2)$. The resulting equations are $\alpha(\tau_1' \oplus \tau_2) \geq \alpha_1' + \beta_2 - |D_a|$ and $\beta_1 + \beta_2 - |D_a| \geq \beta(\tau_1 \oplus \tau_2)$. Because we know that $\alpha_1' > \beta_1$ by definition, we can combine the expressions into one: $\alpha(\tau_1' \oplus \tau_2) \geq \alpha_1' + \beta_2 - |D_a| > \beta_1 + \beta_2 - |D_a| \geq \beta(\tau_1 \oplus \tau_2)$. Finally, applying the transitive property of inequalities, we can conclude that $\alpha(\tau_1' \oplus \tau_2) > \beta(\tau_1 \oplus \tau_2)$, and hence $\tau_1 \oplus \tau_2 \succ \tau_1' \oplus \tau_2$.

Since the task has been shown as separable under the three conditions of Theorem 4.7, and because the recurrence relation we presented is a narrowed-down version of Eq. (1), we can conclude, according to Theorem 4.8, that the algorithm returns optimal trees.

5.3 Brute force algorithm

In order to validate the results of the algorithm above, as well as provide a benchmark algorithm, we developed a brute-force algorithm that recursively generates all possible trees, and uses the same adversary model as described above. The brute-force algorithm itself might be competitive with state-of-the-art solutions, as it takes advantage of the recursive nature of decision trees, pruning feature splits that are already present in parent nodes, however, it does not make use of memoization like the dynamic programming approach.

We also developed another brute-force algorithm uses a the conventional way of modelling the adversary, where leaf labels are predetermined, and total misclassifications correspond to the number of samples that could be moved to a leaf node bearing the opposing label.

6 Experimental Setup and Results

In this research, we sought to demonstrate the scalability of a given algorithm under various conditions, controlling for a range of variables. The primary objective of our experiments

was to determine how the algorithm's performance, measured in runtime, scales in relation to specific controlled variables. These variables encompassed the max depth, the number of features, the number of instances, and the proportion of cells that an adversary was permitted to manipulate. To conduct a thorough analysis, we executed the algorithm for every possible combination of these variables, generating a comprehensive dataset for further investigation.

To understand the specific influence of each variable, we performed an analysis in which all but one variable were held constant. This methodology enabled us to isolate the impact of each variable on the algorithm's scalability. For instance, we assessed how the algorithm's runtime scales with the number of features, at a constant depth, the number of instances, and the number of cells the adversary is allowed to manipulate.

By isolating each variable, we were able to derive specific insights into the individual and collective impact of these variables on the algorithm's performance. The outcomes of this research provide a detailed understanding of the algorithm's scalability and potential performance under varying conditions, showing significant improvements in terms of scalability, when compared to the brute-force approach. For clarity, when we refer to the brute-force approach, we are refering to the approach that uses our proposed adversary model.
We carried out experiments on 18 datasets which were provided to us by another student, with 1800 variations for each dataset, on an HP ZBook Studio x360 G5, with 2.60Ghz processing power, of which approximately 15% of the CPU was used at a time. A timeout was set to 5 minutes due to the large number of results that needed to be generated.

6.1 Results

The results contain sections that investigate the scalability of the algorithm when isolating for certain conditions. Conditions include max depth (which ranges from 1 to 5), number of instances in the dataset (ranging from 100 to 1000), number of features (ranging from 10 to 100), and the attack power, which specificies the proportion of cells that the adversary is allowed to manipulate (ranging from 0.005 to 0.05). The runtime is displayed logarithmically. There is also a section that investigates the general scalability of the algorithm compared to a brute-force approach. For the sections investigating runtime when isolating certain conditions, the ranges of conditions are sometimes cut short, as the results are dominated by timeouts. For example, most results with a max depth of 4 or larger take longer than 5 minutes; therefore, no results containing analysis of the runtime itself for depth 4 or higher are shown; instead, results on the percentage of timeouts are presented.

Overall scalability of the algorithm

To evaluate the holistic scalability of the algorithm, we sorted the results on runtime for both the brute-force algorithm, and the dynamic algorithm. Then we computed the cumulative runtime and percentage of searches covered, for both algorithms. The results are presented in the following figures 5 and 6.

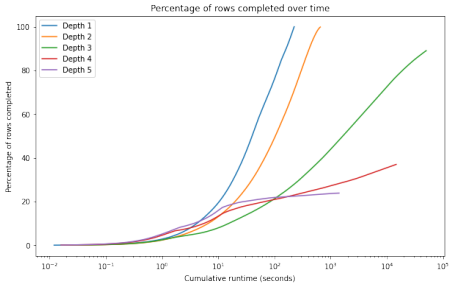

Figure 5: Percentage of optimal decision trees generated. Results were sorted on runtime first, then the cumulative runtime was calculated.

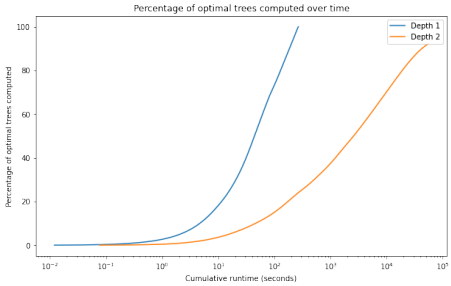

Figure 6: Percentage of optimal decision trees generated. Results were sorted on runtime first, then the cumulative runtime was calculated.

As shown in Figure 5, all optimal decision trees with a max depth equal to 1 and 2, were completed in less than five minutes. For trees with a max depth of 3, the algorithm found the majority of decision trees within the five-minute time frame. Contrast this with the results of the brute-force algorithm in figure 6, where all searches for optimal trees with a max depth of 3 or greater timed out, and even a few searches for max-depth 2 trees timed out as well. It is clear that the dynamic programming algorithm scales far better than the baseline.

Runtime gain

To demonstrate the added value our algorithm provides, we measured the runtime gain of the proposed algorithm over the brute-force algorithm. The runtime gain is the runtime of the brute-force algorithm divided by the runtime of the proposed algorithm.

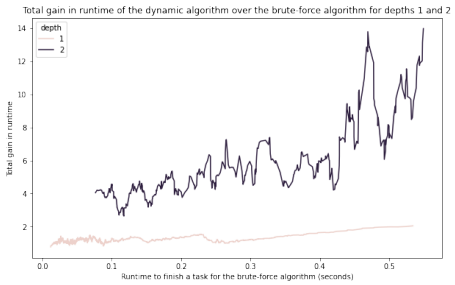

Figure 7: Runtime gain for trees of depth 1 and 2.

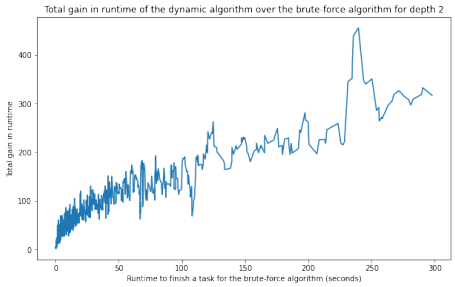

Figure 8: Runtime gain for trees of depth 2.

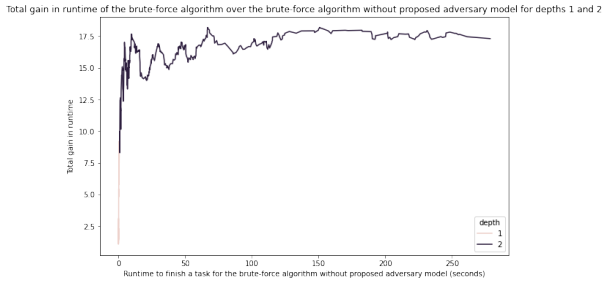

Figure 9: Runtime gain between brute force algorithm (which uses our proposed adversary model), and brute-force algorithm that uses the conventional adversary approach; for trees of depth 1 and 2.

Figures 7 and 8 show the runtime gains for trees of depth 1 and 2. We notice that the runtime gain between trees of depth 1 and 2 in figure 7, differ significantly. The reason behind this lies in the fact that there is very little pruning the model can do, when limited to only depth 1 trees, compared to depth 2 trees. This comparison is significant, when speculating on what the runtime gain might be for searches limited to trees of max depth 3. If depth 2 trees allow for significantly more pruning than depth 1 trees, the same might hold true for depth 3 trees when compared to depth 2 trees.

Figure 9 illustrates the benefits of our proposed adversary model. The figure depicts the runtime gain of the brute-force algorithm, with a brute-force algorithm that uses the conventional adversary model, where leaf labels are predetermined, and total misclassifications correspond to the number of samples that could be moved to a leaf node bearing the opposing label.

Note that the increase in runtime for depth 1 trees varies between a factor of 2 and 8, while for depth 2 trees, the runtime increase is approximately 16. These observations bolster our claim of an exponential speedup relative to the number of leaves in a tree. This is evidenced by the fact that depth 1 trees have two leaves and hence four potential leaf label combinations, and a fully matured depth 2 trees with four leaves contain 16 such combinations. Both corresponding with the runtime gain.

Runtime increase with attack power

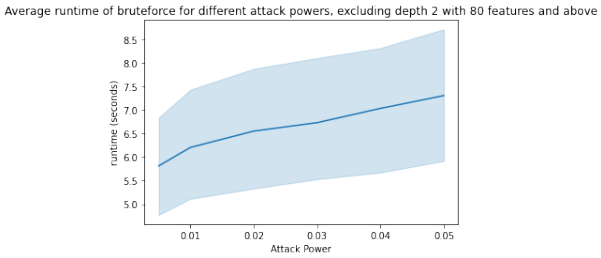

Figure 10: Average runtime for the brute-force algorithm iwth varying attack power. Only includes results for trees of depth 1. As well as results for trees of depth 2 with maximally 70 features, as there are timeouts in the other results

Further insights are garnered from Figure 10 that shows the runtime increases marginally with attack power for the brute-force algorithm. This is evidence that the push-relabel algorithm scales well with the type of network described in Figure 4. In theory the push-relabel algorithm, scales polynomially with the amount of vertices, however, in practice it is known to be very efficient.

7 Responsible Research

The methodology and experimental design of this study adhered strictly to the tenets of FAIR principles. The FAIR principles are a set of guidelines designed to enhance the usability of data. They encompass four core tenets detailed below, alongside the ways in which they were implemented in this study:

- Findable: We have data readily accessible in the same repository as the source code, ensuring straightforward replication of the experiments detailed in our paper.
- Accessible: There is no requirement for special authentication to access the data.
- Interoperable: We've stored our data in the widely-used CSV format.
- Reusable: The code contains instructions to run the code.

By adhering to these guidelines, we aim to increase the potential usability of the data garnered in our study. This, we hope, will promote further experimentation and enhance our collective understanding of algorithms and NP-Hard problems.

It is also relevant to acknowledge that LLM's like chatGPT were used to improve the quality of the writing, help debug some of the code, and write a few helper functions for further debugging.

8 Conclusions and Future Work

In conclusion, we present to you an algorithm, that treats the task of finding optimal robust decision trees, as separable optimization tasks, as well as a max-flow formulation for the adversary, that can be efficiently computed using the push-relable algorith, and offers exponential speedup with respect to the number of leaf nodes in a tree. The algorithm scales comfortably up to depth 3 trees, and can feasibly compute a considerable amount of optimal robust trees with a maximum depth of up to 5. In contrast with the brute-force approach which does not scale past depth 2.

8.1 Limitations and possible improvements

This study, while advancing the field, acknowledges certain limitations. The current research exclusively assumes binary data and binary labels and confines its focus to robust decision trees in the context of minimizing misclassifications.

Another limitation of this paper is the difficulty of comparing this solution with other state-of-the-art methods such as ROCT, which assumes only continuous data and, hence, cannot be directly compared.

8.2 Future research

Despite these limitations, there is ample scope for future research. For example, this work could be extended to consider continuous and discrete data. This would allow for more direct comparisons with different state-of-the-art methods that deal with continuous data. Additionally, considering other definitions of adversaries could provide a richer understanding of how the method performs under various threat models.

Furthermore, research could be extended to optimizing robust decision trees on other metrics such as MSE, F-score, survival analysis.

Finally, the definitions of adversaries in the current work and other state-of-the-art methods differ. While in essence both definitions allow adversaries to move samples to different leaf nodes, the rules that determine whether a sample can be moved to a leaf node, differ. Future work in this context could include translating the adversaries in other works to the adversary we defined, when binarizing the data.

References

[1] Push-relabel algorithm — set 2 (implementation), Mar 2023.

[2] R. K. Ahuja and James B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[3] Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: training evasion-aware decision trees. *Data Mining and Knowledge Discovery*, 34(5):1390–1420, Sep 2020.

[4] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 99–108, New York, NY, USA, 2004. Association for Computing Machinery.

[5] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *J. Mach. Learn. Res.*, 23(1), jan 2022.

[6] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, page 43–58, New York, NY, USA, 2011. Association for Computing Machinery.

[7] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 641–647, New York, NY, USA, 2005. Association for Computing Machinery.

[8] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 372–387, 2016.

[9] Jacobus G. M. Van der Linden, Mathijs M. De Weerdt, and Emir Demirovic. *Optimal Decision Trees for Separable Objectives: Pushing the Limits of Dynamic Programming*, May 2023.

[10] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10586–10595. PMLR, 18–24 Jul 2021.

[11] Daniël Vos and Sicco Verwer. Robust optimal classification trees against adversarial examples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8520–8528, Jun. 2022.

[12] Daniël Vos and Sicco Verwer. Adversarially robust decision tree relabeling. In Massih-Reza Amini, Stéphane Canu, Asja Fischer, Tias Guns, Petra Kralj Novak, and Grigorios Tsoumakas, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 203–218, Cham, 2023. Springer Nature Switzerland.