Synthesizing Fair Decision Trees via Iterative Constraint Solving

Jingbo Wang(✉), Yannan Li, and Chao Wang

University of Southern California, Los Angeles, CA 90089, USA
jingbow@usc.edu

Abstract. Decision trees are increasingly used to make socially sensitive decisions, where they are expected to be both accurate and fair, but it remains a challenging task to optimize the learning algorithm for fairness in a predictable and explainable fashion. To overcome the challenge, we propose an iterative framework for choosing decision attributes, or *features*, at each level by formulating *feature selection* as a series of mixed integer optimization problems. Both fairness and accuracy requirements are encoded as numerical constraints and solved by an off-the-shelf constraint solver. As a result, the trade-off between fairness and accuracy is quantifiable. At a high level, our method can be viewed as a generalization of the entropy-based greedy search techniques such as `CART` and `C4.5`, and existing fair learning techniques such as `IGCS` and `MIP`. Our experimental evaluation on six datasets, for which *demographic parity* is used as the fairness metric, shows that the method is significantly more effective in reducing bias than other methods while maintaining accuracy. Furthermore, compared to non-iterative constraint solving, our iterative approach is at least 10 times faster.

1 Introduction

Decision trees are one of the most widely used machine learning models in statistical analysis, data mining and decision making. Compared to other predictive models such as deep neural networks, decision trees have the advantage of being easily understandable by humans, which makes them a favorite building block in systems that require interpretability [34]. However, when they are used to make socially sensitive decisions in business, finance and law enforcement, decision trees may introduce bias against certain groups [16]. In this context, a widely used group fairness metric is *demographic parity* [11,38], also known as the *80% rule* [8]. Bias against demographic groups, in general, comes from two sources. First, historical data used to learn models may be biased. Second, learning algorithms may be biased even if they operate on unbiased data.

State-of-the-art decision tree learning algorithms such as `CART` and `C4.5` [10, 29], which are the ones used by popular machine learning toolkits, rely on a

This work was partially funded by the U.S. National Science Foundation grants CNS-1813117 and CNS-1702814.

S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13372, pp. 364–385, 2022.
https://doi.org/10.1007/978-3-031-13188-2_18

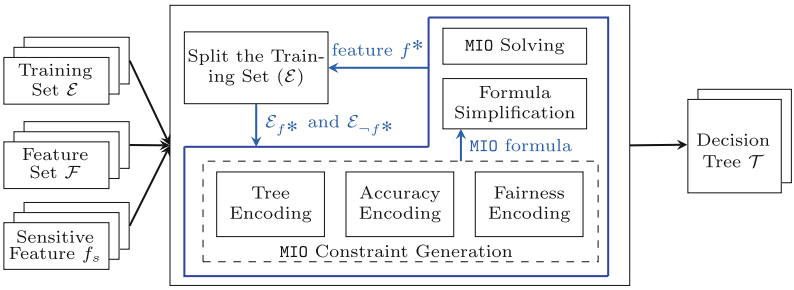

Fig. 1. SFTree – our symbolic method for synthesizing a fair decision tree.

greedy search technique that is optimized solely for high learning speed and classification accuracy. Since they do not consider fairness as an optimization requirement at all, they often produce decision trees that are severely biased. To mitigate the bias, modifications have been proposed to make the greedy search discrimination-aware [24] (e.g., `IGCS`). Unfortunately, these modifications are not always effective as shown by our own experimental evaluation in Sect. 5 and, more importantly, the impact of *ad hoc* modifications is often unpredictable and difficult to explain.

Meanwhile, there is a line of work in operational research that formulates decision tree learning as a mixed-integer optimization (`MIO`) problem [7,35]. Given a finite set $\mathcal{F}$ of decision attributes, or features, and a maximum tree depth K, the set of all possible decision trees is captured symbolically as a set of numerical constraints, which is then fed to a solver to compute the *globally-optimal* decision tree. While optimality was defined initially to minimize the tree size and accuracy loss [7,35], later on, fairness was added as a goal of the optimization [1,5]. However, the approach remains largely theoretical due to its limited scalability: since the entire decision tree must be encoded as a monolithic `MIO` problem, only small training datasets (with sample sizes in the 1000s) and small decision trees (with depths up to 4 or 5) can be handled [2,7].

To overcome the limitations of the existing approaches, we propose an *iterative constraint solving* technique for synthesizing decision trees in a practically efficient fashion while simultaneously optimizing for fairness and accuracy. Instead of encoding the decision tree as a monolithic `MIO` formula, we break it down to a series of small steps to avoid the scalability bottleneck. Specifically, starting from the root node, we use constraint solving to conduct a depth-bounded look-ahead search at each level of the decision tree, to compute the best feature. Within the look-ahead search, we encode both fairness and accuracy requirements explicitly as numerical constraints, to make the fairness-accuracy trade-off not only predictable but also easy to explain.

The overall flow of our method, SFTree, is shown in Fig. 1. Given a set of training examples ($\mathcal{E}$), a set of features ($\mathcal{F}$), and a sensitive feature ($f_s \in \mathcal{F}$) as input, SFTree returns the synthesized decision tree ($\mathcal{T}$) as output. Internally, SFTree encodes the hierarchical structure of a partial decision tree symbolically

starting from the current node and its training set $\mathcal{E}$, covering a fixed number of tree levels. Then, it uses an `MIO` solver to compute the optimal feature, f^*, that minimizes the bias against the protected group, the classification error, and the tree size. Assuming that $f^* \in \{0, 1\}$ is a Boolean predicate, the training set is partitioned into subsets $\mathcal{E}_{f^*}$ and $\mathcal{E}_{\neg f^*}$, one for each child node. Our method iteratively partitions the child nodes until the training subset becomes empty, or all examples in $\mathcal{E}$ belong to the same class, or all features in $\mathcal{F}$ have been used.

To demonstrate its effectiveness, we have implemented SFTree and evaluated it on six supervised learning datasets, consisting of three small datasets and three large ones. Since the small datasets can be handled even by the monolithic `MIO` approach (named `MIP` [1]) to obtain globally-optimal and fair solutions, we used them to evaluate the quality of decision trees learned by our method. The large datasets, which are out of the reach of `MIP`, were used to evaluate scalability. For comparison, we also evaluated `CART` [27], a mainstream decision tree learning algorithm, and `IGCS` [24], a discrimination-aware learning algorithm.

The experimental results show that, among all methods (`CART`, `IGCS`, `MIP`, and SFTree), SFTree produces the best overall solution in terms of fairness and accuracy. In contrast, `CART` produces unfair decision trees in most cases and, while `IGCS` does well on the small datasets, it produces mostly unfair decision trees for the large datasets. Neither `CART` nor `IGCS` is effective in satisfying the well-known *80% Rule* [8] for demographic parity [11,38]. In contrast, SFTree satisfies the *80% Rule* in all cases. In terms of scalability, `MIP` fails to handle any of the large datasets, while SFTree handles all of them. In fact, among all four methods, SFTree is the only one that produces fair and accurate decision trees for datasets with $>40,000$ training samples.

To sum up, this paper makes the following contributions:

- We propose an iterative constraint-solving method for synthesizing fair decision trees:
  - By formulating feature selection as a series of mixed integer optimization subproblems, we make the constraints efficiently solvable.
  - By encoding fairness and accuracy explicitly as symbolic constraints, we make the trade-off quantifiable and easy to explain.
- We demonstrate the advantages of SFTree over existing approaches (`CART`, `IGCS`, and `MIP`) using six popular datasets in the fairness literature.

The remainder of this paper is organized as follows. In Sect. 2, we review the basics of decision tree learning and group fairness. In Sect. 3, we present our method. In Sect. 4, we present generalization and performance enhancement techniques. In Sect. 5, we present our experimental results. After reviewing the related work in Sect. 6, we give our conclusions in Sect. 7.

2 Background

2.1 Training Dataset $\mathcal{E}$

The training dataset is a finite set of examples, $\mathcal{E} = \{(x_i, y_i)\}$, where $i \in \mathcal{N}$ is the index, input $x_i = \langle f_1, \ldots, f_k \rangle$ is a vector of features, and output y_i is a class

i	Input(x_i)				Output(y_i)
	gender(f_1)	rank(f_2)	experi(f_3)	interv(f_4)	offer
1	0	0	1	1	1
2	1	0	1	1	1
3	1	1	0	1	1
4	0	1	0	0	0
5	1	1	0	1	1
6	1	0	1	0	0
7	0	1	0	1	0

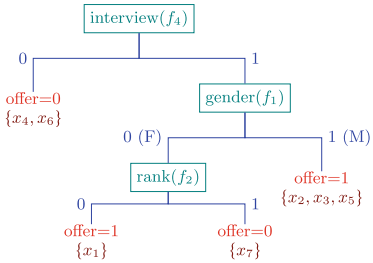

Fig. 2. An example training dataset $\mathcal{E}$ (left) and the related decision tree $\mathcal{T}$ (right).

label. Let $\mathcal{F}$ be the set of all features. For ease of comprehension, let us assume for now that all input features and the output class label are Boolean. In this case, every input $x_i \in \{0,1\}^k$ is a k-bit vector in the feature space, the output $y_i \in \{0,1\}$ is a bit, and a decision tree trained using $\mathcal{E}$ is a k-input Boolean function. To make the presentation clear, we may also use $y_i \in \{-,+\}$ instead of $y_i \in \{0,1\}$ as the output, where $-$ means "no" and $+$ means "yes".

Figure 2 shows a training set $\mathcal{E}$, where each row in the table represents an example. The input features are a job candidate's *gender* (0 = Female, 1 = Male), *college rank* (0 = Low, 1 = High), *experience* (0 = No, 1 = Yes), and *interview score* (0 = Not-Good, 1 = Good), while the output shows whether the job is offered (0 = No, and 1 = Yes). At the root of the decision tree, for instance, the input goes to the left branch when ($f_4 = 0$) and to the right branch when ($f_4 = 1$). The example illustrates three important notions associated with the training set: (1) partition of $\mathcal{E}$ (2) entropy, and (3) conditional entropy.

Partition. Given a set $\mathcal{E}$ and a feature f_j, we can partition $\mathcal{E}$ into subsets $\mathcal{E}_{f_j=0}$ and $\mathcal{E}_{f_j=1}$, or $\mathcal{E}_{\neg f_j}$ and $\mathcal{E}_{f_j}$, respectively, in shorthand notation. Here, $\mathcal{E}_{\neg f_j} = \{(x_i, y_i) \in \mathcal{E} \mid f_j(x_i) = 0\}$ consists of examples whose f_j is 0, and $\mathcal{E}_{f_j} = \{(x_i, y_i) \in \mathcal{E} \mid f_j(x_i) = 1\}$ consists of examples whose f_j is 1. By definition, we have $\mathcal{E}_{\neg f_j} \subseteq \mathcal{E}$ and $\mathcal{E}_{f_j} \subseteq \mathcal{E}$, $\mathcal{E}_{\neg f_j} \cap \mathcal{E}_{f_j} = \varnothing$ and $\mathcal{E}_{\neg f_j} \cup \mathcal{E}_{f_j} = \mathcal{E}$.

For our example in Fig. 2, partitioning the dataset by *gender* (f_1) results in subsets $\mathcal{E}_{f_1=F} = \mathcal{E}_{\neg f_1} = \{(x_1, y_1)(x_4, y_4)(x_7, y_7)\}$ and $\mathcal{E}_{f_1=M} = \mathcal{E}_{f_1} = \{(x_2, y_2)(x_3, y_3)(x_5, y_5)(x_6, y_6)\}$.

Entropy. The diversity (or purity) of a set $\mathcal{E}$ may be measured by Shannon entropy. Let $|\mathcal{E}^+|$ be the number of examples in $\mathcal{E}$ with positive output label, and $|\mathcal{E}^-|$ be the number of examples with negative output label. The percentage of positive examples is $|\mathcal{E}^+|/|\mathcal{E}|$, and the percentage of negative examples is $|\mathcal{E}^-|/|\mathcal{E}|$. Thus, the entropy is $H(\mathcal{E}) = -\frac{|\mathcal{E}^+|}{|\mathcal{E}|} log(\frac{|\mathcal{E}^+|}{|\mathcal{E}|}) - \frac{|\mathcal{E}^-|}{|\mathcal{E}|} log(\frac{|\mathcal{E}^-|}{|\mathcal{E}|})$.

For our example in Fig. 2, since $|\mathcal{E}^-| = 3$ and $|\mathcal{E}^+| = 4$, the entropy is $H(\mathcal{E}) = -\frac{3}{7} log(\frac{3}{7}) - \frac{4}{7} log(\frac{4}{7}) \approx 0.985$.

Conditional Entropy. Given a partition of the set $\mathcal{E}$ by the feature f_j, the entropy of each subset, $\mathcal{E}_{\neg f_j}$ or $\mathcal{E}_{f_j}$, is defined similarly. For our example, since $\mathcal{E}_{\neg f_1}$ has 2/3 negative examples and 1/3 positive examples, the

entropy is $H(\mathcal{E}_{\neg f_1}) = -\frac{2}{3}log(\frac{2}{3}) - \frac{1}{3}log(\frac{1}{3}) = 0.918$. Similarly, since $\mathcal{E}_{f_1}$ has 1/4 negative examples and 3/4 positive examples, the entropy is $H(\mathcal{E}_{f_1}) = -\frac{1}{4}log(\frac{1}{4}) - \frac{3}{4}log(\frac{3}{4}) = 0.811$.

The conditional entropy of $\mathcal{E}$, with respect to f_j, is defined as follows:

$$H(\mathcal{E} \mid f_j) = \frac{|\mathcal{E}_{\neg f_j}|}{|\mathcal{E}|} H(\mathcal{E}_{\neg f_j}) + \frac{|\mathcal{E}_{f_j}|}{|\mathcal{E}|} H(\mathcal{E}_{f_j})$$

For our running example, since there are 3 female and 4 male candidates, we have $|\mathcal{E}_{\neg f_1}|/|\mathcal{E}| = 3/7$ and $|\mathcal{E}_{f_1}|/|\mathcal{E}| = 4/7$. Thus, the conditional entropy is $H(\mathcal{E} \mid f_1) = \frac{3}{7}H(\mathcal{E}_{\neg f_1}) + \frac{4}{7}H(\mathcal{E}_{f_1}) \approx 0.857$.

The difference between $H(\mathcal{E})$ and $H(\mathcal{E} \mid f_j)$ is called the *information gain*, a metric for evaluating how effective f_i is in separating positive examples from negative examples in $\mathcal{E}$. For our example, since $H(\mathcal{E}) \approx 0.985$ and $H(\mathcal{E} \mid f_1) \approx 0.857$, the information gain (of partitioning $\mathcal{E}$) by *gender* (f_1) is $0.985 - 0.857 = 0.128$. In contrast, the information gain by *interview* (f_4) is $0.985 - 0.516 = 0.469$. Thus, f_4 is more effective as a decision attribute.

Real-Valued Features. It is important to note that, while the above examples use Boolean features, our method is more general in that it allows all features have real values, i.e., $x_i \in [0,1]^k$ instead of $x_i \in \{0,1\}^k$. We accomplish this by applying one-hot encoding to any categorical feature and normalizing any real-valued feature to the $[0,1]$ domain. Thus, the branch predicates become $(f_j < b_v)$ and $(f_j \geqslant b_v)$, instead of $(f_j = 0)$ and $(f_j = 1)$, where $b_v \in (0,1]$ is a threshold computed by our method. For example, if f_j is the (normalized) salary and $b_v = 0.5$, the branch predicates are $(f_j < 0.5)$ and $(f_j \geqslant 0.5)$.

2.2 Decision Tree Learning

A decision tree $\mathcal{T}$ is a binary tree consisting of a set of nodes and a set of edges. Let the set of nodes be $\mathcal{V} \cup \mathcal{L}$, where $\mathcal{V}$ is the subset of branch nodes (including the root) and $\mathcal{L}$ is the subset of leaf nodes. Let E be the set of edges between these nodes. A path in $\mathcal{T}$ is a sequence of nodes and edges, denoted $v_0, e_1, v_1 \ldots v_n, e_n, l_n$, where v_0 is the root, l_n is a leaf node, $v_1 \ldots v_n$ are the internal nodes, and $e_1, \ldots, e_n$ are the edges.

Each edge has a branch condition. The edge is activated only if the condition holds for a given input x. In Fig. 2, for example, the left-most path of the decision tree has the condition $f_4(x) = 0$ and output *offer* $= 0$, while the right-most path has the condition $(f_4(x) = 1) \wedge (f_1(x) = M)$ and output *offer* $= 1$.

Given a training set $\mathcal{E} = \{(x_i, y_i)\}$, where x_i is an input and y_i is the known output, mainstream algorithms aim to learn a decision tree $\mathcal{T}$ that minimizes the classification error. They also aim to minimize the tree size which, in general, allows $\mathcal{T}$ to generalize well on the test examples.

The Baseline Algorithm. Algorithm 1 shows the top-level procedure of these mainstream algorithms. It takes the training set $\mathcal{E}$ and the feature set $\mathcal{F}$ as input, and returns a decision tree ($\mathcal{T}$) as output. These mainstream algorithms use a

Algorithm 1. The baseline decision tree learning procedure $\mathcal{T} = \texttt{DTL}(\mathcal{E}, \mathcal{F})$.

1: **Input:** training set $\mathcal{E} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and feature set $\mathcal{F} = \{f_1, f_2, \ldots, f_k\}$
2: **Output:** decision tree $\mathcal{T}$
3: **if** all examples in $\mathcal{E}$ have the same label l=$\textsc{Label}(\mathcal{E})$
4: **return** $\mathcal{T}$ = LeafNode(l)
5: **else if** $\mathcal{F} = \varnothing$ and the most common label of $\mathcal{E}$ is $l^* = \textsc{MostCommonLabel}(\mathcal{E})$
6: **return** $\mathcal{T}$ = LeafNode(l^*)
7: **else if** $\mathcal{E} = \varnothing$ and in $\mathcal{E}$.parent, we have $l^* = \textsc{MostCommonLabel}(\mathcal{E}.\text{parent})$
8: **return** $\mathcal{T}$ = LeafNode(l^*)
9: **else**
10: $\mathcal{T}$ = BranchNode(f^*), where $f^* = \textsc{FindNextFeature}(\mathcal{E}, \mathcal{F})$
11: **foreach** value $i \in \{0, 1\}$ of the chosen feature f^*
12: $\mathcal{T}_i = \texttt{DTL}(\mathcal{E}_{f^*=i}, \mathcal{F} \setminus \{f^*\})$
13: Add an edge from $\mathcal{T}$ to $\mathcal{T}_i$ with label $(f^*(x) = i)$
14: **return** $\mathcal{T}$

Algorithm 2. Subroutine $\textsc{FindNextFeature}(\mathcal{E}, \mathcal{F})$ used in CART.

1: Let $H(\mathcal{E}) := -\sum_{l \in \{-,+\}} \frac{|\mathcal{E}^l|}{|\mathcal{E}|} \log(\frac{|\mathcal{E}^l|}{|\mathcal{E}|})$ ▷ Entropy
2: Let $H(\mathcal{E} \mid f) := \sum_{i \in \{0,1\}} \frac{|\mathcal{E}_{f=i}|}{|\mathcal{E}|} H(\mathcal{E}_{f=i})$ ▷ Conditional Entropy
3: **return** $f^* = \text{argmax}_{f \in \mathcal{F}} H(\mathcal{E}) - H(\mathcal{E} \mid f)$

greedy method to recursively select decision attributes from $\mathcal{F}$ and use them to partition the training set $\mathcal{E}$. At each step, it selects the best feature f^* using the subroutine FindNextFeature.

In CART, for example, FindNextFeature is entropy-based, to maximize the information gain of partitioning $\mathcal{E}$ by f as shown in Algorithm 2. While this is fast and often leads to high classification accuracy, it does not consider fairness and thus often produces biased decision trees. In this work, we use *iterative constraint solving* to overcome the limitation.

After f^* is computed by FindNextFeature, Algorithm 1 uses it to partition the training set $\mathcal{E}$, and recursively process the two subsets: $\texttt{DTL}(\mathcal{E}_{f^*=0}, \mathcal{F} \setminus \{f^*\})$ and $\texttt{DTL}(\mathcal{E}_{f^*=1}, \mathcal{F} \setminus \{f^*\})$. The recursion ends when

- all training examples in the set $\mathcal{E}$ have the same class label (*Lines 3–4*);
- there are no features left in $\mathcal{F}$ to split $\mathcal{E}$ further (*Lines 5–6*); or
- the set $\mathcal{E}$ is empty (*Lines 7–8*).

2.3 Fairness Metric

Given a training set $\mathcal{E}$ and a sensitive feature $f_s \in \mathcal{F}$, e.g., race or gender, the goal is to construct a decision tree $\mathcal{T}$ that maximizes classification accuracy while minimizing bias. The metric concerned in this work, *demographic parity* [11,38], comes from the legal guideline in the United States for avoiding employment discrimination. Known as the *80% rule* [8], it says the percentage at which

Algorithm 3. Subroutine FindNextFeature($\mathcal{E}, \mathcal{F}$) in our method.

1: Let f_s be the sensitive feature
2: (O, Φ) = DtlEncoding($\mathcal{E}, \mathcal{F}, f_s$)
3: f^* = MioSolver(O, Φ)
4: **return** f^*

candidates from one protected group are offered jobs should be at least 80% of the percentage at which candidates from another group are offered jobs.

This is formalized using the fairness index, $F_s(\mathcal{T}, \mathcal{E})$, defined as follows:

$$F_{f_s}(\mathcal{T}, \mathcal{E}) = \frac{Pr[\mathcal{T}(x) = + \mid f_s(x) = 0]}{Pr[\mathcal{T}(x) = + \mid f_s(x) = 1]} \tag{1}$$

where $Pr[\mathcal{T}(x) = + \mid f_s(x) = 0]$, or $Pr^+_{\neg f_s}$ in short, is the probability of positive examples under the condition $f_s(x) = 0$, and $Pr[\mathcal{T}(x) = + \mid f_s(x) = 1]$, or Pr^+_{fs} in short, is the probability of positive examples under the condition $f_s(x) = 1$. Thus, we have $Pr^+_{\neg f_s} = \frac{|\{x \in \mathcal{E} \mid f_s(x)=0 \wedge \mathcal{T}(x)=+\}|}{|\{x \in \mathcal{E} \mid f_s(x)=0\}|}$ and $Pr^+_{f_s} = \frac{|\{x \in \mathcal{E} \mid f_s(x)=1 \wedge \mathcal{T}(x)=+\}|}{|\{x \in \mathcal{E} \mid f_s(x)=1\}|}$.

Demographic parity means $0.8 \leqslant F_s(\mathcal{T}, \mathcal{E}) \leqslant (1/0.8) = 1.25$. For the example in Fig. 2, since $F_{f_1}(\mathcal{T}, \mathcal{E}) = 0.44$ for *gender* (f_1), the tree fails to satisfy the *80% rule* due to bias against female. The bias is explicit in that f_1 is actually used in the edge labels of the right most two paths of the decision tree. However, even if f_1 is not used in $\mathcal{T}$ explicitly, $\mathcal{T}$ may still be biased against female, for example, if other non-sensitive features (or their combinations) are statistically correlated to f_1 and, as a result, introduce bias against female. This is the reason why mitigating bias during decision tree learning is a challenging task.

3 Our Method

To minimize the bias and, at the same time, maximize the classification accuracy, we proposed to follow the top-level procedure in Algorithm 1, but formulate *feature selection* as a series of mixed-integer optimization (MIO) subproblems.

As shown in Algorithm 3, each of our MIO subproblems consists of an objective function O and a constraint Φ, and the solution is an assignment of the numerical variables (shared by O and Φ) that minimizes O while satisfying Φ. In the remainder of this section, we present our symbolic encoding of the objective function, O, and the constraint, Φ, respectively.

3.1 The Objective Function O

We define the function as $O := O_{accu} + \alpha O_{tree} - \beta O_{fair}$, consisting of components for accuracy loss (O_{accu}), tree size (O_{tree}), and fairness score (O_{fair}),

respectively. The constants, α and β, are used to make trade-offs. In our implementation, α is fixed to $1/(2^{K+1}-2)$ while β is the optimal value in $[0, 1]$ selected using n-fold cross-validation.

Specifically, we test the values 0.02, 0.04, 0.06, ... to 1.00 and, for each fold of the dataset, we compute the objective function and choose β with the minimal objective value. In general, a bigger β means more fairness. Our experiments show that, as β gets larger, O_{fair} remains constant initially and then starts increasing while O_{accu} remains constant, and then O_{accu} starts increasing.

Since the decision tree structure is not known *a priori*, we encode a complete binary tree while allowing all branch and leaf nodes to be activated or de-activated. Recall that $\mathcal{L}$ is the subset of leaf nodes, $\mathcal{V}$ is the subset of branch nodes, $l \in \mathcal{L}$ denotes a leaf node, and $v \in \mathcal{V}$ denotes a branch node.

Tree Size ($O_{tree} := \sum_{v \in \mathcal{V}} p_v$). We assign a variable p_v to each branch node $v \in \mathcal{V}$, to indicate if a feature is used to split v. Thus, $p_v = 1$ means v is split, while $p_v = 0$ means v is not split. To get a valid decision tree, p_v must be constrained also by formula Φ (Sect. 3.2) . Assuming the number of p_v variables is $|\mathcal{V}|$, the tree size is the number of p_v variables with value 1.

Accuracy Loss ($O_{accu} := \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} L_l$). We assign a variable L_l to each leaf node $l \in \mathcal{L}$ to represent the misclassification error at l. Since we start with a complete tree, each leaf node corresponds to a distinct path. The actual value of L_l is defined by formula Φ (Sect. 3.3). Assuming the number of L_l variables is $|\mathcal{L}|$, the accuracy loss is measured by averaging the L_l values.

Fairness Score ($O_{fair} := F$). We assign a variable F to represent the overall fairness score of the decision tree. The value of F is defined by formula Φ (Sect. 3.4) according to the definition of demographic parity.

Next, we present our encoding of formula $\Phi := \Phi_{tree} \wedge \Phi_{accu} \wedge \Phi_{fair}$, where Φ_{tree} encodes the hierarchical structure of the tree, Φ_{accu} encodes the accuracy requirement, and Φ_{fair} encodes the fairness requirement. They share variables with O_{tree}, O_{accu} and O_{fair} in the objective function, such as p_v, L_l, and F. Note that, since the constraint will be solved by an off-the-shelf `MIO` solver, Φ must be encoded as a conjunction of equality/inequality constraints. If *logical-or* operators are needed, they must be converted to equality/inequality operators.

3.2 Encoding of the Decision Tree (Φ_{tree})

Given a node, which may be the root of the decision tree under construction, or any of its branch nodes, we consider a depth-K *complete binary tree* rooted at that node. Since it is a complete binary tree, there are precisely $T_K = 2^{K+1} - 1$ nodes with indices $1 \ldots T_K$ and, for any node n, the left and right child nodes have indices $2n$ and $2n + 1$, respectively. Furthermore, the set of leaf nodes is $\mathcal{L} = \{2^K, 2^K + 1 \ldots 2^{K+1} - 1\}$, where $|\mathcal{L}| = 2^K$, and the set of branch nodes is $\mathcal{V} = \{1, 2 \ldots 2^K - 1\}$, where $|\mathcal{V}| = 2^K - 1$.

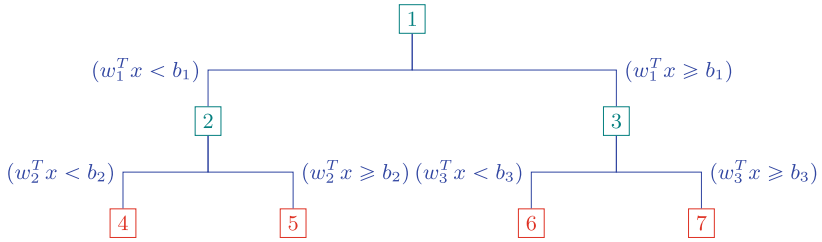

Fig. 3. Example of a complete binary tree, where $\mathcal{V} = \{1, 2, 3\}$ are branch nodes, $\mathcal{L} = \{4, 5, 6, 7\}$ are leaf nodes, and the decision thresholds b_1, b_2 and b_3 belong to $[0, 1]$. (Color figure online)

- Every leaf node $l \in \mathcal{L}$ has an output class label, and the path from root to l represents a classification rule, which assigns any input x that goes through the path to the output class.
- Every branch node $v \in \mathcal{V}$ has a vector w_v of bits for selecting the feature. Thus, at most one bit in w_v is 1, and $w_v[i] = 1$ means feature f_i is selected. For input x, the value of the selected feature is $f_i(x) = w_v^T x$.
- When node v is split by a feature, its outgoing edges are labeled $(w_v^T x < b_v)$ and $(w_v^T x \geqslant b_v)$, respectively. Here, $b_v \in (0, 1]$ is a symbolic threshold. When $f_i(x) = w_v^T x$ is a Boolean feature and $b_v = 1$, for example $(w_v^T x < 1)$ means $f_i(x) = 0$, and $(w_v^T x \geqslant 1)$ means $f_i(x) = 1$.

Figure 3 shows a depth-2 binary tree whose branch nodes are colored in *teal* and leaf nodes are colored in *red*. The thresholds b_1, b_2 and b_3 may be either 0 or a value in $(0, 1]$: only when they are non-zero, the corresponding nodes are split by features.

For instance, when b_2 is set to 1, if edge condition $(w_2^T x < 1)$ holds, input x goes to the left child, and if $(w_2^T x \geqslant 1)$ holds, x goes to the right child. When b_2 is set to 0, however, since edge condition $(w_2^T x < 0)$ is always false and $(w_2^T x \geqslant 0)$ is always true, input x always goes to the right child. In other words, $b_2 = 0$ disallows splitting at node $v = 2$.

Symbolic Variables. To model how a feature splits the training set, we define some symbolic variables first.

- **Input ($\mathbf{x}_{ij}$):** We use $\mathbf{x}_{ij}$ to model the j-th feature of the i-th input in $\mathcal{E}$. Thus, $i \in [1 \ldots n]$, $j \in [1 \ldots k]$, $n = |\mathcal{E}|$, and $k = |\mathcal{F}|$. The value of $\mathbf{x}_{i,j}$ may be any real number from 0 to 1, i.e., $\mathbf{x}_{i,j} \in [0, 1]$.
- **Split (p_v):** For every branch node $v \in \mathcal{V}$, we use p_v to model if v is split by a feature. The value of p_v is either 0 (no) or 1 (yes).
- **Selection (w_{vj}):** We use w_{vj} to model if the j-th feature is selected by node $v \in \mathcal{V}$. The value of w_{vj} is either 0 (no) or 1 (yes). Since both w and x are k-bit vectors, $w_v^T x$ is the value of the selected feature for a given input x.
- **Threshold (b_v):** We use b_v to control the activation of branch conditions at node $v \in \mathcal{V}$. When $b_v = 0$, input x always goes to the right child since

condition ($w_v^T x < 0$) is unsatisfiable. Otherwise, x goes to the left child when ($w_v^T x < b_v$), and to the right child when ($w_v^T x \geqslant b_v$).
- **Input Association (z_{it}):** We use z_{it} to model if the i-th input, x_i, is associated with node $t \in \{\mathcal{L} \vee \mathcal{V}\}$. The value of z_{it} is either 0 (no) or 1 (yes).
- **Empty Association (I_t):** For every leaf node $t \in \mathcal{L}$, we use I_t to model if t has any associated input. The value of I_t is either 0 (no) or 1 (some).

Formula Φ_{tree}. We define the formula as $\Phi_{tree} := \Pi_{split} \wedge \Pi_{edge} \wedge \Pi_{leaf} \wedge \Pi_{branch}$ where Π_{split} encodes how features are used to split branch nodes, Π_{edge} encodes the constraints on edges, Π_{leaf} encodes the constraints on leaf nodes, and Π_{branch} encodes the constraints on branch nodes.

Subformula Π_{split}. We construct Π_{split} by constraining p_v, w_{vj}, and b_v:

1. If $p_v = 1$, meaning $v \in \mathcal{V}$ is split, we require $(\sum_{j\in\{1,\ldots,k\}} w_{vj} = 1)$ to ensure exactly one feature is selected. We also require $(b_v > 0)$ to activate the branch conditions on the outgoing edges, $(w_v^T x < b_v)$ and $(w_v^T x \geqslant b_v)$.
2. If $p_v = 0$, meaning v is not split, we require $(\sum_{j\in\{1,\ldots,k\}} w_{vj} = 0)$ to ensure no feature is selected, and $(b_v = 0)$ to de-activate the left branch. That is, input x always goes to the right, while the left subtree stops growing.

Thus, we have $\Pi_{split} := \bigwedge_{v\in\mathcal{V}} \ (\sum_{j\in\{1,\ldots.k\}} w_{vj} = p_v) \wedge (0 \leqslant b_v \leqslant p_v)$.

Subformula Π_{edge}. We construct Π_{edge} by constraining the p_v variables: If node $v \in \mathcal{V}$ stops splitting, its child nodes also stop splitting. That is, when $p_v = 0$, both p_{2v} and p_{2v+1} must also be 0.
Thus, we have $\Pi_{edge} = \bigwedge_{v\in\mathcal{V}} \ (p_v \geqslant p_{2v}) \wedge (p_v \geqslant p_{2v+1})$.

Subformula Π_{leaf}. We construct Π_{leaf} by constraining variables z_{it} and I_t:

1. For each input x_i, where $i \in \{1, \ldots, n\}$ and $n = |\mathcal{E}|$, we require that x_i is associated with exactly one leaf node $l \in \mathcal{L}$, i.e., $(\sum_{l\in\mathcal{L}} z_{il} = 1)$.
2. If $I_l = 0$, meaning no input is associated with l, we require that $(z_{il} = 0)$ for all $i \in \{1, \ldots, n\}$. This is encoded as $\bigwedge_{l\in\mathcal{L}}(z_{il} \leqslant I_l)$.

Thus, we have $\Pi_{leaf} := \bigwedge_{i\in\{1,\ldots,n\}} \ (\sum_{l\in\mathcal{L}} z_{il} = 1) \wedge \bigwedge_{l\in\mathcal{L}}(z_{il} \leqslant I_l)$.

Subformula Π_{branch}. We construct Π_{branch} by constraining w_{vj}, b_v, and z_{it}:

1. In a complete binary tree, the depth-d nodes are $v \in \{2^d, \ldots, 2^{d+1} - 1\}$. Since exactly one of them is associated with input x_i, we require that condition $\Pi_{br1} := (\sum_{v\in\{2^d,\ldots,2^{d+1}-1\}} z_{iv} = 1)$ holds.
2. At each node $v \in \mathcal{V}$, since input x_i is associated with either the left child $L = 2v$ or the right child $R = 2v + 1$, but not both, we require that the following three conditions hold:
   - $\Pi_{br2} := \bigwedge_{v\in\{2^d,\ldots,2^{d+1}-1\}} \ (z_{iv} = z_{i(2v)} + z_{i(2v+1)})$
   - $\Pi_{br3} := \bigwedge_{v\in\{2^d,\ldots,2^{d+1}-1\}} \ (\sum_{j\in\{1,\ldots,k\}} w_{vj}x_{ij} - \gamma_L(1 - z_{iL}) < b_v)$
   - $\Pi_{br4} := \bigwedge_{v\in\{2^d,\ldots,2^{d+1}-1\}} \ (\sum_{j\in\{1,\ldots,k\}} w_{vj}x_{ij} + (1 - z_{iR}) \geqslant b_v)$

Thus, we have $\Pi_{branch} := \bigwedge_{i\in\{1,\ldots,n\}} \bigwedge_{d\in\{1,\ldots,K-1\}} (\Pi_{br1} \wedge \Pi_{br2} \wedge \Pi_{br3} \wedge \Pi_{br4})$.

Explanation of Π_{br3} and Π_{br4}. What we would like to encode in Π_{br3} is the fact that branch condition ($\sum w_{vj}x_{ij} < b_v$) may be either `TRUE` (x_i goes to the left child L when $z_{iL} = 1$ and $b_v \in (0, 1]$) or `FALSE` (x_i goes to the right child R when $z_{iL} = 0$ and $b_v \in (0, 1]$, or when $b_v = 0$). However, since off-the-shelf `MIO` solvers do not support *logical-or* operators, we have to encode these different scenarios in a single inequality constraint. This is accomplished by adding a slack value, $-\gamma_L(1 - z_{iL})$, to the branch condition. Similarly, in Π_{br4}, we add a slack value, $(1 - z_{iR})$, to the branch condition ($\sum w_{vj}x_{ij} \geqslant b_v$).

3.3 Encoding of the Accuracy Requirement (Φ_{accu})

To minimize the accuracy loss defined in $O_{accu} := \frac{1}{|\mathcal{L}|} \sum_{l\in\mathcal{L}} L_l$ (Sect. 3.1), we need to constrain the L_l variables in Φ_{accu} such that L_l models the misclassification error at the leaf node $l \in \mathcal{L}$. In the depth-K complete binary tree, there are $|\mathcal{L}| = 2^K$ leaf nodes. For each leaf node l, variable L_l represents the number of misclassified examples $(x_i, y_i) \in \mathcal{E}$: it is misclassified if the given output y_i does not match the predicted output $\mathcal{T}(x_i)$.

The formula $\Phi_{accu} := \Phi_p \wedge \Phi_N \wedge \Phi_\theta \wedge \Phi_{loss}$ consists of four subformulas.

Subformula Φ_P. For each $(x_i, y_i) \in \mathcal{E}$, where $i \in \{1, \ldots, n\}$ and $n = |\mathcal{E}|$, and for each output value $m \in \{0, 1\}$, we use p_{im} to model if $(y_i = m)$. The value of p_{im}, which is either 0 or 1, is $const_{im} := (y_i = m)\ ?\ 0 : 1$.

Thus, we have $\Phi_p := \bigwedge_{i=1}^{n} \bigwedge_{m=0}^{1} (p_{im} = const_{im})$.

Subformula Φ_N. We use variable N_l to represent the number of examples associated with leaf node l, and N_{lm} to represent those with output value m.

Thus, we have $\Phi_N := \bigwedge_{l\in\mathcal{L}} (N_l = \sum_{i=1}^{n} z_{il}) \wedge (N_{lm} = \frac{1}{2} \sum_{i=1}^{n} z_{il}(1 + p_{im}))$.

Subformula Φ_θ. According to Lines 5–8 of Algorithm 1, each leaf node has an output class label $\theta_l = \mathbf{argmax}_{m\in\{0,1\}}\ N_{lm}$. Since **argmax** cannot be directly encoded, we define a matrix of θ_{lm} variables in $\{0, 1\}$, where $\theta_{lm} = 1$ means the output label of node l is m. By definition, only one θ_{lm} variable can be 1.

Thus, we have $\Phi_\theta := \bigwedge_{l\in\mathcal{L}}\ (\sum_{m\in\{0,1\}}\ \theta_{lm} = 1)$.

Subformula Φ_{loss}. Assuming that m is the output label predicted by the leaf node l. The misclassification error, L_l, is equal to the number of examples associated with l, denoted N_l, minus the number of examples that have the most common label m, denoted $\mathbf{max}_{m\in\{0,1\}} N_{lm}$.

To avoid **max/min** in $L_l = N_l - \mathbf{max}_{m\in\{0,1\}} N_{lm} = \mathbf{min}_{m\in\{0,1\}}(N_l - N_{lm})$, we use θ_{lm} variables and constant $n = |\mathcal{E}|$ to rewrite the constraint as :

$$(L_l \geqslant 0) \wedge \bigwedge_{m\in\{0,1\}} (L_l \geqslant N_l - N_{lm} - n(1 - \theta_{lm})) \wedge (L_l \leqslant N_l - N_{lm} + n\theta_{lm})$$

Thus, we have $\Phi_{loss} := \bigwedge_{l\in\mathcal{L}}((L_l \geqslant 0) \wedge \bigwedge_{m\in\{0,1\}} (L_l \geqslant N_l - N_{lm} - n(1-\theta_{lm})) \wedge (L_l \leqslant N_l - N_{lm} + n\theta_{lm}))$.

3.4 Encoding of the Fairness Requirement

Formula $\Phi_{fair} := \Phi_{F_s} \wedge \Phi_{FM}$ has two subformulas. Here, Φ_{F_s} encodes the fairness index and Φ_{FM} encodes the constraints on variables used in Φ_{F_s}.

According to Eq. 1 (Sect. 2.3), the fairness index is defined as $F_s = (Pr^+_{\neg f_s}/Pr^+_{f_s})$, where f_s is a sensitive feature such that $f_s(x)$, for any input $x \in \mathcal{E}$, may be 0 or 1 (e.g., female and male) while $\mathcal{T}(x) = +$ means the output generated by $\mathcal{T}$ is positive (e.g., a job is offered). According to the *"80% rule"*, demographic parity is achieved if F_s is above 80%. In this work, our goal is to find a solution that (1) satisfies ($F_s > 0.8$) and, at the same time (2) maximizes the value of F_s.

However, the definition of F_s shown in Eq. 1 has division operators, which are not supported by off-the-shelf `MIO` solvers. Furthermore, the divisor part of the equation varies even for a fixed set $\mathcal{E}$ of examples, which makes the encoding a challenging task. To overcome the challenge, we refine the definition of as follows:

$$\frac{Pr^+_{f_s=0}}{Pr^+_{f_s=1}} = \frac{|\{x \in \mathcal{E} \mid f_s(x) = 0, \mathcal{T}(x) = +\}| \;/\; |\{x \in \mathcal{E} \mid f_s(x) = 0\}|}{|\{x \in \mathcal{E} \mid f_s(x) = 1, \mathcal{T}(x) = +\}| \;/\; |\{x \in \mathcal{E} \mid f_s(x) = 1\}|} = \frac{S^+_0/S_0}{S^+_1/S_1} \tag{2}$$

For each of the four components, we create a symbolic variable. Variable S_0 represents the number of examples whose sensitive feature has the value 0 (e.g., *female*) for the *gender (f_1)* feature. Variable S^+_0 represents the number of examples in S_0 that have the positive output (e.g., a job is offered). Variable S_1 represents the number of examples whose sensitive feature has the value 1 (e.g., *male*) for the *gender (f_1)* feature. Variable S^+_1 represents the number of examples in S_1 that have the positive output.

Subformula Φ_{F_s}. We use Φ_{F_s} to enforce the *80% rule*: $F_s = \frac{S^+_0/S_0}{S^+_1/S_1} \geqslant 0.8$. Assuming $S_0 > 0$, $S^+_0 > 0$, $S_1 > 0$, and $S^+_1 > 0$, we encode the rule as follows:

$$\Phi_{F_s} := (S^+_0 \times S_1 - 0.8 \times S_0 \times S^+_1 \geqslant 0)$$

There are two advantages of this encoding. First, the resulting constraint can be solved by off-the-shelf `MIO` solvers, whereas a direct encoding of Eq. 2 cannot. Second, the value of $(S^+_0 \times S_1 - 0.8 \times S_0 \times S^+_1)$ increases as F_s increases; therefore, it can be used as part of the objective function, O_{fair}, to maximize F_s.

Subformula Φ_{FM}. We use Φ_{FM} to constrain the variables S_0, S^+_0, S_1, and S^+_1. Toward this end, we need to define the following variables:

- S_{0i}*:* We use variable $S_{0i} \in \{0,1\}^n$ to model if the value of $f_s(x_i)$ is 0. Thus, we require $S_{0i} = 1$ when $f_s(x_i) = 0$, and $S_{0i} = 0$ otherwise.
- S^+_{0il}*:* We use variable $S^+_{0il} \in \{0,1\}^{n \times |\mathcal{L}|}$ to model, at each leaf node $l \in \mathcal{L}$, if $x_i \in \mathcal{E}$ is given the positive output. Thus, we require $S^+_{0il} = 1$ when the following condition holds, and $S^+_{0il} = 0$ otherwise:

  $$(\theta_{lm} = 1 \;\wedge\; m = 1 \;\wedge\; z_{il} = 1 \;\wedge\; S_{0i} = 1)$$

  In the condition above, $(\theta_{lm} = 1)$ means the output label produced by the leaf node l is m, and $(m = 1)$ means m is the positive output ("+").

– S_{1i} and S_{1il}^{+}: We define variables S_{1i} and S_{1il}^{+} similar to S_{0i} and S_{0il}^{+}.

Thus, we have $\Phi_{FM} := (S_0 = \sum_{i\in\{1,\ldots,n\}} S_{0i}) \wedge (S_0^{+} = \sum_{i\in\{1,\ldots,n\}} \sum_{l\in\mathcal{L}} S_{0il}^{+}) \wedge (S_1 = \sum_{i\in\{1,\ldots,n\}} S_{1i}) \wedge (S_1^{+} = \sum_{i\in\{1,\ldots,n\}} S_{1il}^{+})$.

Putting It All Together. Recall that, in Sect. 3.3, we have constrained the accuracy loss, L_l, in the objective function O_{accu}, and defined the objective function O_{tree} in Sect. 3.1, which is used to minimize the tree size and thus reduce over-fitting. As for the objective function O_{fair} (Sect. 3.1), we define the fairness score as follows: $F = (S_0^{+} \times S_1 - 0.8 \times S_0 \times S_1^{+})$.

Thus, we have the entire MIO problem as follows:

$$\begin{aligned} &\textit{minimize} && \frac{1}{|\mathcal{L}|}\sum_{l\in\mathcal{L}} L_l + \alpha \sum_{v\in\mathcal{V}} p_v - \beta F \\ &\textit{subject to} && \Phi_{accu}(L_l) \wedge \Phi_{tree}(p_v) \wedge \Phi_{fair}(F) \end{aligned} \tag{3}$$

4 Generalization and Performance Enhancement

In this section, we first explain how our method relates to various existing algorithms (Sect. 4.1). Next, we present techniques for speeding up constraint solving while maintaining the quality of the solution (Sect. 4.2). Finally, we show that, beyond *demographic parity*, our method can encode other group fairness metrics, such as *equal opportunity* and *equal odds* (Sect. 4.3).

4.1 Relating to Existing Algorithms

Recall that our method performs feature selection by symbolically encoding a depth-K binary tree, to perform a bounded look-ahead search of the optimal feature using the MIO solver. For ease of presentation, let us call the selected feature *depth-K optimal*, where $K \in \{1, \ldots, +\infty\}$.

Depth-1 Optimal. When $K = 1$, the tree consists of the root node only and, as a result, look-ahead search is disabled. In this case, our method is the same as a purely greedy search method. Depending on whether fairness is encoded, there are two cases.

– Without the fairness component, our method would compute the *depth-1 optimal* feature that minimizes only the tree size and the accuracy loss. This is similar to mainstream decision tree learning algorithms such as CART.
– With the fairness component, our method would compute the *depth-1 optimal* feature that minimizes the tree size and the accuracy loss, and maximizes the fairness score. This is similar to IGCS [24], an discrimination-aware technique for learning decision trees.

Our experimental evaluation (in Sect. 5) shows that neither CART nor IGCS is effective in improving fairness, especially for larger datasets, primarily due to their inability to look beyond the current node.

Depth-∞ Optimal. When K is set to a sufficiently-large number, our method is able to find the globally optimal feature for not only the root node, but also other nodes in the decision tree. Thus, it would compute the entire decision tree in one shot.

- Without the fairness component, our method would act like the technique introduced by Bertsimas and Dunn [7], which laid the ground work for encoding an optimal classification tree as a monolithic `MIO` problem.
- With the fairness component, our method would act like `MIP`, a fair learning technique introduced by Aghaei et al. [1].

Our experimental evaluation (in Sect. 5) shows that the computational overhead of the monolithic `MIO` approach or `MIP` is too high to be practically useful. We discuss how to set the value of K in our method in the next subsection.

4.2 Performance Enhancement

We propose two techniques for speeding up our method by (1) choosing the K value adaptively and (2) sampling the training examples in $\mathcal{E}$.

Choosing the K Value Adaptively. There is a trade-off between looking further ahead and reducing the constraint solving time. Given $n = |\mathcal{E}|$ training examples, and 2^K leaf nodes in a depth-K binary tree, the number of decision variables (such as S_{0il}) would be $(n \times 2^K)$. Since mixed-integer optimization is NP-hard, the complexity of constraint solving is $O(2^{n \times 2^K})$. Empirically, we have found that Gurobi, a state-of-the-art solver, may take 1–2 h to solve a problem for $n = 1000$ training examples and tree depth $K = 7$—this is consistent with prior experimental results, e.g., Bertsimas and Dunn [7]. Unfortunately, supervised learning datasets in practice often bring as many as 50,000 training examples to the root node of a decision tree, although the number decreases gradually and may reach 0 for some leaf nodes. Therefore, setting K to 7, or any predetermined value, would not work well in practice.

Instead, we propose to set the K value adaptively. Given a time-out limit (`T/O`) for learning a decision tree, we start with a relatively small K value, say $K = 2$, to synthesize a decision tree. Then, we increase the K value to synthesize a better decision tree. We keep increasing the K value as long as the time limit is not yet reached, and the quality of the decision tree is improved. We measure the quality of the tree using the value of the objective function, O, which consists of the tree size, the accuracy loss, and the fairness score.

Sampling the Training Examples. We propose to reduce the size of the constraints in Φ by sampling the training examples in $\mathcal{E}$, before using them to construct the formula Φ. Our experience shows that sampling can reduce the value of n significantly and, at the same time, maintaining the quality of the `MIO` solution. For the `adult` dataset, which has 48,842 training examples, even with a small K value, the symbolic constraints would take more than 1 h to solve.

Algorithm 4. Subroutine FindNextFeature$(\mathcal{E}, \mathcal{F})$ with our enhancement.

1: Let f_s be the sensitive feature
2: **if** $|\mathcal{E}| \leqslant 8000$ **then** $(O, \Phi) =$ DtlEncoding$(\mathcal{E}, \mathcal{F}, f_s)$
3: **else** $(O, \Phi) =$ DtlEncoding$(\mathcal{E}|_{sampled}, \mathcal{F}, f_s)$
4: **return** $f^* =$ MioSolver(O, Φ)

Empirically, we have observed that the feature computed by depth-K look-ahead using 8,000 randomly-chosen examples is almost as good as the feature computed using all examples. Based on this observation, we set the threshold ($n \leqslant 8000$), i.e., at most 8,000 examples from $\mathcal{E}$ are used in the symbolic constraints in Algorithm 4, where $\Phi =$ DtlEncoding$(\mathcal{E}, \mathcal{F}, f_s)$ is invoked if $|\mathcal{E}| \leqslant 8000$. Otherwise, $\mathcal{E}$ is replaced by the randomly-sampled subset $\mathcal{E}\,|_{sampled}$.

Our sampling method is not directly applicable to the original MIP approach because, if sampled data are used as input, the MIP solving procedure would permanently discard the rest of the data, which would significantly degrade its accuracy. In contrast, sampling in our method only causes the rest of the data to be ignored temporarily (for this particular node) but, for the child nodes in the subtree, the entire data will still be used in the subsequent computation.

4.3 Encoding Other Group Fairness Metrics

Beyond *demographic parity*, there are two popular metrics for group fairness, of which one is *equal opportunity* and the other is *equalized odds*.

Equal Opportunity. In addition to the sensitive feature f_s, there is a decision-critical feature f_c. Let $P^+_{f_s=0,f_c=1} = \frac{|x \in \mathcal{E} \mid f_s(x)=0,\ f_c(x)=1,\ \mathcal{T}(x)=+|}{|x \in \mathcal{E} \mid f_s(x)=0,\ f_c(x)=1|} = \frac{S^+_0}{S_0}$ and $P^+_{f_s=1,f_c=1} = \frac{|x \in \mathcal{E} \mid f_s(x)=1,\ f_c(x)=1,\ \mathcal{T}(x)=+|}{|x \in \mathcal{E} \mid f_s(x)=1,\ f_c(x)=1|} = \frac{S^+_1}{S_1}$. A decision tree $\mathcal{T}$ satisfies *equal opportunity* if the following condition holds (for a small ϵ).

$$P^+_{f_s=1,f_c=1} - P^+_{f_s=0,f_c=1} \leqslant \epsilon \tag{4}$$

In our method, Eq. 4 may be encoded as $\Phi_{eq} := S^+_1 S_0 - S^+_0 S_1 - \epsilon S_0 S_1 \leqslant 0$, to replace Φ_{F_s} in the fairness requirement $\Phi_{fair} := \Phi_{F_s} \wedge \Phi_{FM}$. The definitions of variables S_0, S^+_0, S_1 and S^+_1 are analogous to that in Sect. 3.4. Similarly, we can define fairness decision variables S_{0i}, S_{0il}, S_{1i}, and S_{1il}. For example, the value of S_{0i} is set to 1 if $f_s(x_i) = 0 \wedge f_c(x_i) = 1$ and is set to 0 otherwise.

Equalized Odds. To satisfy *equalized odds*, we must satisfy Eq. 4, as well as the condition below:

$$P^+_{f_s=1,f_c=0} - P^+_{f_s=0,f_c=0} \leqslant \epsilon. \tag{5}$$

Since Eq. 5 can be encoded similarly to Eq. 4, the details are omitted for brevity.

Table 1. Comparing our method with existing methods on small benchmarks.

Benchmark	SFTree (ours)		CART [27]		IGCS [24]		MIP [1]	
	Accuracy	Fairness	Accuracy	Fairness	Accuracy	Fairness	Accuracy	Fairness
German Fold1	77.5%	0.82	83.0%	0.65	74.0%	0.84	80.5%	0.82
German Fold2	80.5%	0.81	85.0%	0.67	78.5%	**0.78**	83.5%	0.82
German Fold3	76.0%	0.84	79.0%	0.71	73.5%	0.80	78.5%	0.84
German Fold4	81.0%	0.80	83.5%	0.65	76.0%	0.84	81.0%	0.89
German Fold5	80.5%	0.81	85.0%	0.66	77.0%	0.81	83.0%	0.81
Salary Fold1	81.8%	0.82	90.9%	0.59	81.8%	0.82	81.8%	0.82
Salary Fold2	72.7%	0.83	90.9%	0.57	81.8%	**0.77**	81.8%	0.84
Salary Fold3	72.7%	0.83	81.8%	0.62	72.7%	0.83	81.8%	0.83
Salary Fold4	81.8%	0.82	90.9%	0.61	81.8%	0.82	81.8%	0.82
Salary Fold5	81.8%	0.81	81.8%	0.57	72.7%	**0.73**	72.7%	0.83
Student Fold1	71.2%	0.84	75.9%	0.58	72.1%	**0.78**	72.8%	0.87
Student Fold2	70.3%	0.81	75.1%	0.63	69.3%	0.82	72.8%	0.85
Student Fold3	70.9%	0.81	73.6%	0.57	71.4%	0.81	73.6%	0.85
Student Fold4	69.1%	0.82	75.1%	0.61	69.3%	**0.77**	71.3%	0.84
Student Fold5	71.5%	0.84	77.5%	0.53	72.0%	0.81	75.1%	0.84

5 Experiments

We have implemented our method, SFTree, using Python, Julia 1.5.1 [15], and Gurobi 9.03 [21], where Julia is used to encode the `MIO` constraints and Gurobi is used to solve the constraints. We compared SFTree with three state-of-the-art techniques: `CART`, which is a mainstream algorithm for decision tree learning, `IGCS`, which is a discrimination-aware learning algorithm, and `MIP`, which is a monolithic `MIO` approach to learning fair tress. We conducted all experiments with Catalina running on a macOS with 2.4 GHz 8-Core CPU and 64G RAM.

Benchmarks. Our evaluation uses six popular benchmarks from the fairness literature. They are divided to three small datasets and three large datasets. Since the small datasets can be handled by the less-scalable but more-accurate `MIP` to obtain globally optimal solutions, they are useful in evaluating the quality of our method. The large datasets, in contrast, are out of the reach of `MIP` and thus useful in evaluating the scalability of our method.

- Among the small datasets, `German` [23] (predicting credit risks) has 1000 training examples and 20 features; `Student` [12] (predicting student performance) has 649 training examples and 33 features; and `Salary` [36] (predicting the salary level) has 52 training examples and 16 features. In these datasets, the sensitive feature is *gender*.
- Among the large datasets, `Adult` [14] (predicting the earning power) has 48,842 training examples and 14 features (with *race* as the sensitive feature); `Default` [37] (predicting loan default risk) has 30,000 training examples and 23 features (with *gender* as the sensitive feature); and `Compas` [13] (predicting the recidivism risk) has 10,500 training examples and 16 features (with *race* as the sensitive feature).

Table 2. Comparing our method with existing methods on large benchmarks.

Benchmark	SFTree (ours)		CART [27]		IGCS [24]		MIP [1]	
	Accuracy	Fairness	Accuracy	Fairness	Accuracy	Fairness	Accuracy	Fairness
Adult Fold1	80.3%	0.81	83.0%	0.54	82.8%	**0.51**	-	-
Adult Fold2	77.4%	0.86	80.0%	0.57	81.9%	**0.68**	-	-
Adult Fold3	75.7%	0.84	79.8%	0.57	81.3%	**0.72**	-	-
Adult Fold4	78.1%	0.83	82.1%	0.55	83.0%	**0.62**	-	-
Adult Fold5	77.1%	0.86	82.6%	0.55	75.7%	**0.68**	-	-
Default Fold1	80.5%	0.81	84.7%	0.64	81.3%	**0.77**	-	-
Default Fold2	84.7%	0.81	86.3%	0.61	84.0%	**0.73**	-	-
Default Fold3	80.5%	0.83	83.2%	0.66	82.7%	**0.75**	-	-
Default Fold4	78.8%	0.85	84.1%	0.64	81.5%	**0.73**	-	-
Default Fold5	81.4%	0.82	83.9%	0.64	81.7%	**0.71**	-	-
Compas Fold1	86.4%	0.89	92.8%	0.63	86.7%	0.81	-	-
Compas Fold2	89.8%	0.96	92.5%	0.61	87.5%	0.83	-	-
Compas Fold3	85.3%	0.94	90.4%	0.67	88.9%	**0.74**	-	-
Compas Fold4	87.2%	0.96	92.6%	0.63	92.0%	**0.61**	-	-

During learning, we apply the standard 5-fold cross validation expect for `Compas`, to which we apply 4-fold cross validation to be consistent with prior work.

Results on the Small Benchmarks. We compare the quality of the decision trees learned by our method and three existing methods on the small benchmarks. The results are shown in Table 1, where Column 1 shows name of the dataset, Columns 2–3 shows the result of our method in terms of accuracy and fairness, computed by cross-validation, Columns 4–5 show the result of `CART`, Columns 6–7 show the result of `IGCS`, and Columns 8–9 show the result of `MIP`. Since the datasets are small, `MIP` is able to compute the best solutions: without violating the *80% Rule*, it maximizes accuracy.

The result shows that, overall, `CART` has the best accuracy but the worst fairness score. `IGCS` improves over `CART`, but still violates the *80% Rule* in 5 out of the 15 cases. In contrast, `SFTree` satisfies the fairness requirement in all 15 cases and, at the same time, achieves high accuracy. Furthermore, it runs more than 10 times faster than `MIP`.

Results on the Large Benchmarks. We use these benchmarks to evaluate both the quality and the scalability of our method. Table 2 shows the result of the quality comparison, which has the same format as Table 1. `CART` has the highest accuracy but fails to satisfy the fairness requirement in all 14 cases. Although `IGCS` is somewhat effective for the small benchmarks in Table 1, here, it fails to satisfy the fairness requirement in 12 of the 14 cases. In contrast, our method is the only one that satisfies the fairness requirement in all cases and, at the same time, has accuracy comparable to `CART` and `IGCS`.

Table 3 shows the execution time comparison. `MIP` times out in all 14 cases (`T/O = 3h`), while our method finishes each within 1 h. Thus, our method runs more than 10 times faster than `MIP`. Although `CART` and `IGCS` are faster, they are equivalent to depth-1 look-ahead search in our method and, due to the limited ability to look ahead, they almost never satisfy the fairness requirement.

Table 3. Comparing the run time of methods on large benchmarks (`T/O = 3h`).

Benchmark	SFTree	CART [27]	IGCS [24]	MIP [1]	Benchmark	SFTree	CART [27]	IGCS [24]	MIP [1]
Adult Fold1	2064s	39s	40s	T/O	Default Fold1	2499s	28s	28s	T/O
Adult Fold2	2119s	39s	39s	T/O	Default Fold2	2478s	29s	29s	T/O
Adult Fold3	2075s	39s	40s	T/O	Default Fold3	2526s	29s	29s	T/O
Adult Fold4	2090s	39s	40s	T/O	Default Fold4	2536s	28s	29s	T/O
Adult Fold5	2091s	39s	39s	T/O	Default Fold5	2531s	28s	29s	T/O
Compas Fold1	2115s	15s	16s	T/O	Compas Fold2	2137s	15s	15s	T/O
Compas Fold3	2129s	15s	15s	T/O	Compas Fold4	2166s	15s	15s	T/O

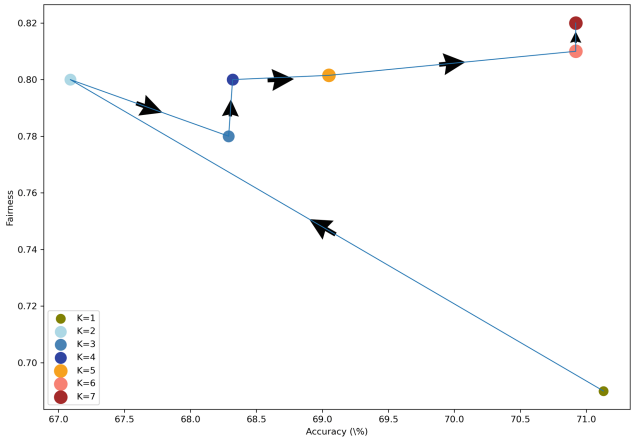

Fig. 4. How accuracy and fairness of the learned decision tree change with the K value for the *Student* dataset. For each $K = 1, \ldots, 7$, we plot the fairness and accuracy scores.

Evaluating the Impact of the K-value. We have also evaluated how the K value affects the quality of the learned decision tree using the `Student Fold1` benchmark. Since the benchmark is small enough, we set K to fixed values $1, \ldots, 7$ instead of letting it adapt, so we can assess the impact. Figure 4 shows the result, where the x-axis is accuracy and the y-axis is the fairness score. Thus, the closer a dot is to the right-top corner, the higher the overall quality is. The result shows that the quality of our solution increases dramatically as the K value increases from 1 to 7, due to the increasingly deeper look-ahead search.

Summary of Additional Results. While we have also evaluated the scalability of our method with respect to the dataset size, we omit the results for brevity and instead provide a summary. What we have found is that, as the dataset gets larger, the execution time of our method increases modestly at first, and then

stops increasing after a threshold is reached. This is due to the use of performance enhancement techniques presented in Sect. 4. Thus, our method does not have scalability issues. In fact, among all four methods, SFTree is the only one that consistently produces fair and accurate decision trees for datasets with >40,000 training samples.

6 Related Work

At a high level, our method can be viewed as an *in-processing* approach to mitigating bias in machine learning models. Broadly speaking, there are three approaches: *pre-processing* [17,25,31], *in-processing* [11,19,24,30,33] and *post-processing* [18,22], depending on whether the focus is on de-biasing the training data, the learning algorithm, or the classification output.

Since the *pre-processing* approach focuses on de-biasing the training data [17, 25,31], it is applicable to any machine learning model; however, it cannot remove bias introduced by the learning algorithms, which is problematic because, even if the training data is not biased, learning algorithms may introduce new bias. While the *post-processing* approach can remove such bias by modifying the predicted output [18,22], the result is often hard to predict and difficult to explain. In contrast, our method does not have these limitations.

Compared to other *in-processing* techniques for fair learning decision trees, including `IGCS` [24] and similar greedy search methods [11,19,30,33], our method has the advantage of being more systematic and quantifiable. This is because we encode both accuracy and fairness requirements explicitly as numerical constraints. Thus, it would be easy to explain, at every step, why a feature is chosen over another feature, and quantify how much more effective it is in minimizing bias and accuracy loss at the same time. Compared to the monolithic constraint solving approach, including `MIP` [1] and similar methods [5,35], our method has the advantage of being significantly more scalable.

Our method differs from the recent work of Torfah et al. [32] in that their method uses a small training set sampled from a known distribution and thus does not need techniques such as incremental solving. Furthermore, their method assumes the decision predicates are given, but in our method, the predicates are synthesized from real-valued features. Finally, our fairness constraint is also different from the explainability constraint.

Besides synthesis, there are techniques for improving fairness by repairing an existing machine learning model [4,9,20,26], and techniques for verifying that an existing machine learning model is indeed fair, e.g., by using probabilistic analysis methods [3,6,28]. While these techniques are related, they differ from our method in that they cannot synthesize new decision trees from training data while ensuring the decision trees are fair by construction.

7 Conclusion

We have presented a method for synthesizing a fair and accurate decision tree, by formulating feature section as a series of mixed-integer optimization problems and solve them using an off-the-shelf constraint solver. The method is flexible in expressing group fairness metrics including *demographic parity*, *equal opportunity*, and *equal odds*. On popular datasets, it is able to learn decision trees that satisfy the fairness requirement and, at the same time, achieve a high classification accuracy.

References

1. Aghaei, S., Azizi, M.J., Vayanos, P.: Learning optimal and fair decision trees for non-discriminative decision-making. In: AAAI Conference on Artificial Intelligence (2019)
2. Aghaei, S., Gómez, A., Vayanos, P.: Strong optimal classification trees. CoRR abs/2103.15965 (2021). https://arxiv.org/abs/2103.15965
3. Albarghouthi, A., D'Antoni, L., Drews, S., Nori, A.V.: FairSquare: probabilistic verification of program fairness. In: Proceedings of the ACM on Programming Languages (OOPSLA), pp. 1–30 (2017)
4. Albarghouthi, A., D'Antoni, L., Drews, S.: Repairing decision-making programs under uncertainty. In: International Conference on Computer Aided Verification (2017)
5. Azizi, M.J., Vayanos, P., Wilder, B., Rice, E., Tambe, M.: Designing fair, efficient, and interpretable policies for prioritizing homeless youth for housing resources. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (2018)
6. Bastani, O., Zhang, X., Solar-Lezama, A.: Probabilistic verification of fairness properties via concentration. In: Proceedings of the ACM on Programming Languages (OOPSLA) (2019)
7. Bertsimas, D., Dunn, J.: Optimal classification trees. Mach. Learn. **106**(7), 1039–1082 (2017)
8. Biddle, D.: Adverse Impact and Test Validation: A Practitioner's Guide to Valid and Defensible Employment Testing. Routledge, London (2017)
9. Bolukbasi, T., Chang, K.W., Zou, J.Y., Saligrama, V., Kalai, A.T.: Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In: Advances in Neural Information Processing Systems (2016)
10. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Routledge, New York (2017)
11. Calders, T., Kamiran, F., Pechenizkiy, M.: Building classifiers with independency constraints. In: IEEE International Conference on Data Mining Workshops (2009)
12. Cortez, P., Silva, A.M.G.: Using data mining to predict secondary school student performance (2008)
13. Dieterich, W., Mendoza, C., Brennan, T.: COMPAS risk scales: demonstrating accuracy equity and predictive parity. Northpointe Inc. (2016)
14. Dua, D., Karra Taniskidou, E.: UCI machine learning repository. School of Information and Computer Science (2017). http://archive.ics.uci.edu/ml
15. Dunning, I., Huchette, J., Lubin, M.: JuMP: a modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)

16. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: Innovations in Theoretical Computer Science (2012)
17. Feldman, M., Friedler, S.A., Moeller, J., Scheidegger, C., Venkatasubramanian, S.: Certifying and removing disparate impact. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2015)
18. Fish, B., Kun, J., Lelkes, Á.D.: A confidence-based approach for balancing fairness and accuracy. In: SIAM International Conference on Data Mining (2016)
19. Friedler, S.A., Scheidegger, C., Venkatasubramanian, S., Choudhary, S., Hamilton, E.P., Roth, D.: A comparative study of fairness-enhancing interventions in machine learning. In: Conference on Fairness, Accountability, and Transparency (2019)
20. Grari, V., Ruf, B., Lamprier, S., Detyniecki, M.: Achieving fairness with decision trees: an adversarial approach. Data Sci. Eng. **5**(2), 99–110 (2020)
21. Gurobi Optimization, LLC: Gurobi optimizer reference manual (2021). https://www.gurobi.com
22. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. In: Annual Conference on Neural Information Processing Systems (2016)
23. Hofmann, H.: Statlog (German Credit Data) Data Set (2021). https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
24. Kamiran, F., Calders, T., Pechenizkiy, M.: Discrimination aware decision tree learning. In: IEEE International Conference on Data Mining (2010)
25. Kamiran, F., Karim, A., Zhang, X.: Decision theory for discrimination-aware classification. In: IEEE International Conference on Data Mining (2012)
26. Kamishima, T., Akaho, S., Asoh, H., Sakuma, J.: Fairness-aware classifier with prejudice remover regularizer. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012. LNCS (LNAI), vol. 7524, pp. 35–50. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33486-3_3
27. Lewis, R.J.: An introduction to classification and regression tree (CART) analysis. In: Annual Meeting of the Society for Academic Emergency Medicine (2000)
28. Meyer, A., Albarghouthi, A., D'Antoni, L.: Certifying robustness to programmable data bias in decision trees. In: Advances in Neural Information Processing Systems (2021)
29. Quinlan, J.R.: C4.5: Programs for Machine Learning. Elsevier, Amesterdam (2014)
30. Raff, E., Sylvester, J., Mills, S.: Fair forests: regularized tree induction to minimize model bias. In: AAAI/ACM Conference on AI, Ethics, and Society (2018)
31. Thanh, B.L., Ruggieri, S., Turini, F.: k-NN as an implementation of situation testing for discrimination discovery and prevention. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2011)
32. Torfah, H., Shah, S., Chakraborty, S., Akshay, S., Seshia, S.A.: Synthesizing pareto-optimal interpretations for black-box models. In: International Conference on Formal Methods in Computer Aided Design (2021)
33. Valdivia, A., Sánchez-Monedero, J., Casillas, J.: How fair can we go in machine learning? Assessing the boundaries of accuracy and fairness. Int. J. Intell. Syst. **36**(4), 1619–1643 (2021)
34. Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: International Conference on Machine Learning (2018)
35. Verwer, S., Zhang, Y.: Learning decision trees with flexible constraints and objectives using integer optimization. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2017)

36. Weisberg, S.: Applied Linear Regression. Wiley, Hoboken (2005)
37. Yeh, I.C., Lien, C.H.: The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Exp. Syst. Appl. **36**(2), 2473–2480 (2009)
38. Zafar, M.B., Valera, I., Rogriguez, M.G., Gummadi, K.P.: Fairness constraints: mechanisms for fair classification. In: Artificial Intelligence and Statistics (2017)