Fair and Optimal Decision Trees: A Dynamic Programming Approach

Jacobus G.M. van der Linden **Mathijs M. de Weerdt** **Emir Demirović**
Delft University of Technology, Department of Computer Science
{J.G.M.vanderLinden, M.M.deWeerdt, E.Demirovic}@tudelft.nl

Abstract

Interpretable and fair machine learning models are required for many applications, such as credit assessment and in criminal justice. Decision trees offer this interpretability, especially when they are small. Optimal decision trees are of particular interest because they offer the best performance possible for a given size. However, state-of-the-art algorithms for fair and optimal decision trees have scalability issues, often requiring several hours to find such trees even for small datasets. Previous research has shown that dynamic programming (DP) performs well for optimizing decision trees because it can exploit the tree structure. However, adding a global fairness constraint to a DP approach is not straightforward, because the global constraint violates the condition that subproblems should be independent. We show how such a constraint can be incorporated by introducing upper and lower bounds on final fairness values for partial solutions of subproblems, which enables early comparison and pruning. Our results show that our model can find fair and optimal trees several orders of magnitude faster than previous methods, and now also for larger datasets that were previously beyond reach. Moreover, we show that with this substantial improvement our method can find the full Pareto front in the trade-off between accuracy and fairness.

1 Introduction

As machine learning (ML) is used in more domains that involve discrimination-sensitive decisions, demand for *fair*, *interpretable* and *accurate* models increases. ML, for example, is used in criminal justice [8], credit assessment [28], and housing appointments [7], each of which requires fair decisions. Simply removing the discrimination-sensitive attribute from the dataset does not necessarily result in less discrimination [33]. Instead, fairness is obtained by adding a constraint on the classifier: often either an individual or a group fairness constraint (see [10, 20] for a comparison of the two). The focus in this paper is on *group fairness*. With group fairness, a binary classifier is considered fair if the probability of receiving the positive outcome is the same for all protected groups. This fairness metric is formulated as a global constraint over the full dataset.

To increase confidence and prevent errors in these sensitive-decision domains, it also important that the ML models are interpretable [39]. Decision trees offer inherent interpretability and do not carry some of the risks of post-hoc attempts at explaining black-box models [36]. This is especially the case when decision trees are small [34]. On average, *optimal* decision trees, i.e., trees that for a given size provably maximize an objective such as accuracy, provide better out-of-sample performance than trees generated by heuristics when comparing trees of equal size [9, 15]. Therefore small optimal decision trees are ideal when searching for accurate and interpretable models.

There are several approaches for finding fair and optimal decision trees [2, 3, 25, 40], all of which use mixed integer programming (MIP). However, none of these models scale well. If fairness is not considered, small optimal decision trees can be found efficiently by using dynamic programming (DP)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

methods that exploit the separability present in the tree structure [4, 5, 15, 23, 31]. Demirović and Stuckey [14] also use DP to find the Pareto front of optimal decision trees for biobjective nonlinear metrics.

However, a global fairness constraint cannot trivially be added to these algorithms, because the fairness constraint breaks the separability assumption by introducing dependency between subproblems. When optimizing for accuracy, for example, left and right subtrees can be solved independently after choosing what feature to branch on in the current decision node. In contrast, when optimizing for fairness, the left and right subtree cannot be optimized independently, since imbalance in one part of the tree can be cancelled out in another part of the tree.

Motivated by the success of DP methods for conventional optimal trees, we developed a novel DP method that successfully incorporates a global fairness constraint. We solve the subproblem dependency by computing upper and lower bounds on the final fairness values of partial solutions, thus enabling early pruning of the search space. Moreover, we present algorithmic improvements in the merging and comparing of subproblems, and perform a study of scalability. Overall our approach provides orders of magnitude improvements over state-of-the-art MIP methods.

This has several benefits. First, our algorithm can find fair and optimal decision trees for datasets that were previously too large to consider, often even within seconds. Second, it can find deeper and more accurate trees, while still satisfying the fairness constraint. Last, our algorithm can find the full Pareto-front in the trade-off between accuracy and fairness. This enables the responsible domain expert to make a well-informed decision and select a policy from the solution set that fits the context and application best.

As is commonly done in optimal decision tree search, we assume that all features have been binarized in advance [12]. Similarly, it is assumed that the label of every instance in the dataset is binary: either the positive/preferred outcome or the negative outcome. This is a common assumption in fair classification [16] and was also assumed in previous fair and optimal decision tree methods [25, 40].

The paper setup is as follows: we first discuss related work and preliminaries, and then we present our method. Finally, we evaluate the performance of the new method on twelve datasets from different application domains and we compare its performance to a leading MIP method from the literature.

2 Related work

This section discusses, in order, the related work of 1) algorithms for finding optimal decision trees, 2) MIP models for finding fair optimal decision trees, and finally 3) other (non-optimal) methods for finding fair decision trees.

Algorithms for Optimal Decision Trees. The search for optimal decision trees is an NP-Hard problem [24]. Therefore decision tree search was traditionally done by means of heuristics such as the much-used CART algorithm [11]. Because of increasing computation power, searching for optimal decision trees has become feasible and Bertsimas and Dunn [9] showed that optimal decision trees can offer 1-5% better out-of-sample accuracy scores than previous heuristics. Their MIP model was consecutively improved by several others [41, 42].

During the same period, custom search and bound-based approaches, using techniques such as caching and branching, for finding (binary) classification trees were developed. These methods often outperformed the MIP models in terms of runtime and scalability, sometimes reducing runtimes of several hours to several seconds, by utilizing dynamic programming in a depth-first [4, 5, 15] or breadth-first approach [23, 31].

Of particular interest for this paper is the method presented in [14] which extends on these methods and finds decision trees for biobjective optimisation, minimizing the misclassification score for both classes in binary classification at the same time and returning the Pareto front of nondominated solutions. This method is further discussed in the preliminaries.

Fairness in Optimal Decision Trees. Currently, the only methods for finding fair and optimal decision trees are MIP-based. Verwer and Zhang [40] developed a MIP model with a soft group fairness constraint in the objective. They report how their method can find a fair depth-three tree for a

dataset of 1000 instances in 15 minutes, when the model is given an initial solution from CART. But for the fair tree case they do not report being able to prove optimality within this time limit.

Aghaei et al. [2] present a MIP model that co-optimizes fairness and error for both classification and regression. Apart from group fairness, they also consider a notion of individual fairness separately. However, they are criticized by Ranzato et al. [35] that their model often provides accuracy not much better than a constant classifier. In [3] and [25], they continue their work and present a MIP model based on a max-flow formulation that results in a considerable improvement of the runtime. However, these models still have scalability issues because of the large number of binary variables.

Other methods for Fair Decision Trees. For completeness the following section mentions a number of related fair decision tree methods that do not guarantee a globally optimal solution. These can be divided into pre-, in-, and postprocessing methods.

Kamiran and Calders [26] present a preprocessing method that changes the labels in the dataset according to a group fairness metric. The updated dataset can then be used to train any regular classifier.

Some of the in-processing methods consider fairness in the classifier itself. Others use a regular classifier, combined with a meta- or master problem that takes care of the fairness constraint. Pedreschi et al. [33], for example, develop a method for finding fair association rules that can then be used to compile decision trees. Agarwal et al. [1] iteratively solve a number of cost-sensitive classification problems, with the costs determined by the number of violations of the fairness constraint. Grari et al. [21] propose a gradient tree boosting approach with an adversarial fairness constraint. Detassis et al. [17] use a MIP master problem that changes the labels for the next training iteration to satisfy the fairness constraint. There is, however, no guarantee of convergence. Valdivia et al. [38] developed an evolutionary algorithm that returns a Pareto front of decision trees with the multiobjective of both fairness and accuracy.

Kamiran et al. [27] test a CART-based approach with a split criterion that is a mix of information gain in the label and the sensitive feature. They did not observe a significant improvement when using this approach. As an alternative, they suggest a postprocessing relabeling method that uses a greedy knapsack algorithm to decide which leaf nodes to relabel to improve fairness at a minimum cost of accuracy.

Summary of related work. In summary, the current state-of-the-art has scalable custom methods for finding optimal conventional decision trees, but adding fairness constraints to these methods is non-trivial. MIP-based solutions exist for this problem, but they do not scale well. The alternative is to use heuristic methods, which result in suboptimal solutions.

3 Preliminaries

This section discusses the preliminaries of our method: an introduction of notation, a formal introduction of group fairness, a formal problem definition, and a dynamic programming formulation for optimal decision tree search that does not consider fairness.

Notation. A dataset $\mathcal{D}$ contains instances $(\mathbf{x}, a, y)$, with $\mathbf{x}$ the feature vector, a the sensitive attribute and y the label value. The number of instances in the dataset $\mathcal{D}$ is $N = |\mathcal{D}|$. Let y_i be the label of instance i and a_i the value of the protected feature. Let $\mathcal{F}$ be the feature set and $f \in \mathcal{F}$ a feature in the feature set. We use f_i to denote the value of this feature for instance i. Since we are only considering binary features and binary classification, we introduce the shorter notation f_i and $\bar{f}_i$ to mean respectively that instance i satisfies or does not satisfy feature f. The same idea applies for y and $\bar{y}$, and a and $\bar{a}$. We use subscript notation to refer to a subgroup of the dataset: for example, $\mathcal{D}_{\bar{f}}$ refers to all instances in dataset $\mathcal{D}$ that do not satisfy feature f; N_a is the number of instances in the dataset that belong to group a, and $N_{y,\bar{a}}$ is the number of instances in the dataset with positive outcome that does not belong to group a.

Group Fairness Formalized. The specific group fairness definition considered in this paper is *demographic parity*, although our method can easily be extended to support other notions of group fairness, such as equality of opportunity. Demographic parity requires that the performance (the

expected percentage of positive outcomes) per group is the same [20]. More formally, if $\hat{y}$ is the predicted outcome, 1 is the positive outcome, and a represents a (binary) sensitive/protected group, then demographic parity holds iff:

$$P(\hat{y} = 1 \mid a = 1) = P(\hat{y} = 1 \mid a = 0) \tag{1}$$

In binary classification, this constraint can be measured by considering the relative average performance of the two groups. Let $\hat{p}$ be the percentage of instances that receive the positive outcome, so $\hat{p}_a = N_{\hat{y},a}/N_a$. The difference in the performance between two groups is what we will call the imbalance $I = \hat{p}_a - \hat{p}_{\bar{a}}$. When the imbalance is limited to some value δ, we say that it satisfies demographic parity up to bias δ:

$$|I| = |\hat{p}_a - \hat{p}_{\bar{a}}| \leq \delta \tag{2}$$

According to the definition of demographic parity, if a leaf node n receives label 0, the partial imbalance I_n in that node is also 0. Otherwise, the partial imbalance in a leaf node n can be expressed as follows:

$$I_n = \frac{N_{n,a}}{N_a} - \frac{N_{n,\bar{a}}}{N_{\bar{a}}} \tag{3}$$

The partial imbalance of a branching node is the sum of the imbalance of both sub-nodes.

Problem Definition. The task of learning an optimal fair decision tree is to find the feature value tests in the branch nodes and the leaf node assignments for a tree of a given maximum size that minimize the number of misclassifications in a training data set while observing a fairness constraint. The percentage of correctly classified instances is called accuracy. This can be formalized as follows. The task is to find a decision tree classifier $h : \{0,1\}^{|\mathcal{F}|} \to \{0,1\}$ of depth d that minimizes the misclassification score for a given dataset $\mathcal{D}$ while observing a fairness constraint up to bias δ:

$$\begin{aligned} &\min_h \sum_{(\mathbf{x},a,y)\in\mathcal{D}} |h(\mathbf{x}) - y| \\ &\text{s.t.} \left| \sum_{(\mathbf{x},a,y)\in\mathcal{D}_a} \frac{h(\mathbf{x})}{N_a} - \sum_{(\mathbf{x},a,y)\in\mathcal{D}_{\bar{a}}} \frac{h(\mathbf{x})}{N_{\bar{a}}} \right| \leq \delta \end{aligned} \tag{4}$$

Alternatively, when searching for the full Pareto front, the parameter δ is not necessary, and the problem can be written as a multiobjective problem:

$$\min_h \left\{ \sum_{(\mathbf{x},a,y)\in\mathcal{D}} |h(\mathbf{x}) - y|, \left| \sum_{(\mathbf{x},a,y)\in\mathcal{D}_a} \frac{h(\mathbf{x})}{N_a} - \sum_{(\mathbf{x},a,y)\in\mathcal{D}_{\bar{a}}} \frac{h(\mathbf{x})}{N_{\bar{a}}} \right| \right\} \tag{5}$$

The Pareto front for Eq. 5 consists of all nondominated solutions (M, I) with M the misclassification score and I the imbalance. A solution is considered dominated if there is another solution that performs better or similar on all objectives. Therefore, we can define the dominance relation $(\succ)$ for two solutions (M_1, I_1) and (M_2, I_2), and the function nondom as follows:

$$(M_1, I_1) \succ (M_2, I_2) \text{ iff } M_1 \leq M_2 \wedge |I_1| \leq |I_2| \wedge (M_1, I_1) \neq (M_2, I_2) \tag{6}$$

$$\mathrm{nondom}(S) = \{s_1 \in S \mid \neg\exists\, s_2 \in S, s_2 \succ s_1\} \tag{7}$$

Dynamic Programming Formulation for Decision Trees. The problem of finding accurate decision trees has been formulated as a DP problem before in [15]. Of interest to this paper is the formulation for biobjective optimization in [14]. They minimize the misclassification score of two classes at the same time and return the Pareto front of nondominated solutions:

$$T_{\mathrm{BI}}(\mathcal{D}, d) = \begin{cases} \{(0, |\mathcal{D}_y|), (|\mathcal{D}_{\bar{y}}|, 0)\} & d = 0 \\ \mathrm{nondom}\left(\cup_{f\in\mathcal{F}} \mathrm{merge}\left(T_{\mathrm{BI}}(\mathcal{D}_f, d-1), T_{\mathrm{BI}}(\mathcal{D}_{\bar{f}}, d-1)\right)\right) & \text{else} \end{cases} \tag{8}$$

At the leaf node $(d = 0)$, the function returns two solutions consisting of two values: the misclassification score per class (e.g., $(0, |D_y|)$ when label $\hat{y} = 0$ is selected, because all instances with $y = 0$ will be correctly classified and all instances with $y = 1$ will cause $|D_y|$ misclassifications for class 1). At a branching node $(d > 0)$, the function should return the merge of two solution sets and retain only the nondominated solutions resulting from that merge. The function merge combines two sets of partial solutions (from the left and right subtree) by element-wise addition:

$$\mathrm{merge}(S_1, S_2) = \{(a_1 + a_2, b_1 + b_2) \mid (a_1, b_1) \in S_1, (a_2, b_2) \in S_2\} \tag{9}$$

4 Method Description

In this section, we present our method DPF (DP Fair) and show how to deal with a global fairness constraint that introduces subproblem dependency. Similar to Eq. 8, we define a recursive function T_{F} that returns a set of all nondominated (partial) solutions, resulting in a full Pareto front of fair and optimal decision trees. In order to guarantee that the method returns the full Pareto front we must do an exhaustive search of all combinations of subtrees, pruning only those solutions for which we can prove that they will never be part of an optimal (nondominated) solution.

Dependency. The recursive function 8 introduced in the preliminaries assumes that the left and the right tree can be independently optimized, but this is not the case when considering group fairness, which is a global constraint on the whole tree. When comparing two possible subtrees, one of which discriminates against group A and the other discriminating against group B, it is not clear which one is better, because it will depend on what happens in the rest of the tree. Even when both subtrees discriminate against the same group A, it is not immediately clear that the one with lower discrimination score will be better, but again it will depend on what happens in the rest of the tree.

More formally, the previous proposed biobjective method requires objectives to be *additive* and *monotonic* in order to create independent subproblem. The additive property holds if two partial solutions (a, b) and (a', b') can be combined by addition: $(a + a', b + b')$. The monotonic property holds for a biobjective function $f(a, b)$ if for any two possible different inputs a, b and a', b', the following dominance relation holds:

$$a \leq a' \wedge b \leq b' \rightarrow f(a, b) \succ f(a', b') \tag{10}$$

However, because fairness is measured by the absolute value of the imbalance, both the additive property and the monotonic property are not satisfied. Our method addresses this problem.

Upper and lower bounds for fairness. The key idea in our method is to compute upper and lower bounds for the imbalance value of partial solutions to enable comparison. When upper and lower bounds $\bar{I}_R$ and $\underline{I}_R$ are known for the imbalance in the rest (R) of the tree, then the final imbalance value of a tree that contains node n will be in the range: $[\underline{I}_R + I_n, \bar{I}_R + I_n]$. The lower bound ($\underline{I}$) and the upper bound ($\bar{I}$) for the final *absolute* imbalance value can now be computed:

$$\underline{I}(I_n, [\underline{I}_R, \bar{I}_R]) = \begin{cases} 0 & \text{if } 0 \in [\underline{I}_R + I_n, \bar{I}_R + I_n] \\ \min\left(|\underline{I}_R + I_n|, |\bar{I}_R + I_n|\right) & \text{else} \end{cases} \tag{11}$$

$$\bar{I}(I_n, [\underline{I}_R, \bar{I}_R]) = \max\left(|\underline{I}_R + I_n|, |\bar{I}_R + I_n|\right) \tag{12}$$

In Eq. 11, the lower bound is zero if the permitted range contains zero; otherwise it is the minimum absolute value in that range. In Eq. 12, the upper bound is the maximum absolute value in the permitted range.

With these upper and lower bounds on the final absolute imbalance value, we can redefine the dominance relation of Eq. 6:

$$\begin{aligned} (M_1, I_1) \succ (M_2, I_2)) \text{ iff } & (M_1, I_1) \neq (M_2, I_2) \\ & \wedge\ M_1 \leq M_2 \\ & \wedge\ \left(\bar{I}(I_1, [\underline{I}_R, \bar{I}_R]) \leq \underline{I}(I_2, [\underline{I}_R, \bar{I}_R])\ \vee\ I_1 = I_2\right) \end{aligned} \tag{13}$$

A solution is dominated by another solution if 1) the other solution has a lower misclassification score; and 2) either has an upper bound on the absolute imbalance lower than the lower bound on the absolute imbalance of this solution, or has precisely the same imbalance score.

Finding the Pareto front of optimal fair trees. Now we can define a new function for finding fair trees $T_F(\mathcal{D}, d, [\underline{I}_R, \bar{I}_R])$ that optimizes Eq. 5:

$$T_{\mathrm{F}}(\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]) = \begin{cases} \mathrm{leaf}(\mathcal{D}, [\underline{I}_R, \bar{I}_R]) & d = 0 \\ \mathrm{branch}(\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]) & \text{else} \end{cases} \tag{14}$$

The functions leaf and branch will be introduced next. Both use the nondom function, which is now based on our new dominance comparison as described in Eq. 13 (but for ease of notation, $[\underline{I}_R, \bar{I}_R]$ will often be left out).

In a leaf node n, the function should return the following (M, I) solutions, with imbalance values based on Eq. 3:

$$\text{leaf}(\mathcal{D}, [\underline{I}_R, \bar{I}_R]) = \text{nondom}\left(\left\{(|\mathcal{D}_y|, 0), \left(|\mathcal{D}_{\bar{y}}|, \frac{N_{n,a}}{N_a} - \frac{N_{n,\bar{a}}}{N_{\bar{a}}}\right)\right\}\right) \tag{15}$$

Eq. 15 returns up to two solutions, each described by two values: first the misclassification score as before, and second the imbalance when assigning the positive outcome.

In a branching node, the data set is split on some feature f, resulting in two subproblems. Each subproblem is solved independently by passing the best known bounds for the other subproblem. Let $U(\mathcal{D}_{\bar{f}}, [\underline{I}_R, \bar{I}_R], d-1)$ denote the function that returns these best known bounds. These bounds can be based on dataset inspection, or based on previous (cached) solutions. Bounds from inspecting the dataset can be derived by assigning one label to all instances of one group and the other label to all other instances and vice versa. The results provide the maximum discrimination that could possibly happen. When (cached) partial solutions for the rest of the tree are known, the bounds can be derived by taking the minimum and maximum imbalance values among solutions. With these bounds, the solutions from the two subproblems are then merged as follows:

$$\begin{aligned}\text{branch}(\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]) = \text{nondom}\,(\cup_{f\in\mathcal{F}} \text{merge}\,(&\\ T_F(\mathcal{D}_{\bar{f}}, d-1, \quad U(\mathcal{D}_f, [\underline{I}_R, \bar{I}_R], d-1)),&\\ T_F(\mathcal{D}_f, d-1, \quad U(\mathcal{D}_{\bar{f}}, [\underline{I}_R, \bar{I}_R], d-1)))&)\end{aligned} \tag{16}$$

Finding a single best tree. It is also possible with this same DP formulation to search for a single optimal and fair tree up to bias δ, as defined in Eq. 4. This can be achieved by pruning all (partial) solutions and retaining only those that are *possibly* fair. Then, in the root node of the search, the tree with minimum misclassification score is selected from the list of solutions. A solution is possibly fair if the lower bound on the absolute imbalance value is less than or equal to δ. For this, we introduce the function prune, which should filter a set of solutions S before it is passed to the nondom function.

$$\text{prune}(S, [\underline{I}_R, \bar{I}_R]) = \left\{(M_n, I_n) \mid (M_n, I_n) \in S,\ \underline{I}(I_n, [\underline{I}_R, \bar{I}_R]) \leq \delta\right\} \tag{17}$$

Pseudocode. This section gives the pseudocode for DPF. In addition to the description above, the pseudocode also considers upper and lower bounds and cache. Therefore, the function T_{F} receives the current best known upper bound ub on the misclassification score as an extra parameter.

The function prune must be redefined to take into account this upper bound on the misclassification score.

$$\begin{aligned}\text{prune}(S, \text{ub}, [\underline{I}_R, \bar{I}_R]) = \{(M_n, I_n) \mid (M_n, I_n) \in S,&\\ \underline{I}(I_n, [\underline{I}_R, \bar{I}_R]) \leq \delta, M_n < \text{ub}\}&\end{aligned} \tag{18}$$

For shorter notation, define:

$$\text{filter}(S, \text{ub}, [\underline{I}_R, \bar{I}_R]) = \text{nondom}\left(\text{prune}\left(S, \text{ub}, [\underline{I}_R, \bar{I}_R]\right), [\underline{I}_R, \bar{I}_R]\right) \tag{19}$$

Also we define the functions $\underline{M}$ to return the best misclassification score within a set of solutions:

$$\underline{M}(S) = \min\{M \mid (M, I) \in S\} \tag{20}$$

Furthermore, let LB return the best known lower bound on the misclassification score for a subproblem, or zero if no such lower bound is known.

With these changes, see Algorithm 1 for the resulting pseudocode of DPF. In this algorithm S is the set of solutions, and lb and ub are the lower and upper bounds, with subscripts L and R denoting left and right subtrees. For a full Pareto front, the algorithm must be called with a fairness cut-off value of $\delta = 1$.

Other algorithmic improvements. There are a number of other improvements in DPF for reducing the runtime. To reduce the runtime cost of the merge function, before merging, the two sets of partial solutions are first pruned based on bounds on the misclassification score and discrimination score derived from the other set. Then the two sets are sorted by misclassification score to enable early termination of the merge if it can be proven that all consecutive combinations of partial solutions will

Algorithm 1: Tree search of depth d with fairness on a dataset $\mathcal{D}$ for a feature set $\mathcal{F}$.

$T_{\mathrm{F}}(\mathcal{D}, d, \mathrm{ub}, [\underline{I}_R, \bar{I}_R])$
  if $d = 0$ **then**
    $S \leftarrow \left\{(|\mathcal{D}_y|, 0), \left(|\mathcal{D}_{\bar{y}}|, \frac{N_{n,a}}{N_a} - \frac{N_{n,\bar{a}}}{N_{\bar{a}}}\right)\right\}$
    return $\mathrm{filter}(S, \mathrm{ub}, [\underline{I}_R, \bar{I}_R])$
  $\langle S, \mathrm{lb}, \mathrm{stat}\rangle \leftarrow \mathrm{cache}[\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]]$
  if $\mathrm{lb} \geq \mathrm{ub}$ **then return** $\emptyset$
  if $\mathrm{stat} = \mathrm{optimal}$ **then return** $\mathrm{filter}(S, \mathrm{ub}, [\underline{I}_R, \bar{I}_R])$
  $S \leftarrow \emptyset$
  for $f \in \mathcal{F}$ **do**
    $\mathrm{lb}_R \leftarrow \mathrm{LB}(\mathcal{D}_f, d-1, [\underline{I}_R, \bar{I}_R])$
    $S_L \leftarrow T_{\mathrm{F}}(\mathcal{D}_{\overline{f}}, d-1, \mathrm{ub} - \mathrm{lb}_R, U(\mathcal{D}_f, [\underline{I}_R, \bar{I}_R], d-1))$
    if $S_L = \emptyset$ **then continue**
    $\mathrm{lb}_L \leftarrow \mathrm{LB}(\mathcal{D}_{\overline{f}}, d-1, [\underline{I}_R, \bar{I}_R])$
    $S_R \leftarrow T_{\mathrm{F}}(\mathcal{D}_f, d-1, \mathrm{ub} - \mathrm{lb}_L, U(\mathcal{D}_f, [\underline{I}_R, \bar{I}_R], d-1))$
    if $S_R = \emptyset$ **then continue**
    $S \leftarrow S \cup \mathrm{prune}\left(\mathrm{merge}(S_L, S_R, [\underline{I}_R, \bar{I}_R]), \mathrm{ub}, [\underline{I}_R, \bar{I}_R]\right)$
  $S \leftarrow \mathrm{nondom}(S)$
  if $S = \emptyset$ **then**
    $\mathrm{cache}[\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]] \leftarrow \langle \emptyset, \mathrm{ub}, \text{lower bound}\rangle$
    return $\emptyset$
  $\mathrm{cache}[\mathcal{D}, d, [\underline{I}_R, \bar{I}_R]] \leftarrow \langle S, \underline{M}(S), \mathrm{optimal}\rangle$
  return S

exceed the misclassification upper bound. In the root node of the search, the solutions are sorted on the imbalance value to enable faster elimination of non-fair solutions.

The runtime of the nondom function can be reduced by observing that partial solutions with a lower bound imbalance of 0 do not need to be compared with other solutions because they will always be nondominated. For the sake of brevity, all these have been left out of the method description. See our code repository for full implementation details.[1]

Similar to [14, 15] our method also uses a special depth-two solver to increase the runtime performance. Our source code contains the full details. Appendix A presents a complexity analysis of DPF. Appendix B shows how our algorithm can easily be modified to also optimize tree sparsity.

5 Experimental Results

The following section analyzes the performance of DPF. We focus on the analysis of the runtime and scalability of DPF: 1) How does DPF compare to the state-of-the-art in terms of runtime? 2) What impact do our pruning method and merge improvements have on the runtime? 3) What is the runtime cost to find the full Pareto front? 4) What impact do parameters such as number of features, instances, etc. have on the runtime of DPF?

For an out-of-sample analysis, see Appendix D.

Experiment setup. We evaluate DPF on all datasets mentioned in the survey by Le Quy et al. [30]. The data preprocessing and binarization is also done as described in that paper. Categorical variables are encoded through one-hot encoding, except for variables with twenty or more categories, which were removed (this applies to the KDD census dataset). The binarized datasets are included in our code repository (if the license permits redistribution). The Diabetes and Law-school dataset are left out of the evaluation because the best tree of depth three is (almost) identical to the trivial solution (assign the majority label to all instances). See Appendix C for more details.

[1] `https://gitlab.tudelft.nl/jgmvanderlinde/dpf`

The algorithm receives the whole dataset as input and is asked to find a tree with an unfairness limit of $\delta = 1\%$. In a later experiment, we also examine the impact of this parameter on the runtime. The values shown are the average of five runs and a time limit of one hour is set for every run. All experiments are run on a 2.6Ghz Intel i7 CPU with 8GB RAM using only one thread.

The FairOCT model To our knowledge, the only methods available for finding fair and optimal decision trees are MIP models [2, 25, 40]. Verwer and Zhang [40] limit their model to solving problems with less than 1000 instances. In our initial comparison of [2] and [25], the latter outperformed the first by a large margin. Therefore, we compare our method with their FairOCT model. They model the decision tree as a flow graph where all instances must flow from the source (connected to the root node of the tree) to one of the sink nodes (one for each class), to which all nodes are connected. In our experiments, the FairOCT model is solved with Gurobi 9.0 using the default parameters.

Runtime comparison. Table 1 shows the runtime comparison between FairOCT and our method DPF. DPF can find optimal trees for $d = 2$ within one second for all datasets, about four to five orders of magnitude faster than FairOCT. FairOCT is able to find $d = 2$ trees only for the smallest datasets within one hour. It struggles with an increasing number of data instances because every data instance is represented by a binary variable. Also for $d = 3$, DPF can find optimal trees within one second for several datasets and all problems are solved within a minute. For $d = 4$, the exponential nature of the problem becomes clearly visible. Datasets with both large number of features and instances are no longer solvable within one hour. However, DPF still succeeds in finding the best tree for a number of datasets, among which Adult, with over 45000 instances.

The impact of our novel pruning method and the merge improvements Table 2 shows the runtime results when DPF is run 1) without our novel pruning method (not applying the prune and nondom methods of Eqs. 7 and 17; 2) without our improvements to the merge method; 3) without our improvement to the nondom method, and 4) to generate a full Pareto front.

For several datasets, our pruning method decreases the runtime by one or more orders of magnitude. This is specifically the case for depth four, where the subproblem dependency is more significant than for depth three. Our method can find optimal trees for depth four for eight out of twelve datasets within one hour, but without the pruning method, this is possible for only four datasets.

Our improvements to the merge function in some cases significantly reduce the runtime. This is specifically the case for datasets that have a high initial bias (the bias of an optimal tree if fairness would not be considered): Adult, COMPAS recid. and Dutch census have an initial bias of 18%, 16%, and 14% respectively, and for these datasets the merge improvements reduce the runtime by an order of magnitude. The other data sets in Table 2 have an initial bias of 1-6% and the merge improvements result in only a small reduction of the runtime. The improvements to the nondom function are also essential for the performance of our pruning mechanism.

Table 1: Runtime comparison of DPF and FairOCT. Runtime is in seconds. FairOCT resulted in an out of memory error for KDD census income, so no runtime can be reported.

				FairOCT	DPF		
Dataset	a	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$	$d=2$	$d=2$	$d=3$	$d=4$
Adult	Gender	45222	17	> 1h	< 1	< 1	1012
Bank Marketing	Married	45211	46	> 1h	< 1	2	> 1h
Communities & Crime	Race	1994	97	> 1h	< 1	4	1261
COMPAS recid.	Race	6172	9	> 1h	< 1	< 1	9
COMPAS viol. recid.	Race	4020	9	1594	< 1	< 1	< 1
Dutch census	Gender	60420	58	> 1h	< 1	5	> 1h
German credit	Gender	1000	69	> 1h	< 1	2	741
KDD census income	Race	284556	117	-	< 1	32	> 1h
OULAD	Gender	21562	45	> 1h	< 1	3	> 1h
Ricci	Race	118	4	< 1	< 1	< 1	< 1
Student-Mathematics	Gender	395	55	284	< 1	< 1	24
Student-Portuguese	Gender	649	55	866	< 1	< 1	32

Table 2: DPF runtimes without the pruning and dominance checks (¬P), without the merge improvements (¬M), without the nondom improvements (¬D), for finding the full Pareto-front (Pr), and for comparison the default runtime (Df). Datasets for which DPF has a default runtime below one second or over one hour, are left out.

	$d = 3$						$d = 4$				
Dataset	¬P	¬M	¬D	Pr	Df	Dataset	¬P	¬M	¬D	Pr	Df
Bank	5	3	319	4	2	Adult	> 1h	> 1h	> 1h	> 1h	1012
Com.&Cr.	13	5	19	7	4	Com.&Cr.	> 1h	1329	2473	1939	1261
Dutch	> 1h	101	1814	510	5	COMP. r.	> 1h	394	1658	1659	9
German	8	2	22	2	2	German	> 1h	770	1971	738	741
KDD	144	35	1254	50	32	Stud. Math	144	24	29	412	24
OULAD	5	4	1448	3	3	Stud. Port.	1834	32	38	259	32

Generating the full Pareto-front Because of the runtime improvements of DPF, it is also able to find a *full*, fair and optimal decision tree Pareto front. In contrast, Valdivia et al. [38] find a partial and non-optimal Pareto front and the FairOCT model is used to find an optimal, but still partial, Pareto front. Table 2 shows the runtime performance of DPF when tasked to find the full Pareto front. In comparison to FairOCT, DPF is several orders of magnitude faster. For COMPAS recid., for example, DPF found the Pareto front for depth two consisting of 29 solutions in 0.12 ± 0.04 seconds. The FairOCT method generates the partial Pareto front by repeatedly solving the problem with a different maximum bias, resulting in a runtime several times higher than the values reported in Table 1.

See Figure 1 for a number of full Pareto fronts generated by DPF for depth 2-4. These plots confirm the trade-off between accuracy and fairness as reported in previous works.

Method evaluation. To provide more insight into our method we evaluate the impact of the number of features, the number of dataset instances, the minimum leaf node size and the maximum allowed bias in the fairness constraint on the runtime of DPF for three datasets when searching for trees of depth three. Figure 2 shows the results. It can be observed from these experiments that the number of features is the most important factor, resulting in an exponential increase, which is in line with previous studies [15, 31]. One might expect that the method would scale linearly with increasing dataset size, but the experiments show almost no increase in runtime after a certain size. The reason for this is that the runtime of the method is mostly dependent on the number of unique partial solutions that need to be merged by Eq. 9. This number of unique solutions is strongly dependent on the number of features and is only weakly related to the number of instances.

DPF can easily be changed to limit the minimum leaf node size, by returning an empty set in Eq. 15 when the number of instances does not exceed the required leaf node size. Increasing the minimum leaf node size decreases the amount of unique partial solutions, and consequently we see a significant decrease in the runtime when the minimum leaf node size is increased. Adding such a minimum leaf node size often does not have a significant impact on the accuracy score.

Relaxing the fairness constraint to values larger than $\delta = 1\%$ causes the runtime to increase for the Dutch census dataset. This can be explained because a more relaxed constraint means a larger search space and less strong fairness bounds. However, when the constraint is even more relaxed, the problem

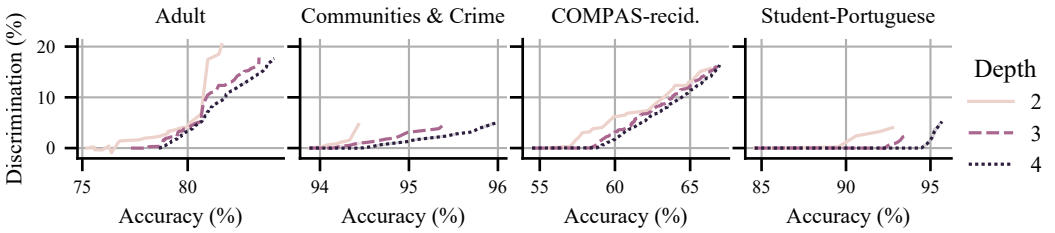

Figure 1: The full Pareto front of training accuracy and discrimination for four datasets for trees of depth $2, 3$ and 4, generated by DPF.

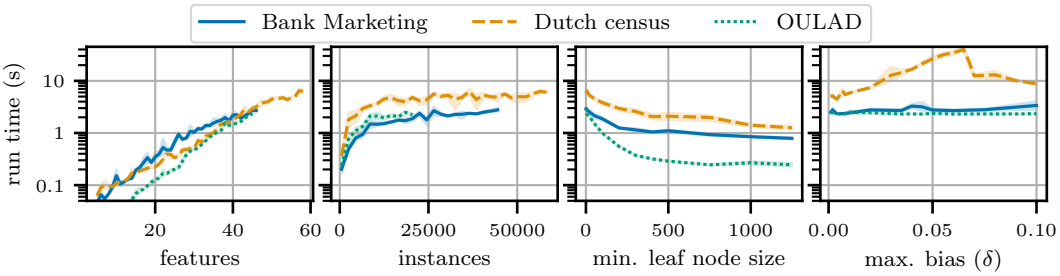

Figure 2: The impact of the number of features, instances, minimum leaf node size and maximum allowed bias in the fairness constraint, δ, on the runtime of DPF with $d = 3$. The shaded area shows the values within one standard deviation. Note the logarithmic scale of the y-axis.

is closer to a normal optimal tree search, and the accuracy bounds become stronger, again resulting in a shorter runtime. For the Bank marketing and OULAD datasets, the default discrimination when no fairness constraint is in place, is lower: 1.2% and 3.0%, versus Dutch census: 14.1%. This explains why these datasets show almost no difference in runtime when changing the value δ.

6 Conclusion

We show how dynamic programming can be used to find fair and optimal decision trees, even when considering a global fairness constraint that introduces subproblem dependency. We solve the subproblem dependency by computing upper and lower bounds on the final fairness value and thus enable comparison of partial solutions. The results show that our method DPF can find fair optimal trees several orders of magnitude faster than the state-of-the-art. As a result, DPF succeeds in finding fair and optimal decision trees even for large datasets that were previously beyond reach. Moreover, we present the first specialized algorithm for finding fair optimal decision trees that can also find the full Pareto front in terms of accuracy and fairness.

An extension of DPF would be to support other group-based notions of fairness, such as equality of opportunity. To better deal with unbalanced datasets, another extension of the presented research would be to replace the accuracy objective with an objective that can cope better with unbalanced datasets, for example, with the biobjective method presented in [14], or by using balanced accuracy. Future work could also investigate the possibility of combining top-down search with bottom-up search. Finally, based on the evaluation results, new heuristic methods could be investigated, for example, by considering not all possibly-fair partial solutions, but only a representative subset.

References

[1] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *Proceedings of ICML-18*, 2018.

[2] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making. In *Proceedings of AAAI-19*, 2019.

[3] Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Strong Optimal Classification Trees. *arXiv preprint arXiv:2103.15965*, 2021.

[4] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. In *Proceedings of AAAI-20*, 2020.

[5] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. PyDL8.5: a Library for Learning Optimal Decision Trees. In *Proceedings of IJCAI-20*, 2020.

[6] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine Bias, May 2016. URL `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`.

[7] Mohammad Javad Azizi, Phebe Vayanos, Bryan Wilder, Eric Rice, and Milind Tambe. Designing Fair, Efficient, and Interpretable Policies for Prioritizing Homeless Youth for Housing Resources. In *Proceedings of CPAIOR-18*, 2018.

[8] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. Fairness in criminal justice risk assessments: The state of the art. *Sociological Methods & Research*, 50(1):3–44, 2021.

[9] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[10] Reuben Binns. On the apparent conflict between individual and group fairness. In *Proceedings of FAT* '20*, 2020.

[11] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[12] Emilio Carrizosa, Cristina Molero-Río, and Dolores Romero Morales. Mathematical optimization in classification and regression trees. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 29(1):5–33, 2021.

[13] Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. In A. Brito and J. Teixeira, editors, *Proceedings of 5th FUture BUsiness TEChnology Conference*, 2008.

[14] Emir Demirović and Peter J. Stuckey. Optimal Decision Trees for Nonlinear Metrics. In *Proceedings of AAAI-21*, 2021.

[15] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. MurTree: Optimal Classification Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26), 2022.

[16] Christophe Denis, Romuald Elie, Mohamed Hebiri, and François Hu. Fairness guarantee in multi-class classification. *arXiv preprint arXiv:2109.13642*, 2021.

[17] Fabrizio Detassis, Michele Lombardi, and Michela Milano. Teaching the Old Dog New Tricks: Supervised Learning with Constraints. In *Proceedings of AAAI-21*, 2021.

[18] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[19] Dutch Central Bureau for Statistics. Dutch Census 2001 public use files (anonymized 1% samples from the microdata files). DANS, 2001. URL `https://easy.dans.knaw.nl/ui/datasets/id/easy-dataset:32357`.

[20] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness Through Awareness. In *Proceedings of ICTS '12*, 2012.

[21] Vincent Grari, Boris Ruf, Sylvain Lamprier, and Marcin Detyniecki. Achieving fairness with decision trees: An adversarial approach. *Data Science and Engineering*, 5(2):99–110, 2020.

[22] Gurobi Optimization LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

[23] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In *Advances in NeurIPS-19*, 2019.

[24] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information processing letters*, 5(1):15–17, 1976.

[25] Nathanael Jo, Sina Aghaei, Jack Benson, Andrés Gómez, and Phebe Vayanos. Learning Optimal Fair Classification Trees. *arXiv preprint arXiv:2201.09932*, 2022.

[26] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *2nd International Conference on Computer, Control and Communication*, 2009.

[27] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. Discrimination Aware Decision Tree Learning. In *Proceedings of ICDM-10*, 2010.

[28] Nikita Kozodoi, Johannes Jacob, and Stefan Lessmann. Fairness in credit scoring: Assessment, implementation and profit implications. *European Journal of Operational Research*, 297(3):1083–1094, 2022.

[29] Jakub Kuzilek, Martin Hlosta, and Zdenek Zdrahal. Open university learning analytics dataset. *Scientific data*, 4(1), 2017.

[30] Tai Le Quy, Arjun Roy, Vasileios Iosifidis, Wenbin Zhang, and Eirini Ntoutsi. A survey on datasets for fairness-aware machine learning. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1452, 2022.

[31] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *Proceedings of ICML-20*, 2020.

[32] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[33] Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Discrimination-aware Data Mining. *Proceedings of SIGKDD-08*, 2008.

[34] Rok Piltaver, Mitja Luštrek, Matjaž Gams, and Sandra Martinčić-Ipšić. What makes classification trees comprehensible? *Expert Systems with Applications*, 62:333–346, 2016.

[35] Francesco Ranzato, Caterina Urban, and Marco Zanella. Fair Training of Decision Tree Classifiers. *arXiv preprint arXiv:2101.00909*, 2021.

[36] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[37] Beata Strack, Jonathan P DeShazo, Chris Gennings, Juan L Olmo, Sebastian Ventura, Krzysztof J Cios, and John N Clore. Impact of HbA1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014.

[38] Ana Valdivia, Javier Sánchez-Monedero, and Jorge Casillas. How fair can we go in machine learning? Assessing the boundaries of accuracy and fairness. *International Journal of Intelligent Systems*, 36(4): 1619–1643, 2021.

[39] Kush R Varshney and Homa Alemzadeh. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big data*, 5(3):246–255, 2017.

[40] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *Proceedings of CPAIOR-17*, 2017.

[41] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of AAAI-19*, 2019.

[42] Haoran Zhu, Pavankumar Murali, Dzung T. Phan, Lam M. Nguyen, and Jayant R. Kalagnanam. A Scalable MIP-based Method for Learning Optimal Multivariate Decision Trees. In *Advances in NeurIPS-20*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] The limitations of the work are as follows: 1) the algorithm only works for *binary* classification with only *binary* features; 2) the algorithm can only generate *small* decision trees; 3) the algorithm's runtime performance scales exponentially with increasing number of features. All of these are mentioned in the text.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] This paper uses demographic parity as a fairness metric. Readers are informed that this is just one of the metrics commonly used to determine what is fair. Other papers that further discuss these issues are referenced.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] 1) Datasets are referenced and licenses are included in the repository. 2) This paper focuses on *demographic parity* as a fairness metric. Relevant papers that discuss the value and limitations of this fairness metric are referenced.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The link to the source code will be made publicly available upon acceptance
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the code repository.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Complexity analysis

An upperbound on the runtime complexity of DPF as presented in Algorithm 1 can be formulated by multiplying the size of the dataset by the maximum number of leaf nodes that can be considered. Without pruning based on bounds or cache the number of possible leaf nodes is $O(|\mathcal{F}|^{2^d})$. Therefore $O(|\mathcal{D}||\mathcal{F}|^{2^d})$ is an upper bound on the runtime complexity. In practice this number is much smaller because of pruning.

B Non-complete trees

For the sake of brevity, the main text of this paper only shows how to search for complete trees, that is, trees of depth d with $2^d - 1$ branch nodes and 2^d leaf nodes. Our method also allows to search for smaller trees, as was done similarly in [15].

The DP formulation in Eq. 16 can be extended as follows to also allow for incomplete/sparse trees. Here n signifies the number of nodes in the tree.

$$\begin{aligned}&\text{nondom}\left(\text{prune}\left(\cup_{f\in\mathcal{F},i\in[0,n-1]}\,\text{merge}\left(\right.\right.\right.\\&\qquad T_F(\mathcal{D}_{\bar{f}}, d-1, n-i-1, \quad U(\mathcal{D}_f, [\underline{I}_R, \bar{I}_R], d-1, i)),\\&\qquad \left.\left.\left.T_F(\mathcal{D}_f, d-1, i, \quad U(\mathcal{D}_{\bar{f}}, [\underline{I}_R, \bar{I}_R], d-1, n-i-1))\right)\right)\right)\end{aligned} \tag{21}$$

With this change in place, the solver can search for incomplete trees. This also allows to add a parameter α to prevent overfitting. The misclassification score now becomes $M + \alpha n$. The parameter α describes how much the misclassification score should at least decrease in order to justify adding another node to the tree. The addition of this parameter to the algorithm is trivial.

C Dataset details

Table 3 shows detailed information about every dataset considered in this study. The references for the original datasets can be found here [6, 13, 18, 19, 29, 32, 37].

Table 3: Datasets used for evaluation. Preprocessed as described in [30]. Training accuracy (%) and discrimination results (%) are shown for the best tree of depth three that does not consider fairness when training with the full dataset. The sign of the discrimination score tells which of the two groups is discriminated against.

			protected	y=1		y=0		Acc.	Disc.
Name	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$	feature	$a=1$	$a=0$	$a=1$	$a=0$	d=3	d=3
Adult	45222	17	Gender	9539	1669	20988	13026	83.4	-17.8
Bank	45211	46	Married	2755	2534	24459	15463	90.0	1.2
Com.&Cr.	1994	97	Race	1017	855	7	115	95.4	-4.5
COMP. r.	6172	9	Race	1281	2082	822	1987	66.7	-16.4
COMP. v.r.	4020	9	Race	1285	2083	174	478	84.1	-1.6
Dutch	60420	58	Gender	18860	9903	11287	20370	81.4	-14.1
German	1000	69	Gender	499	201	191	109	75.3	-5.5
KDD	284556	117	Race	15926	1475	223155	44000	94.4	-1.1
OULAD	21562	45	Gender	7727	6928	3841	3066	69.1	-3.0
Ricci	118	4	Race	41	15	27	35	100	-30.3
Stud. Math	395	55	Gender	124	141	63	67	93.4	-6.3
Stud. Port.	649	55	Gender	228	321	38	62	93.7	3.4

D Test Evaluation

Experiment setup. The evaluation in the main text is focused on analyzing the runtime of our DPF method. This section further analyzes the out-of-sample performance of DPF and compares it with two heuristics. The pre-processing (massaging) approach presented in [26], and a post-processing approaches proposed in [27]. We will call the pre-relabelling method KamPre and the post-relabelling method KamPost, after their first author (Kamiran).

KamPre pre-processes the training data by relabeling a number of instances in the training data such that the training data no longer is biased. Like them, we use a Naive Bayes classifier to decide which labels to change. Unlike them, we use the newly labeled data to train a standard decision tree with CART.

KamPost differs from CART by using a different splitting criterion and by its post-relabelling of the leaf nodes. KamPost uses a splitting criterion that is a mix of information gain in the class label and information gain in the sensitive attribute (IGC+IGS). After generating the tree, it heuristically changes the label of some leaves in such a way that discrimination is minimized with the least loss of accuracy.

We do not compare to the optimal MIP method FairOCT because its optimal solutions would either be the same as those generated by DPF, or -if multiple optimal solutions exists- any difference could only be attributed to a random selection of one out of several optimal models.

We first evaluate the out-of-sample performance by running DPF for $d = 2, 3, 4$ and KamPost for $d = 3, 4$ for different maximum allowed bias $\delta = 1\%, 5\%, 100\%$ on three datasets that are commonly evaluated in the literature. Note that KamPost with $\delta = 100\%$ is the same as plain CART. We run each case 10 times on random stratified train-test splits of 75% vs 25%. We here do not compare to KamPre because it does not take a maximum allowed bias as an input factor. Figure 3 shows the resulting distribution of test accuracy and test discrimination. The sign of the discrimination score tells which of the two groups is discriminated against.

Evaluation. From the results in Figure 3 it can be observed that the optimal decision trees generated by DPF in general have better out-of-sample accuracy than KamPost for the same or smaller depth. In several cases DPF $d = 2$ even outperforms KamPost, even though it uses four times less nodes. However, in general the variance in test accuracy is high.

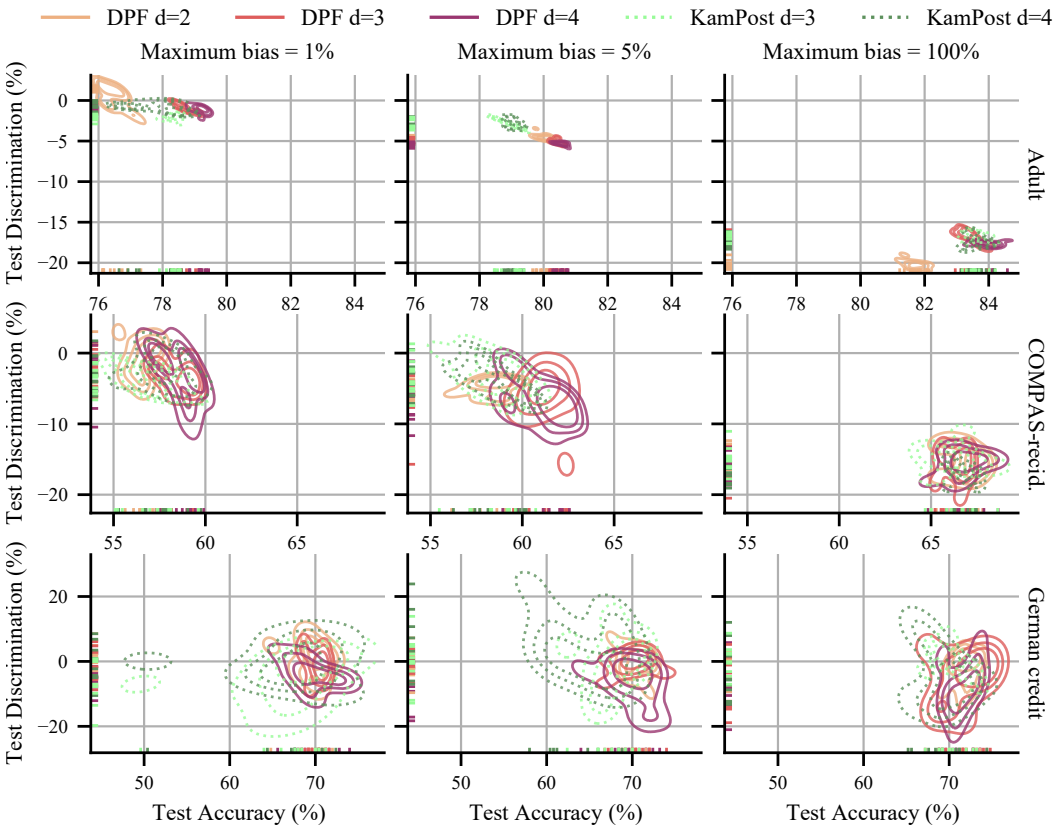

Figure 3: Out-of-sample accuracy and discrimination for three datasets, for maximum bias $\delta = 1\%, 5\%, 100\%$. The figure shows the distribution of 10 runs.

The variance in test discrimination is also high, and both methods often exceed the imposed discrimination limit in the test evaluation. This problem is less visible with the Adult dataset, probably because of its larger number of data instances.

For both the COMPAS-recid. and German credit dataset almost maximum accuracy can already be obtained with a decision tree of depth two, so the addition of extra nodes does not help much. For the Adult dataset, however, deeper trees can provide better accuracy.

Tuned for number of nodes. In our next analysis, we prevent overfitting during training for all three methods by using 10 random validation splits of 25% of the training data to find what number of branching nodes is best. The tree size with the best average accuracy in the validation set, while on average respecting the discrimination constraint is selected as best. The full training dataset is then used to generate a tree of that size. Table 4 shows the results when all three algorithms are used to find trees of a maximum bias of 5%.

Discussion. DPF searches for optimal decision trees, which means it will always find the tree with maximum accuracy for the training dataset, and thus always outperform heuristics on performance in the training dataset. The results in Table 4 show that when DPF is tuned for selecting the right number of nodes, this on average also generalizes to better performance than KamPre and KamPost in the test set.

We compare the results of the methods that achieve (on average) a test discrimination lower than 5%, and among those select the method with highest test accuracy. There are seven datasets for which DPF is significantly better than KamPre and KamPost ($p < 5\%$), with differences in accuracy even as large as 11.6% (Student-Mathemtacis) or 9.2% (Dutch census). KamPost only scores best for Ricci and Adult. Ricci is the smallest of all datasets with only 118 instances and 4 features, but KamPost's

Table 4: Out-of-sample average accuracy and discrimination $\pm$ the standard deviation (%) for solutions with a maximum training bias of $\delta = 5\%$ and a maximum depth of $d = 3$. Whenever the 5% discrimination threshold is exceeded on average, the result is marked in red. Best performing accuracy score per dataset is marked bold, if significantly better than other methods that also stay within the 5% discrimination limit (p-value $< 5\%$).

	DPF		KamPre		KamPost	
Dataset	Accuracy	Disc.	Accuracy	Disc.	Accuracy	Disc.
Adult	80.4 $\pm$ 0.3	-5.1 $\pm$ 0.4	75.2 $\pm$ 0.0	0.0 $\pm$ 0.0	**78.0** $\pm$ 1.5	-1.9 $\pm$ 1.1
Bank	**89.7** $\pm$ 0.1	1.7 $\pm$ 0.6	89.1 $\pm$ 0.3	0.9 $\pm$ 0.6	89.0 $\pm$ 0.4	0.5 $\pm$ 0.3
Com.&Cr.	**94.6** $\pm$ 0.3	-3.2 $\pm$ 1.0	93.0 $\pm$ 1.1	-3.3 $\pm$ 2.6	93.9 $\pm$ 0.3	-1.3 $\pm$ 2.3
COMP. r.	**59.1** $\pm$ 2.4	-4.6 $\pm$ 3.1	61.4 $\pm$ 1.8	-9.8 $\pm$ 2.8	55.6 $\pm$ 1.9	-1.1 $\pm$ 2.1
COMP. v.r.	83.7 $\pm$ 0.2	-0.3 $\pm$ 0.5	82.2 $\pm$ 1.1	-4.0 $\pm$ 2.1	83.8 $\pm$ 0.1	-0.3 $\pm$ 0.8
Dutch	**77.4** $\pm$ 0.3	-5.0 $\pm$ 1.0	76.0 $\pm$ 0.2	-9.9 $\pm$ 0.4	68.2 $\pm$ 10.9	-0.6 $\pm$ 0.6
German	70.3 $\pm$ 1.7	-2.1 $\pm$ 3.0	69.8 $\pm$ 1.3	-0.7 $\pm$ 3.9	70.3 $\pm$ 1.0	-0.9 $\pm$ 3.4
KDD	**94.3** $\pm$ 0.0	-1.0 $\pm$ 0.1	93.9 $\pm$ 0.0	0.0 $\pm$ 0.0	93.9 $\pm$ 0.0	0.0 $\pm$ 0.1
OULAD	**68.7** $\pm$ 0.3	-2.1 $\pm$ 1.0	68.2 $\pm$ 0.6	-1.8 $\pm$ 1.1	68.0 $\pm$ 0.1	0.1 $\pm$ 0.2
Ricci	66.0 $\pm$ 4.4	-13.4 $\pm$ 12.3	100.0 $\pm$ 0.0	-28.1 $\pm$ 0.0	**53.3** $\pm$ 0.0	0.0 $\pm$ 0.0
Stud. Math	**85.5** $\pm$ 6.6	-2.8 $\pm$ 8.3	89.7 $\pm$ 2.8	-5.5 $\pm$ 3.2	73.9 $\pm$ 11.8	-2.5 $\pm$ 4.2
Stud. Port.	90.5 $\pm$ 2.8	0.8 $\pm$ 5.0	91.2 $\pm$ 4.2	6.1 $\pm$ 4.9	89.4 $\pm$ 4.2	2.3 $\pm$ 3.1

result is only slightly better than random. KamPost also performs best for Adult, with DPF exceeding the limit by 0.1%. For three datasets, no method is significantly better than the others.

The results also confirm the findings from Figure 3 that the variance in the discrimination value is often high, specifically for the small datasets. This means that for those instances it is difficult to generalize and overfitting in terms of discrimination is still happening. It is an open question how this can be reduced.