Optimal Decision Diagrams for Classification

Alexandre M. Florio[1,2*], **Pedro Martins**[3*], **Maximilian Schiffer**[4*], **Thiago Serra**[5*], **Thibaut Vidal**[1,2*]

[1] CIRRELT & SCALE-AI Chair in Data-Driven Supply Chains,
[2] Department of Mathematical and Industrial Engineering, Polytechnique Montréal, Canada,
[3] Department of Computer Science, Pontifical Catholic University of Rio de Janeiro, Brazil,
[4] School of Management & Munich Data Science Institute, Technical University of Munich, Germany,
[5] Freeman College of Management, Bucknell University, USA
aflorio@gmail.com, pmartins@inf.puc-rio.br, schiffer@tum.de, thiago.serra@bucknell.edu, thibaut.vidal@polymtl.ca

Abstract

Decision diagrams for classification have some notable advantages over decision trees, as their internal connections can be determined at training time and their width is not bound to grow exponentially with their depth. Accordingly, decision diagrams are usually less prone to data fragmentation in internal nodes. However, the inherent complexity of training these classifiers acted as a long-standing barrier to their widespread adoption. In this context, we study the training of optimal decision diagrams (ODDs) from a mathematical programming perspective. We introduce a novel mixed-integer linear programming model for training and demonstrate its applicability for many datasets of practical importance. Further, we show how this model can be easily extended for fairness, parsimony, and stability notions. We present numerical analyses showing that our model allows training ODDs in short computational times, and that ODDs achieve better accuracy than optimal decision trees, while allowing for improved stability without significant accuracy losses.

Introduction

Decision diagrams, also known as decision graphs or decision streams, have a long history in logic synthesis and formal circuit verification (Lee 1959; Bryant 1986, 1992) as well as in optimization (Behle 2007; Bergman et al. 2016; Lange and Swoboda 2021) and artificial intelligence topics such as planning (Sanner, Uther, and Delgado 2010; Castro et al. 2019), knowledge compilation (Abío et al. 2012; Lai, Liu, and Wang 2013; Serra 2020), and constraint propagation (Andersen et al. 2007; Perez and Régin 2015; Verhaeghe, Lecoutre, and Schaus 2018). In machine learning, decision diagrams have recurrently emerged as a possible classification model (Oliver 1993; Oliveira and Sangiovanni-Vincentelli 1996; Mues et al. 2004; Shotton et al. 2013; Ignatov and Ignatov 2018) or as a by-product of model compression algorithms applied on decision trees (Breslow and Aha 1997; Gossen and Steffen 2019; Choudhary et al. 2020). A decision diagram for classification is represented as a rooted directed acyclic graph in which each internal node represents a splitting hyperplane, and each terminal node is uniquely associated to a class. The topology of the graph remains a free parameter of the model, such that decision diagram learning requires to *jointly* determine the splitting hyperplanes and the node-connecting arcs.

Decision diagrams possess notable advantages over decision trees. Firstly, their width is not bound to grow exponentially with their depth, which allows training deep but narrow decision diagrams without quickly facing issues of data fragmentation (Shotton et al. 2013; Ignatov and Ignatov 2018). Moreover, additional degrees of freedom in their topology design permit to express a richer set of concepts and to achieve better model compression in memory-constrained computing environments (Breslow and Aha 1997; Kumar, Goyal, and Varma 2017).

Despite these advantages, decision diagrams have been more rarely used than decision trees, as learning them remains inherently complex. A decision diagram topology cannot be easily optimized by construction or local optimization algorithms based on impurity measures. However, recent enhancements in global optimization techniques for decision tree training motivate us to reevaluate this issue. Indeed, optimal decision tree training through mathematical programming is becoming practical due to the formidable progress of hardware and mixed-integer linear programming solvers, which collectively led to speed-ups as high as 10^{11} between 1991 and 2015—most of which due to algorithmic improvements rather than hardware (Bixby 2012). In view of this, we reevaluate the problem of searching for optimal decision diagrams (ODDs) through modern combinatorial optimization lenses and propose new mathematical models and efficient solution techniques to learn ODDs. Specifically, our contributions are threefold:

1. We propose the first mixed-integer linear program (MILP) to train decision diagrams for classification. This model effectively represents the decision diagram topology and the flow of samples within it, employing a limited number of binary variables. In practice, it includes exponentially fewer binary variables than (Bertsimas and Dunn 2017) when applied to decision tree topologies.

*These authors contributed equally.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Furthermore, we include additional symmetry-breaking constraints that speed up the solution process, and provide efficient heuristic search strategies to obtain good primal solutions quickly.

2. We conduct an extensive computational study to evaluate our approach's scalability and compare the resulting decision diagrams with classical decision trees. We observe that training ODDs requires a computational effort comparable to optimal decision-tree training but leads to more parsimonious and accurate models.
3. As our MILP semantic permits to express various additional constraints with minimal adaptation, we discuss possible extensions to capture fairness, parsimony and stability requirements. We show the efficacy of such extensions in our numerical study for the stability case.

Related Work

Optimal training of decision trees. Standard construction algorithms for decision-tree training based on local impurity measures are not guaranteed to find the most accurate tree of a given size. To circumvent this issue, several works have been focused on global optimization algorithms. Optimal decision-tree training is known to be NP-hard (Hyafil and Rivest 1976), but solution methods for this problem went a long way from early dynamic programming algorithms (Meisel and Michalopoulos 1973; Payne and Meisel 1977) to modern solution approaches. Nowadays, these algorithms can find optimal trees for datasets with thousands of samples and hundreds of features (Bertsimas and Dunn 2017; Demirović et al. 2020; Firat et al. 2020).

Carrizosa, Molero-Río, and Morales (2021) recently conducted a comprehensive survey of mathematical programming approaches for optimal decision-tree training. Among the works surveyed, the paper of Bertsimas and Dunn (2017) represents a turning point, as it proposed a compact MILP formulation that could be solved within reasonable time for a wide range of datasets from the UCI machine learning repository. Subsequent methodological improvements occurred through sophisticated model reformulations and decomposition methods (Aghaei, Gomez, and Vayanos 2020; Firat et al. 2020), permitting to achieve better linear programming bounds and quickly prune sub-optimal regions of the search space. Among other benefits, MILP-based approaches can handle combinatorial splits on categorical features, and they can easily include notions of fairness (Aghaei, Azizi, and Vayanos 2019; Ye and Xie 2020) and parsimony (Blanquero et al. 2020) through additional global objectives and constraints. Research in this field strives towards more effective solution approaches, based on combinations of branch-and-bound with dynamic programming (Aglin, Nijssen, and Schaus 2020; Demirović et al. 2020), constraint programming (Verhaeghe et al. 2020), or exploiting Boolean satisfiability (SAT) problem solvers (Narodytska et al. 2018). Some of these solution approaches also open new perspectives for other related training tasks, for example, in model compression (Vidal and Schiffer 2020).

Decision diagrams. Decision diagrams for classification have regularly reappeared in the machine learning domain. Early studies on the topic (Oliver 1993; Kohavi and Li 1995; Oliveira and Sangiovanni-Vincentelli 1996; Akers 1978) followed a minimum description length perspective and led to a first generation of learning algorithms – often transforming an initial decision tree into a decision diagram. Oliver (1993) argued that decision diagrams are particularly suitable to express disjunctive concepts (e.g., exclusive OR) and proposed an iterative node-merging algorithm. Kohavi (1994) and Kohavi and Li (1995) first designed a bottom-up learning approach and then opted to train an oblivious decision tree and post-process it into a decision diagram through iterative merging. Oliveira and Sangiovanni-Vincentelli (1996) exploited efficient algorithms known from logic synthesis (Bryant 1986, 1992) to manipulate ordered decision diagrams. They design an iterative greedy approach that merges nodes with similar sub-graphs to achieve a greater reduction of message length. An extension of support vector machines towards multi-class settings through decision diagrams was also presented in Platt, Cristianini, and Shawe-Taylor (2000). Decision diagrams have also been used for model compression (Breslow and Aha 1997; Gossen and Steffen 2019) and as a surrogate model for neural networks in Chorowski and Zurada (2011). They were generalized into ensembles called decision jungles in Shotton et al. (2013). Ignatov and Ignatov (2018) considered training deep decision diagrams, called decision streams, and reported a good performance on a range of datasets for credit scoring, aircraft control, and image classification. Recently, Cabodi et al. (2021) studied binary decision diagrams (BDDs) in the context of interpretable machine learning, while Hu, Huguet, and Siala (2022) focused on optimizing BDDs via MaxSAT. Different from works on BDDs, however, we find optimal diagrams for classification without assuming neither Boolean features nor a fixed feature evaluation order.

Most likely, the biggest obstacle towards the effective use of decision diagrams remains the ability to learn them efficiently. Indeed, most aforementioned learning algorithms first generate a decision tree and then transform it into a decision diagram. Unfortunately, these methods also often fix the order of the predicates through the tree, as this choice is known to lead to a difficult combinatorial optimization problem (Bollig and Wegener 1996). Accordingly, design decisions related to the graph topology (permitting to learn complex concepts) are ineffectively learned through trial and error. Our approach closes this significant methodological gap and, for the first time, allows to derive efficient algorithms for training decision diagrams.

Mathematical Formulation

In this section, we mathematically formulate the ODD training problem as a MILP. We assume that we have a training dataset $\{(\mathbf{x}^i, c^i)\}_{i=1}^n$ in which each $\mathbf{x}^i \in \mathbb{R}^d$ corresponds to a sample characterized by a d-dimensional feature vector and a class $c^i \in \mathcal{C}$. Our formulation takes as input the diagram's depth D, i.e., the diagram's number of decision layers, and the width w_l of each layer $l \in \{0, \ldots, D-1\}$. This input constitutes the *skeleton* of the decision diagram, and it guarantees that any final topology found during training (i.e., number of activated nodes per layer and their mutual connections) is contained in the skeleton. W.l.o.g., we can assume that $w_{l+1} \leq 2w_l$ for all l. The MILP optimizes the decision diagram's topology and the splitting hyperplane of each internal node. We allow connections between internal nodes of consecutive layers and direct connections via *long arcs* to terminal nodes representing the classes, as this permits to progressively assign a final classification for specific samples without necessarily passing through the complete decision diagram. Accordingly, the training algorithm can progressively send samples to the final leaves to fully exploit the remaining layers for classifying the other samples.

We designed our approach for numerical data, such that processing other data types requires prior transformation, e.g., one-hot encoding for categorical data, which is a common practice in decision-tree-based models. For numerical stability, we assume that each feature has been normalized within $[0, 1]$. Our model handles multiclass classification with a dedicated leaf for each class. It also naturally finds optimal multivariate (i.e., diagonal) splits without extra computational burden. We further show how it can be restricted to produce only univariate splits if necessary.

Canonical Formulation

To represent the decision diagram, as illustrated on Figure 1, we define an acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with $\mathcal{V} = \mathcal{V}^{\mathrm{I}} \cup \mathcal{V}^{\mathrm{C}}$. Each node $v \in \mathcal{V}^{\mathrm{I}}$ represents an internal node, and each node $v \in \mathcal{V}^{\mathrm{C}}$ represents a terminal node corresponding to a class c_v. We represent nodes by indices $\mathcal{V} = \{0, \dots, |\mathcal{V}^{\mathrm{I}}| + |\mathcal{V}^{\mathrm{C}}| - 1\}$; node $0 \in \mathcal{V}^{\mathrm{I}}$ represents the root of the decision diagram and the remaining nodes are listed by increasing depth (from left to right on the figure). Let $\mathcal{V}^{\mathrm{I}}_l$ be the set of nodes at depth l, and let $\delta^-(v)$ and $\delta^+(v)$ be the sets of predecessors and successors of each node $v \in \mathcal{V}$. With these definitions, $\delta^-(0) = \varnothing$ and $\delta^+(0) = \mathcal{V}^{\mathrm{I}}_1 \cup \mathcal{V}^{\mathrm{C}}$. For $v \in \mathcal{V}^{\mathrm{I}}_l$ with $1 \leq l < D - 1$, $\delta^-(v) = \mathcal{V}^{\mathrm{I}}_{l-1}$ and $\delta^+(v) = \mathcal{V}^{\mathrm{I}}_{l+1} \cup \mathcal{V}^{\mathrm{C}}$. Finally, for $v \in \mathcal{V}^{\mathrm{C}}$, $\delta^-(v) = \mathcal{V}^{\mathrm{I}}$ and $\delta^+(v) = \varnothing$. The decision diagram produced by our model will be a subgraph of $\mathcal{G}$.

To formulate the training problem as a MILP, we start by defining the flow variables that represent the trajectory of the samples within the graph. We then connect these flows to the design variables defining the structure of the decision diagram and to those characterizing the splits.

Flow variables. Each sample i and internal node $u \in \mathcal{V}^{\mathrm{I}}$ is associated to a pair of flow variables $w^-_{iu} \in [0, 1]$ and $w^+_{iu} \in [0, 1]$. A non-zero value in w^-_{iu} (respectively w^+_{iu}) means that sample i passes through node u on the negative side of the separating hyperplane (on the positive side, respectively). Moreover, variables $z^-_{iuv} \in [0, 1]$ (respectively $z^+_{iuv} \in [0, 1]$) characterize the flow going from the negative and positive sides of u to other nodes v. With these definitions, we can express flow conservation within the graph $\mathcal{G}$ through the following conditions for each $u \in \mathcal{V}^{\mathrm{I}}$ and $i \in \{1, \dots, n\}$:

$$w^+_{iv} + w^-_{iv} = \begin{cases} 1 & \text{if } v = 0 \\ \sum_{u \in \delta^-(v)} (z^+_{iuv} + z^-_{iuv}) & \text{otherwise} \end{cases} \tag{1}$$

$$w^-_{iu} = \sum_{v \in \delta^+(u)} z^-_{iuv} \tag{2}$$

$$w^+_{iu} = \sum_{v \in \delta^+(u)} z^+_{iuv}. \tag{3}$$

However, due to the interaction of the constraints coming from the hyperplanes (described later in this section), integrality of the flow variables is not guaranteed. To obtain integer sample flows, we add an additional binary variable $\lambda_{il} \in \{0, 1\}$ for each sample $i \in \{1, \dots, n\}$ and level $l \in \{0, \dots, D - 1\}$, along with the following constraints for each $l \in \{0, \dots, D - 1\}$ and $i \in \{1, \dots, n\}$:

$$\sum_{u \in \mathcal{V}^{\mathrm{I}}_l} w^-_{iu} \leq 1 - \lambda_{il} \tag{4}$$

$$\sum_{u \in \mathcal{V}^{\mathrm{I}}_l} w^+_{iu} \leq \lambda_{il}. \tag{5}$$

With these constraints, sample i can only go to the negative (respectively positive) side of any node u of level $\mathcal{V}^{\mathrm{I}}_l$ if $\lambda_{il} = 0$ (respectively $\lambda_{il} = 1$). This allows us to use fewer binary variables compared to a direct definition of the w^-_{iu} and w^+_{iu} as binary (see Theorem 2), which is a desirable characteristic to allow for efficiently solving the MILP.

Decision diagram topology. We now connect the flow variables to the binary design variables that characterize the topology of the diagram. Here, we define one binary variable $d_u \in \{0, 1\}$ for each $u \in \mathcal{V}$ that takes value 1 if this node is used in the classification (i.e., samples can pass through it). The terminal nodes and the root are always activated, so we impose $d_u = 1$ for $u \in 0 \cup \mathcal{V}^{\mathrm{C}}$. For the negative and positive sides of each node $u \in \mathcal{V}^{\mathrm{I}}$, we create binary design variables $y^-_{uv} \in \{0, 1\}$ and $y^+_{uv} \in \{0, 1\}$ taking value 1 if and only if u links towards v on the negative and positive sides, respectively. The following constraints connect the design variables and the sample flows:

$$d_u = \sum_{v \in \delta^+(u)} y^+_{uv} = \sum_{v \in \delta^+(u)} y^-_{uv} \qquad u \in \mathcal{V}^{\mathrm{I}} \tag{6}$$

$$d_v \leq \sum_{u \in \delta^-(v)} (y^+_{uv} + y^-_{uv}) \qquad v \in \mathcal{V}^{\mathrm{I}} - \{0\} \tag{7}$$

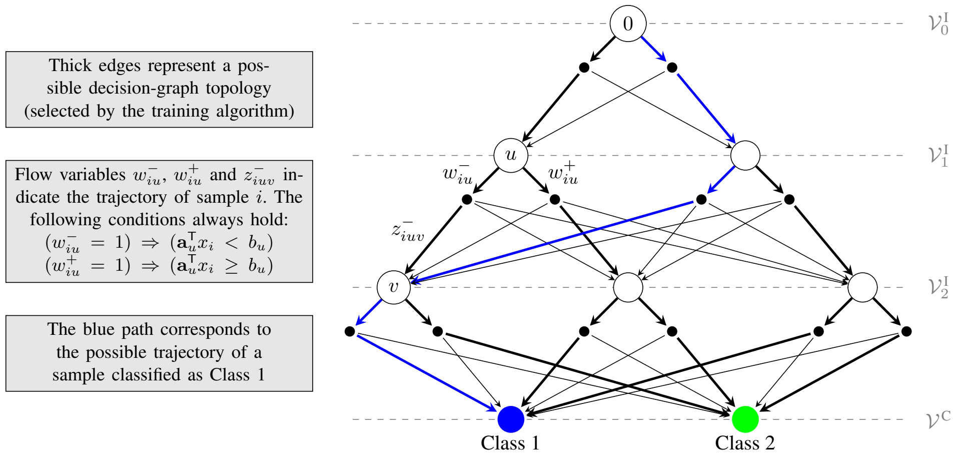
$$y^+_{uv} + y^-_{uv} \leq d_v \qquad u \in \mathcal{V}^{\mathrm{I}}, v \in \delta^+(u) \tag{8}$$

$$z^+_{iuv} \leq y^+_{uv}, \qquad u \in \mathcal{V}^{\mathrm{I}}, v \in \delta^+(u), i \in \{1, \dots, n\} \tag{9}$$

$$z^-_{iuv} \leq y^-_{uv} \qquad u \in \mathcal{V}^{\mathrm{I}}, v \in \delta^+(u), i \in \{1, \dots, n\} \tag{10}$$

Symmetry-breaking constraints. Without any constraint for breaking symmetry, $2^{|\mathcal{V}^{\mathrm{I}}|}$ equivalent topologies can be obtained by switching the positive- and negative-side arcs of each internal node and using opposite hyperplanes. Such symmetry has a dramatic negative impact on branch-and-bound-based MILP solution approaches. To circumvent this issue, we impose that arcs (u, v) and (u, w) such that $y^-_{uv} = 1$ and $y^+_{uw} = 1$ satisfy $v < w$ for each internal node $u \in \mathcal{V}^{\mathrm{I}}$. This corresponds to the logical constraint $(y^-_{uv} = 1) \Rightarrow (y^+_{uw} = 0 \; \forall w \leq v)$, formulated as:

$$y^-_{uv} + \sum_{w \in \delta^+(u), w \leq v} y^+_{uw} \leq 1 \qquad u \in \mathcal{V}^{\mathrm{I}}, v \in \delta^+(u) \tag{11}$$


Thick edges represent a possible decision-graph topology (selected by the training algorithm)

Flow variables w_{iu}^-, w_{iu}^+ and z_{iuv}^- indicate the trajectory of sample i. The following conditions always hold: $(w_{iu}^- = 1) \Rightarrow (\mathbf{a}_u^\intercal x_i < b_u)$ $(w_{iu}^+ = 1) \Rightarrow (\mathbf{a}_u^\intercal x_i \geq b_u)$

The blue path corresponds to the possible trajectory of a sample classified as Class 1
0
u
v
w_{iu}^-
w_{iu}^+
z_{iuv}^-
$\mathcal{V}_0^\mathrm{I}$
$\mathcal{V}_1^\mathrm{I}$
$\mathcal{V}_2^\mathrm{I}$
$\mathcal{V}^\mathrm{C}$
Class 1
Class 2

Figure 1: Example of a graph $\mathcal{G}$ with three layers of internal nodes ($w_1 = 2$ and $w_2 = 3$) and two terminal nodes. The thick edges indicate a possible decision diagram. The black connectors permit to illustrate flow-conservation within the graph. For clarity, the long arcs between the black connectors of layers $\mathcal{V}_0^\mathrm{I}$ and $\mathcal{V}_1^\mathrm{I}$ and the terminal nodes of $\mathcal{V}^\mathrm{C}$ are not displayed.

To further reduce model symmetry and the number of equivalent topologies, we impose that the nodes along each layer must respect a weak in-degree ordering by including the following constraint for each $l \in \{2, \dots, D-1\}$ and each $u \in \mathcal{V}_l^\mathrm{I}$, $v \in \mathcal{V}_l^\mathrm{I}$ such that $u < v$:

$$\sum_{w \in \delta^-(u)} (y_{wu}^+ + y_{wu}^-) \geq \sum_{w \in \delta^-(v)} (y_{wv}^+ + y_{wv}^-) \tag{12}$$

Linear separator variables and consistency with the sample flows. We associate to each internal node $v \in \mathcal{V}^\mathrm{I}$ a vector of variables $\mathbf{a}_v \in [-1,1]^d$ and a variable $b_v \in [-1,1]$ to characterize the splitting hyperplane. Samples $i \in \{1,\dots,n\}$ following the negative-side path should satisfy $\mathbf{a}_v^\intercal \mathbf{x}^i < b_v$, whereas samples taking the positive-side path should satisfy $\mathbf{a}_v^\intercal \mathbf{x}^i \geq b_v$. This is done by including *indicator constraints* in our MILP that express the following implication logic for each $i \in \{1,\dots,n\}$ and $v \in \mathcal{V}^\mathrm{I}$:

$$(w_{iv}^- = 1) \Rightarrow (\mathbf{a}_v^\intercal \mathbf{x}^i + \varepsilon \leq b_v) \tag{13}$$
$$(w_{iv}^+ = 1) \Rightarrow (\mathbf{a}_v^\intercal \mathbf{x}^i \geq b_v) \tag{14}$$

In Constraint (13), ε should be a small constant greater than the numerical precision of the solver (set to $\varepsilon = 10^{-4}$ in our experiments) that permits to express strict inequality. These logical constraints could be reformulated as linear constraints using a big-M transformation, leading to $\mathbf{a}_v^\intercal \mathbf{x}^i + \varepsilon \leq b_v + M(1 - w_{iv}^-)$ and $\mathbf{a}_v^\intercal \mathbf{x}^i \geq b_v - M(1 - w_{iv}^+)$. However, as seen in (Belotti et al. 2016), modern MILP solvers generally benefit from directly specifying the implication logic and apply a big-M reformulation with tailored bound tightening.

Constraints (13–14) represent general multivariate splits. We can further restrict the model to produce univariate splits by adding for each feature $j \in \{1,\dots,d\}$ and internal node $v \in \mathcal{V}^\mathrm{I}$ a binary variable $e_{vj} \in \{0,1\}$, along with constraints that impose the selection of a single feature:

$$\sum_{j=1}^{d} e_{vj} = 1 \qquad v \in \mathcal{V}^\mathrm{I} \tag{15}$$
$$-e_{vj} \leq a_{jv} \leq e_{vj} \qquad j \in \{1,\dots,d\}, v \in \mathcal{V}^\mathrm{I} \tag{16}$$

Objective function. In a similar fashion as in (Bertsimas and Dunn 2017), we optimize accuracy and an additional regularization term that favors simple decision diagrams with fewer internal nodes. The accuracy of the model can be computed by means of variables $w_{iv} \in [0,1]$ for each sample $i \in \{1,\dots,n\}$ and leaf $v \in \mathcal{V}^\mathrm{C}$ expressing the amount of flow of i reaching terminal node v with class c_v. These variables must satisfy the following constraints for $v \in \mathcal{V}^\mathrm{C}$ and $i \in \{1,\dots,n\}$:

$$w_{iv} = \sum_{u \in \delta^-(v)} (z_{iuv}^+ + z_{iuv}^-) \tag{17}$$

With the help of these variables, we state our objective as

$$\min \frac{1}{n} \sum_{i=1}^{n} \sum_{v \in \mathcal{V}^\mathrm{C}} \phi_{iv} w_{iv} + \frac{\alpha}{|\mathcal{V}^\mathrm{I}| - 1} \sum_{v \in \mathcal{V}^\mathrm{I} - \{0\}} d_v, \tag{18}$$

where ϕ_{iv} represents the mismatch penalty when assigning sample i to terminal node v (typically defined as 0 if $c^i = c_v$ and 1 otherwise), while α is a regularization parameter. The first term of the objective penalizes misclassified samples, whereas the second term penalizes complex models. With those conventions, the objective lies in $[0, 1+\alpha]$ and an hypothetical value of 0 would correspond to a model that achieves perfect classification with a single split at the root node.

Theorem 1 Formulation (1–18) produces solutions in which all variables $\mathbf{w}$ and $\mathbf{z}$ take binary values, leading to a feasible and optimal decision diagram.

Theorem 2 Our formulation includes $\mathcal{O}(n \log |\mathcal{V}|)$ binary decision variables when applied to decision-tree skeletons.

While Theorem 1 establishes optimality for our MILP formulation, Theorem 2 highlights its efficiency, illustrated in the context of decision tree skeletons where it improves upon the formulation of (Bertsimas and Dunn 2017), which requires $\mathcal{O}(n|\mathcal{V}|)$ binary decision variables. We give proofs to Theorems 1 and 2 in the supplemental material.

Discussion

Whereas classical training algorithms are often tailored to a specific setting and difficult to adapt to new requirements, our MILP approach for ODD training provides an extensible mathematical framework for expressing new requirements. We show how our MILP can be extended to fairness, parsimony, or stability measures. All extensions require minimal effort and only require extra linear constraints over the existing variables.

Fairness. The confusion (or error) matrix can be directly calculated from the binary variables w_{iv} of our model, which take value one if and only if sample i reaches terminal node v with class c_v. Hence, we can introduce classical fairness metrics into our MILP, either as an additional term in the objective or as a constraint, without significantly changing its complexity. To illustrate this, consider a binary classification dataset in which outcome 1 is the most desirable. For any subgroup $g \subset \{1, \dots, n\}$, the number of samples classified as 1 by the ODD can be calculated as $Z_g^{\text{P}} = \sum_{i \in g} w_{iv_{\text{P}}}$, where v_{P} is the terminal node associated to the positive class. This permits to express demographic parity (see, e.g., Mehrabi et al. 2019) with the following linear constraints:

$$\sum_{i \in g_1} w_{iv_{\text{P}}} \geq \xi \sum_{i \in g_2} w_{iv_{\text{P}}}, \tag{19}$$

where g_1 and g_2 are two (non-necessarily disjoint) subgroups and ξ represents a minimal discrepancy ratio between two subgroups. Compliance with the classic *four-fifths* rule (Biddle 2006; Zafar et al. 2017) is achieved with $\xi = 0.8$. In a similar fashion, the number of false-positives and false-negatives for any subgroup g can be calculated as $Z_g^{\text{FP}} = \sum_{i \in g, c^i \neq \text{P}} w_{iv_{\text{P}}}$ and $Z_g^{\text{FN}} = \sum_{i \in g, c^i = \text{P}} (1 - w_{iv_{\text{P}}})$, permitting in turn to express most other classical fairness notions. e.g., equal opportunity or predictive equality (Mehrabi et al. 2019).

Parsimony and stability. Our MILP's flexibility for ODD training is not limited to fairness. It is possible to bound the number of activated nodes to a predefined limit D to ensure parsimony by imposing $\sum_{i \in \mathcal{V}^{\text{I}}} d_i \leq D$. Finally, it is possible to impose that a minimum number S of samples passes through each activated internal node to enhance stability, through the following conditions:

$$\sum_{i=1}^{n} \sum_{u \in \delta^-(v)} (z_{iuv}^+ + z_{iuv}^-) \geq S d_v \qquad v \in \mathcal{V}^{\text{I}}. \tag{20}$$

Training Strategy

We introduce a two-step search strategy, which permits to train an initial decision diagram quickly, and then refine it to eventually reach an optimal topology. First, we use an efficient multi-start construction and improvement heuristic to derive an initial topology. Then, we solve the complete MILP using an off-the-shelf branch-and-cut implementation (Gurobi in our case), where the value of the solution obtained in the first step is used as a cutoff value in the branch-and-bound search.

Step 1—Initial construction and improvement. We use a top-down construction approach that shares some common traits with CART (Breiman et al. 1984). For each layer $l \in \{0, \dots, D-2\}$ and internal node $u \in \mathcal{V}_l^{\text{I}}$, we select the univariate split that maximizes the information gain. In contrast to CART, we determine the internal connections of the decision diagram at the training stage. Therefore, additional decisions need to be taken regarding the destination of the sample flows emerging from each layer. To this end, we adopt a greedy merging policy: as long as the number of sample flows is greater than w_{l+1}, we merge the pair of flows that least decreases the information gain. In the last layer connecting to the terminal nodes $\mathcal{V}^{\text{C}}$, we finally directly connect each sample flow to the terminal node of its most represented class.

To obtain a better initial diagram, we repeat the construction process for 60 seconds and consider only a random subset of 60% of the features during each split selection. Additionally, we apply a bottom-up pruning strategy that consists of iteratively eliminating any internal (i.e., splitting) node that is (i) connected only to terminal nodes, and (ii) such that the removal improves Objective (18).

Step 2—Solution of the MILP. In this step, we apply Gurobi on the complete MILP described in the previous section. Gurobi is a state-of-the-art solver for MILPs that utilizes a branch-and-cut process to derive lower and upper bounds on the objective value and to prune regions of the search space that have no chance to contain an optimal solution. To further guide the branch-and-bound search towards promising regions of the feasible space, we set an objective cutoff value equal to the value of the solution found in Step 1. The completion of this process gives a global optimum of the training model for the considered objective and regularization parameter. We set a CPU time limit of $T_{\text{MAX}} = 600$ seconds for this phase. The process terminates with the best solution found so far if T_{MAX} is attained.

We remark that our training method always terminates after a fixed amount of time, regardless of dataset size, and returns the best-found decision diagram so far.

Experimental Analyses

We focus our experiments on the same selection of 54 datasets as in Bertsimas and Dunn (2017). All these datasets are publicly available from the UCI machine learning repository (Dua and Graff 2017). They reflect a wide range of classification applications and contain between 47 to 6435 data points with 2 to 484 features. The list of datasets is accessible along with our detailed experimental results in the supplemental material. We split each dataset into a training, validation, and testing subset of samples with respective proportions of 50%, 25%, and 25%. We repeat all our experiments five times for each dataset, using a different seed and

	Proven optimality					Improved solution					Total
Skeleton	$\alpha = 0.01$	0.1	0.2	0.5	1.0	0.01	0.1	0.2	0.5	1.0	
I	152	155	162	179	225	35	91	134	202	202	1537
II	150	154	162	179	227	36	91	124	194	193	1510
III	148	152	161	177	223	38	89	129	200	205	1522
IV	149	156	163	178	226	62	124	150	226	221	1655
I (tree)	155	158	163	177	225	30	98	131	208	207	1552
Total	754	775	811	890	1126	201	493	668	1030	1028	7776

Table 1: Performance of the MILP-based training method

thus a different random separation of the samples for each run.

All our algorithms have been implemented in Python 3.8 and can be readily executed from a single script. We use Gurobi 9.1.0 (via gurobipy) for solving the mathematical models. Each validation and test experiment has been run on a single thread of an Intel Gold 6148 Skylake @2.40GHz CPU. Overall, the experiments of this paper took eight hours on 10 CPUs of the mentioned type for five seeds. All data, source code, and additional details on the computational results are provided in the supplemental material and will be provided as a public Git repository upon publication.

We divide our computational experiments into two stages. Firstly, we conduct a calibration experiment using only the training and validation sets, considering different decision diagram skeletons and different levels of the regularization parameter α. The goal of this experiment is twofold: it permits to find the best skeleton and α hyperparameter for each dataset, and allows us to evaluate our methodology's computational performance, scalability, and sensitivity for different parameters. Finally, in the second stage of experiments, we use the test set to compare our optimal decision diagram models against the optimal decision tree model, using the best-found α hyperparameter for each model and dataset.

Hyperparameters Calibration and Computational Performance

We study the computational tractability of our approach for $\alpha \in \{0.01, 0.1, 0.2, 0.5, 1\}$ and for four decision diagram skeletons: (1–2–4–8), (1–2–4–4–4), (1–2–3–3–3–3–3) and (1–2–2–2–2–2–2–2), hereby referred to as Skeletons I to IV, respectively. All these skeletons have 15 internal nodes; therefore, the final trained decision diagrams will contain no more than 15 active nodes. We further employ Skeleton (1–2–4–8) to find optimal decision trees (ODTs). To this end, we specialize our model by fixing the decision variables representing the internal topology to match a decision tree; hence, generating optimal ODT solutions similar to Bertsimas and Dunn (2017).

Computational performance. We run our algorithm for each dataset, split type (univariate and multivariate), random seed, skeleton, and value of α. First, we evaluate our ability to find either the global optimum or an improved solution to the training problem, compared to the solution found in Step 1. Table 1 shows, for each skeleton and α combination, the number of runs (out of 54 datasets $\times$ 5 seeds $=$ 270 runs) for which a global optimum or an improved solution was found. Our algorithm can find optimal topologies in 32% of all runs (4356 out of 13500 runs). Model difficulty is inversely proportional to α values, as large values of the regularization parameter discourages complex topologies with many active nodes. The difficulty of the training problem is generally sensitive to the number of samples in the dataset but relatively insensitive to the number of features (see results in the supplemental material). Because of the large size of some datasets and their resulting training models, in many cases optimality is not achieved within the short computational budget of 600 seconds used in our experiments. Still, Step 2 of our methodology improves upon the initial heuristic solution in 58% of all runs, which demonstrates the value of the mathematical programming-based training approach and indicates that it might be possible to solve more instances to optimality when allocating larger computational budgets to each instance.

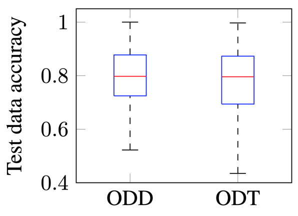


Figure 2: Test data accuracy

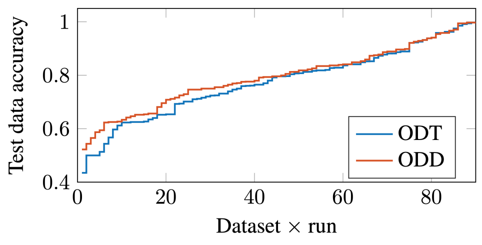


Figure 3: Accuracy per dataset $\times$ run, ascending order

Performance Analyses and Model Comparison

Accuracy of ODDs. We now compare the performance and structure of ODDs with those of ODTs. We use the best skeleton for each ODD, as obtained during the hyperparam-

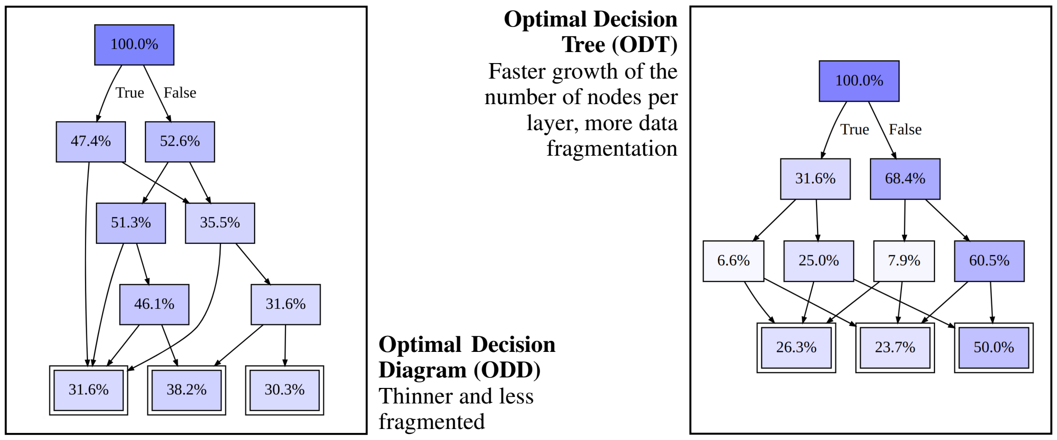

100.0%
True
False
47.4%
52.6%
51.3%
35.5%
46.1%
31.6%
31.6%
38.2%
30.3%
Optimal Decision Diagram (ODD)
Thinner and less fragmented
Optimal Decision Tree (ODT)
Faster growth of the number of nodes per layer, more data fragmentation
100.0%
True
False
31.6%
68.4%
6.6%
25.0%
7.9%
60.5%
26.3%
23.7%
50.0%

Figure 4: Fragmentation of an ODT and an ODD trained on the "teaching-assistant evaluation" dataset

eter calibration phase. We apply the same α hyperparameter calibration process for ODTs. To compare both methods based on near-optimal topologies, we focus on a subset of 18 datasets for which optimal ODDs and ODTs can be consistently obtained, and for which the final topologies are non-trivial (i.e., use at least two internal nodes). We refer the interested reader to the supplemental material for a more detailed discussion on the selection of those datasets. Moreover, we note that we do not extend this analysis to other heuristic tree models such as CART, since the superior performance of ODTs over CART has already been discussed for the same dataset collection in Bertsimas and Dunn (2017).

Overall, ODDs and ODTs with univariate splits exhibit comparable accuracy as can be seen in the detailed numerical results in the supplemental material. With multivariate splits, however, ODDs show a higher accuracy compared to their ODT counterparts. Figure 2 shows the distribution of the classification accuracy on the test data over the selected datasets for the respective best ODT and ODD models, highlighting that the respective ODDs show fewer low-accuracy outliers. Their accuracy distribution is generally more compact and exhibits higher first, second, and third quartiles. Figure 3 complements this analysis by representing the accuracy of all 18×5 instances for ODD and ODT in the multivariate splits case, sorted in ascending order. The sorted accuracy of the ODDs superposes that of the best ODTs, which points towards a better classification performance.

The improved performance of ODDs over ODTs results from the additional freedom in its topology, which permits to express more complex concepts and generally avoids data fragmentation. Figure 4 illustrates the resulting different topologies and visualizes the data fragmentation for an ODT and an ODD trained on the "teaching-assistant evaluation" dataset. Both models contain seven internal nodes, but the ODD has a much more balanced data fragmentation, i.e., the share of samples processed through each node is more balanced for the ODD.

Stability of ODDs. Stability is a desirable characteristic of tree and diagram classifiers, as it preserves effective classification rules in face of new data and enhances model interpretability. Table 2 measures the impact on accuracy when enforcing a certain level of stability when training ODDs with our MILP approach. This is achieved by activating Constraints (20) and varying the minimum number of samples S required to pass through any active node in the optimal topology. A tradeoff is expected as tighter stability constraints reduce the solution space when constructing the ODD. As observed, more stable ODDs can be obtained at a marginal out-of-sample accuracy loss.

Split	$S = 0.05n$	$0.10n$	$0.15n$	$0.20n$
Multivariate	0.828	0.827	0.823	0.823
Univariate	0.840	0.838	0.837	0.835

Table 2: Test accuracy when enforcing minimum flow of samples through active nodes

Conclusions

We studied the training of optimal decision diagrams from a combinatorial optimization perspective. Specifically, we proposed the first MILP that allows to train ODDs for classification. We conducted an extensive numerical study on 54 benchmark datasets reflecting a wide variety of practical classification tasks. The degrees of freedom of the model permit to find good ODD topologies that exhibit less data fragmentation than ODTs. Concerning out-of-sample accuracy, we showed that ODDs perform favorably when compared to ODTs for the case of multivariate splits.

The flexibility of our mathematical programming approach permits to extend the proposed model to address a variety of important side requirements. Therefore, we believe that it can serve as an important building block for more elaborate models, suited for a variety of application domains. As future research perspective, we suggest the investigation of heuristic construction techniques that permit to generate decision diagrams for classification independently of dataset size.

Acknowledgments

This research was enabled in part by support provided by Calcul Québec and Compute Canada.

References

Abío, I.; Nieuwenhuis, R.; Oliveras, A.; Rodríguez-Carbonell, E.; and Mayer-Eichberger, V. 2012. A New Look at BDDs for Pseudo-Boolean Constraints. *Journal of Artificial Intelligence Research*, 45: 443–480.

Aghaei, S.; Azizi, M.; and Vayanos, P. 2019. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1418–1426.

Aghaei, S.; Gomez, A.; and Vayanos, P. 2020. Learning optimal classification trees: Strong max-flow formulations. *arXiv preprint arXiv:2002.09142*.

Aglin, G.; Nijssen, S.; and Schaus, P. 2020. Learning optimal decision trees using caching branch-and-bound search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(4): 3146–3153.

Akers, S. B. 1978. Binary decision diagrams. *IEEE Transactions on computers*, 27(06): 509–516.

Andersen, H.; Hadzic, T.; Hooker, J.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, 118–132. Springer.

Behle, M. 2007. *Binary Decision Diagrams and Integer Programming*. Ph.D. thesis, Saarland University.

Belotti, P.; Bonami, P.; Fischetti, M.; Lodi, A.; Monaci, M.; Nogales-Gómez, A.; and Salvagnin, D. 2016. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3): 545–566.

Bergman, D.; Cire, A.; Van Hoeve, W.-J.; and Hooker, J. 2016. *Decision diagrams for optimization*. Springer International Publishing. ISBN 9783319428475.

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

Biddle, D. 2006. *Adverse impact and test validation: A practitioner's guide to valid and defensible employment testing*. Gower Publishing, Ltd.

Bixby, R. 2012. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 107–121.

Blanquero, R.; Carrizosa, E.; Molero-Río, C.; and Romero Morales, D. 2020. Sparsity in optimal randomized classification trees. *European Journal of Operational Research*, 284: 255–272.

Bollig, B.; and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9): 993–1002.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and regression trees*.

Breslow, L.; and Aha, D. 1997. Simplifying decision trees: A survey. *Knowledge Engineering Review*, 12(1): 1–40.

Bryant, R. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8): 677–691.

Bryant, R. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3): 293–318.

Cabodi, G.; Camurati, P. E.; Ignatiev, A.; Marques-Silva, J.; Palena, M.; and Pasini, P. 2021. Optimizing binary decision diagrams for interpretable machine learning classification. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1122–1125. IEEE.

Carrizosa, E.; Molero-Río, C.; and Morales, D. 2021. Mathematical optimization in classification and regression trees. *Top*, 29(1): 5–33.

Castro, M.; Piacentini, C.; Cire, A.; and Beck, J. 2019. Relaxed BDDs: An admissible heuristic for delete-free planning based on a discrete relaxation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 77–85.

Chorowski, J.; and Zurada, J. 2011. Extracting rules from neural networks as decision diagrams. *IEEE Transactions on Neural Networks*, 22(12): 2435–2446.

Choudhary, T.; Mishra, V.; Goswami, A.; and Sarangapani, J. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 1–43.

Demirović, E.; Lukina, A.; Hebrard, E.; Chan, J.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Stuckey, P. 2020. MurTree: Optimal classification trees via dynamic programming and search. *arXiv preprint arXiv:2007.12652*.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, http://archive.ics.uci.edu/ml, Accessed 2023-04-03.

Firat, M.; Crognier, G.; Gabor, A.; Hurkens, C.; and Zhang, Y. 2020. Column generation based heuristic for learning classification trees. *Computers and Operations Research*, 116: 104866.

Gossen, F.; and Steffen, B. 2019. Large random forests: Optimisation for rapid evaluation. *arXiv preprint arXiv:1912.10934*.

Hu, H.; Huguet, M.-J.; and Siala, M. 2022. Optimizing Binary Decision Diagrams with MaxSAT for Classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 3767–3775.

Hyafil, L.; and Rivest, R. 1976. Constructing optimal decision trees is NP-complete. *Information Processing Letters*, 5(1): 1–3.

Ignatov, D.; and Ignatov, A. 2018. Decision stream: Cultivating deep decision trees. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 905–912.

Kohavi, R. 1994. Bottom-up induction of oblivious read-once decision graphs: strengths and limitations. *Proceedings of AAAI*, 613–618.

Kohavi, R.; and Li, C.-H. 1995. Oblivious decision trees, graphs, and top-down pruning. In *Proceedings of IJCAI*, 1071–1079.

Kumar, A.; Goyal, S.; and Varma, M. 2017. Resource-efficient machine learning in 2 KB RAM for the Internet of Things. *34th International Conference on Machine Learning, ICML 2017*, 4: 3062–3071.

Lai, Y.; Liu, D.; and Wang, S. 2013. Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach. *Knowledge and Information Systems*, 35: 665–712.

Lange, J.-H.; and Swoboda, P. 2021. Efficient message passing for 0–1 ILPs with binary decision diagrams. In *International Conference on Machine Learning*, 6000–6010. PMLR.

Lee, C. Y. 1959. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technical Journal*, 38(4): 985–999.

Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A. 2019. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635*.

Meisel, W.; and Michalopoulos, D. 1973. A partitioning algorithm with application in pattern classification and the optimization of decision trees. *IEEE Transactions on Computers*, C-22(1): 93–103.

Mues, C.; Baesens, B.; Files, C. M.; and Vanthienen, J. 2004. Decision diagrams in machine learning: An empirical study on real-life credit-risk data. *Expert Systems with Applications*, 27(2): 257–264.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning optimal decision trees with SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 1362–1368.

Oliveira, A.; and Sangiovanni-Vincentelli, A. 1996. Using the minimum description length principle to infer reduced ordered decision graphs. *Machine Learning*, 25(1): 23–50.

Oliver, J. 1993. Decision graphs – An extension of decision trees. In *Proceedings of the 4th international workshop on artificial intelligence and statistics (AISTATS)*, 343—350.

Payne, H.; and Meisel, W. 1977. An algorithm for constructing optimal binary decision trees. *IEEE Computer Architecture Letters*, 26(09): 905–916.

Perez, G.; and Régin, J.-C. 2015. Efficient operations on MDDs for building constraint programming models. In *IJCAI 2015*.

Platt, J.; Cristianini, N.; and Shawe-Taylor, J. 2000. Large margin DAGs for multiclass classification. *Advances in Neural Information Processing Systems*, 547–553.

Sanner, S.; Uther, W.; and Delgado, K. 2010. Approximate dynamic programming with affine ADDs. In *AAMAS*, 1349–1356.

Serra, T. 2020. Enumerative branching with less repetition. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 399–416. Springer.

Shotton, J.; Nowozin, S.; Sharp, T.; Winn, J.; Kohli, P.; and Criminisi, A. 2013. Decision jungles: Compact and rich models for classification. *Advances in Neural Information Processing Systems*, 1–9.

Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2018. Compact-MDD: Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. In *IJCAI*, 1383–1389.

Verhaeghe, H.; Nijssen, S.; Pesant, G.; Quimper, C.-G.; and Schaus, P. 2020. Learning optimal decision trees using constraint programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 4765–4769. ISBN 9780999241165.

Vidal, T.; and Schiffer, M. 2020. Born-Again Tree Ensembles. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 9743–9753. Virtual: PMLR.

Ye, Q.; and Xie, W. 2020. Unbiased subdata selection for fair classification: A unified framework and scalable algorithms. *arXiv preprint arXiv:2012.12356*.

Zafar, M.; Valera, I.; Rodriguez, M.; and Gummadi, K. 2017. Fairness constraints: Mechanisms for fair classification. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, 54.