Journal of Artificial Intelligence Research 80 (2024) 875-918 Submitted 01/2024; published 07/2024

SAT-based Decision Tree Learning for Large Data Sets

André Schidler ASCHIDLER@AC.TUWIEN.AC.AT
Stefan Szeider SZ@AC.TUWIEN.AC.AT
Algorithms & Complexity,
TU Wien,
Favoritenstrasse 9-11,
1040 Vienna, Austria

Abstract

Decision trees of low depth are beneficial for understanding and interpreting the data they represent. Unfortunately, finding a decision tree of lowest complexity (depth or size) that correctly represents given data is NP-hard. Hence known algorithms either (i) utilize heuristics that do not minimize the depth or (ii) are exact but scale only to small or medium-sized instances. We propose a new hybrid approach to decision tree learning, combining heuristic and exact methods in a novel way. More specifically, we employ SAT encodings repeatedly to local parts of a decision tree provided by a standard heuristic, leading to an overall reduction in complexity. This allows us to scale the power of exact SAT-based methods to comparatively very large data sets. We evaluate our new approach experimentally on a range of real-world instances that contain up to several thousand samples. In almost all cases, our method successfully decreases the complexity of the initial decision tree; often, the decrease is significant.

1. Introduction

Decision trees are indispensable tools for the description, classification, and generalization of data (Rudin, 2019; Larose, 2005; Murthy, 1998; Quinlan, 1986). Since decision trees are easy to understand, they are particularly attractive for providing interpretable models for the data they represent. This aspect has been emphasized in recent years (Darwiche & Hirth, 2020; Doshi-Velez & Kim, 2017; Goodman & Flaxman, 2017; Lipton, 2018; Monroe, 2018). In this context, one prefers trees of *low complexity*, which are trees of low depth (length of a longest path from the root to a leaf) and/or of small size (total number of nodes). A decision tree of low depth guarantees a low number of tests required to classify a sample (desired if tests are expensive or even intrusive (Podgorelec, Kokol, Stiglic, & Rozman, 2002); a decision tree of small size limits the overall effort of understanding its decision-making and thus promotes transparency and interpretability. Decision trees are also used as part of larger implementations like functional synthesis, where the size of the decision tree can have a large impact on the runtime of the application (Golia, Roy, & Meel, 2020; Golia, Slivovsky, Roy, & Meel, 2021). However, inducing decision trees of the lowest complexity is NP-hard (Hyafil & Rivest, 1976) but fixed-parameter tractable under suitable parameterization (Eiben, Ordyniak, Paesani, & Szeider, 2023; Kobourov, Löffler, Montecchiani, Pilipczuk, Rutter, Seidel, Sorge, & Wulms, 2023; Ordyniak, Paesani, Rychlicki, & Szeider, 2024; Dabrowski, Eiben, Ordyniak, Paesani, & Szeider, 2024; Gahlawat & Zehavi, 2024).

Therefore, several exact methods (SAT and CP-based) have been proposed for that purpose (Bessiere, Hebrard, & O'Sullivan, 2009; Narodytska, Ignatiev, Pereira, & Marques-Silva, 2018; Avellaneda, 2020; Janota & Morgado, 2020; Hu, Siala, Hebrard, & Huguet, 2020). Although reducing the depth of the tree decreases the number of required tests, it does not guarantee that the number of tests is minimal (Izza, Ignatiev, & Marques-Silva, 2022). In this paper we focus on finding low-complexity decision trees.

Scalability is often an issue with exact methods. For this reason, we propose a novel approach to learning decision trees of low complexity. We combine the scalability of heuristic methods with the strength of encoding-based exact methods, thus taking the best of the two worlds. Our approach follows the principle of *SAT-based Local Improvement (SLIM)* which starts with a solution provided by a fast heuristic. It then repeatedly applies a SAT-based exact method locally to improve the solution. SLIM has shown to be effective in graph decomposition problems (Lodha, Ordyniak, & Szeider, 2017b; Fichte, Lodha, & Szeider, 2017; Lodha, Ordyniak, & Szeider, 2017a; Ganian, Lodha, Ordyniak, & Szeider, 2019), graph coloring (Schidler, 2022), boolean circuits (Kulikov, Pechenev, & Slezkin, 2022; Reichl, Slivovsky, & Szeider, 2023, 2024), anytime MaxSAT (Schidler & Szeider, 2024) and Bayesian network structure learning (Ramaswamy & Szeider, 2021a, 2021b, 2022). SLIM is similar to Large Neighborhood Search (Pisinger & Ropke, 2010), where SLIM distinguishes itself by combining a structurally constrained notion of neighbourhood with a complete method (SAT).

Key to our approach is a suitable notion of a *local instance* which is based on using the decision tree's subtrees. Given a subtree of the decision tree, a SAT solver tries to find a subtree of lower complexity that correctly classifies all the samples the selected subtree classifies correctly. Whenever the SAT solver is successful, the improved subtree can then replace the selected subtree. This, in turn, may lower the complexity of the whole decision tree. By adding weights and new classification categories to the local instance, we ensure that this replacement does not introduce misclassifications or increase the decision tree's complexity.

Because of the new classification categories and weights, a local instance poses a more complex classification problem. We show how SAT encodings can be generalized to accommodate non-binary weighted classification instances and propose a subtree selection strategy that avoids weights (Corollary 1). We further propose a new encoding based on a characterization of decision trees in terms of partitions (Theorem 3), which allows us to handle local instances of higher depth than it is possible with known encodings.

We establish a prototype implementation of our approach (DT-SLIM) and empirically evaluate it on data sets from the UCI Machine Learning Repository and prior work. Our experimental results are very encouraging: we can lower the complexity of heuristically obtained decision trees in almost all cases, in some cases significantly, without sacrificing much in terms of accuracy on unseen data.

We further apply DT-SLIM in a setting closer to the typical use of decision trees, where we apply pruning after reducing the complexity of the the decision tree using DT-SLIM. Pruning is a method where parts of the decision tree are removed that will most likely not generalize well to unseen data. Our results are again very encouraging, showing SLIMed and pruned decision trees often outperform pruned decision trees in both complexity and accuracy.

This paper is structured as follows. We will first give some necessary background and notation in Section 2. Next, we discuss the key contribution: how SLIM works in the context of decision trees (Section 4). In Section 5, we discuss how SAT encodings can induce decision trees, which includes a description of our new encoding DT_pb. Data reductions are necessary to solve many instances and are the topic of Section 6. We combine the previously introduced concepts in our approach DT-SLIM in Section 7 and give some practical considerations in Section 8. The last remaining piece in our approach is pruning, which is applied after SLIM and is briefly introduced in Section 9. Finally, we thoroughly test our approach against the state-of-the-art by a set of experiments in Section 10.

2. Preliminaries

Classification problems A *classification instance* over a set F of *features* is a pair $\mathcal{I} = (E, c)$ where E is a finite set of *samples* and c is a mapping that assigns each sample $e \in E$ a *class* $c(e)$. $C_\mathcal{I} = \{\, c(e) : e \in E \,\}$ is the set of *classes* of the instance $\mathcal{I}$. A sample $e \in E$ is a mapping that assigns each feature $f \in F$ a *value* $e(f)$. The *domain* of a feature $f \in F$ is denoted by $D(f)$ and the feature's domain relative to E is the set $D_\mathcal{I}(f) = \{\, e(f) : e \in E \,\}$. We call $\mathcal{I}$ a *binary classification instance* if $|C_\mathcal{I}| = 2$. We assume w.l.o.g. that the classes are ordered, particularly that we can determine a maximum over the set of classes. We say a feature is *numerical* if its domain contains only real numbers and otherwise we say it is *categorical*. We denote the set of numerical features by $N(F)$ and the categorical features by $R(F)$.

Decision Trees A *decision tree* is a rooted binary tree T with node set $V(T)$, arc set $A(T)$, and root r; T_v denotes the subtree of T rooted at $v \in V(T)$. The *depth* $d_T(v)$ of a node v in T is the length (i.e., number of edges) of the path from T's root r to v; thus r has depth 0. T's depth $d(T)$ is the largest depth over all its nodes; T's *size* is its number of nodes.

Each internal node $v \in V(T)$ is labeled by a feature $\text{feat}(v)$, a *threshold* $\text{thresh}(v) \in D(\text{feat}(v))$, and a *comparison operator* $\circ_v \in \{=, \leq\}$. If $\text{feat}(v)$ is numerical, then $\circ_v$ is $\leq$; if $\text{feat}(v)$ is categorical, then $\circ_v$ is $=$.[1] Each internal node has exactly two children, a left one and a right one. $L(T)$ denotes the set of T's leaves. We write $\text{feat}(T) = \{\, \text{feat}(v) : v \in V(T) \setminus L(T) \,\}$.

Consider a classification instance (E, c) over a set F of features and a decision tree T with $\text{feat}(T) \subseteq F$. For each $v \in V(T)$, we define a set $E_T(v) \subseteq E$: if r is the root of T, we put $E_T(r) = E$; if v_1 is the left and v_2 the right child of a node u, we put $E_T(v_1) = \{\, e \in E_T(u) : e(\text{feat}(u)) \circ_u \text{thresh}(u) \,\}$ and $E_T(v_2) = E_T(u) \setminus E_T(v_1)$. We define the *classification* $c_{\mathcal{I},T}(\ell)$ of a leaf $\ell \in L(T)$, with respect to the classification instance $\mathcal{I} = (E, c)$ as

$$c_{\mathcal{I},T}(\ell) = \arg\max_{c \in C_\mathcal{I}} |\{\, e \in E_T(\ell) : c(e) = c \,\}|,$$

ties are broken arbitrarily. We say T *correctly classifies* a sample $e \in E$ if for the unique leaf $\ell \in L(T)$ with $e \in E_T(\ell)$ we have $c(e) = c_{\mathcal{I},T}(\ell)$. T correctly classifies $\mathcal{I} = (E, c)$ if it

1. Note that in practice, one can enforce any of the two comparison operator for a specific feature by preprocessing. I.e., ordering a categorical domain and using the order as values, or use non-real representations of the real values.

°C	Weather	Day	Length	Hike
29.1	Sunny	Sat	3 h	Yes
22.3	Thunder	Mon	2 h	No
21.5	Rain	Thu	1 h	Yes
23.7	Rain	Fri	3 h	Yes
14.3	Rain	Wed	4 h	No
14.7	Sunny	Tue	3 h	Yes

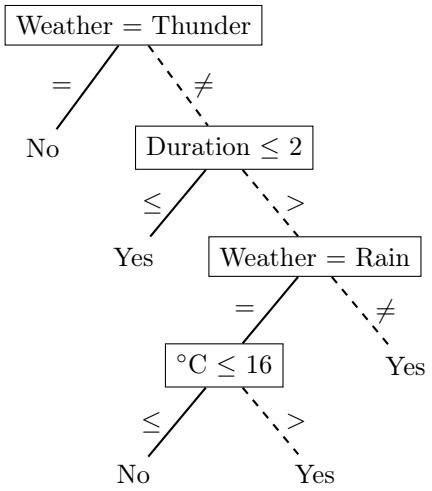

Figure 1: Left: A classification instance on when to hike and when not, with $|E| = 6$, $|F| = 4$, using numeric and categorical features. Right: a decision tree of depth 4 and size 9 for the instance on the left.

correctly classifies all the samples $e \in E$; in that case, we simply say T *is a decision tree for* $\mathcal{I}$. Figure 1 shows an example decision tree.

We use a *training set* (E, c) to induce the decision tree and a *test set* (E', c') to determine the *accuracy* of the tree, where $E \cap E' = \emptyset$. The accuracy of a decision tree on a given instance is the fraction of samples from E' it correctly classifies. We use the term *training accuracy* if the accuracy is measured on E instead of E'.

Problems In this paper we consider optimization methods for *complexity-optimal* decision trees:

Definition 1 (Complexity-optimal decision trees)**.** *Given a classification instance* $\mathcal{I}$ *and let* T *be a decision tree that correctly classifies* $\mathcal{I}$*:*

- T *is* size-optimal *if* T *has minimum* $|V(T)|$*.*
- T *is* depth-optimal *if* T *has minimum* $d(T)$*.*
- T *is* complexity-optimal *if it is either size-optimal or depth-optimal.*

Hence, we explore exact and heuristic methods for the following problem:

Definition 2 (Bounded-Complexity Decision Tree Induction)**.** *Given a classification instance* $\mathcal{I}$ *find a depth-optimal decision tree (*Bounded-Depth Decision Tree Induction*), or a size-optimal (*Bounded-Size Decision Tree Induction*) for* $\mathcal{I}$*.*

We extend the depth-optimality definition to *weighted classification instances*. A weighted classification instance is a tuple $\mathcal{I}_w = (E, c, d)$ where $\mathcal{I} = (E, c)$ is a classification instance, and d is a mapping that assigns each $c \in C_{\mathcal{I}}(E)$ a positive integer $d(c)$ called its *weight*. We purposfully overload d here, as we will use the depth as the weight. $\mathcal{I}$ and $\mathcal{I}_w$ have the same

decision trees, just the depth for these decision trees are defined differently for $\mathcal{I}$ and $\mathcal{I}_w$. Consider a decision tree T for $\mathcal{I}_w$. For a leaf ℓ of T with classification c (i.e., $c_{\mathcal{I},T}(\ell) = c$), we define the *weighted depth of ℓ in T* as $d_{w,T}(\ell) = d_T(\ell) + d(c)$. The *weighted depth* $d_w(T)$ of T is the maximum weighted depth over all its leaves.

In this paper we discuss SAT-based methods for complexity-optimal decision trees. These methods are both exact, finding either size-optimal or depth-optimal decision trees, and heuristic, finding decision trees of low depth or small size that correctly classify the instance.

3. Related Work

We limit the discussion in this section to the induction of optimal decision trees and refer the reader to the survey by Costa and Pedreira (2023) for a comprehensive summary of state-of-the-art methods for different decision tree induction tasks and algorithms. We discuss two different concepts of optimality: *complexity-optimal* decision trees and *accuracy-optimal* decision trees.

3.1 Complexity-Optimal Decision Trees

Complexity-optimal decision trees correctly classify all training set samples and have minimal depth and/or size. Hence, inducing complexity-optimal decision trees is a knowledge compilation task. Pruning is usually required for good accuracy on unseen data, as correctly classifying all training samples often causes overfitting. Inducing decision trees that correctly classify the training set and then pruning the decision tree follows the approach, which is not accuracy optimal, of widely used decision tree heuristics like CART and C4.5 (Quinlan, 1993; Breiman, Friedman, Olshen, & Stone, 1984). We discuss pruning in Section 9.

Several SAT-based methods for inducing complexity-optimal decision trees have been proposed. Initially, Bessiere et al. (2009) explored the use of a SAT encoding, but it performed poorly compared to their constraint programming approach. The first success was achieved by Narodytska et al. (2018): their proposed SAT encoding scaled to small instances. This success sparked a series of further research into SAT encodings for complexity-optimal decision trees (Avellaneda, 2020; Janota & Morgado, 2020; Schidler & Szeider, 2021; Ignatiev, Marques-Silva, Narodytska, & Stuckey, 2021; Shati, Cohen, & McIlraith, 2021), and theoretical work (Ordyniak & Szeider, 2021; Kobourov et al., 2023; Ordyniak et al., 2024; Dabrowski et al., 2024; Eiben et al., 2023).

The encoding by Narodytska et al. (2018) determines whether there is a decision tree that correctly classifies the instance with at most k nodes. Repeated calls with varying k can then determine a size-optimal decision tree for the instance. The encoding does not scale well if the number of samples or number of nodes increases. This observation led Janota and Morgado (2020) to try a different approach and encoding. Their encoding can find both size- and depth-optimal decision trees. The authors observed that the SAT solver fails to find a decision tree even if the number of possible tree structures is low given the depth or size limit, but that the SAT solver is much faster at labeling a given structure. Hence, their approach enumerates part of the structures separately and then asks the SAT solver to complete and label the structure. Avellaneda (2020) achieves good scalability by fixing the whole structure of the decision tree for a given depth d to a perfect binary tree. The SAT solver now only has to label the nodes and leaves. Hence, the encoding finds depth-optimal decision trees

(but can also optimize for size). The encoding by Shati et al. (2021) is based on Avellaneda's (2020) encoding. Here, they encounter a different scalability issue: the SAT solver has to consider each feature value as a possible threshold. Shati et al. (2021) encode this part differently: instead of explicitly considering every threshold, the SAT solver only labels the internal nodes with features. The threshold is then implicitly enforced by checking whether the paths the samples take are consistent with the used features. The encoding's scalability still dependents on the number of features, but much less on the number of different values. This makes the encoding the state-of-the-art SAT encoding for depth-optimal decision trees. We discuss more details of these approaches in Section 5 and evaluate them in Section 10.

3.2 Accuracy-Optimal Decision Trees

Related to complexity-optimal decision trees are accuracy-optimal decision trees: an accuracy-optimal decision tree has a constrained structure and maximizes the training accuracy. Maximizing the training accuracy of small decision trees follows the idea that good accuracy on the training samples correlates with good accuracy on unknown samples for small decision trees. Many approaches for inducing accuracy-optimal decision trees have been proposed using SAT (Hu et al., 2020; Shati et al., 2021), constraint programming (Verhaeghe, Nijssen, Pesant, Quimper, & Schaus, 2020), mixed integer programming (Bertsimas & Shioda, 2007; Bertsimas & Dunn, 2017; Zhu, Murali, Phan, Nguyen, & Kalagnanam, 2020; Aghaei, Gómez, & Vayanos, 2021; Hua, Ren, & Cao, 2022; Verwer & Zhang, 2017), binary integer programming (Verwer & Zhang, 2019), and using dedicated algorithms based on dynamic programming/branch & bound (Hu, Rudin, & Seltzer, 2019; Aglin, Nijssen, & Schaus, 2020a, 2020b; Lin, Zhong, Hu, Rudin, & Seltzer, 2020; Demirovic, Lukina, Hebrard, Chan, Bailey, Leckie, Ramamohanarao, & Stuckey, 2022; van der Linden, de Weerdt, & Demirovic, 2023).

A generalization of accuracy-optimality are sparsity-optimal decision trees. These do not maximize the accuracy, but minimize $R(v) = err + \alpha|L(T)|$, where err is the error rate, i.e., the inverse of the accuracy. The parameter α controls how small the induced decision tree will be. The idea is that parts of the decision tree that do not contribute much to the training accuracy will be removed by increasing α. This is advantageous as these parts are also less likely to contribute much to the testing accuracy and can cause overfitting. Several accuracy-optimal methods support the induction of sparsity-optimal decision trees (Bertsimas & Dunn, 2017; Hu et al., 2019; Lin et al., 2020; Demirovic et al., 2022; van der Linden et al., 2023). Further, several of the dynamic programming approaches support optimization goals beyond accuracy and sparsity, including non-linear metrics (Lin et al., 2020; Demirovic et al., 2022; van der Linden et al., 2023).

In the context of accuracy-optimal decision trees, the methods vary in how efficiently they can find an optimal decision tree. SAT, mixed integer programming, and CP approaches don't scale well to instances with larger numbers of samples (van der Linden et al., 2023). For SAT, this is true for both complexity- and accuracy-optimal decision trees, which additionally motivates the work in this paper. The advantage of these methods is that additional constraints can easily be added to encode additional requirements. Dynamic programming approaches scale much better with the number of samples. We will revisit this discussion in Section 10, when we compare the accuracy-optimal method STreeD (van der Linden et al., 2023) to our method.

Accuracy-optimality methods often scale better than complexity-optimal methods, as the corresponding decision trees are usually smaller, reducing the search space for the solver. Hence, finding accuracy-optimal decision trees for instances with many thousands of samples is feasible as long as the decision tree complexity is low. In this paper, we try to make these complexity-optimal methods scale better, albeit in a heuristic manner. Further, we will see in Section 10, that not all instances allow for small decision trees that capture the data well.

Most of the methods discussed in this section require a binary instance, exceptions are the SAT encoding by Shati et al. (2021), GOSDT (Lin et al., 2020) and OCT (Bertsimas & Dunn, 2017). General classification instances are then *binarized*, where a binary feature is introduced for each feature and a value we consider as a possible threshold. Considering all feature values occurring in an instance, can result in a huge binary instance that is hard to solve. Hence, several binarization methods have been proposed that strategically reduce the number of features by discarding some values from consideration. Unfortunately, this can lead to complexity- and accuracy-sub-optimal decision trees (Lin et al., 2020). One method is *bucketization* (Verwer & Zhang, 2019), which for each feature only considers those threshold values that partition the samples according to their class. Another method is picking a constant number of values for each feature at random, or dividing the values for a feature in quantiles and picking a value from each quantile (Günlük, Kalagnanam, Li, Menickelly, & Scheinberg, 2021). *Support sets* (Ibaraki, Crama, & Hammer, 2011; Ordyniak & Szeider, 2021) only select a subset of the binary features that is required to distinguish each sample from each other. Another method is inducing heuristic decision trees and only considering thresholds used in these trees (McTavish, Zhong, Achermann, Karimalis, Chen, Rudin, & Seltzer, 2022). The latter corresponds to the union of several support sets. We discuss this topic further in Section 6.

In practice, decision tree heuristics are commonly used, as they scale very well to very large instances. Since the introduction of CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993), several improvements have been proposed, e.g., *C5.0* which is more memory efficient and can handle larger datasets more effectively (Kuhn & Johnson, 2013), *Conditional Inference Trees* (Hothorn, Hornik, & Zeileis, 2006) differ from traditional decision tree algorithms by using a more statistically rigorous method for selecting splits in the tree. *Rotation Forest* (Rodriguez, Kuncheva, & Alonso, 2006) and *Perfect Random Tree Ensemble* (Cutler & Zhao, 2001) algorithms have been used primarily for ensemble methods; the core concept can enhance decision tree performance when used in isolation. *Hoeffding Trees* (Domingos & Hulten, 2000) are designed for high-speed data streams where storing the entire dataset for analysis is impractical.
However, C4.5 and CART set the standard for decision tree learning and are still used as benchmark models for comparison with newer algorithms. Despite their age, these algorithms are still effective for many problems, especially where the data is not excessively large or complex. While newer algorithms may offer improvements in certain areas (like handling large datasets, speed, or specific types of data), C4.5 and CART remain relevant for their simplicity, interpretability, and benchmark standards in decision tree algorithms.

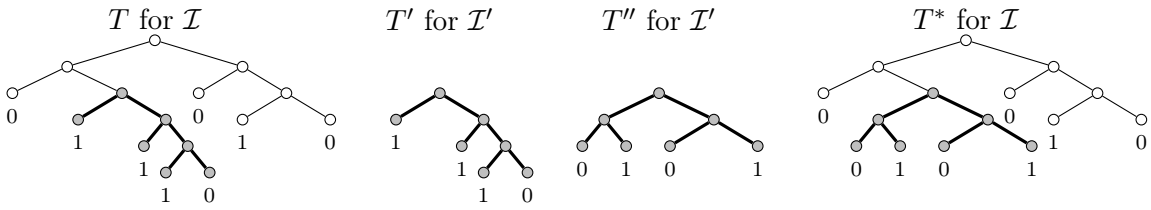

Figure 2: Local improvement workflow.

4. Local Improvement

Assume we are given a classification instance $\mathcal{I} = (E, c)$, which is too large for inducing a decision tree of smallest depth using an exact method such as a SAT encoding. We can use a heuristic method to compute a non-complexity-optimal decision tree T for $\mathcal{I}$. The idea of local improvement is to repeatedly select subtrees T' of T that induce a *local instance* $\mathcal{I}'$ that is small enough (possibly after further simplification and reduction) to be solved by an exact method. Once we have found an improved decision tree T'' for local instance $\mathcal{I}'$ of smallest depth (or at least a depth that is smaller than the depth of T'), we can *replace* T' in T with the improved tree T'', obtaining a new decision tree T^* for $\mathcal{I}$ that improves upon the heuristic tree T. Figure 2 illustrates this process. We will focus on depth reduction in the theoretical part. Size reduction works analogously and will be discussed explicitly in Section 8.

4.1 Local Instances

We need to develop a suitable concept of a local instance to instantiate this general idea. This concept must guarantee that after replacing the subtree, the new decision tree T^* still correctly classifies all samples. This task becomes harder if we allow that some leaves of T' are internal nodes of T, i.e., $L(T') \setminus L(T) \neq \emptyset$. In this case, after replacement T^* generally does not correctly classify $\mathcal{I}$. We achieve correct classification by extending the nodes in $L(T'') \setminus L(T)$ with parts of the original tree T to complete T^*. The key to our solution is based on the introduction of new classes, as follows.

Definition 3 (Local Instance)**.** *Let T be a decision tree for instance $\mathcal{I}$, T' be a subtree of T, and r' be the root of T'. Further, let $\ell_1, \ldots, \ell_k \in L(T') \setminus L(T)$ be those leaves of T' that are internal nodes of T, where possibly $k = 0$, and let $s = \max_{e \in E} c(e)$ be the last class used in $\mathcal{I}$. The* local instance induced by *T' is the pair $\mathcal{I}' = (E', c')$ where $E' = E_T(r')$ and c' is the mapping defined by*

$$c'(e) = \begin{cases} c'(e) = s + i & \text{if } e \in E_T(\ell_i) \text{ for some } 1 \leq i \leq k; \\ c'(e) = c(e) & \text{otherwise.} \end{cases}$$

Although $|E_T(r')|$ is independent of the subtree below r', the local instance size can be further reduced by the reduction methods discussed in Section 6.

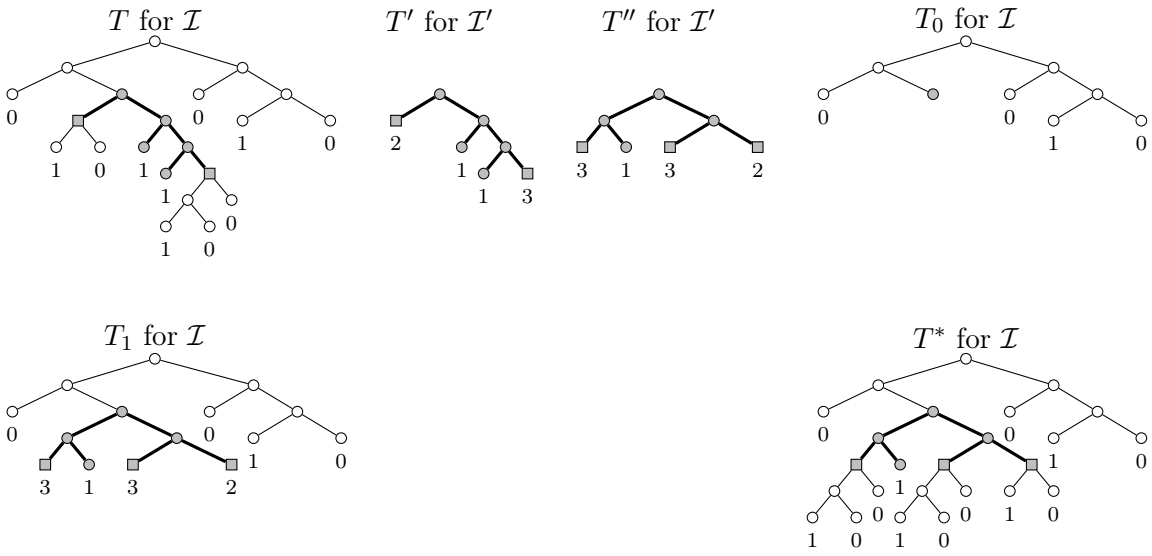

Figure 3: Local improvement workflow using special leaves. The numbers indicate the leaves' classes; squares indicate special leaves.

4.2 Local Replacement

Let T'' be any decision tree for the local instance $\mathcal{I}'$. Since T'' correctly classifies $\mathcal{I}'$, T'' will contain for each $i \in \{1, \ldots, k\}$ at least one leaf m such that $c_{\mathcal{I},T}(m) = \{s + i\}$. We call such a leaf m a *special leaf with classification* $s + i$. We can see an example of a local instance with special leaves in the top left part of Figure 3. Here, the special leaves are marked as squares and are exactly those leaves of the subtree T' that have descendants in T. In this example, the depth of T' is 3, while without special leaves, we would need to include all descendants, raising the depth to 5. Hence, special leaves allow us to reduce the depth of the subtree we consider for the local instance, which makes reducing the depth easier for the SAT encoding, as we will discuss in Section 5.

To describe how the new decision tree T^* is constructed, we need the following operation on decision trees: Let $T^{\uparrow}, T^{\downarrow}$ be decision trees, x a leaf of $T^{\uparrow}$ and y the root of $T^{\downarrow}$. The *extension of* $T^{\uparrow}$ *at* x *with* $T^{\downarrow}$ is the decision tree $T^{\cup}$ obtained from $T^{\uparrow}$ and $T^{\downarrow}$ by taking the vertex-disjoint union of the two trees and identifying x with y.

To construct T^*, we start with the decision tree T_0 obtained from T by deleting all descendants of the root r' of subtree T'. From T_0 we obtain T_1 by extending it at r' by the improved tree T''. Finally, from T_1 we obtain T^* by extending each special leaf m with classification $s + i$ with a new copy $T^m_{\ell_i}$ of T_{ℓ_i}. Figure 3 shows an example of this process.

The next theorem states that this replacement process is sound.

Theorem 1. *T^* correctly classifies $\mathcal{I}$.*

Proof. For showing the claim, let ℓ^* be any leaf of T^*. We will show that $|\{\, c(e) : e \in E_T(\ell^*)\,\}| \leq 1$ and therefore, that all samples at the leaf are classified correctly. Let P be the unique path in T^* from the root of T^* to ℓ^*. We distinguish several cases.

Case 1: Path P does not run through the subtree's root r'. Hence ℓ^* is also a leaf of T. Since the original decision tree T correctly classifies $\mathcal{I}$ by assumption, $1 \geq |\{\, c(e) : e \in E_T(\ell^*)\,\}| = |\{\, c(e) : e \in E_{T^*}(\ell^*)\,\}|$.

Case 2: Path P runs through the subtree's root r'.

Subcase 2.1: ℓ^* is a leaf of the improved tree T''. Since ℓ^* is also a leaf of T^*, it isn't a special leaf. Since T'' correctly classifies $\mathcal{I}'$, the latter implies that $c_{\mathcal{I},T^*}(\ell^*) = c_{\mathcal{I}',T''}(\ell^*)$—the leaves have the same classification in both trees—and, therefore, we have $|\{\, c(e) : e \in E_{T''}(\ell^*)\,\}| \leq 1$, hence again $|\{\, c(e) : e \in E_{T^*}(\ell^*)\,\}| \leq 1$.

Subcase 2.2: ℓ^* is not a leaf of the improved tree T''. Consequently, path P runs through a special leaf m of T''. Let $s+i$ be the classification of m. By construction, the subtree T^*_m of T^* is a copy of the subtree T_{ℓ_i} of the original tree T, and the leaf ℓ^* of T^*_m is the copy of a leaf ℓ of T_{ℓ_i}. Since by definition the classification of m is $c_{\mathcal{I}',T''}(m) = \{s+i\}$, we have $E_{T^*}(m) \subseteq E_T(\ell_i)$. Consequently $E_{T^*}(\ell^*) \subseteq E_T(\ell)$. Since T correctly classifies $\mathcal{I}$, $|\{\, c(e) : e \in E_T(\ell)\,\}| \leq 1$, and from $E_{T^*}(\ell^*) \subseteq E_T(\ell)$ we thus get $|\{\, c(e) : e \in E_{T^*}(\ell^*)\,\}| \leq 1$. □

Let us now turn to the question of decreasing the depth of the input decision tree T employing such a local replacement. This does not work out of the box: Even when $d(T'') < d(T')$ it still can happen that $d(T^*) > d(T)$, since the depth of a special leaf v of T'' of classification $s+i$ can be larger than the depth of the corresponding leaf ℓ_i of T', resulting in a larger depth of T^* if the subtree attached to v at T^* is large.

To overcome this problem, we enrich the local instance with additional information, defining a weighted version of the classification problem.

4.3 Weighted Classification

We will use weighted classification instances (see Section 2) to ensure that the depth of the original decision tree never increases. Remember that a weighted classification instance is a tuple $\mathcal{I}_w = (E, c, d)$, that in addition to defining a classification instance also defines a weight for each class $c \in C_{\mathcal{I}}(E)$. We will show how locally decreasing the weighted depth of the weighted local instance within our local improvement setting allows us to decrease the depth of the original decision tree.

Definition 4 (Weighted Local Instance)**.** *Let $\mathcal{I} = (E, C)$ be a classification instance, T a decision tree for $\mathcal{I}$, T' a subtree of T, $\mathcal{I}' = (E', C')$ the local instance induced by T', T'' a decision tree for $\mathcal{I}'$ and T^* the improved decision tree constructed using T, T' and T''. Let $\mathcal{I}'_w = (E', C', d)$ denote the* weighted local instance*, where the weights for $c \in c'(E')$ are defined as follows: if $c = s + i$ then $d(c) = d(T_{\ell_i})$; if $c \leq s$, then $d(c) = 0$.*

We note that T' is a decision tree of the weighted local instance, and hence $d_w(T')$ is defined.

Theorem 2. *If $d_w(T'') \leq d_w(T')$ then $d(T^*) \leq d(T)$.*

Proof. Assume $d_w(T'') \leq d_w(T')$ and consider a longest path P^* in T^* between the root of T^* and a leaf ℓ^* of T^*. We denote the length of a path P^* by $\mathrm{len}(P^*)$.

If P^* does not pass through r'', the root of T'', then it is also a root-to-leaf path of T, and so $d(T^*) = \mathrm{len}(P^*) \leq d(T)$, and the claim of is established.

It remains to consider the case where P^* passes through r''. Let P be a longest path in T which passes through r''. Consequently, $\text{len}(P) \leq d(T)$.

We can write $\text{len}(P^*) = \text{len}_0^* + \text{len}_1^* + \text{len}_2^*$ where len_0^* is the length of the part of P^* between the root of T^* and r'', len_1^* is the length of the part of P^* between r'' and a leaf of T'', and len_2^* is the length of the part of P^* between a leaf of T'' and ℓ^*. It is possible that $\text{len}_2^* = 0$.

Similarly, we can write $\text{len}(P) = \text{len}_0 + \text{len}_1 + \text{len}_2$, where the three integers are defined similarly, using len_1 for the length of the part of P inside T'.

By the definition of the weights, we have $\text{len}_1 + \text{len}_2 = d_w(T')$, and $\text{len}_1^* + \text{len}_2^* = d_w(T'')$. Since $d_w(T'') \leq d_w(T')$, $\text{len}_1^* + \text{len}_2^* \leq \text{len}_1 + \text{len}_2$. Since $\text{len}_0^* = \text{len}_0$ by construction, this gives $d(T^*) = \text{len}(P^*) \leq \text{len}(P) \leq d(T)$, as claimed. □

We now identify a special case of Theorem 2 where we only need to consider the unweighted local instance and still ensure that $d(T^*) \leq d(T)$. Let us call a subtree T' of T to be *safe* if for every leaf ℓ of T' it holds that $d(T') \leq d_w(T') - d_T(\ell)$.

Corollary 1. *If T' is safe and $d(T'') \leq d(T')$ then $d(T^*) \leq d(T)$.*

Proof. Let T' be a safe subtree with $d(T'') \leq d(T')$. Let ℓ'' be a leaf of T'' with $d_{w,T''}(\ell'') = d_w(T'')$ and let c be the classification of ℓ'' in T''. From the definitions we get $d_w(T'') = d_{w,T''}(\ell'') = d(c) + d_{T''}(\ell'') \leq d(c) + d(T'') \leq d(c) + d(T')$. Since T' is safe, we have $d(T') \leq d_w(T') - d(c)$, and so we get from $d_w(T'') \leq d(c) + d(T')$ that $d_w(T'') \leq d(c) + d_w(T') - d(c) = d_w(T')$. By Theorem 2, $d(T^*) \leq d(T)$ follows. □

This concludes our concept of local improvement for decision trees. Next, we discuss how SAT-based decision tree induction works.

5. SAT Encodings

The subsequent SAT encodings address the problem *Bounded-Depth Decision Tree Induction*: Given a classification instance $\mathcal{I}$ over a classification scheme, find a decision tree of minimal depth that correctly classifies $\mathcal{I}$. The encodings also allow for minimizing the size for a given depth.

A SAT approach to Bounded-Depth Decision Tree Induction, given a classification instance $\mathcal{I}$ and an integer d, entails formulating a propositional formula $F(\mathcal{I}, d)$, called the *encoding*, which is satisfiable if and only if there exists a decision tree of depth at most d that correctly classifies $\mathcal{I}$. The encoding is then tested by a SAT solver with increasing values for d until it becomes satisfiable, i.e., represents a (depth-optimal) decision tree. Given a size z, we can extend this idea to a formula $F(\mathcal{I}, d, z)$, which is satisfiable if and only if there exists a decision tree of depth at most d that has at most z many nodes and correctly classifies $\mathcal{I}$. Introducing the size as a second minimization goal offers the possibility for different optimization strategies, prioritizing one or the other.

Different encodings have been proposed that mainly differ in two respects: (i) how they model the decision tree and (ii) how they assign thresholds to internal nodes. A common way to represent thresholds is via binary encodings, where each possible threshold value is represented by one binary variable. This requires $\sum_{f \in F} |D_{\mathcal{I}}(f)|$ many binary variables if all values are considered as thresholds, resulting in large encodings for instances with large

feature domains. Not considering all values as thresholds can cause the decision tree to be deeper or larger than necessary.

We will first discuss our encoding DT_pb, and then discuss the main ideas for encodings from related work.

5.1 DT_pb

The idea behind DT_pb is to formulate the problem in terms of partitions: Starting from a partition containing only the set of all samples, a decision tree refines this partition more and more, until at the leaves, each set in our partition contains only samples of a single class. This approach has been used successfully for different graph-related problems and was introduced by Heule and Szeider (2015) for clique-width computation. We first reformulate the problem of finding a decision tree of a given depth for a classification instance $\mathcal{I}$ by partitioning the set of samples (Theorem 3). We then directly convert this definition into a propositional CNF formula $F(\mathcal{I}, d)$, that is satisfiable if and only if a decision tree of depth d that correctly classifies $\mathcal{I}$ exists (Theorem 4).

Let $\mathcal{I} = (E, c)$ be a classification instance and $\mathcal{S} = (S_0, \ldots, S_d)$ a sequence of partitions of E. We refer to the classes as *groups* and partitions $S_0, \ldots, S_d$ as the *levels* $0, \ldots, d$. $\mathcal{S}$ is a *DT-sequence that correctly classifies* $\mathcal{I}$ if the following conditions hold.

DT1 $S_0 = \{E\}$.

DT2 For all $1 \leq m \leq d$ it holds that, for each group $g \in S_{m-1} \setminus S_m$, there are groups $g', g'' \in S_m$ with $g = g' \cup g''$, such that for some $f \in \text{feat}(E)$ and $t \in D_{\mathcal{I}}(f)$ it holds that $e'(f) \circ t$ for all $e' \in g'$, and there exists no $e'' \in g''$, such that $e''(f) \circ t$, where $\circ$ is $\leq$ if f is numerical and $=$ if f is categorical.

DT3 For each $g \in S_d$ it holds that for all $e_1, e_2 \in g$ we have $c(e_1) = c(e_2)$.

We note that the definition implies that S_m is a refinement of S_{m-1}, for $1 \leq m \leq d$. The definition of DT-sequences corresponds to the definition of $E_T(v)$ and it is easy to see that a decision tree of depth d can be converted into a DT-sequence of length $d + 1$; and the other way around. This leads us to the following theorem.

Theorem 3. *A classification instance can be classified by a decision tree of depth d if and only if it can be classified by a DT-sequence of length d.*

Figure 4 shows an example DT-sequence that follows the example shown in Figure 1.

We encode a DT-sequence of length d for $\mathcal{I} = (E, c)$ where $E = \{e_1, \ldots, e_n\}$ and $F = \text{feat}(E) = \{f_1, \ldots, f_k\}$. The result of our encoding is a propositional formula $F(\mathcal{I}, d)$. This formula is satisfiable if and only if there exists a DT-sequence of length d, and therefore a decision tree of depth d that correctly classifies $\mathcal{I}$.

We use the variables

- $g_{i,j,m}$, for $1 \leq i < j < n$, $0 \leq m \leq d$, with the semantics that $g_{i,j,m}$ is true if and only if samples e_i and e_j are in the same group at level m,
- $s_{i,m,\ell}$ for $1 \leq i \leq n$, $0 \leq m < d$, $1 \leq \ell \leq k$, where $s_{i,m,\ell}$ is true if and only if the group of sample e_i at level m is split into two groups using feature f_ℓ.

Id	°C	Weather	Day	Length	Hike
e_1	29.1	Sunny	Sat	3 h	Yes
e_2	22.3	Thunder	Mon	2 h	No
e_3	21.5	Rain	Thu	1 h	Yes
e_4	23.7	Rain	Fri	3 h	Yes
e_5	14.3	Rain	Wed	4 h	No
e_6	14.7	Sunny	Tue	3 h	Yes

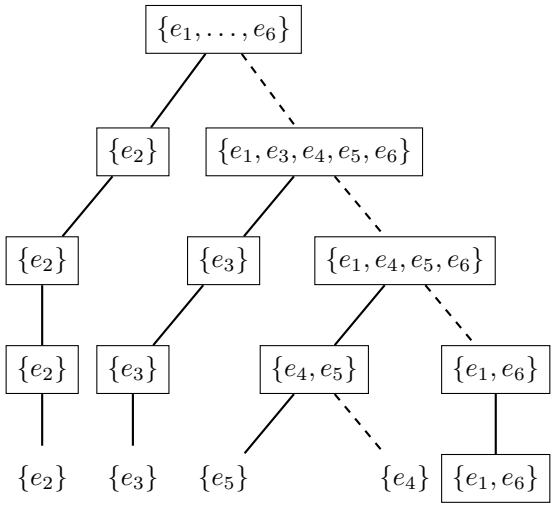

Figure 4: Left: The example classification instance used in Figure 1. Right: a DT-sequence corresponding to the decision tree in Figure 1.

- $l_{i,m}$ for $1 \leq i \leq n$, $0 \leq m < d$, that is true if and only if the sample e_i at level m is in the group where $e(f) \circ t$.

At the start, i.e., the first set of partitions in the sequence, all samples belong to the same group (DT1). We add the unary clauses

$$\bigwedge_{1 \leq i < j \leq n} g_{i,j,0}.$$

At the last level, all samples in one group must belong to the same class (DT3). We enforce this by adding the unary clauses

$$\bigwedge_{\substack{1 \leq i < j \leq n \\ c(e_1) \neq c(e_2)}} \neg g_{i,j,d}$$

.

The remaining clauses enforce DT2. As S_m is a refinement of S_{m-1}, we have to ensure that samples in different groups cannot be in the same group at a higher level. We state this by adding the clauses

$$\bigwedge_{\substack{1 \leq i < j \leq n \\ 0 \leq m < d}} g_{i,j,m} \vee \neg g_{i,j,m+1}.$$

Next, we ensure that at each level m, for every sample i there exists a corresponding feature to satisfy. For this purpose, we add the clauses

$$\bigwedge_{\substack{1 \leq i \leq n \\ 0 \leq m < d}} \bigvee_{1 \leq \ell \leq k} s_{i,m,\ell},$$

and ensure consistency within groups by adding the clauses

$$\bigwedge_{\substack{1\leq i<j\leq n,\\ 0\leq m<d,\\ 1\leq \ell\leq k}} \neg g_{i,j,m} \vee \neg s_{i,m,\ell} \vee s_{j,m,\ell}.$$

The next step in encoding DT2 is determining in which group the sample is in the next level. We distinguish between numerical and categorical features and add the following clauses. First, we handle categorical features and samples that agree on this feature value:

$$\bigwedge_{\substack{f_\ell\in R(F),\\ 1\leq i,j\leq n, i\neq j\\ e_i(f)=e_j(f),\\ 0\leq m<d}} \neg g_{i,j,m} \vee \neg s_{i,m,\ell} \vee \neg l_{i,m} \vee l_{j,m}.$$

Next, we handle those samples that disagree on the feature value:

$$\bigwedge_{\substack{f_\ell\in R(F),\\ 1\leq i<j\leq n\\ e_i(f)\neq e_j(f),\\ 0\leq m<d}} \neg g_{i,j,m} \vee \neg s_{i,m,\ell} \vee \neg l_{i,m} \vee \neg l_{j,m}.$$

For numerical features, we handle the sample based on the ordering of the feature value. We use the shorthand notation g^* as follows:

$$g^*_{i,j,m} = \begin{cases} g_{i,j,m} & \text{if } i < j; \\ g_{j,i,m} & \text{otherwise.} \end{cases}$$

We can now state that the split in two groups has to be consistent with the ordering of the feature values:

$$\bigwedge_{\substack{f_\ell\in N(F),\\ 1\leq i,j\leq n\\ e_i(f)\leq e_j(f),\\ 0\leq m<d}} \neg g^*_{i,j,m} \vee \neg s_{i,m,\ell} \vee \neg l_{j,m} \vee l_{i,m}.$$

For the case the two features have the same feature value, we add the following clauses:

$$\bigwedge_{\substack{f_\ell\in N(F),\\ 1\leq i<j\leq n\\ e_i(f)=e_j(f),\\ 0\leq m<d}} \neg g^*_{i,j,m} \vee \neg s_{i,m,\ell} \vee \neg l_{i,m} \vee l_{j,m}.$$

We finalize the encoding of DT2 by ensuring that the refinement into groups is according to the values in l:

$$\bigwedge_{\substack{1\leq i<j\leq n,\\ 0\leq m<d}} g_{i,j,m+1} \leftrightarrow (g_{i,j,m} \wedge (l_{i,m} \leftrightarrow l_{j,m})).$$

By construction of the formula and from Theorem 3 we obtain the following result.

Theorem 4. *$F(\mathcal{I}, d)$ is satisfiable if and only if there exists a decision tree of depth at most d that correctly classifies $\mathcal{I}$.*

The number of clauses in $F(\mathcal{I}, d)$ is $O(|E|^2 \cdot |\text{feat}(E)| \cdot d)$. While most of these clauses are short, the number of literals per clause is in $O(|\text{feat}(E)|)$. Therefore, the main factor determining the encoding size is the number of samples and not the depth.

Encoding Weights. We can encode weights with DT_pb by using different maximum depths for the different classes. Let $d_{\min}$ be the lowest weight among all classes. Given a class c, for all $e_i, e_j \in E$ such that $C(e_i) = c, C(e_j) \neq c$, we add the clauses $\neg g_{i,j,w}$, where $w = d - d(c) + d_{\min}$ is the allowed depth in regards to the weight of c.

Minimizing Size. We define the formula $F(\mathcal{I}, d, z)$ for limiting the size of the decision tree. The following modification does not guarantee that the decision tree has at most z nodes but that the decision tree has at most z leaves. For this purpose, we add variables z_i for $1 \leq i \leq n$, where z_i is true if e_i is the first sample in its group, i.e., e_i is not in any leaf that correctly classifies any e_j with $1 \leq j < i$. We express these semantics with the following clauses:

$$\bigwedge_{\substack{1 \leq j \leq n, \\ 0 \leq m < d}} z_j \vee \bigvee_{\substack{1 \leq i < j, \\ c(e_i) = c(e_j)}} g_{i,j,d}.$$

The number of leaves can then be restricted using any cardinality constraint encoding. We use *totalizer* constraints (Bailleux & Boufkhad, 2003), as they performed best for our purposes.

5.2 Further Encodings

We discuss representative encodings from the literature. Each of these encodings consists of three parts that they approach differently:

1. Encoding the decision tree structure.
2. Encoding the labeling of the internal nodes.
3. Encoding the paths that the samples take. This is then used to verify the correct classification of the sample.

The classes for the leaves implicitly follow from the paths of the samples.

The encoding by Narodytska et al. (2018) defines a propositional Formula $F(\mathcal{I}, z)$, which is satisfiable if and only if there exists a decision tree for $\mathcal{I}$ with at most z nodes. In the encoding, the structure of the z many nodes is not fixed, and the solver can arrange them as needed. Further, the encoding requires the instance to be binarized and, consequently, only labels the internal nodes with features. The dynamic structure of the decision tree limits the performance of this encoding, as it requires many clauses to express the relationship between nodes, more precisely, the encoding size is quadratic in the number of nodes. The dynamic structure also makes encoding the samples' paths less efficient. Consequently, finding a model for $F(\mathcal{I}, z)$ is comparatively hard, as our experiments show (Section 10). The advantage of this encoding is that it minimizes the size of the decision tree without fixing the depth.

Janota and Morgado (2020) devised a new encoding that does not encode the structure of the decision tree, but the paths of the decision tree. The encoding then enforces that these

paths are consistent with a decision tree. Hence, the structure of the decision tree is encoded implicitly, which makes the encoding more succinct. Indeed, the encoding's size is now only linear in the number of nodes. The samples are then associated with exactly one path. As in the previous encoding, the instance must be binary. Further, the encoding is parameterized by the depth and the number of leaves of the decision tree, where the number of leaves corresponds to the number of paths in the decision tree. Janota and Morgado (2020) noticed that both their encoding and the encoding by Narodytska et al. (2018) often struggle even for very simple instances. Specifically, the encodings struggle on instances, where it would be possible to enumerate all possible decision trees in a reasonably short time. Additionally, the encodings are good at labeling the nodes of a decision tree if the decision tree's structure is fixed. Janota and Morgado (2020) used these observations in an incremental approach: instead of computing a complexity-optimal decision tree in one SAT call, they enumerate all possible tree structures of a specific depth. These enumerated structures are usually incomplete and for each enumerated structure they call the SAT solver. The SAT solver then completes the decision tree. Their experiments show that this "topology enumeration" indeed speeds up the search for both encodings.

Avellaneda (2020) achieved even better scalability by fixing the structure of the decision tree to a perfect tree of a given depth d. Therefore, enumerating the structures is not necessary and the SAT solver only has to label the nodes with binary features, as this encoding expects a binary instance. The paths of the samples are encoded explicitly, which is straightforward, due to the fixed structure. Since the encoding uses a perfect tree, it requires $\Omega(2^d)$ many clauses. In comparison to the encoding by Narodytska et al. (2018), the encoding's size is now linear and not quadratic in the number of nodes.

Shati et al. (2021)'s encoding also fixes the structure of the decision tree to a perfect tree. The key feature is that the encoding does not require a binary instance. Instead, the internal nodes of the decision tree are only labeled with features. The thresholds are encoded implicitly using the samples' paths, as encoded in the previous encoding. The encoding enforces that these paths are consistent with a threshold as follows. For an internal node and potential feature, the paths that the samples take at this node have to be consistent with a single split, i.e., for categorical features all samples going to the left have to have the same feature value and for numerical features all the samples going to the left have to have a feature value lower than the samples going to the right. The threshold is then implied. This encoding's size is, therefore, only dependent on the number of features, not the domain size, which can severely decrease the overall size compared to the encoding by Avellaneda (2020).

Our encoding DT_pb differs in several ways from the other encodings. Similar to the encoding by Shati et al. (2021), DT_pb does not require a binary instance. Hence, the encoding size depends (linearly) on the number of features. DT_pb does not encode the structure of the decision tree besides the maximum depth. The structure is implicit in the refinement, which makes the depth only a linear factor in the encoding size. This is a drastic reduction as compared to the previous encodings. Further, while the previous encodings require an explicit encoding of each leaf's class, this is also implicit in DT_pb, making the size independent of the number of classes. DT_pb also does not encode the samples' paths, as these are implicit in the refinement. The weakness of this encoding is that it requires one variable per pair of samples, making the encoding size quadratic in the number of samples, while the size of all other encodings is linear in the number of samples. Therefore, DT_pb

performs better than other encodings, whenever the complexity-optimal decision tree has large depth but few samples, e.g., instances that require one very deep branch in a comparatively small decision tree. This case does not occur often for full classification instances, but occurs frequently for local instances created by DT-SLIM. When using DT-SLIM, instances may also have many classes due to special leaves, which is also comparatively easy for DT_pb.

All the SAT encodings discussed here do not scale well to instances with many samples, this is due to the explicit encoding of a path for each sample in all encodings but DT_pb. These constraints are necessary to ensure the decision tree is correctly classifying the instance. We discuss the empirical evaluation of how different encodings perform in Section 10. Further, the encodings can easily be adapted to induce accuracy-optimal decision trees using MaxSAT, as has been shown by Hu et al. (2020) and Shati et al. (2021). Instead of forcing the samples to be classified correctly, the corresponding clause becomes a soft clause and the MaxSAT solver then optimizes the accuracy. A similar approach could also be used for our encoding.

6. Instance Size Reduction

As we have seen in the previous section, the number of samples is one of the main factors determining encoding size. Encoding size, in turn, often correlates with the solving time. Since we cannot reduce the optimal size or depth, we try to reduce the number of samples. Hence, we try to speed up solving time by reducing the encoding size, and since we cannot reduce the required depth, we have to reduce the number of samples using the data reduction methods proposed subsequently. Additionally, our methods often remove features, further decreasing the encoding size.

The main idea behind our methods is that we can ignore duplicate samples: given two samples with the same values for all features, we can remove one of them and still obtain a classifying decision tree for the original instance. Hence, these data reduction methods strategically modify the samples to reduce the number of values in each feature's domain. This, in turn, increases the chance that two samples have the same values. The drawback of these data reduction methods is that we lose optimality: a complexity-optimal decision tree for the reduced instance might be deeper and/or larger than a complexity-optimal decision tree for the unreduced instance. The methods and concerns are closely related to binarization, discussed in Section 3.

We formalize this idea using *active domains* $D^A_{\mathcal{I}}(f) \subseteq D_{\mathcal{I}}(f)$: the set of domain values of a feature f that we consider as possible thresholds. We first show how small active domains help to reduce the number of samples and then introduce our methods for finding small active domains.

6.1 Removing Samples

Given active domains $D^A_{\mathcal{I}}(f) \subseteq D_{\mathcal{I}}(f)$ for each feature f, we create the reduced instance $\mathcal{I}^r = (E^r, c^r)$ as follows. First, we remove all features f with $D^A_{\mathcal{I}}(f) = \emptyset$. Next, we add for each numerical feature f the value $\max D_{\mathcal{I}}(f)$ to the feature's active domain. Finally, we add for each $e \in E$ a reduced sample e^r to E^r defined as

$$e^r(f) := \begin{cases} e(f), & \text{if } e(f) \in D^A_{\mathcal{I}}(f) \\ \min\{\, t \in D^A_{\mathcal{I}}(f) : t > e(f) \,\}, & \text{if } f \text{ is numerical} \\ \text{unused}, & \text{otherwise.} \end{cases}$$

Further, c^r is defined such that for each $e^r \in E^r$ where e^r corresponds to an unreduced sample $e \in E$, it holds that $c^r(e^r) = c(e)$.

Here, the *unused* value for categorical features is a special value that is not used as a threshold when inducing a decision tree for $\mathcal{I}^r$. Hence, at every node that uses the feature, the corresponding samples will always take the right path. The reduction in the number of samples is then automatically achieved whenever several samples in E map to one sample in E^r.

Since in a decision tree for $\mathcal{I}^r$ each sample's path is, by construction, the path that the unreduced sample would take, we observe the following:

Observation 1. *A decision tree that correctly classifies the reduced instance $\mathcal{I}^r$ also correctly classifies $\mathcal{I}$.*

Next, we discuss our methods for computing small active domains. Our first method applies only to numerical features, while the second method applies to both numerical and categorical features.

6.2 Bucketization

Bucketization (Verwer & Zhang, 2019) aims to reduce the active domain size of a single numerical feature independent of the other features. This can already drastically reduce the encoding size for encodings that use a binary representation for the possible thresholds.

For each numerical feature $f \in F$, we denote by $E_f[i]$ the i-th sample in E sorted by the values for f. We now define

$$D^A_{\mathcal{I}}(f) := \{\, E_f[i](f) : 1 \leq i \leq |E| - 1, c(E_f[i]) \neq c(E_f[i+1]) \,\}.$$

In other words: for each numerical feature, we only consider those values that partition the samples according to the samples' class.

The runtime of this method is in $O(|F| \cdot |E| \cdot \log |E|)$, as we have to sort the samples according to each feature. However, for most non-trivial instances, it holds that $|F| \ll |E|$. Although using these methods' active domains may lead to more complex decision trees (Lin et al., 2020), we could not observe this sub-optimality in our experiments. We do not use other related methods that could reduce even more values, such as uniformly picking a constant number of values, or using quantiles (Günlük et al., 2021; van der Linden et al., 2023), as this could cause a decrease in training accuracy.

6.3 Feature Reduction

Classification instances with fewer features result in smaller SAT encodings, and it is, therefore, useful to remove features as long as we do not misclassify samples in the original instance. A *support set*, for an instance $\mathcal{I} = (E, c)$ over a feature set F, is a subset $F' \subseteq F$, such that for any two samples that have a different class, the support set contains at least

one feature the two samples disagree on (Ibaraki et al., 2011; Ordyniak & Szeider, 2021). Removing features as long as the remaining features form a support set guarantees that the induced decision tree will still classify the original instance. However, we observed that it is rarely possible to remove features this way for instances with few features and large domains, as in this case, the set of features is often already a minimal support set.

We, therefore, refine the notion of support sets to *threshold support sets*, which gives us more potential for data reduction. A threshold support set S for instance $\mathcal{I}$ is a set of pairs (f, t) with $f \in F$ and $t \in D_{\mathcal{I}}(f)$. Further, for any two samples $e_1, e_2 \in E$ with $c(e_1) \neq c(e_2)$, S contains a tuple (f, t) such that t *separates* e_1 *and* e_2 *on feature* f, which means that, if f is numerical, then $e_1(f) \leq t < e_2(f)$ or $e_1(f) > t \geq e_2(f)$, and if f is categorical, then $e_1(f) = t \neq e_2(f)$ or $e_1(f) \neq t = e_2(f)$. Given a threshold support set S, we define the active domain as follows:

$$D^A_{\mathcal{I}}(f) := \{\, t : (f', t) \in S, f' = f \,\}.$$

We can compute a subset-minimal threshold support S set by starting with $S = \emptyset$, and by comparing all samples of different classes, adding a new pair to S whenever none of the existing thresholds separate the two samples. Let $D = \max_{f \in F} |D_{\mathcal{I}}(f)|$, then this algorithm runs in time $O(|E|^2 \cdot |F| \cdot D)$ where the actual running time strongly depends on the instance. Which features and thresholds are chosen for S in what order can determine the quality of the result. We choose these elements randomly.

A quick way to find a support set is given by the following observation:

Observation 2. *For a given decision tree T the set $\{\, (\mathit{feat}(v), \mathit{thresh}(v)) : v \in V(T) \setminus L(T) \,\}$ is a threshold support set.*

Although the resulting support set is usually not as useful for our purposes, as improvements often require using different thresholds, we can compute this support set much faster, making it an attractive alternative, whenever the instance is too large for the heuristic above. This concept is also used in binarization, often using an ensemble of trees rather than a single tree (McTavish et al., 2022).

In general, threshold support sets result in smaller active domains compared to bucketization. Nonetheless, it is useful to apply the simpler method, as it is usually much faster, and it speeds up and informs the second method, providing overall better threshold support sets.

We now combine all discussed components in our approach DT-SLIM, the topic of the next section.

7. DT-SLIM

In this section, we describe DT-SLIM, the overall algorithm that facilitates the SAT-based local improvement, building upon the theoretical results of Section 4, the encodings described in Section 5, and using the data reductions from the previous section.

As before, let T be a decision tree for a classification instance $\mathcal{I} = (E, c)$. Our aim is to select a subtree T' which gives rise to a local instance $\mathcal{I}' = (E', c')$ and a weighted local instance $\mathcal{I}'_w = (E', c', d)$. Since we will try to find a better decision tree T'' for $\mathcal{I}'_w$ with a SAT encoding, we need to select T' in such a way that the encoding size remains feasible.

We ensure that the local instance is small enough using a *budget* given by integer-valued parameters $\hat{d}$ and $\hat{c}$. The maximum depth of T' is limited by $\hat{d}$, thereby limiting one of the main factors for the encoding size. The other main factor, the number of samples, is limited by $\hat{c}$. Further, since the depth of T' gives us an upper bound, we search for an improved decision tree starting with $F(\mathcal{I}', d(T') - 1)$ and incrementally decrementing the depth limit. Therefore, with each successful SAT call, we improve upon T', even if we cannot find the optimal depth.

7.1 Operations

Given an internal node r' of T that is the root of subtree T', we have three different operations we can perform:

Leaf select: Whenever $d(T_{r'}) \leq \hat{d}$ and $|E_T(r')| \leq \hat{c}$, hence, the subtree's depth and local instance's number of samples are within the budget, we can simply create a local instance using all the samples in $E_T(r')$ as they are.

Leaf reduce: In case $d(T_{r'}) \leq \hat{d}$ and $|E_T(r')| > \hat{c}$, we need to reduce the number of samples in $E_T(r')$ using the reduction methods discussed in Section 6. We can use the local instance if the number of samples after the reduction does not exceed $\hat{c}$.

Mid reduce: In case both $d(T_{r'}) > \hat{d}$ and $|E_T(r')| > \hat{c}$, we need to select a subtree of $T_{r'}$ as T' in a way such that $d(T') \leq \hat{d}$. Afterwards, we introduce special leaves as discussed in Section 4, and modify $E_T(r')$ accordingly to obtain our local instance. Finally, we apply Section 6's data reductions to the local instance and thereby try to get the size of the local instance to conform to $\hat{c}$. We only select safe subtrees in this operation.

We try to apply the three operations in the above order, and whenever one is applicable, after performing the respective reductions, we do not need to try the others for the same r'.

7.2 Subtree Selection

We propose the following search strategy for local instances. We find a leaf ℓ of maximum depth and proceed on the path $P(\ell)$ from the root to ℓ. We try each internal node in turn as the subtree's root r' until we reach ℓ or we can apply leaf selection or reductions, at which point we know that no further improvements on this path are possible. We then proceed with the next leaf and never use a node in $P(\ell)$ as a possible root r' again. In general, we never choose a node for r' as the subtree's root that has already been tried unsuccessfully in a prior iteration. Once we tried all nodes, we exhausted our options. Whenever we find an improvement, we discard the path $P(\ell)$ and start again, as there might be a different leaf of maximum depth after the improvement. The reason why we proceed from top to bottom is that changes in the top affect the path that samples take later on, and the top-down order requires fewer tries.

The structure of the subtree T' for leaf selection and leaf reduction is clear; for mid reduction we propose the following strategy. We start with T' consisting only of the root r' and then incrementally grow T' as follows: in each iteration, we add all nodes v from $V(T_{r'}) \setminus V(T')$ where depth $d(T_v)$ is maximal until we either hit $\hat{d}$ or after data reductions $|E'| > \hat{c}$. Maximizing the depth has two advantages: (i) T' is usually unbalanced, which

creates more opportunities for depth reduction, and (ii) T' is always safe. We, therefore, do not have to use weights.

Data Reductions The heuristic for computing threshold support sets discussed in Section 6.3 is quadratic in the number of samples and therefore becomes prohibitively slow when the number of samples becomes too high. Whenever the runtime of the heuristic becomes infeasible, we use the thresholds used in the subtree T' as our threshold support set.

7.3 Algorithm

We can now formulate the entire algorithm, which we refer to as DT-SLIM($\mathcal{H}$), where $\mathcal{H}$ denotes the heuristic used to generate the initial decision tree T. The pseudo-code for DT-SLIM($\mathcal{H}$) is shown in Algorithm 1. It iteratively selects a leaf ℓ with maximum depth, ignoring those in the set D of completed nodes. The algorithm proceeds top-down from the root, as any improvements, affect the subtrees. In a bottom-up approach, later improvements could change already explored parts of the decision tree, which would necessitate re-exploring these parts. Hence, the top-down approach avoids the need for re-exploration.

The three operations are stated as an if-then-else construct. Leaf select in Line 8, leaf reduce in Line 12, and mid reduce in Line 17. These operations are performed for each node v on the path from the root to ℓ. Each operation constructs a local instance $\mathcal{I}'$, which is optionally reduced in size by a call to *reduce*. Whenever $\mathcal{I}'$ is within the budget, the SAT solver call is triggered by *induce_dt*. In case one operation succeeds in finding an improved subtree, the other operations are skipped. The method *construct_subtree* constructs the subtree for mid reduction as described in the previous section.

In case an improved tree T'' has been found, it is used to improve T^*. Since this changes the structure of the subtree below v, the path to the leaf ℓ is invalid. Hence, we recompute the path and start again. Further, the change in the subtree may enable further improvements. Therefore, v is not marked as done.

Depending on how the SAT encoding is used in the *induce_dt* call, the guarantees on T^* change. Independent of the SAT encoding, DT-SLIM never increases the depth, hence it always holds that $d(T^*) \leq d(T)$. The size of T^* depends on what the SAT encoding optimizes for. Whenever size is optimized, before depth, it holds that $|V(T^*)| \leq |V(T)|$. Otherwise, the size can theoretically increase to $O(2^{d(T)})$. We discuss the practical considerations in Section 8.3.

Whenever no improvements are found, the node is marked as done. The algorithm terminates when no improvements can be found for any node. Without the mid reduce operation, the algorithm terminates after $O(2^{d(T)})$ many iterations, as DT-SLIM never increases the depth. The increase in size is usually not drastic, hence, the number of iterations is usually in $O(|V(T)|)$. The mid reduce operation has to choose from a large number of possible subgraphs. Hence, each improvement can enable another improvement, making the number of iterations unpredictable. In practice, we never observed this behaviour and the number of iterations is usually in $O(|V(T)|)$.

Algorithm 1: DT-SLIM($\mathcal{H}$)

Data: An instance $\mathcal{I} = (E, c)$, a decision tree $T = (V, A)$ with root r induced using $\mathcal{H}$, a depth limit $\hat{d}$, a sample limit $\hat{c}$.
Result: A new decision tree T^* with $d(T^*) \leq d(T)$.

$D \leftarrow \emptyset$ // D keeps track of visited roots.
$T^* = T$ // T^* contains the current best decision tree.
// Run until all nodes are marked as done.
while $V(T^*) \setminus D \neq \emptyset$ **do**
  $\ell \leftarrow \arg\max_{\ell \in L(T) \setminus D} d(\ell)$ // Find deepest leaf ignoring visited roots.
  $P \leftarrow \mathtt{path}(r, \ell)$ // Path starts at the root and leads to the leaf.
  foreach $v \in P$ **do**
    if $v \notin D$ **then**
      // Perform *leaf select*.
      if $d(T^*_v) \leq \hat{d}$ ***and*** $|E_{T^*}(v)| \leq \hat{c}$ **then**
        $\mathcal{I}' \leftarrow (E_{T^*}(v), c)$
        $T'' \leftarrow \mathtt{induce\_dt}(\mathcal{I}', d(T_v) - 1)$
        $D \leftarrow D \cup P$
      // Perform *leaf reduce*.
      else if $d(T^*_v) \leq \hat{d}$ **then**
        $\mathcal{I}' \leftarrow (E_{T^*}(v), c)$
        $\mathcal{I}' \leftarrow \mathtt{reduce}(\mathcal{I}')$
        if $|E'_{T^*}(v)| \leq \hat{c}$ **then**
          $T'' \leftarrow \mathtt{induce\_dt}(\mathcal{I}', d(T_v) - 1)$
      // Perform *mid reduce*.
      else
        $T', E', c' \leftarrow \mathtt{construct\_subtree}(v)$
        $\mathcal{I}' \leftarrow (E'_{T'}(v), c')$
        $\mathcal{I}' \leftarrow \mathtt{reduce}(\mathcal{I}')$
        if $|E'_{T'}(v)| \leq \hat{c}$ **then**
          $T'' \leftarrow \mathtt{induce\_dt}(\mathcal{I}', d(T') - 1)$
      /* Whenever we change the structure of the subtree, we do not mark v as done. Further, since P becomes invalid, we find a new leaf ℓ and path P. */
      if *any* T'' *found* **then**
        $T^* = \mathtt{replace}(T^*, T', T'')$
        break
      $D \leftarrow D \cup \{v\}$

8. Implementation

So far, we have discussed our approach conceptually. In this section, we will discuss the practical aspects: which encoding we use, why we use it, and how we determine good values for the budget.

8.1 Choosing an Encoding

We tested all the encodings we discussed so far (see Section 10). Overall, the encoding by Shati et al. (2021) performed best. As expected from the asymptotics, at higher depths, when the encoding can handle only a limited number of samples, our encoding DT_pb starts to perform better. For this reason, we use a *Dynamic Encoding* in our implementation: up to depth 9, we use Shati et al.'s encoding, and for higher depths, we use DT_pb.

DT-SLIM is designed for instances that can not be tackled directly by an encoding. In general, local instances could be encoded in two ways: (i) each local instance is encoded separately and then solved by a new SAT solver instance, or (ii) we encode the full instance once and use one SAT solver instance throughout the whole DT-SLIM run. Local instances are then run using assumptions in an incremental call. Since we target instances that cannot be solved directly by an encoding, very often because the encoding would be too large, we use option (i).

8.2 Determining the Budget

With the encodings in hand, the next question is, which value to use for our depth budget $\hat{d}$ and sample budget $\hat{c}$, and at which depth should we switch encodings.

We used randomly generated subsets of increasing size from different instances (see Section 10 for details on the instances). We increased the size until the respective encoding failed.

Further, we also used a heuristically computed decision tree T and used different nodes $v \in V(T)$ to generate local instances from $E_T(v)$. Our results show that instead of an absolute sample limit, it works best to define for each $1 \leq m \leq \hat{d}$ a specific $\hat{c}_m$. With increasing m, the value of $\hat{c}_m$ decreases until it reaches 0 when $m > \hat{d}$.

The depth and number of samples provide a crude estimate for the solver's running time, as the solving time can greatly vary for the same depth and number of samples. Therefore, we picked our limits such that it is reasonable to expect that the solver will finish within a given timeout, in our case five minutes. While choosing higher values allows DT-SLIM to try more improvements, it also means that the timeout will be exceeded very often, and, therefore, a lot of time is spent on SAT-solver runs that will not find a reduction in depth or size.

We use a sample limit of 25000 samples for low depths and incrementally lower it to 250 until the maximal depth of 14.

8.3 Size Minimization

Minimizing only the depth can severely increase the decision tree's size, as the solver balances the tree and often requires more nodes to achieve lower depths. The use of special leaves aggravates this problem, as any duplication of a special leaf effectively duplicates a whole subtree. We propose several options that directly address that issue and limit the size increase in a single DT-SLIM iteration.

The core technique uses the size minimization offered by all the encodings. This gives us the choice of size as an additional optimization goal, either before or after depth. Alternatively, we can enforce that the size does not increase without minimizing it. Special leaves can

be constrained such that they can only occur once by stating that all samples whose class corresponds to the special leaf have to be classified by the same leaf. These options highlight another benefit of using a SAT-based approach: adding extra constraints that constrain the decision tree to fit application-specific requirements is easy.

These options directly tackle the size of a decision tree after a single DT-SLIM iteration. It is important to mention that this does not automatically lead to smaller decision trees after a full DT-SLIM run for several iterations. For example, duplicating special leaves may be beneficial, as it might enable more improvements in the subtrees, decreasing the overall size later on. Similar effects occur when minimizing the size first, where later improvements are made impossible. In fact, the only configuration that showed an overall benefit, compared to only minimizing the depth, was minimizing the size as a secondary objective. For this reason, we only consider the minimization of depth and depth then size in our experiments.

8.4 Unknown Values

Sometimes $e(f)$ is unknown for a feature f and a sample e, i.e., e is a partial function. In such a case, we use the most prevalent value for f among the other samples. In case this causes inconsistencies, i.e., using the most prevalent value for sample e_i causes it to be equal to another sample e_j that was previously different, but $c(e_i) \neq c(e_j)$, we ignore this sample when inducing the decision tree. Without using a replacement for the unknown value, we could not consider samples with unknown values, which can lead to very low training accuracy on instances with many unknown values.

9. Decision Tree Pruning

Decision tree pruning is well known to be an essential tool for generalization, i.e., high accuracy on unseen data. Surprisingly, it has not been considered in the literature in conjunction with SAT-based methods. We provide the first integration of SAT-based methods and state-of-the-art pruning techniques. We use established post-pruning methods, i.e., methods that are applied after inducing a classifying decision tree. We first briefly discuss these techniques and then their integration with SAT-based methods.

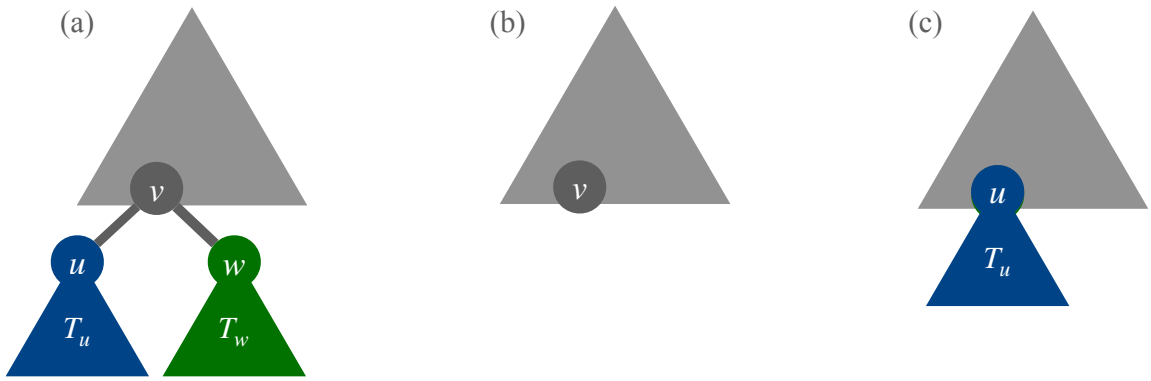

Figure 5: The illustration shows the original tree (a) and result of subtree-replacement (b) and subtree-lifting (c).

Pruning heuristically removes subtrees from a decision tree to achieve a better generalization. In other words, pruning is a method that trades an increased error rate on the training data for a lower estimated error rate on unseen data. Pruning of a decision tree T is performed relative to a specific non-leaf node $v \in V(T) \setminus L(T)$; there are two *pruning operations* illustrated in Figure 5: make v a leaf and thereby omit T_v (subtree-replacement), or, let u and w be the children of v in any order, replace v by u, thereby lifting T_u and removing T_w (subtree-lifting). We consider two different *pruning methods*. Each method defines its metric and checks the tree's nodes in turn. A pruning operation is then performed whenever it improves the metric. More specifically, we use the following methods. Let f denote the error rate of the decision tree (misclassifications divided by the number of samples) after pruning.

Cost-complexity pruning (Breiman et al., 1984). We prune a node v if the pruning decreases $R(v) = f + \alpha|L(T)|$. The parameter α controls the trade-off between error rate and the number of leaves; the higher α, the smaller the resulting tree.

C4.5's pruning (Quinlan, 1993). We prune a node v if $e_T(v) < e'_T(v)$, where $e'_T(v)$ and $e_T(v)$ are the estimates for error rates before and after pruning, respectively, defined as follows: $e_T(v) = \left(f + \frac{z^2}{2|E_T(v)|} + z \cdot \sqrt{\frac{f}{|E_T(v)|} - \frac{f^2}{|E_T(v)|} + \frac{z^2}{4|E_T(v)|^2}}\right) \cdot \left(1 + \frac{z^2}{|E_T(v)|}\right)^{-1}$ where z is the percent point function value for confidence c (Tan, Steinbach, Karpatne, & Kumar, 2019). The error estimate for the current subtree is calculated by $e'_T(v) = \left(\sum_{\ell \in L(T_v)}(|E_T(\ell)| \cdot e_T(\ell))\right) \cdot \left(|E_T(v)|\right)^{-1}$. C4.5's pruning uses another parameter m: all leaves $\ell \in L(T)$ with $|E_T(\ell)| < m$ are pruned immediately.

Cost-complexity pruning and C4.5's pruning come with hyperparameters α and (c, m) respectively. These hyperparameters have to be tuned to good values using for example cross validation. We assume that the hyperparameter settings are part of the input.

Pruning for SAT-based methods SAT-based methods offer several options for pruning. The straightforward approach is to induce a decision tree first and then prune it, using the same process as widely used heuristics like C4.5 and CART. There is also the option to encode the pruning itself. Shati et al. (2021) showed that their SAT encoding could be modified, such that not the depth but the training accuracy is minimized. Yu, Ignatiev, Stuckey, and Bodic (2021) encoded cost-complexity pruning for decision lists and sets. The two ideas could, therefore, be combined to directly apply cost-complexity pruning for SAT encodings for inducing decision trees.

Pruning for DT-SLIM The options for DT-SLIM look different. Since we run the SAT encoding many times on different parts of the decision tree and pruning decreases the training accuracy, the pruned decision tree does not necessarily correctly classify the instance. Non-classifying decision trees are generally not an issue to the SLIM approach, and DT-SLIM handles them by correctly classifying the samples that were correctly classified before, thereby not decreasing the training accuracy. Therefore, it is possible to integrate pruning directly into DT-SLIM, but it is not trivial, as there are different ways of handling non-classifying decision trees, applying pruning, handling hyperparameters, developing selection strategies, picking termination criteria, etc., which would exceed the scope of this paper. We, therefore,

keep to the established process of improving the decision tree, keeping it classifying, and then pruning it afterwards.

In the final section, we will evaluate DT-SLIM with and without pruning.

10. Experimental Results[2]

In our experiments we aim to answer several questions:

1. How well do different SAT encodings induce complexity-optimal decision trees? (Section 10.1)
2. Can DT-SLIM reduce the complexity of decision trees? (Section 10.2)
3. How important are the different DT-SLIM operations? (Section 10.3)
4. How does pruning impact the results of DT-SLIM? (Section 10.4)
5. How do DT-SLIM decision trees compare against accuracy-optimal decision trees? (Section 10.5)

Instances We collected a comprehensive set of instances used in related work (Bessiere et al., 2009; Olson, La Cava, Orzechowski, Urbanowicz, & Moore, 2017; Narodytska et al., 2018; Verwer & Zhang, 2019; Avellaneda, 2020; Schidler & Szeider, 2021), totaling 69 base instances. Eight of these instances come with a test set. We split the other 61 instances into five folds for cross validation, resulting in 313 instances in total. This section reports results for the 69 base instances, averaged over the folds. The instances provide a good variety in the number of samples, features, and classes, as Table 1 shows. Features vary from all-numerical to mixed and to all-categorical. As we will discuss below, SAT encodings were able to optimally solve 37 of the base instances, and we only consider the remaining 32 base instances for our DT-SLIM experiments.

Setup Our implementation uses Python 3.9 and PySAT 0.1.7[3] (Ignatiev, Morgado, & Marques-Silva, 2018). We use an adapted version of the Glucose 3.0[4] (Audemard & Simon, 2009) SAT solver, as it is among PySAT's solvers that performed best and has a low memory profile. We induce heuristic initial decision trees using C4.5 implemented in Weka 3.8.5[5] (Frank, Hall, Holmes, Kirkby, & Pfahringer, 2005) and CART implemented in scikit-learn 0.24.1[6] (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, VanderPlas, Passos, Cournapeau, Brucher, Perrot, & Duchesnay, 2011). Additionally, we compare DT-SLIM to the dynamic programming tool STreeD (van der Linden et al., 2023)[7]. We ran the experiments with a memory limit of 12 GB on servers with two Intel Xeon E5-2640 v4 CPUs running at 2.40 GHz and using Ubuntu 18.04. DT-SLIM and STreeD were limited to 12 hours per run.

2. Results and source code are available at `https://doi.org/10.5281/zenodo.11314227` and the current version of the source code is available at `https://github.com/ASchidler/decision_tree`.
3. `https://pysathq.github.io/`
4. `https://www.labri.fr/perso/lsimon/glucose/`
5. `https://www.cs.waikato.ac.nz/~ml/weka/`
6. `https://scikit-learn.org/`
7. `https://github.com/AlgTUDelft/pystreed`, Commit 6aed94a

Instance	$\|E\|$	$\|F\|$	$\|C\|$	Solved	Instance	$\|E\|$	$\|F\|$	$\|C\|$	Solved
anneal	798	38	5		iris	150	4	3	x
appendicitis	106	7	2	x	irish	500	5	2	x
audiology	200	70	24	x	kr-vs-kp	3196	36	2	
australian	690	14	2		letter recognition	20000	16	26	
backache	180	32	2	x	lymphography	148	18	4	x
balance-scale	625	4	3	x	magic04	19020	10	2	
banknote	1372	4	2	x	messidor	1151	19	2	
bank conv	4521	51	2		meteo	14	4	2	x
biodeg	1055	41	2		monks-1	124	6	2	x
breast-cancer-wisconsin	699	10	2	x	monks-2	169	6	2	x
cancer	683	9	2	x	monks-3	122	6	2	x
car	1728	6	2	x	mouse	70	5	2	x
ccdefault	30000	23	2		mushroom	8124	22	2	x
cleve	303	13	2	x	musk1	476	166	2	
cleveland	303	13	5	x	musk2	6598	166	2	
colic	368	22	2	x	mux6	128	6	2	x
compas-scores	11752	16	11		new-thyroid	215	5	3	x
corral	160	6	2	x	objectivity	1000	59	2	
german-credit	1000	24	2		pendigits	7494	16	10	
haberman	306	3	2	x	primary-tumor	339	17	21	x
hand posture	78095	36	5		promoters	106	58	2	x
heart-statlog	270	13	2	x	segment	2310	19	7	
heloc dataset	10459	23	2		seismic bumps	2584	18	2	
hepatitis	155	19	2	x	shuttleM	14500	9	2	x
hiv 1625	1625	8	2		soybean-large	307	35	19	x
hiv 746	746	8	2		spambase	4601	57	2	
hiv impens	947	8	2		spect	267	22	2	x
hiv schilling	3272	8	2		splice	3190	60	3	
house-votes-84	435	16	2	x	Statlog satellite	4435	36	6	
HTRU 2	17898	8	2		tic-tac-toe	958	9	2	
hungarian	294	13	2	x	vehicle	846	18	4	
hypothyroid	3163	25	2	x	wine	178	13	3	x
ida	60000	170	2		yeast	1484	8	10	
IndiansDiabetes	768	8	2		zoo	101	16	7	x
Ionosphere	351	34	2	x					

Table 1: A list of all instances considered for our experiments. *Solved* indicates that a depth-optimal decision tree for the instance could be induced by a SAT encoding.

10.1 Encodings

In our first experiment, we were interested in the performance of the encodings themselves. We did not include the approach by Janota and Morgado (2020) as the approach is not just an encoding and their results show that the approach performs worse than the encodings by Avellaneda (2020) and Shati et al. (2021).

We can see in Table 2 that the encodings alone can already solve a majority of the instances. The encoding by Shati et al. (2021) and the Dynamic Encoding performed best overall. Besides solving the most instances, they were also the fastest, solving many of the instances within the first minute. This is unsurprising, as almost all induced decision trees were below depth 10, therefore, the Dynamic Encoding was not using DT_pb most of the time. On closer look, the Dynamic Encoding solved two more slices than Shati et al. (2021)'s encoding, exactly where a decision tree of depth 10 was required.

Our encoding DT_pb performed similarly to the encoding by Avellaneda (2020). The encoding by Narodytska et al. (2018) solved the fewest instances but is the only encoding that minimizes the size independent of a depth limit.

	<1m		<5m		<10m		<1h		<3h		<6h	
Instance	S	P	S	P	S	P	S	P	S	P	S	P
Narodytska et al.	8	0	10	0	10	0	11	1	12	1	12	2
Avellaneda	18	0	19	0	20	0	27	0	29	1	30	2
Avellaneda + Size	16	0	17	0	18	0	21	1	25	2	26	2
DT_pb	15	0	20	0	20	0	26	0	29	1	30	2
DT_pb + Size	11	0	14	0	16	0	18	0	20	1	20	1
Shati et al.	23	0	26	0	27	0	31	0	35	0	36	2
Shati et al. + Size	18	0	20	0	20	1	24	1	26	2	27	2
Dynamic Encoding	23	0	27	0	27	0	32	0	35	0	36	2
Dynamic Encoding + Size	17	0	20	0	21	1	24	1	26	2	27	2

Table 2: Solved instances by encoding grouped by runtime. For each encoding, S (Solved) indicates the number of instances where all slices were solved and P (Partial) the number of instances where at least one but not all slices were solved. For each depth-minimizing encoding, the table also shows the variant, where the size is minimized as a secondary objective.

Adding size minimization, after finding the optimal depth, makes the problem considerably harder for each of the depth-minimizing encodings, severely decreasing the number of solved instances. Interestingly, with size minimization, the encodings by Avellaneda and Shati et al. perform very similarly.

Depth Limit To gauge the maximum depth any of the encodings can handle, we devised a small experiment. We incrementally created an artificial instance with $d + 1$ samples, such that any decision tree needs exactly a branch with depth d to correctly classify the instance. Hence, it is the smallest instance that requires depth d. We gave each encoding up to one hour per instance. The encoding by Avellaneda (2020) managed depth 11 in 635 seconds. Further, the encoding by Shati et al. (2021) was able to handle depth 12 in 2247 seconds. The encoding by Narodytska et al. (2018) had an advantage here, as although the depth was high, the number of nodes was low. This encoding managed 29 nodes or depth 14 in 1263 seconds. Finally, DT_pb was able to handle depth 15 in 342 seconds. This shows the unique property of our encoding to handle high depths comparatively well, as long as the number of samples is low.

10.2 DT-SLIM

In this experiment, we tested how well DT-SLIM can improve the decision trees induced by the state-of-the-art decision tree heuristics, C4.5 and CART, and the results are listed in Tables 3 and 4. We consider plain DT-SLIM, where only the depth is minimized, and DT-SLIM + Size, where in each iteration, after trying to improve the depth, the solver gets the same time for improving the size. The tables are sorted by the number of samples. In

both tables, it is noteworthy that for the largest initial decision trees, the size after running DT-SLIM is actually higher than without DT-SLIM.

Figures 6 and 7 show why this is the case, illustrated by two instances *HTRU 2* and *ida*, plotting the trajectories of depth, size, and accuracy over time. While the first figure shows one of the larger initial decision trees in our experiments, the second figure shows the decision tree for one of the largest instances in our set in terms of the number of samples. In both cases, the depth decreases monotonically, but the size goes up and down in waves. The reason for this is that DT-SLIM balances the tree, since we minimize the depth (first), and might require more nodes than before to achieve this. This also means that after the improvement, the same number of samples is spread over more branches. This, in turn, gives DT-SLIM more chances for improvement. Hence, the number of nodes goes up when the subtrees are balanced and down once DT-SLIM improves the new branches.

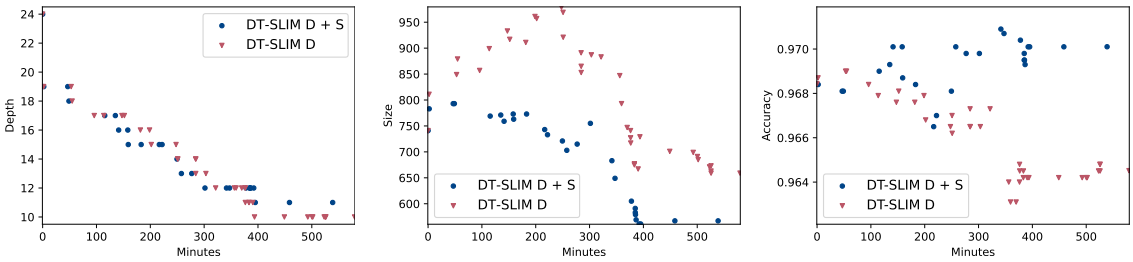

Figure 6: The DT-SLIM progression for instance *HTRU 2*.

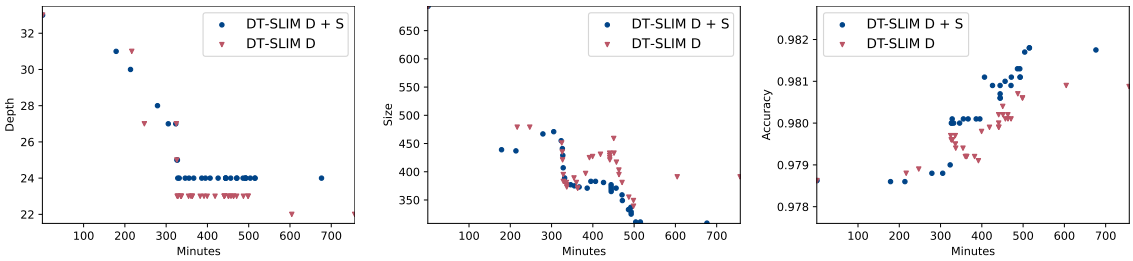

Figure 7: The DT-SLIM progression for instance *ida*.

In comparison, Figure 8 shows the progression for the largest initial decision tree. Here, DT-SLIM never got to the phase where it reduces the branches, as the run did not finish. Indeed, looking at Tables 3 and 7 we see that a large increase in the number of nodes only happens for decision trees where DT-SLIM timed out, which occurs for decision trees with more than 3000 nodes and the CART decision tree for instance *ida*, which has a very large number of features. Extending the runtime for this instances could still result in a smaller decision tree. The obvious solution, immediately trying the improve the new, larger, subtree, turns out to perform worse, as it drastically increases the overall runtime.

DT-SLIM works well for initial decision trees induced by C4.5. The depth is almost always significantly reduced, and even without minimizing the size, DT-SLIM is often able to reduce the size. With size minimization, DT-SLIM reduces the size of almost all C4.5 decision trees. In terms of accuracy, C4.5 performs a little better, although, on the instances

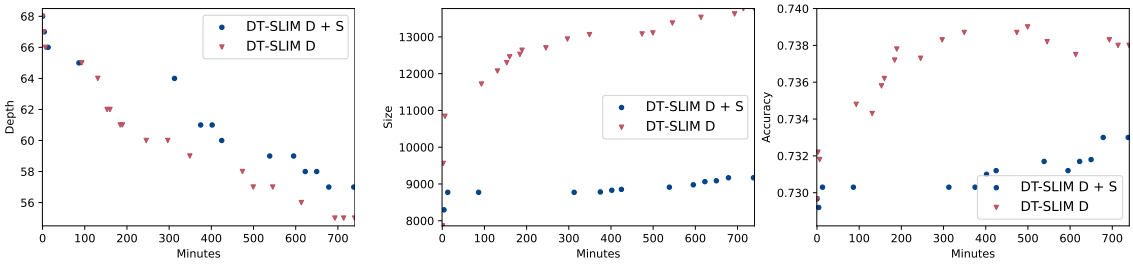

Figure 8: The DT-SLIM progression for instance *ccdefault*.

where the accuracy decreases, it does so only slightly. Overall, the accuracy stays relatively stable, while depth and size are often greatly reduced.

The results are different for CART. When comparing CART and C4.5, the decision tree sizes are similar—with CART decision trees being slightly smaller—but the depth differs significantly. CART produces more balanced decision trees, hence, we expected that DT-SLIM would not work as well here, since, as we have seen above, balancing the subtrees is an important mechanism, and CART trees do not give many opportunities for it. Indeed, while DT-SLIM still manages to improve the depth of almost all instances, the size reduction was not as successful. In terms of size, DT-SLIM can achieve better size, if it minimizes the depth and size. Particularly, again, on the largest instances, where DT-SLIM does not finish, CART achieves better results. Interestingly, although CART achieves, in general, a better accuracy on most instances, for the largest instances, the accuracy using DT-SLIM is better.

The complexity of the tree is visible to the user. Hence, although DT-SLIM does not perform better than the heuristic on all instances, the user can, in these cases, decide to either give DT-SLIM more time, or use the original decision tree instead.

10.3 Operations

Table 5 shows how often the three different DT-SLIM operations are applied per run. The results also contain the average change in depth and number of nodes. All three operations are necessary for most instances to achieve the best possible depth reduction. This highlights the importance of the instance size reductions—required for leaf reduce and mid reduce—and the importance of special leaves that enable the mid reduce operation. Particularly the instances *ccdefault*, *heloc dataset*, and *compas-scores* would allow almost no reductions without mid reduce.

All three operations consistently reduce the depth of the decision tree. While leaf select reduces the decision tree size on most instances, leaf reduce only does so on about half the instances, and mid reduce almost always increases the decision tree size. Hence, the mid reduce operation requires the other to operations to eventually decrease the decision tree size. Whenever this is possible, the overall size decreases, such as with the instances *pendigits*, *HTRU 2*, or *spambase*.

10.4 Pruning

C4.5 and CART both perform their pruning after inducing the decision tree, in a separate step. This allows us to insert DT-SLIM between the decision tree induction and the pruning.

Instance	C4.5 Size	C4.5 Depth	C4.5 Acc.	DT-SLIM(C4.5) Size	DT-SLIM(C4.5) Depth	DT-SLIM(C4.5) Acc.	DT-SLIM(C4.5) t [h]	DT-SLIM (C4.5) + Size Size	DT-SLIM (C4.5) + Size Depth	DT-SLIM (C4.5) + Size Acc.	DT-SLIM (C4.5) + Size t [h]
musk1	67.4	18.6	**0.74**	51.4	6.0	0.54	1.8	**37.4**	**5.2**	0.61	1.5
australian	139.8	16.0	**0.82**	145.8	**7.0**	0.80	1.5	**114.6**	**7.0**	0.79	2.0
hiv 746	168.6	47.0	0.73	172.6	**24.0**	0.76	4.9	**152.6**	25.6	**0.79**	5.5
IndiansDiabetes	225.4	18.6	**0.71**	227.0	7.8	0.66	2.4	**195.4**	**7.4**	0.69	3.0
vehicle	238.2	34.8	**0.71**	272.2	9.4	0.68	6.6	**188.2**	**9.0**	0.68	6.0
anneal	123.0	23.0	0.72	109.0	**7.0**	0.80	1.5	**79.0**	**7.0**	**0.90**	1.4
hiv impens	183.4	21.6	0.83	201.0	**15.6**	0.84	1.8	**176.6**	16.2	**0.85**	1.7
tic-tac-toe	125.4	11.6	0.77	139.4	**7.0**	**0.80**	0.6	**115.8**	**7.0**	0.79	0.8
german-credit	349.4	22.6	**0.68**	344.6	10.0	0.66	4.5	**259.8**	**9.6**	0.67	6.0
objectivity	201.0	18.4	0.73	203.0	**8.4**	**0.74**	4.6	**157.0**	**8.4**	0.72	5.5
biodeg	195.4	19.6	**0.79**	187.8	**7.8**	0.78	3.1	**159.4**	8.8	0.78	4.7
messidor	370.2	32.4	**0.62**	376.6	**10.6**	0.59	6.0	**311.4**	11.6	0.60	7.2
yeast	831.0	26.8	**0.47**	925.0	13.4	**0.47**	5.7	**753.4**	**13.2**	**0.47**	6.5
hiv 1625	208.2	21.8	**0.87**	220.6	**11.6**	**0.87**	0.3	**176.6**	**11.6**	**0.87**	0.4
segment	125.4	13.6	**0.96**	121.4	**7.6**	0.95	1.7	**96.6**	8.2	**0.96**	1.9
seismic bumps	360.2	24.2	0.82	339.4	**11.4**	**0.83**	3.8	**274.6**	**11.4**	0.82	4.0
splice	253.0	14.4	0.91	308.2	10.0	0.90	3.3	**237.0**	**9.8**	**0.92**	4.0
kr-vs-kp	**87.4**	15.2	**0.98**	139.4	**10.0**	0.93	1.0	106.6	**10.0**	0.95	1.0
hiv schilling	539.0	30.8	0.73	601.8	**18.0**	**0.74**	3.2	**497.8**	**18.0**	**0.74**	3.9
Statlog satellite	610.6	32.8	**0.81**	675.8	12.8	0.79	10.5	**503.4**	**11.8**	0.79	11.5
bank conv	664.2	27.8	**0.87**	640.2	16.4	0.86	11.3	**502.2**	**16.0**	**0.87**	11.4
spambase	483.8	29.2	**0.88**	455.0	21.6	0.87	5.0	**364.6**	**21.2**	0.87	4.5
musk2	**233.8**	19.8	**0.66**	**233.8**	**11.6**	0.60	8.6	248.2	12.4	0.64	8.7
heloc dataset	**3107.4**	44.8	**0.66**	5406.2	**36.2**	0.65	12.0	4105.0	36.6	**0.66**	12.0
pendigits	507.0	17.0	**0.92**	457.0	11.0	**0.92**	6.1	**407.0**	**10.0**	0.91	6.1
compas-scores	**3846.2**	58.8	0.76	5487.4	**33.6**	**0.77**	12.0	4611.8	37.0	**0.77**	12.0
HTRU 2	702.2	23.2	**0.97**	661.8	**10.6**	0.96	9.3	**563.4**	**10.6**	0.96	8.4
magic04	**3120.2**	38.0	**0.82**	6029.8	**27.8**	**0.82**	12.0	4219.0	28.4	**0.82**	12.0
letter recognition	**3751.8**	27.0	**0.87**	5088.6	**22.0**	**0.87**	12.0	4243.4	22.2	**0.87**	12.0
ccdefault	**7800.2**	67.8	0.73	12036.2	**55.4**	**0.74**	12.0	9362.6	57.6	0.73	12.0
ida	693.0	33.0	**0.98**	391.0	**22.0**	**0.98**	12.0	**309.0**	24.0	**0.98**	11.3
hand posture	**3010.6**	36.4	**0.72**	3561.4	31.0	**0.72**	12.0	3226.2	**30.4**	**0.72**	12.0

Table 3: Size, depth, test accuracy, and DT-SLIM runtime of decision trees induced by C4.5 and improved by DT-SLIM(C4.5). The decision trees are unpruned.

In this experiment, we compare the decision trees obtained from the heuristics with pruning to the decision trees obtained by running DT-SLIM before pruning. Pruning without DT-SLIM is performed by the respective heuristics implementations, and the pruning after DT-SLIM is performed by our implementations of C4.5 pruning and cost-complexity pruning.

Hyperparameters are required by both pruning methods discussed in Section 9, and we tune them as as follows: in addition to the test set, we withhold an additional slice, the validation set[8]. We induce the decision tree using the remaining slices. Afterwards, we vary the hyperparameters and pick the values that have the highest accuracy on the validation

8. For the eight instances that provided a test set, the validation set is created by splitting off 25% of the samples.

Instance	CART Size	CART Depth	CART Acc.	DT-SLIM(CART) Size	DT-SLIM(CART) Depth	DT-SLIM(CART) Acc.	DT-SLIM(CART) t [h]	DT-SLIM (CART) + Size Size	DT-SLIM (CART) + Size Depth	DT-SLIM (CART) + Size Acc.	DT-SLIM (CART) + Size t [h]
musk1	73.4	15.4	**0.70**	67.8	7.2	0.65	2.2	**58.6**	**6.8**	0.63	2.6
australian	143.4	12.0	**0.81**	147.4	**7.0**	0.79	1.2	**119.8**	**7.0**	**0.81**	1.5
hiv 746	**137.4**	14.6	**0.81**	172.2	**10.2**	0.75	0.7	151.4	**10.2**	0.79	1.0
IndiansDiabetes	217.8	15.0	**0.72**	221.4	**7.6**	0.66	1.9	**190.6**	7.8	0.66	2.2
vehicle	230.2	15.4	**0.71**	262.2	**8.4**	0.69	3.0	**209.8**	8.8	0.69	3.4
anneal	129.0	20.0	0.92	147.0	**7.0**	**0.93**	1.0	**101.0**	**7.0**	0.83	1.6
hiv impens	**170.2**	18.2	0.81	208.6	13.6	**0.82**	1.0	177.8	**13.4**	**0.82**	1.2
tic-tac-toe	**112.6**	10.6	0.79	131.8	**7.0**	**0.81**	0.6	115.0	**7.0**	0.80	0.9
german-credit	333.0	16.4	**0.68**	335.4	**8.6**	0.65	3.9	**293.4**	9.2	0.66	4.5
objectivity	207.4	19.2	**0.75**	195.8	**8.4**	0.72	4.2	**161.8**	8.8	0.73	5.8
biodeg	195.8	14.8	**0.80**	195.8	8.0	0.78	3.4	**153.8**	**7.4**	0.78	3.7
messidor	357.0	20.4	**0.62**	369.8	**10.4**	0.59	4.7	**292.2**	**10.4**	0.60	5.1
yeast	825.0	25.2	**0.50**	926.2	**11.8**	0.46	6.3	**820.6**	12.2	0.45	8.1
hiv 1625	188.2	14.0	**0.86**	228.2	**10.8**	0.84	0.2	**181.8**	11.0	**0.86**	0.4
segment	127.4	15.6	**0.96**	119.0	8.6	**0.96**	2.2	**107.0**	**8.0**	**0.96**	2.2
seismic bumps	351.4	19.8	**0.81**	336.6	9.8	**0.81**	2.9	**267.0**	**9.6**	0.80	3.3
splice	**245.0**	14.2	**0.92**	297.8	**9.8**	0.89	3.1	255.0	10.0	0.90	3.9
kr-vs-kp	**94.6**	14.6	**0.98**	163.8	**9.6**	0.94	1.2	133.8	**9.6**	0.93	1.4
hiv schilling	**502.6**	20.0	**0.77**	588.6	**16.6**	**0.77**	3.7	507.4	**16.6**	0.76	4.6
Statlog satellite	618.6	22.2	**0.80**	607.8	**11.6**	0.78	10.0	**527.0**	11.8	**0.80**	11.3
bank conv	**623.8**	26.8	**0.87**	807.4	**15.2**	0.86	12.0	658.2	16.2	0.86	12.0
spambase	465.8	32.0	**0.89**	463.0	**17.6**	0.87	7.1	**437.4**	19.2	0.87	9.2
musk2	226.6	18.4	**0.68**	225.8	**10.4**	0.67	8.6	**189.0**	11.4	0.67	9.4
heloc dataset	**2913.4**	32.4	0.61	4586.2	**23.0**	**0.62**	12.0	3560.2	23.8	**0.62**	12.0
pendigits	479.0	17.0	0.91	501.0	**10.0**	0.90	6.8	**373.0**	11.0	**0.92**	6.8
compas-scores	**3659.8**	32.4	0.76	5075.0	23.0	**0.77**	12.0	4340.6	**22.8**	**0.77**	12.0
HTRU 2	685.4	21.8	**0.97**	665.0	**9.8**	**0.97**	8.1	**578.2**	10.0	0.96	9.6
magic04	**3210.2**	34.4	**0.82**	5351.4	**24.0**	**0.82**	12.0	4025.8	25.2	**0.82**	12.0
letter recognition	**3897.0**	27.2	**0.88**	5077.8	**22.8**	0.87	12.0	4556.2	**22.8**	0.87	12.0
ccdefault	**7505.0**	42.4	0.73	10380.2	**34.0**	**0.74**	12.0	8954.6	34.4	0.73	12.0
ida	**659.0**	32.0	**0.99**	1261.0	23.0	**0.99**	12.0	1189.0	**20.0**	**0.99**	12.0
hand posture	**3226.6**	43.4	**0.70**	4041.0	**30.8**	**0.70**	12.0	3667.8	31.6	**0.70**	12.0

Table 4: Size, depth, test accuracy, and DT-SLIM runtime of decision trees induced by CART and improved by DT-SLIM(CART). The decision trees are unpruned.

set. In case of DT-SLIM, we first perform DT-SLIM and then pruning, and in this section, we give the results of the pruning method with the hyperparameter settings that performed best according to the validation set. The accuracy in the results is then again measured on the test set that was never visible to the heuristic, DT-SLIM, or pruning method during the whole process.

Table 6 shows the results for C4.5. We can see that the DT-SLIM's depth reductions are maintained even after pruning. In terms of size, the results are more mixed. Interestingly, using size minimization does not automatically achieve a smaller size after pruning, hinting that balancing the subtrees may actually help pruning. Overall, DT-SLIM improved decision trees are overall smaller and less deep than C4.5 decision trees, as C4.5 achieved the smallest

Instance	Leaf Select			Leaf Reduce			Mid Reduce		
	Count	Depth	Nodes	Count	Depth	Nodes	Count	Depth	Nodes
musk1	3.6	-0.6	-14.8	1.4	-3.4	12.6	1.0	-5.6	19.6
australian	2.6	-1.8	-28.8	0.6	-1.7	6.0	1.8	-1.9	42.9
hiv 746	3.8	-4.8	-13.4	0.4	-1.5	-24.0	1.2	-3.3	53.7
IndiansDiabetes	3.6	-1.8	-35.3	0.4	-0.5	-33.0	2.4	-1.7	59.2
vehicle	7.2	-0.4	-38.4	1.4	-0.9	-39.7	10.4	-2.0	35.2
anneal	1.0	0.0	-24.0	2.0	-2.5	0.0	2.0	-5.5	5.0
hiv impens	7.0	-0.5	-2.5	1.0	-1.6	17.2	0.6	-1.0	30.0
tic-tac-toe	2.2	-0.7	-5.1	1.0	-3.0	25.2	0.0	0.0	0.0
german-credit	7.6	-1.1	-19.8	1.8	-0.6	-9.1	2.0	-1.6	81.0
objectivity	9.2	-0.3	-22.9	2.4	-1.0	-3.5	3.2	-1.6	69.0
biodeg	4.6	-0.8	-35.1	2.0	-1.8	-21.4	2.2	-2.0	89.5
messidor	10.2	-0.8	-39.4	1.6	-1.1	-56.2	6.0	-2.0	83.0
yeast	9.0	-0.6	-17.2	1.6	-0.4	-50.5	5.0	-1.6	65.9
hiv 1625	6.6	-1.4	0.1	0.6	-1.0	18.0	0.2	-1.0	4.0
segment	3.0	-0.3	-18.7	1.4	-2.4	-18.9	1.4	-1.1	56.0
seismic bumps	7.0	-0.4	-26.1	1.8	-2.3	3.8	3.6	-1.7	43.0
splice	11.0	-0.2	-2.2	3.8	-0.5	14.2	0.4	-1.0	63.0
kr-vs-kp	1.4	-1.3	9.1	1.2	-2.8	32.7	0.0	0.0	0.0
hiv schilling	17.2	-0.1	0.4	4.6	-2.3	12.0	0.0	0.0	0.0
Statlog satellite	14.4	-0.2	-25.4	3.8	-0.7	-9.2	9.0	-1.6	51.8
bank conv	18.8	-0.2	-18.9	3.4	-0.5	-37.1	7.2	-0.9	63.4
spambase	16.0	-0.1	-8.5	4.6	-1.0	11.8	1.4	-1.4	38.0
musk2	17.4	-0.1	-7.7	5.8	-0.6	-13.2	4.0	-1.0	52.5
heloc dataset	0.0	0.0	0.0	0.0	0.0	0.0	12.4	-0.7	185.4
pendigits	13.0	-0.2	-22.0	4.0	0.0	12.5	4.0	-1.0	46.5
compas-scores	0.0	0.0	0.0	0.2	0.0	56.0	24.4	-1.0	66.8
HTRU 2	22.8	-0.2	-19.5	3.6	-0.3	-15.2	8.4	-0.8	54.8
magic04	1.0	0.0	3.2	0.4	0.0	4.0	13.2	-0.8	220.1
letter recognition	2.2	-0.4	42.9	1.0	-0.2	61.6	9.8	-0.4	120.5
ccdefault	0.0	0.0	0.0	0.0	0.0	0.0	13.0	-1.0	325.8
ida	22.0	-0.2	-10.3	5.0	0.0	11.2	5.0	-1.4	-26.4
hand posture	2.2	-0.1	-9.3	1.4	-2.0	14.3	3.8	-0.6	145.1

Table 5: DT-SLIM(C4.5) (without size reduction) operations performed per instance (*Count*), the average change in depth per operation (*Depth*), and the average change in the number of nodes (*Nodes*) per operation. Negative numbers indicate reductions.

size on 13 and the smallest depth on 3 of the 32 instances. In terms of accuracy, C4.5 decision trees do a little better. While the difference is mostly small, for instances with many unknown values, like *anneal*, the combination of DT-SLIM and pruning is less accurate. Here, DT-SLIM with size minimization achieves higher accuracy than DT-SLIM without size minimization.

Comparing CART and C4.5 after pruning, C4.5 decision trees are smaller and more accurate, while CART decision trees remain less deep. The results are shown in Table 7. Here, the size varies: sometimes CART does better without DT-SLIM, while the depth is almost always better using DT-SLIM. Overall, CART without DT-SLIM achieves the best

	C4.5			DT-SLIM(C4.5)			DT-SLIM(C4.5) + Size		
Instance	Size	Depth	Acc.	Size	Depth	Acc.	Size	Depth	Acc.
musk1	**30.2**	9.4	0.65	39.8	**5.2**	**0.69**	36.2	5.6	0.58
australian	16.6	6.2	0.84	16.2	**4.0**	0.84	**14.2**	4.6	**0.86**
hiv 746	36.2	16.0	**0.82**	**30.2**	9.0	0.75	32.6	**6.8**	0.76
IndiansDiabetes	**13.0**	**4.4**	**0.75**	16.6	**4.4**	0.73	27.0	5.0	0.73
vehicle	**61.4**	9.2	**0.71**	92.6	**7.0**	0.67	100.2	7.8	0.66
anneal	**53.0**	12.0	**0.98**	59.0	**6.0**	0.86	59.0	**6.0**	0.77
hiv impens	**44.2**	10.4	**0.85**	64.2	9.4	**0.85**	50.2	**9.2**	**0.85**
tic-tac-toe	43.4	6.0	**0.80**	**41.0**	5.2	0.76	**41.0**	**5.0**	0.72
german-credit	49.0	12.0	**0.73**	50.6	7.8	0.70	**48.2**	**5.8**	0.71
objectivity	47.0	7.6	**0.79**	**16.2**	4.2	0.76	23.0	**3.4**	0.78
biodeg	**29.8**	8.0	**0.82**	44.6	**5.0**	0.75	60.2	6.8	0.79
messidor	**31.0**	7.8	**0.64**	37.4	**5.4**	0.62	66.6	7.8	**0.64**
yeast	55.0	8.6	**0.56**	**33.4**	**6.0**	0.53	63.0	7.4	0.53
hiv 1625	53.4	12.0	**0.84**	**39.8**	**6.8**	0.83	41.0	7.8	**0.84**
segment	**69.8**	11.6	**0.96**	92.6	**7.2**	0.95	81.0	**7.2**	**0.96**
seismic bumps	**33.8**	6.6	0.91	39.4	**4.8**	**0.93**	55.8	6.2	**0.93**
splice	56.6	10.4	**0.94**	**33.0**	**6.8**	0.93	40.6	7.0	0.93
kr-vs-kp	**41.4**	10.4	**0.97**	54.2	**7.8**	0.94	52.2	8.4	0.93
hiv schilling	57.0	8.6	**0.87**	**21.0**	**4.2**	**0.87**	27.0	5.2	**0.87**
Statlog satellite	123.8	12.8	**0.82**	186.6	10.2	0.79	**115.4**	**9.8**	0.81
bank conv	**35.0**	7.8	**0.90**	76.2	**7.0**	**0.90**	45.4	7.6	**0.90**
spambase	**64.2**	12.8	**0.89**	91.4	**10.6**	**0.89**	83.0	**10.6**	**0.89**
musk2	38.2	9.2	0.81	40.6	5.0	0.77	**28.2**	**3.6**	**0.83**
heloc dataset	92.2	13.2	**0.74**	66.2	9.8	0.73	**33.0**	**6.6**	0.73
pendigits	325.0	15.0	**0.92**	317.0	**11.0**	0.91	**281.0**	**11.0**	**0.92**
compas-scores	**101.8**	**11.8**	**0.81**	445.0	16.4	**0.81**	119.0	12.6	**0.81**
HTRU 2	**20.6**	6.0	**0.98**	25.0	**5.4**	**0.98**	30.2	6.2	**0.98**
magic04	**325.8**	**15.6**	**0.85**	659.8	20.6	0.84	484.2	20.0	**0.85**
letter recognition	**2721.8**	24.8	**0.86**	4185.8	21.6	0.85	3557.8	**21.0**	0.85
ccdefault	78.2	9.8	**0.82**	**17.8**	**5.0**	**0.82**	25.0	7.0	**0.82**
ida	**179.0**	21.0	**0.99**	249.0	**16.0**	**0.99**	189.0	20.0	**0.99**
hand posture	713.0	21.8	**0.68**	788.2	19.2	0.67	**605.4**	**17.0**	0.67

Table 6: Size, depth, and test accuracy of decision trees induced by C4.5 and processed by DT-SLIM(C4.5). All decision trees have been pruned.

size on 11 and the best depth on 6 of the 32 instances. DT-SLIM(CART)-improved decision trees are comparatively accurate. The two DT-SLIM configurations found the most accurate decision tree for all but 8 instances, and on these 8 instances, the accuracy was only slightly lower than that of the CART-induced decision tree.

Pruning is important, as can be seen by comparing the result with and without pruning. Independent of DT-SLIM, the decision trees become much smaller, less deep, and most of the time, much more accurate. Particularly the largest unpruned decision tree for instance *ccdefault* with around 7000 nodes, can achieve better generalization with only 22 nodes after pruning. Therefore, we should always compare our methods with and without pruning.

Instance	CART Size	CART Depth	CART Acc.	DT-SLIM Depth Size	DT-SLIM Depth Depth	DT-SLIM Depth Acc.	DT-SLIM Depth + Size Size	DT-SLIM Depth + Size Depth	DT-SLIM Depth + Size Acc.
musk1	43.4	8.6	0.62	45.0	5.0	**0.67**	**25.8**	**4.2**	0.63
australian	35.0	5.4	0.82	10.2	3.0	0.83	**4.2**	**1.4**	**0.86**
hiv 746	41.0	7.2	0.77	55.4	6.8	0.77	**16.6**	**6.0**	**0.78**
IndiansDiabetes	53.4	9.6	**0.74**	15.0	4.0	0.70	**9.4**	**2.8**	0.73
vehicle	101.4	12.0	0.68	104.6	6.6	**0.69**	**61.4**	**6.0**	0.68
anneal	123.0	19.0	0.91	95.0	**7.0**	**0.93**	**69.0**	**7.0**	0.91
hiv impens	75.0	10.0	**0.85**	**57.8**	7.6	**0.85**	65.8	**7.4**	**0.85**
tic-tac-toe	53.4	6.6	0.76	65.4	**5.4**	0.76	**52.6**	6.4	**0.80**
german-credit	59.4	9.4	0.72	31.4	5.0	**0.73**	**15.0**	**4.0**	0.71
objectivity	43.4	8.2	**0.80**	59.4	6.2	0.78	**41.0**	**5.0**	0.79
biodeg	87.8	11.2	**0.80**	62.6	5.8	0.78	**38.2**	**5.4**	**0.80**
messidor	**34.2**	8.6	**0.65**	160.2	8.2	0.63	117.4	**6.8**	0.63
yeast	45.8	8.4	**0.55**	**31.8**	**5.8**	0.54	62.6	7.2	0.53
hiv 1625	58.2	8.4	0.81	**37.0**	**6.0**	**0.84**	44.2	6.6	0.83
segment	102.2	15.4	**0.95**	97.0	**7.8**	**0.95**	**72.6**	**7.8**	**0.95**
seismic bumps	31.4	7.2	**0.93**	**8.2**	**1.8**	**0.93**	33.8	2.6	**0.93**
splice	45.8	8.0	**0.95**	42.6	7.4	0.94	**38.2**	**7.2**	0.94
kr-vs-kp	**48.6**	10.0	**0.97**	49.8	**7.6**	0.94	54.6	8.4	0.95
hiv schilling	88.6	6.2	**0.87**	99.0	4.4	**0.87**	**47.4**	**3.6**	**0.87**
Statlog satellite	**88.2**	9.0	0.79	148.6	**8.8**	0.79	145.0	9.6	**0.81**
bank conv	**35.8**	**7.8**	**0.90**	45.8	**7.8**	0.89	58.6	**7.8**	**0.90**
spambase	173.8	17.4	0.88	104.6	**9.0**	**0.89**	**90.2**	9.6	0.88
musk2	36.2	8.0	**0.77**	39.8	**4.4**	0.72	**35.4**	5.6	0.69
heloc dataset	**50.2**	**7.8**	**0.73**	130.6	8.2	0.72	75.4	8.8	0.72
pendigits	393.0	14.0	0.90	469.0	**10.0**	0.91	**359.0**	11.0	**0.92**
compas-scores	137.5	10.5	**0.81**	248.0	11.8	0.80	**85.0**	**9.0**	**0.81**
HTRU 2	**17.4**	**3.8**	**0.98**	205.4	9.0	0.97	100.6	8.2	**0.98**
magic04	**249.4**	**13.2**	**0.85**	315.0	14.2	0.84	449.4	15.4	0.84
letter recognition	**3128.2**	28.8	**0.86**	4243.8	**21.8**	**0.86**	3637.4	23.2	**0.86**
ccdefault	22.2	5.2	**0.82**	**13.0**	**4.6**	**0.82**	15.0	**4.6**	**0.82**
ida	**67.0**	**9.0**	**0.99**	871.0	20.0	**0.99**	547.0	19.0	**0.99**
hand posture	**535.0**	**15.2**	**0.67**	1679.0	20.2	0.66	1909.4	22.2	0.65

Table 7: Size, depth, and test accuracy of decision trees induced by CART and processed by DT-SLIM(CART). All decision trees have been pruned.

DT-SLIM still reduces the complexity and increases the accuracy of many of the instances, making it a valuable addition before pruning.

10.5 Accuracy-Optimal Decision Trees

Accuracy-optimal methods aim for a similar goal as DT-SLIM: small and accurate decision trees, but try to directly compute them instead of applying pruning. Hence, we wanted to compare DT-SLIM induced decision trees—including pruned decision trees—to accuracy-optimal decision trees. We used STreeD (van der Linden et al., 2023), as it is among the fastest methods, which is important for the large instances. Indeed, STreeD is faster than

DL8.5 (Aglin et al., 2020b), GOSDT (Lin et al., 2020) and only by a factor of 1.25 slower than MurTree (Demirovic et al., 2022). Since we use much larger timeouts (van der Linden et al. (2023) used a timeout of 600 seconds), the slightly slower performance of STreeD compared to MurTree should not impact the results.

We leave the binarization to STreeD, which uses quintiles as a default. Although this ignores some feature values and can cause a small decrease in (training) accuracy, the number of binary features severely impacts STreeD's performance, and ignoring some feature values is important for scalability. We used the *cost-complexity-accuracy* mode of STreeD with a sparsity coefficient of 10^{-10}, as STreeD does not accept 0. This induces for a given depth the smallest decision tree with (almost) maximum accuracy. We ran STreeD for 12 hours on each instance, starting with a depth of 2 and increasing the depth, whenever STreeD induced an accuracy-optimal decision tree.

The maximum depth of the induced decision trees varies. For two instances (*ida* and *musk2*), STreeD could only find a decision tree of depth 3 within the time limit, as the number of features was too high. For two other instances (*ccdfault*, *hand posture*), STreeD timed out after inducing a depth 4 decision tree. STreeD found an accuracy-optimal decision tree of at least depth 5—where it was needed to correctly classify the instance—for all other instances, with the overall deepest decision tree having depth 12. While the deepest decision tree had depth 12, STreeD could handle depth limits up to 20, where the accuracy-optimal decision tree had a depth lower than the depth limit.

Table 8 shows a comparison between DT-SLIM-improved decision trees and STreeD-induced decision trees. We have several decision trees induced by both methods: for DT-SLIM we have pruned decision trees based on C4.5 and CART, both improved with and without size optimization; for STreeD we have decision trees of different depths. We use two different methods for choosing one of them for the comparison. The *virtual best* decision tree is the one with the highest testing accuracy. This is the best decision tree the method could find, if we knew the testing accuracy. Since in practice, we cannot pick the decision tree based on the testing accuracy, we also give the *validation best*: we measure the validation accuracy of each decision tree on a validation set and use the decision tree with the highest validation accuracy.

The results show that, in general, DT-SLIM achieves better testing accuracy but the decision trees are usually more complex. Particularly looking at the virtual best results, DT-SLIM finds the more accurate decision tree for almost all instances. Whenever the DT-SLIM induced decision tree has worse testing accuracy, it is at most two percentage points worse, while the STreeD induced decision tree can be to 32 percentage points worse. The underlying idea behind accuracy-optimal decision trees is the correlation between testing and training accuracy. This correlation does not necessarily hold for larger decision trees. Hence, for the instances that require large decision trees, it is not surprising that DT-SLIM performs better, as it can preserve larger decision trees when needed.

The results are more mixed when looking at the validation best results. The low complexity of the accuracy-optimal decision trees seems to introduce less variance in the testing accuracy, compared to the more complex DT-SLIM decision trees.

	Virtual Best						Validation Best					
	DT-SLIM			STreeD			DT-SLIM			STreeD		
Instance	Size	Dep.	Acc.	Size	Dep.	Acc.	Size	Dep.	Acc.	Size	Dep.	Acc.
musk1	40.6	5.0	**0.73**	**18.2**	**3.0**	0.72	34.6	4.8	**0.67**	**27.8**	**3.4**	0.59
australian	11.0	4.2	**0.87**	**10.2**	**2.4**	0.86	11.4	3.6	**0.85**	**10.2**	**2.4**	**0.85**
hiv 746	**17.8**	6.0	**0.80**	44.2	**4.4**	0.78	**24.2**	8.0	**0.76**	49.4	**4.6**	0.75
IndiansDiabetes	**18.2**	**3.0**	**0.74**	21.0	**3.0**	**0.74**	26.2	5.0	**0.73**	**8.6**	**2.2**	0.72
vehicle	103.8	7.4	**0.71**	**56.2**	**4.6**	0.69	109.4	7.6	**0.69**	**34.2**	**4.0**	0.68
anneal	95.0	7.0	**0.93**	**57.0**	**5.0**	0.87	69.0	7.0	**0.91**	**31.0**	**4.0**	0.84
hiv impens	63.0	7.0	**0.87**	**35.4**	**4.0**	0.86	30.2	7.6	**0.85**	**29.4**	**3.8**	**0.85**
tic-tac-toe	**44.6**	6.2	**0.83**	76.2	**5.8**	0.77	**45.0**	6.2	**0.75**	69.8	**5.2**	**0.75**
german-credit	42.6	6.8	**0.75**	**8.6**	**2.2**	**0.75**	32.6	5.2	0.71	**19.8**	**2.8**	**0.72**
objectivity	20.2	3.8	**0.81**	**10.2**	**2.4**	0.80	34.2	5.4	**0.79**	**15.0**	**2.8**	**0.79**
biodeg	65.8	6.6	0.82	**32.2**	**3.8**	**0.83**	55.4	5.8	0.78	**34.2**	**4.0**	**0.81**
messidor	89.4	7.2	**0.66**	**24.6**	**3.6**	**0.66**	96.6	7.2	0.64	**23.0**	**3.4**	**0.65**
yeast	58.6	7.6	**0.56**	**18.2**	**3.2**	**0.56**	**38.6**	6.8	**0.54**	40.6	**3.8**	**0.54**
hiv 1625	45.4	6.6	**0.85**	**28.6**	**3.6**	0.83	**36.6**	6.4	**0.84**	39.8	**4.4**	0.82
segment	**76.6**	7.4	**0.96**	100.6	**5.6**	**0.96**	**74.2**	7.0	**0.96**	75.8	**5.2**	0.95
seismic bumps	**15.0**	3.0	**0.93**	28.2	**2.8**	0.91	**5.8**	**1.6**	**0.93**	13.4	2.6	0.90
splice	37.0	7.2	**0.94**	**30.2**	**4.2**	0.92	42.6	7.4	**0.94**	**30.2**	**4.2**	0.92
kr-vs-kp	**38.2**	7.4	**0.95**	53.0	**5.2**	0.94	49.8	7.8	**0.94**	**43.8**	**4.6**	0.92
hiv schilling	**20.6**	5.0	**0.88**	21.0	**3.2**	**0.88**	19.8	4.4	**0.87**	**9.0**	**2.4**	**0.87**
Statlog satellite	91.8	8.6	**0.82**	**27.8**	**3.6**	**0.82**	114.6	9.4	**0.80**	**34.2**	**3.8**	**0.80**
bank conv	15.8	4.0	**0.90**	**6.2**	**2.0**	0.89	30.6	6.0	**0.90**	**6.2**	**2.0**	0.89
spambase	42.2	6.4	**0.90**	**34.2**	**3.8**	**0.90**	72.2	8.4	0.88	**37.4**	**4.0**	**0.90**
musk2	37.8	4.2	**0.85**	**11.4**	**2.6**	0.84	51.0	5.8	0.74	**13.0**	**2.8**	**0.75**
heloc dataset	16.2	6.0	**0.74**	**10.2**	**2.4**	0.73	30.6	7.2	**0.73**	**10.2**	**2.4**	**0.73**
pendigits	259.0	11.0	**0.92**	**63.0**	**5.0**	0.85	259.0	11.0	**0.92**	**63.0**	**5.0**	0.85
compas-scores	67.0	8.8	**0.81**	**40.2**	**4.2**	0.77	62.2	10.4	**0.80**	**40.2**	**4.2**	0.77
HTRU 2	9.0	3.6	**0.98**	**5.0**	**2.0**	0.97	7.8	3.2	**0.98**	**5.0**	**2.0**	0.97
magic04	311.8	15.4	**0.85**	**35.0**	**4.2**	0.84	192.2	12.8	**0.84**	**34.2**	**4.2**	**0.84**
letter recognition	3598.6	22.2	**0.86**	**63.0**	**5.0**	0.54	3616.2	22.2	**0.86**	**63.0**	**5.0**	0.54
ccdefault	**8.2**	2.6	**0.82**	**8.2**	**2.2**	**0.82**	20.6	5.6	**0.82**	**8.2**	**2.2**	**0.82**
ida	139.0	12.0	**0.99**	**13.0**	**3.0**	**0.99**	139.0	12.0	**0.99**	**5.0**	**2.0**	0.98
hand posture	760.2	18.0	**0.70**	**21.4**	**3.4**	0.57	548.6	16.0	**0.68**	**24.6**	**3.6**	0.57

Table 8: Comparison between STreeD and DT-SLIM. The left side shows the virtual best in regards to the testing accuracy and the right hand side the best according to the accuracy on the validation set. *Acc.* is always the testing accuracy. *Virtual Best* picks for each method the decision tree that has the highest testing accuracy and *Validation Best* picks for each method the decision tree that has the highest accuracy on a validation set.

11. Concluding Remarks

We introduced DT-SLIM, an anytime method that reduces the complexity of decision trees and scales to very large decision trees and classification instances. Our approach includes the novel SAT encoding DT_pb and novel data reductions.

The experimental results show how effectively DT-SLIM can reduce the size and depth of decision trees induced by a standard heuristic. For very large decision trees, the effectiveness is limited by the speed of DT-SLIM. Hence, improving the efficiency of DT-SLIM could improve the results on these instances. Unsurprisingly, the reduction in size is generally more significant if we minimize not only the depth but also the size. In terms of accuracy, DT-SLIM might slightly reduce the accuracy. Hence, in applications where a smaller or less deep decision tree is essential, DT-SLIM is very applicable.

Pruning is of utmost practical relevance, and the results look very different after applying pruning. Generally, the decision tree's size decreases drastically, independent of whether it is DT-SLIM improved or not. Here, the initial decision tree after pruning can become smaller without DT-SLIM than with DT-SLIM. In terms of accuracy, the results vary greatly, depending on the heuristic used for the initial decision tree. For CART-induced decision trees, DT-SLIM improves the depth consistently, achieves overall good size reduction, and provides comparatively very good accuracy. For C4.5-induced decision trees, DT-SLIM is also able to provide low-depth, small, and accurate decision trees. While DT-SLIM can overall improve CART decision trees very well, it is more instance-dependent for C4.5, where it can nonetheless very often improve the decision tree, not only regarding the complexity but also the accuracy.

Accuracy-optimal methods are more complimentary. While they perform very well on instances that can be captured by small decision trees, for decision trees that are large enough that DT-SLIM becomes interesting, the accuracy-optimal performance decreases.

In short, the results show that DT-SLIM is indeed able to reduce the complexity of decision trees, often drastically. Further, we have shown that when considering pruning, the benefits carry over for many instances. DT-SLIM is, therefore, a viable method for reducing decision tree complexity in a variety of use cases. Particularly if the depth of the decision tree is of importance, as DT-SLIM provides significant reductions in depth.

11.1 Future Work

We see two parts of DT-SLIM that can be improved in future work. The *running time*, as our method still requires a significant time investment and different ways of *pruning* that are more interleaved into the improvement iterations.

Running Time DT-SLIM, in the current implementation, still offers room for improvement. One avenue of improvement is parallelization, as DT-SLIM can run in parallel on independent subtrees. Particularly on large decision trees, this could significantly improve runtime. Additionally, the current implementation is written in Python, but the supporting code around the SAT solver has become relatively complex. A significant part of the runtime is spent selecting the local instances and there, particularly on instance size reduction. Therefore, re-implementing the supporting code in a native language, e.g., C++, could bring considerable improvement in efficiency.

Pruning While post-pruning works well, there are other options. As discussed in Section 9, pruning can be included in the SAT encoding. In the context of DT-SLIM, it would be possible to apply this method instead of finding classifying decision trees. While this can be applied and tuned in many ways, it could improve the overall accuracy of the decision

tree. Furthermore, the pruning results hint at the possibility that DT-SLIM might help post-pruning remove larger parts of the decision tree. The results could be improved when DT-SLIM could be tuned more towards providing a good input for the pruning method.

Acknowledgments

We acknowledge the support from the Austrian Science Fund (FWF), project 10.55776/P36420. This paper is based on and extends our AAAI paper (Schidler & Szeider, 2021), where we proposed an earlier version of our approach.

References

Aghaei, S., Gómez, A., & Vayanos, P. (2021). Strong optimal classification trees. *CoRR*, *abs/2103.15965*.

Aglin, G., Nijssen, S., & Schaus, P. (2020a). Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of AAAI 2020*, pp. 3146–3153. AAAI Press.

Aglin, G., Nijssen, S., & Schaus, P. (2020b). PyDL8.5: a library for learning optimal decision trees. In Bessiere, C. (Ed.), *Proceedings of IJCAI 2020*, pp. 5222–5224. ijcai.org.

Audemard, G., & Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. In Boutilier, C. (Ed.), *Proceedings of IJCAI 2009*, pp. 399–404.

Avellaneda, F. (2020). Efficient inference of optimal decision trees. In *Proceedings of AAAI 2020*. AAAI Press.

Bailleux, O., & Boufkhad, Y. (2003). Efficient CNF encoding of boolean cardinality constraints. In Rossi, F. (Ed.), *Conference on Principles and Practice of Constraint Programming (CP) 2003*, Vol. 2833 of *Lecture Notes in Computer Science*, pp. 108–122. Springer.

Bertsimas, D., & Dunn, J. (2017). Optimal Classification Trees. *Machine Learning*, *106*(7), 1039–1082.

Bertsimas, D., & Shioda, R. (2007). Classification and regression via integer optimization. *Oper. Res.*, *55*(2), 252–271.

Bessiere, C., Hebrard, E., & O'Sullivan, B. (2009). Minimising decision tree size as combinatorial optimisation. In *Conference on Principles and Practice of Constraint Programming (CP) 2009*, pp. 173–187, Berlin, Heidelberg. Springer Berlin Heidelberg.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Costa, V. G., & Pedreira, C. E. (2023). Recent advances in decision trees: an updated survey. *Artif. Intell. Rev.*, *56*(5), 4765–4800.

Cutler, A., & Zhao, G. (2001). PERT—perfect random tree ensembles. In *Computing Science and Statistics*, Vol. 33, pp. 490–497.

Dabrowski, K. K., Eiben, E., Ordyniak, S., Paesani, G., & Szeider, S. (2024). Learning small decision trees for data of low rank-width. In Wooldridge, M. J., Dy, J. G., & Natarajan, S. (Eds.), *Proceedings of AAAI 2024*, pp. 10476–10483. AAAI Press.

Darwiche, A., & Hirth, A. (2020). On the reasons behind decisions. In *ECAI 2020*, Vol. 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 712–720. IOS Press.

Demirovic, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., & Stuckey, P. J. (2022). Murtree: Optimal decision trees via dynamic programming and search. *J. Mach. Learn. Res.*, *23*, 26:1–26:47.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80. ACM.

Doshi-Velez, F., & Kim, B. (2017). A roadmap for a rigorous science of interpretability. *CoRR*, *abs/1702.08608*.

Eiben, E., Ordyniak, S., Paesani, G., & Szeider, S. (2023). Learning small decision trees with large domain. In Elkind, E. (Ed.), *Proceedings of IJCAI 2023*, pp. 3184–3192. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Fichte, J. K., Lodha, N., & Szeider, S. (2017). SAT-based local improvement for finding tree decompositions of small width. In *Conference on Theory and Applications of Satisfiability Testing (SAT) 2017*, Vol. 10491 of *Lecture Notes in Computer Science*, pp. 401–411. Springer Verlag.

Frank, E., Hall, M. A., Holmes, G., Kirkby, R., & Pfahringer, B. (2005). WEKA - A machine learning workbench for data mining. In Maimon, O., & Rokach, L. (Eds.), *The Data Mining and Knowledge Discovery Handbook*, pp. 1305–1314. Springer.

Gahlawat, H., & Zehavi, M. (2024). Learning small decision trees with few outliers: A parameterized perspective. In Wooldridge, M. J., Dy, J. G., & Natarajan, S. (Eds.), *Proceedings of AAAI 2024*, pp. 12100–12108. AAAI Press.

Ganian, R., Lodha, N., Ordyniak, S., & Szeider, S. (2019). SAT-encodings for treecut width and treedepth. In Kobourov, S. G., & Meyerhenke, H. (Eds.), *Proceedings of ALENEX 2019*, pp. 117–129. SIAM.

Golia, P., Roy, S., & Meel, K. S. (2020). Manthan: A data-driven approach for boolean function synthesis. In Lahiri, S. K., & Wang, C. (Eds.), *Proceedings of CAV 2020, Part II*, Vol. 12225 of *Lecture Notes in Computer Science*, pp. 611–633. Springer.

Golia, P., Slivovsky, F., Roy, S., & Meel, K. S. (2021). Engineering an efficient boolean functional synthesis engine. In *Proceedings of ICCAD 2021*, pp. 1–9. IEEE.

Goodman, B., & Flaxman, S. R. (2017). European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, *38*(3), 50–57.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., & Scheinberg, K. (2021). Optimal decision trees for categorical data via integer programming. *J. Glob. Optim.*, *81*(1), 233–260.

Heule, M., & Szeider, S. (2015). A SAT approach to clique-width. *ACM Trans. Comput. Log.*, *16*(3), 24.

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, *15*(3), 651–674.

Hu, H., Siala, M., Hebrard, E., & Huguet, M.-J. (2020). Learning optimal decision trees with maxsat and its integration in adaboost. In *Proceedings of IJCAI 2020*, pp. 1170–1176. ijcai.org. Main track.

Hu, X., Rudin, C., & Seltzer, M. (2019). Optimal sparse decision trees. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., & Garnett, R. (Eds.), *Conference on Neural Information Processing Systems (NeurIPS) 2019*, pp. 7265–7273.

Hua, K., Ren, J., & Cao, Y. (2022). A scalable deterministic global optimization algorithm for training optimal decision tree. In *Conference on Neural Information Processing Systems (NeurIPS) 2022*.

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, *5*(1), 15–17.

Ibaraki, T., Crama, Y., & Hammer, P. L. (2011). *Partially defined Boolean functions*, p. 511–563. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Ignatiev, A., Marques-Silva, J., Narodytska, N., & Stuckey, P. J. (2021). Reasoning-based learning of interpretable ML models. In *Proceedings of IJCAI 2021*, pp. 4458–4465. ijcai.org.

Ignatiev, A., Morgado, A., & Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. In *Conference on Theory and Applications of Satisfiability Testing (SAT) 2018*, pp. 428–437.

Izza, Y., Ignatiev, A., & Marques-Silva, J. (2022). On tackling explanation redundancy in decision trees. *J. Artif. Intell. Res.*, *75*, 261–321.

Janota, M., & Morgado, A. (2020). SAT-based encodings for optimal decision trees with explicit paths. In *Conference on Theory and Applications of Satisfiability Testing (SAT) 2020*, Vol. 12178 of *Lecture Notes in Computer Science*, pp. 501–518. Springer Verlag.

Kobourov, S. G., Löffler, M., Montecchiani, F., Pilipczuk, M., Rutter, I., Seidel, R., Sorge, M., & Wulms, J. (2023). The influence of dimensions on the complexity of computing decision trees. In Williams, B., Chen, Y., & Neville, J. (Eds.), *Proceedings of AAAI 2023*, pp. 8343–8350. AAAI Press.

Kuhn, M., & Johnson, K. (2013). *Classification Trees and Rule-Based Models*, pp. 369–413. Springer New York, New York, NY.

Kulikov, A. S., Pechenev, D., & Slezkin, N. (2022). SAT-based circuit local improvement. In Szeider, S., Ganian, R., & Silva, A. (Eds.), *Proceedings of MFCS 2022*, Vol. 241 of *LIPIcs*, pp. 67:1–67:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Larose, D. T. (2005). *Discovering knowledge in data*. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ. An introduction to data mining.

Lin, J., Zhong, C., Hu, D., Rudin, C., & Seltzer, M. I. (2020). Generalized and scalable optimal sparse decision trees. In *Proceedings of ICML 2020*, pp. 6150–6160. PMLR.

Lipton, Z. C. (2018). The mythos of model interpretability. *Communications of the ACM*, *61*(10), 36–43.

Lodha, N., Ordyniak, S., & Szeider, S. (2017a). A SAT approach to branchwidth. In *Proceedings of IJCAI 2017*, pp. 4894–4898.

Lodha, N., Ordyniak, S., & Szeider, S. (2017b). SAT-encodings for special treewidth and pathwidth. In *Conference on Theory and Applications of Satisfiability Testing (SAT) 2017*, Vol. 10491 of *Lecture Notes in Computer Science*, pp. 429–445. Springer Verlag.

McTavish, H., Zhong, C., Achermann, R., Karimalis, I., Chen, J., Rudin, C., & Seltzer, M. I. (2022). Fast sparse decision tree optimization via reference ensembles. In *Proceedings of AAAI 2022*, pp. 9604–9613. AAAI Press.

Monroe, D. (2018). AI, explain yourself. *AI Communications*, *61*(11), 11–13.

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, *2*(4), 345–389.

Narodytska, N., Ignatiev, A., Pereira, F., & Marques-Silva, J. (2018). Learning optimal decision trees with SAT. In *Proceedings of IJCAI 2018*, pp. 1362–1368. ijcai.org.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., & Moore, J. H. (2017). PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, *10*(1), 36.

Ordyniak, S., Paesani, G., Rychlicki, M., & Szeider, S. (2024). A general theoretical framework for learning smallest interpretable models. In Wooldridge, M. J., Dy, J. G., & Natarajan, S. (Eds.), *Proceedings of AAAI 2024*, pp. 10662–10669. AAAI Press.

Ordyniak, S., & Szeider, S. (2021). Parameterized complexity of small decision tree learning. In *Proceedings of AAAI 2021*, pp. 6454–6462. AAAI Press.

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

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pp. 399–419. Springer.

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision trees: An overview and their use in medicine. *Journal of Medical Systems*, *26*(5), 445–463.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ramaswamy, V. P., & Szeider, S. (2021a). Learning fast-inference Bayesian networks. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., & Vaughan, J. W. (Eds.), *Conference on Neural Information Processing Systems (NeurIPS) 2021, virtual*, pp. 17852–17863.

Ramaswamy, V. P., & Szeider, S. (2021b). Turbocharging treewidth-bounded Bayesian network structure learning. In *Proceedings of AAAI 2021*, pp. 3895–3903. AAAI Press.

Ramaswamy, V. P., & Szeider, S. (2022). Learning large Bayesian networks with expert constraints. In Cussens, J., & Zhang, K. (Eds.), *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, Vol. 180 of *Proceedings of Machine Learning Research*, pp. 1592–1601. PMLR.

Reichl, F., Slivovsky, F., & Szeider, S. (2023). Circuit minimization with QBF-based exact synthesis. In Williams, B., Chen, Y., & Neville, J. (Eds.), *Proceedings of AAAI 2023*, pp. 4087–4094. AAAI Press.

Reichl, F.-X., Slivovsky, F., & Szeider, S. (2024). eSLIM: Circuit minimization with SAT based local improvement. In *Conference on Theory and Applications of Satisfiability Testing (SAT) 2024, August 21-24, 2024, Pune, India*. To appear.

Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, *28*(10), 1619–1630.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, *1*(5), 206–215.

Schidler, A. (2022). SAT-based local search for plane subgraph partitions (CG challenge). In Goaoc, X., & Kerber, M. (Eds.), *Proceedings of SoCG 2022*, Vol. 224 of *LIPIcs*, pp. 74:1–74:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Schidler, A., & Szeider, S. (2021). SAT-based decision tree learning for large data sets. In *Proceedings of AAAI 2021*, pp. 3904–3912. AAAI Press.

Schidler, A., & Szeider, S. (2024). Structure-guided local improvement for maximum satisfiability. In *Conference on Principles and Practice of Constraint Programming (CP) 2024, September 2-6, 2024, Girona, Catalonia*. To appear.

Shati, P., Cohen, E., & McIlraith, S. A. (2021). SAT-based approach for learning optimal decision trees with non-binary features. In *Conference on Principles and Practice of Constraint Programming (CP) 2021*, Vol. 210 of *LIPIcs*, pp. 50:1–50:16. Schloss Dagstuhl.

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). *Introduction to Data Mining, 2nd Edition*. Pearson.

van der Linden, J. G. M., de Weerdt, M., & Demirovic, E. (2023). Necessary and sufficient conditions for optimal decision trees using dynamic programming. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., & Levine, S. (Eds.), *Conference on Neural Information Processing Systems (NeurIPS) 2023*.

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C., & Schaus, P. (2020). Learning optimal decision trees using constraint programming. *Constraints An Int. J.*, *25*(3-4), 226–250.

Verwer, S., & Zhang, Y. (2017). Learning decision trees with flexible constraints and objectives using integer optimization. In Salvagnin, D., & Lombardi, M. (Eds.), *Proceedings of Integration of AI and OR Techniques in Constraint Programming (CPAIOR) 2017*, Vol. 10335 of *Lecture Notes in Computer Science*, pp. 94–103. Springer.

Verwer, S., & Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. In *Proceedings of AAAI 2019*, pp. 1625–1632. AAAI Press.

Yu, J., Ignatiev, A., Stuckey, P. J., & Bodic, P. L. (2021). Learning optimal decision sets and lists with SAT. *J. Artif. Intell. Res.*, *72*, 1251–1279.

Zhu, H., Murali, P., Phan, D. T., Nguyen, L. M., & Kalagnanam, J. (2020). A scalable mip-based method for learning optimal multivariate decision trees. In Larochelle,

H., Ranzato, M., Hadsell, R., Balcan, M., & Lin, H. (Eds.), *Conference on Neural Information Processing Systems (NeurIPS) 2020, December 6-12, 2020, virtual*.