Fixed-Parameter Tractability of Learning Small Decision Trees (full paper)

Anonymous author
Anonymous affiliation

Abstract

We consider the NP-hard problem of finding a smallest decision tree which represents a given partially defined Boolean formula. We establish fixed-parameter tractability of the problem with respect to the NLC-width of the instance. We formulate a dynamic programming procedure which utilizes the NLC-decomposition of the instance. For this to work, we establish a succinct representation of partial solutions, so that the space and time requirements of each dynamic programming step remain bounded in terms of the NLC-width.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms → Parameterized complexity and exact algorithms → Fixed parameter tractability

Keywords and phrases parameterized complexity, NLC-width, rank-width, decision trees, partially defined Boolean formulas

1 Introduction

Decision trees have proved to be extremely useful tools for the describing, classifying, generalizing data [18, 22, 25]. In this paper, we consider decision trees for *classification instances (CIs)*, consisting of a finite set E of *examples* (also called *feature vectors*) over a finite set F of *features*. Each example $e \in E$ is a function $e : F \to \{0, 1\}$ which determines whether the feature f is true or false for e. Moreover, E is given as a partition $E^+ \uplus E^-$ into positive and negative examples. For instance, examples could represent medical patients and features diagnostic tests; a patient is positive or negative corresponding to whether they have been diagnosed with a certain disease or not. CIs are also called *partially* or *incompletely defined Boolean functions*, as we can consider the features as Boolean variables, and examples as truth assignments that evaluate to 0 (for positive examples) or 1 (for negative examples). CIs have been studied as a key concept for the logical analysis of data and in switching theory [4, 6, 5, 7, 8, 17, 20].

Because of their simplicity, decision trees are particularly attractive for providing interpretable models of the underlying CI, an aspect whose importance has been strongly emphasized over the recent years [10, 12, 15, 19, 21]. In this context, one prefers *small trees*, as they are easier to interpret and require fewer tests to make a classification. Small trees are also preferred in view of the parsimony principle (Occam's Razor) since small trees are expected to generalize better to new data [2]. However, finding a small decision tree, as formulated in the following decision problem, is NP-complete [16].

Minimum Decision Tree Size (DTS): given a CI $E = E^+ \uplus E^-$ and an integer s, is there a decision tree with at most s nodes for E?

Given this complexity barrier, we propose a fixed-parameter algorithm for the problem, which exploits the input CI's hidden structure. The *incidence graph* of a CI is the bipartite graph $G_I(E)$ whose vertices are the examples on one side and the features on the other, where an example e is adjacent with a feature f if and only if $e(f) = 1$. Figure 1 shows a CI and a smallest decision tree for it, as well as the incidence graph.

Key to our algorithm are new notions for succinctly representing decision trees that correspond to subtrees of the incidence graph's tree decomposition. Based on that, we can carry out a dynamic programming (DP) procedure along the tree decomposition.

While the DP approach using treewidth is quite well understood and can often be quite easily designed for problems on graphs (or more generally problems whose solutions can be represented in terms of the graph for which the tree decomposition is given), the same DP approach can become rather involved if applied to problems whose solutions have no or only minor resemblence to the graph for which one is given a tree decomposition. Probably the most prominent example for this is the celebrated result by Bodlaender [3], where he uses a

E	f_1	f_2	f_3	f_4
$e_1 \in E^-$	0	0	1	0
$e_2 \in E^-$	0	0	1	1
$e_3 \in E^-$	0	1	1	0
$e_4 \in E^-$	1	1	0	0
$e_5 \in E^+$	1	0	0	1
$e_6 \in E^+$	1	0	1	1

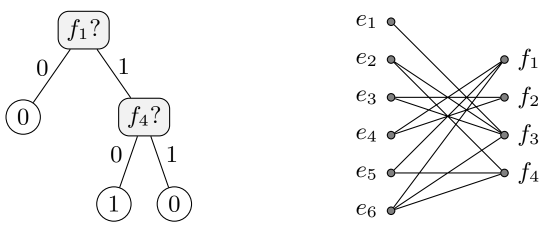

Figure 1 A CI $E = E^+ \uplus E^-$ with six examples and four features (left), a decision tree with 5 nodes that classifies E (middle), the incidence graph $G_I(E)$ (right).

DP approach on an approximate tree decomposition to compute the exact treewidth of a graph; here, the solutions are tree decompositions, which are complex structures that cannot easily be represented in terms of the graph. Other prominent examples include a DP approach to compute the exact treedepth [26] or clique-width [14] using an optimal tree decomposition. We face a similar problem, since solutions in our case are decision trees that do not bear any resemblence to the incidence graph for which we are given the tree decomposition. The main obstacle to overcome, therefore, is the design of the DP-records for our DP algorithm. That is, a record for a node b in a tree decomposition for the incidence graph of E needs to provide a compact representation of partial solutions, i.e. partial solutions in the sense that they represent the part of the solution for the whole instance E that corresponds to the sub-instance induced by all features and examples contained in the bags in the subtree of the tree decomposition rooted at the current node b. We overcome this obstacle in Section 3, where we also provide intuitive descriptions and motivation for the definition of the records (Subsection 3.1).

2 Preliminaries

2.1 Parameterized Complexity

We give some basic definitions of Parameterized Complexity and refer for a more in-depth treatment to other sources [9, 13]. Parameterized complexity considers problems in a two-dimensional setting, where a problem instance is a pair (I, k), where I is the main part and k is the parameter. A parameterized problem is *fixed-parameter tractable* if there exists a computable function f such that instances (I, k) can be solved in time $f(k)\|I\|^{O(1)}$.

2.2 Graphs and NLC-width

We will assume that the reader is familiar with basic graph theory (see, e.g. [11, 1]). We consider (vertex and edge labelled) undirected graphs. Let $G = (V, E)$ be an undirected graph. We write $V(G) = V$ and $E(G) = E$ for the sets of vertices and edges of G, respectively. We denote an edge between $u \in V$ and $v \in V$ as $\{u, v\}$. For a set $V' \subseteq V$ of vertices we let $G[V']$ denote the graph induced by the vertices in V', i.e. $G[V']$ has vertex set V' and edge set $E \cap \{\, \{u, v\} \mid u, v \in V' \,\}$ and we let $G - V'$ denote the graph $G[V \setminus V']$. For a set $E' \subseteq E$ of edges we let denote $G - E'$ the graph with vertex set V and edge set $E \setminus E'$.

A *k-graph* is a pair (G, λ), where $G = (V, E)$ is an undirected graph and $\lambda : V \to [k]$ is a *vertex label mapping* that labels every vertex $v \in V$ with a label $\lambda(v)$ from $[k]$. We call the k-graph consisting of exactly one vertex v (say, labeled by i) an *initial k-graph* and denote it by $i(v)$.

Node label control-width (*NLC-width*) is a graph parameter, defined as follows [28]: Let $k \in \mathbb{N}$ be a positive integer. An *k-NLC-expression tree* of a graph $G = (V, E)$ is a subcubic tree B, where every node b of B is associated with a k-graph (denoted by (G_b, λ_b)), such that:

1. Every leaf represents an initial k-graph $i(v)$ with $i \in [k]$ and $v \in V$.
2. Every non-leaf node b with one child c is a *relabelling node* and is associated with a relabelling function $R_b : [k] \to [k]$. Moreover, G_b is obtained from G_c after relabelling all vertices of G_c with label i to label $R_b(i)$ for every $i \in [k]$.
3. Every non-leaf node b with two children, i.e., a left child l and a right child r, is a *join node* and is associated with a *join matrix*, i.e., a binary $k \times k$ matrix M_b. Moreover,

(G_b, λ_b) is obtained from the disjoint union of (G_l, λ_l) and (G_r, λ_r) after adding an edge from all vertices labeled i in G_l to all vertices labeled j in G_r whenever $M_b[i, j] = 1$.

4. G is equal to the G_r for the root node r of B.

The NLC-width of a graph G, denoted by $nlcw(G)$, is the minimum k for which G has a k-NLC-expression tree. A k-NLC-expression tree is *nice* if every relabelling node has a relabelling function $R : [k] \to [k]$ such that for some $i, j \in [k]$, $R(i) = j$ and $R(\ell) = \ell$ for all $\ell \in [k] \setminus \{i\}$. Clearly, given a k-NLC-expression tree, a nice k-NLC-expression tree can be found in polynomial time; simply replace every relabelling node (that relabels more than one label at a time) by a sequence of relabelling nodes.

Let b be a node in a k-NLC-expression tree of a graph G. We denote by V_b the set of vertices of G_b. By the definition of a k-NLC-expression tree, if $u, v \in V_b$ have the same label in (G_b, λ_b) and $w \in V(G) \setminus V_b$, then u is adjacent to w in G if and only if v is.

Computing the NLC-width of a graph is NP-hard [?]. However, it is sufficient to use the algorithm of Seymour and Oum [?], which returns a c-expression for some $c \leq 2^{3cw(G)+2} - 1$ in $O(n^9 \log n)$ time, or the later improvements of Oum [24] and Hliněný and Oum [?] that provide cubic-time algorithms which yield a c-expression for some $c \leq 8^{cw(G)} - 1$ and $c \leq 2^{cw(G)+1} - 1$, respectively.

should it be *nlcw*, or should we define *cw* and say it's approximation?

2.3 Classification Problems

An *example* e is a function $e : feat(e) \to \{0, 1\}$ defined on a finite set $feat(e)$ of *features*. For a set E of examples, we put $feat(E) = \bigcup_{e \in E} feat(e)$. We say that two examples e_1, e_2 *agree* on a feature f if $f \in feat(e_1)$, $f \in feat(e_2)$ and $e_1(f) = e_2(f)$. If $f \in feat(e_1)$, $f \in feat(e_2)$ but $e_1(f) \neq e_2(f)$, we say that the examples *disagree on* f.

A *classification instance (CI)* (also called a *partially defined Boolean function* [17]) $E = E^+ \uplus E^-$ is the disjoint union of two sets of examples, where for all $e_1, e_2 \in E$ we have $feat(e_1) = feat(e_2)$. The examples in E^+ are said to be *positive*; the examples in E^- are said to be *negative*. A set X of examples is *uniform* if $X \subseteq E^+$ or $X \subseteq E^-$; otherwise X is *non-uniform*.

Given a CI E, a subset $F \subseteq feat(E)$ is a *support set* of E if any two examples $e_1 \in E^+$ and $e_2 \in E^-$ disagree in at least one feature of F. Finding a smallest support set, denoted by $\text{MSS}(E)$, for a classification instance E is an NP-hard task [17, Theorem 12.2].

We define the *incidence graph* of E, denoted by $G_I(E)$, as the bipartite graph with partition $(E, feat(E))$ having an edge between an example $e \in E$ and a feature $f \in feat(e)$ if $f(e) = 1$.

2.4 Decision Trees

A *decision tree* (DT) (or *classification tree*) is a rooted tree T with vertex set $V(T)$ and arc set $A(T)$, where each non-leaf node (called a *test*) $v \in V(T)$ is labelled with a feature $feat(v)$, each non-leaf node v has exactly two out-going arcs, a *left arc* and a *right arc*, and each leaf is either a *positive* or a *negative* leaf. We write $feat(T) = \{ v \in V(T) \mid feat(v) \}$.

Consider a CI E and a decision tree T with $feat(T) \subseteq feat(E)$. For each node v of T we define $E_T(v)$ as the set of all examples $e \in E$ such that for each left (right, respectively) arc (u, v) on the unique path from the root of T to v we have $e(feat(v)) = 0$ $(e(feat(v)) = 1$, respectively). T *correctly classifies* an example $e \in E$ if e is a positive (negative) example and $e \in E_T(v)$ for a positive (negative) leaf. We say that T *classifies* E (or simply that T is a DT for E) if T correctly classifies every example $e \in E$. See Figure 1 for an illustration of a CI, its incidence graph, and a DT that classifies E.

The size of T is its number of nodes, i.e. $|V(T)|$. We consider the following problem.

Minimum Decision Tree Size (DTS)	
Input:	A classification instance E and an integer s.
Question:	Is there a decision tree of size at most s for E?

We now give some simple auxiliary lemmas that are required by our algorithm.

▶ **Lemma 1.** *Let A be a set of features of size a. Then the number of DTs of size at most s that use only features in A is at most a^{2s+1} and those can be enumerated in $\mathcal{O}(a^{2s+1})$ time.*

Proof. We start by counting the number of trees T with n nodes that can potentially underlie a DT with n nodes. Note that there is one-to-one correspondence between trees T that underlie a DT with n nodes and unlabelled rooted ordered binary trees with n nodes (where ordered refers to an ordering of the at most 2 child nodes). Since it is known that the number of unlabelled rooted ordered binary trees with n nodes is equal to the n-th Catalan number C_n and that those trees can be enumerated in $\mathcal{O}(C_n)$ time [27], we already obtain that we can enumerate all of the at most C_n possible trees T underlying a DT of size n in $\mathcal{O}(C_n)$ time. Therefore, there are at most sC_s possible trees of size at most s that can underlie a DT with at most s nodes and those can be enumerated in $\mathcal{O}(sC_s)$ time. It now remains to bound the number of possible feature assignments $\mathit{feat}(f)$ for these trees as well as the number of possibilities for the leave nodes that can be either labelled positive or negative. Since we can assume that $a \geq 2$, we obtain that the number of possible feature assignments (and labellings of leaf-nodes) of a tree T with n nodes is at most a^n. Taking everything together, we obtain that there are at most $sC_sa^s \leq s4^sa^s \leq a^{2s+1}$ many DTs of size at most s using only features in A and those can be enumerated in $\mathcal{O}(a^{2s+1})$ time. ◀

▶ **Lemma 2.** *Let A be a set of features of size a. There are at most $a^{2^{a+1}+3}$ inclusion-wise minimal DTs using only features in A and these can be enumerated in $\mathcal{O}(a^{2^{a+1}+3})$ time.*

Proof. Note that an inclusion-wise minimal DT T that uses only features in A has at most 2^a+1 nodes; this is because every feature appears at most once on every path T. Therefore, we obtain from Lemma 1 that the number of choices for T is at most $a^{2(2^a+1)+1} = a^{2^{a+1}+3}$. ◀

▶ **Lemma 3.** *Let E be a CI. Then one can decide whether E has a DT and if so output a DT of minimum size for E in time $\mathcal{O}((2^{|E|})^{4|E|-1})$.*

Proof. Note first that $|\mathit{feat}(E)| \leq 2^{|E|}$ since we can assume that E does not contain two equivalent features. Moreover, E has a DT if and only if $\mathit{feat}(E)$ is a support set, which can be checked in time $\mathcal{O}(|E|^2|\mathit{feat}(E)|)$ by checking, for every pair of positive and negative examples in E, whether there is a feature that distinguishes them. If this is not the case, we output **NO**, so assume that E has a DT. Note that any inclusion-wise minimal DT for E has at most $|E|$ leaves and therefore size at most $2|E|-1$. We can therefore employ Lemma 1 to enumerate all inclusion-wise minimal potential DTs for E in time $\mathcal{O}((2^{|E|})^{2(2|E|-1)+1}) \in \mathcal{O}((2^{|E|})^{4|E|-1})$. For every such tree we then check whether it is indeed a DT for E and return a DT for E of minimum size found during this process. ◀

3 An FPT-Algorithm for NLC-width

In this section, we present our main result, i.e. we will show that DTS is fixed-parameter tractable parameterized by NLC-width.

▶ **Theorem 4.** *Let E be a CI, let B be an NLC-decomposition of width ω for $G_I(E)$, and let s be an integer. Then, deciding whether E has a DT of size at most s is fixed-parameter tractable parameterized by ω.*

todo: Due to proposition ...

▶ **Corollary 5.** DTS *is fixed-parameter tractable parameterized by NLC-width.*

In principle, we will use a dynamic programming algorithm along the NLC-decomposition (B, χ) of $G_I(E)$ that computes a set of records for every node b of B in a bottom-up manner. Each record will represent an equivalence class of solutions (DTs) for the whole instance restricted to the examples and features contained in the current subtree rooted in b, i.e. the examples and features contained in $\chi(b)$. Before we continue with the formal notions and definitions required to define the records, we want to illustrate the main ideas and motivations. In what follows let B be an NLC-decomposition of $G_I(E)$ of width k. For $b \in V(B)$, we write $\mathit{feat}(b)$ and $\mathit{exam}(b)$ for the sets $\chi(b) \cap \mathit{feat}(E)$ and $\chi(b) \cap E$, respectively.

3.1 Description of the Main Ideas Behind the Algorithm

todo: adjust to NLC-width

Consider a node b of B. To simplify the presentation, we will sometime refer to the features and examples in $\chi(B_b) \setminus \chi(b)$ as *forgotten* features and examples and we refer to the features and examples in $(\mathit{feat}(E) \cup E) \setminus \chi(B_b)$ as *future* features and examples. We start with some simple observations that follow immediately from the properties of tree decompositions.

▶ **Observation 6.** *(1) $e(f) = 0$ for every forgotten example $e \in \mathit{exam}(B_b) \setminus \mathit{exam}(b)$ and future feature $f \in \mathit{feat}(E) \setminus \mathit{feat}(B_b)$,*
(2) $e(f) = 0$ for every future example $e \in E \setminus \mathit{exam}(B_b)$ and forgotten feature $f \in \mathit{feat}(B_b) \setminus \mathit{feat}(b)$;

Proof. Towards showing (1), let e be an example in $\mathit{exam}(B_b) \setminus \mathit{exam}(b)$ and let f be a feature in $\mathit{feat}(E) \setminus \mathit{feat}(B_b)$. We claim that because (T, χ) is a tree decomposition of $G_I(E)$, the graph $G_I(E)$ cannot contain an edge between e and f, which implies that $e(f) = 0$. Suppose for a contradiction that this is not the case, i.e. $\{e, f\} \in E(G_I(E))$. Then, because of property (T1) of a tree decomposition, there must exist a node b' such that $e, f \in \chi(b')$. But then, if $b' \in V(B_b)$ we obtain that $f \notin \chi(b')$. Similarly, if $b' \in V(B \setminus B_b)$, we obtain that $e \notin \chi(b')$ since otherwise e would violate property (T2) of a tree decomposition. This completes the proof for (1); the proof for (2) is analogous. ◀

Informally, Observation 6 shows that forgotten examples cannot be distinguished by future features and future examples cannot be distinguished by forgotten features. Consider a DT T for E and a node b of B. For a set W containing features and examples from E, we denote by $E[W]$ the sub-instance of E induced by the features and examples in W. Our aim is to obtain a compact representation (represented by records) of the partial solution for the sub-instance $E[\chi(B_b)]$ of E induced by the features and examples in $\chi(B_b)$ represented by T.

Intuitively, such a compact representation has to (1) represent a partial solution (DT) for the examples in $\mathit{exam}(B_b)$ and (2) retain sufficient information about the structure of T in order to decide whether it can be extended to a DT that also classifies the examples in $E \setminus \mathit{exam}(B_b)$.

For illustration purposes let us first consider the simplified case that $exam(b) = \emptyset$. Because of Observation 6 (1), this implies that every forgotten example goes to the left child of any node t in T that is assigned a future feature. Therefore, under the assumption that $exam(b) = \emptyset$ the DT T' obtained from T after:

- removing the subtree T_r of T for every right child r of a node t of T with $feat(t) \in feat(E) \setminus feat(B_b)$ and replacing t with an edge from its parent in T to its left child in T

is a DT for $E[\chi(B_b)]$. Note that this means that under the rather strong assumption that $exam(b) = \emptyset$, the part of T that takes care of the sub-instance $E[\chi(B_b)]$ is itself a DT using only features in $feat(B_b)$; we will see later that unfortunately this is no longer the case if $exam(b) \neq \emptyset$. Note that even though T' is a DT for $E[B_b]$, it does not yet constitute a compact representation, since the number of features it uses in $feat(B_b) \setminus feat(b)$ is potentially unbounded. However, we obtain from Observation 6 (2) that every future example will end up in the left child of every node t of T' that is assigned a forgotten feature. This means that to decide whether T' can be extended to a DT for the whole instance, the nodes that are assigned forgotten features are not important. In fact, the only nodes in T' that can be important for the classification of future examples are the nodes that are assigned features in $feat(b)$. That is, it is sufficient to remember the DT T'' obtained from T' after:

- removing the subtree T_r of T' for every right child r of a node t of T' with $feat(t) \in feat(B_b) \setminus feat(b)$ and replacing t with an edge from its parent in T' to its left child in T'.

Since the number of possible DT T'' is clearly bounded in terms of the number of features in $feat(b)$ (and therefore in terms of the treewidth of $G_I(E)$), this would already give us the compact representation that we are looking for. However, this only works in the case that $exam(b) = \emptyset$, which is clearly not the case in general.

So let us now consider the general case with $exam(b) \neq \emptyset$. The first difference now is that the part of T that takes care of the sub-instance $E[\chi(B_b)]$ is no longer a DT that only uses features in $feat(B_b)$. In fact, it could even be the case that $E[\chi(B_b)]$ does not have a DT, because there could exist examples in $exam(b)$ that can only be distinguished using the features in $feat(E) \setminus feat(B_b)$. This means that we have to allow our partial solution for $E[\chi(B_b)]$ to use future features. Fortunately, we do not need to know which exact future feature is used by our partial solution but it suffices to know that a future feature is used and how it behaves w.r.t. the examples in $exam(b)$; this is because Observation 6 (1) implies that a future feature is used in a partial solution only for the purpose of distinguishing examples in $exam(b)$. Moreover, because every forgotten example ends up in the left child of any node t of T that uses a future feature, we only need to remember the left child for those nodes. Also, we only need to remember occurrences of those nodes (using future features) if at least one example in $exam(b)$ ends up to in the right child of such a node; otherwise the node has no influence on the classification of examples in $exam(B_b)$. Finally, we cannot simply forget nodes that use forgotten features (as we could in the case that $exam(b) = 0$). This is because we need to know exactly where the examples in $exam(b)$ end up at. For instance, if such an example in $exam(b)$ ends up in the right child of a node using a future feature, we need to know that this is the case because this means that the example has to be classified in this place at a later stage of the algorithm. Nevertheless, we do not need to remember all occurrences of nodes using forgotten features, but only those for which there is at least one example in $exam(b)$ that ends up in the right child of the node. Similarly, we do not need to remember the exact forgotten feature that is used but only how it behaves towards the examples in $exam(b)$. In summary, we only need to remember the full information about the nodes of T that use a feature in *feat*(b). For all other nodes, i.e. nodes that use either forgotten or future features, we only need to remember such a node, if at least one example in *exam*(b) ends up in its right child. Moreover, even if this is the case, we only need to remember the following for such nodes:

- whether it uses a future or a forgotten feature and
- how it behaves w.r.t. the examples in *exam*(b).

With these ideas in mind, we are now ready to provide a formal definition of the compact representation of the part of T that takes care of the sub-instance $E[\chi(B_b)]$.

3.2 Formal Definition of Records and Preliminary Results

In the following, let E be a CI and let B be a k-NLC-expression tree for $G_I(E)$. Consider a node b of B. Recall that b is either a leaf node associated with a k-graph $i(v)$, a relabelling node with one child and with relabelling function R_b, or a join node with a left child, a right child and a join matrix M_b. Moreover, recall that (G_b, λ_b) is the k-graph associated with b (whose unlabelled version is a subgraph of G) and V_b is the set of vertices of G_b. Additionally, we will use the following notation. We denote by *feat*(b) the set $V_b \cap \mathit{feat}(E)$ of features in V_b and by *exam*(b) the set $V_b \cap E$ of examples in V_b.

Consider a node b of B. Let L be a set of labels (usually $L = [k]$). For a subset $L' \subseteq L$, we denote by $\overline{L'}$ the set $L \setminus L'$. For a label $\ell \in L$, we introduce a new feature f_ℓ, which we will call a *forgotten feature*. Moreover, for a subset $L' \subseteq L$ of labels, we introduce a new feature $f_{L'}$, which we call an *future (or introduce) feature*. Let $F_L = \{\, f_\ell \mid \ell \in L \,\}$ be the set of all forgotten features and let $I_L = \{\, f_{L'} \mid L' \subseteq L \,\}$ be the set of all future features w.r.t. L. To distinguish features in *feat*(E) from forgotten and future features, we will sometimes refer to them as *real features*.

Let T be a DT and $t \in V(T)$. We say that a node t_A is a *left (right) ancestor* of t if t is contained in the subtree of T rooted at the left (right) child of t_A. We denote by $\mathrm{anc}^L_T(t)$ ($\mathrm{anc}^R_T(t)$), or simply $\mathrm{anc}^L(t)$ ($\mathrm{anc}^R(t)$) if T is clear from the context, the set of all left (right) ancestors of t in T. We denote by $\mathrm{anc}(t)$ the set of all *ancestors* of t in T, i.e., $\mathrm{anc}(t) = \mathrm{anc}^L(t) \cup \mathrm{anc}^R(t)$.

Let T be a DT and $t \in V(T)$ be an inner node of T with left child ℓ, right child r, and parent p. We say that T' is obtained from T after *left (right) contracting* t if T' is the DT obtained from T after removing t together with all nodes in T_r/T_ℓ and adding the edge between p and ℓ/r; if t has no parent then no edge is added.

We say that T is a *DT for* b, if T is a DT for *exam*(b) that uses only the features in *feat*(b). We say that an inner node $t \in V(T)$ is *left (right) redundant* in T if $\mathit{feat}(t) \in \mathit{feat}(\mathrm{anc}^L(t))$ ($\mathit{feat}(t) \in \mathit{feat}(\mathrm{anc}^R(t))$). We say that t is redundant if it is either left redundant or right redundant. Intuitively, a node t is left (right) redundant if all examples that end up at t, i.e., the examples in $E_T(t)$, go to the left (right) child of t in T. Therefore, if t is left (right) redundant in T, then the tree obtained after left (right) contracting t is still a DT.

We say that T is a *DT template* for b if T is a DT for *exam*(b) that can additionally use the future features in $I_{[k]}$. Here, we assume that a future feature $f_{L'} \in I_{[k]}$ for some $L' \subseteq [k]$ is 1 at an example $e \in \mathit{exam}(b)$ if $\lambda_b(e) \in L'$ and otherwise it is 0. We say that a DT template is *complete* if it does not use any features in $I_{[k]}$, otherwise we say that it is *incomplete*. Informally, the role of the future features in a DT template is to provide spaceholders for the features in $\mathit{feat}(E) \setminus \mathit{feat}(b)$. Because all of those features behave the same w.r.t. examples in *exam*(b) having the same label, they can be characterised by the set of labels for which those features are 1. Let T be a DT template for b and let $t \in V(T)$. We denote by $A_T(t)$ (or short $A(t)$ if T is clear from the context) the set of *filtered labels* for t, i.e., $A(t) = (\bigcap_{f_{L'} \in feat(\text{anc}_L(t)) \cap I_{[k]}} \overline{L'}) \cap (\bigcap_{f_{L'} \in feat(\text{anc}_R(t)) \cap I_{[k]}} L')$. Informally, $A(t)$ is the set of all labels $\ell \in [k]$ such that an example e with label ℓ would end up at t, if only the effect of the future features on the path to t is considered. We say that t with $f_{L'} = feat(t) \in I_{[k]}$ is *left (right) redundant* in T if $A(t) \subseteq L'$ ($A(t) \subseteq \overline{L'}$). We say that t is *redundant* if it is either left redundant or right redundant. Intuitively, t is left (right) redundant if all examples that can reach t (considering the influence of the future features only) end up in the left (right) child of t. This also implies that if t is left (right) redundant, then the DT template obtained after left (right) contracting t is equivalent with T (all examples end up in the same leaves). Finally, let us extend the definition $E_T(t)$ from DTs to DT templates. That is, for a DT template T for a node b, a node $t \in V(T)$, and a set of examples $E' \subseteq exam(b)$, we denote by $E_T(E', t)$ (or $E_T(t)$ if $E' = exam(b)$) the set of examples $e \in E'$ with $\lambda_b(e) \in A(t)$ and $e \in E'[\tau(t)]$, where $\tau(t)$ is the assignment of the features in $feat(b)$ along the path from the root of T to t.

define τ in prelims

We say that T is a *DT skeleton* for b if T is a DT that can only use features in $F_{[k]} \cup I_{[k]}$. Note that because of the features $F_{[k]}$, whose behaviour w.r.t. the examples in $exam(b)$ is not defined, the behaviour w.r.t. the examples in $exam(b)$ of such a DT skeleton is not necessarily defined. Nevertheless, the behaviour of a feature f_ℓ in $F_{[k]}$ is well-defined w.r.t. the examples in $exam(E) \setminus exam(b)$, i.e., it behaves the same as any feature in $feat(b)$ with label ℓ. Intuitively, DT skeletons are obtained from DT templates after replacing every feature f in $feat(b)$ with the forgotten feature $f_{\lambda_b(f)}$. This allows us to further compress the information contained in DT templates, while still keeping the information about how the DT template behaves w.r.t. future examples in E. In particular, DT skeletons will form the main information stored by our records.

Let T be a DT skeleton and $t \in V(T)$. Similarly as we did for DT templates, we say that T is *complete* if it uses no future features and otherwise we say that it is incomplete. We say that an inner node t with $f_\ell = feat(t) \in F_{[k]}$ is *left (right) redundant* in T if $f_\ell \in feat(\text{anc}^L(t))$ ($f_\ell \in feat(\text{anc}^R(t))$). Similarly, as for DT (templates), if t with $feat(t) \in F_{[k]}$ is left (right) redundant, then we can left (right) contract t without changing the properties of T.

Let T be a DT (skeleton/template). Then, we denote by $r(T)$ the DT obtained from T after left (right) contracting every left (right) redundant node of T. The following lemma shows that $r(T)$ is well-defined, i.e., the order in which the left (right) contractions are performed does not influence the result.

▶ **Lemma 7.** *Let T be a DT (skeleton/template), let $t \in V(T)$ be a left (right) redundant node in T, and let T' be the DT (skeleton/template) obtained from T after left (right) contracting t. Then, a node $t' \in V(T')$ is left (right) redundant in T' if and only if t' is left (right) redundant in T.*

Proof. Clearly, if t' is left (right) redundant in T', then the same is true in T; this is because if t'' is a left (right) ancestor of t' in T', then the same holds in T. So suppose that t' is left (right) redundant in T. If $feat(t')$ is a real or forgotten feature, then t' is left (right) redundant in T because of some left (right) ancestor t_A of t' in T with $feat(t_A) = feat(t')$. If $t_A \neq t$, then t' is also left (right) redundant in T' (because t_A is also in T'). Otherwise, $t_A = t$ and therefore t must also be left (right) redundant in T; because otherwise t' was removed when t was contracted. Therefore, t is left (right) redundant in T because of some left (right) ancestor t'_A of t in T with $feat(t'_A) = feat(t) = feat(t')$, which implies that t' is left (right) redundant in T' because of t'_A.

If, on the other hand, $feat(t')$ is a future feature $f_{L'}$, then $A_T(t') \subseteq \overline{L'}$ ($A_T(t') \subseteq L'$). We will show that $A_T(t') = A_{T'}(t')$, which shows that t' remains left (right) redundant in

T'. This clearly holds if $\mathit{feat}(t)$ is not a future feature. So suppose that $\mathit{feat}(t) = f_L$. Then, because t is left (right) redundant in T (because otherwise t' would have been removed from T when contracting t), we have that $A_T(t) \subseteq \overline{L}$ ($A_T(t) \subseteq L$). Therefore, $A_T(t) = A_T(t) \cap \overline{L}$ ($A_T(t) = A_T(t) \cap L$), which shows that t has no influence on $A_T(t')$ and therefore implies that $A_T(t') = A_{T'}(t')$. ◀

We now show that $r(T)$ shares certain properties with T. In particular, the first observation shows that if T is a DT template for b, then so is $r(T)$.

▶ **Observation 8.** *Let T be a DT template for b, then so is $r(T)$.*

Proof. It suffices to show that if t is left (right) redundant in T and e is in $E_T(t)$, then e goes to the left (right) child of t in T. If $\mathit{feat}(t) \in \mathit{feat}(b)$, then t is left (right) redundant because of some left (right) ancestor t' with $\mathit{feat}(t') = \mathit{feat}(t)$. Moreover, because $e \in E_T(t)$, e went to the left (right) child of t' and therefore e goes to the left (right) child of t (because $\mathit{feat}(t) = \mathit{feat}(t')$). If, on the other hand, $\mathit{feat}(t)$ is some future feature f_L, then $A(t) \subseteq \overline{L}$ ($A(t) \subseteq L$) and because $e \in E_T(t)$, also $\lambda_b(e) \in A(t)$. Therefore, e goes to the left (right) child of t. ◀

The second observation shows the similarity in behaviour of T and $r(T)$ with respect to future examples in $E \setminus \mathit{exam}(b)$.

▶ **Observation 9.** *Let T be a DT (skeleton/template) for b, and let e be an example in $E \setminus \mathit{exam}(b)$ that is correctly classified by T. Then, e is also correctly classified by $r(T)$.*

Proof. The proof is very similar to the proof of Observation 8. That is, again it suffices to show that if t is left (right) redundant in T and e goes to t, then e goes to the left (right) child of t in T. The proof is essentially the same as the proof in Observation 8 for the case that $\mathit{feat}(t)$ is a real feature or a future feature. Moreover, if $\mathit{feat}(t)$ is a forgotten feature f_ℓ, then t is left (right) redundant because of some left (right) ancestor t' with $\mathit{feat}(t') = \mathit{feat}(t) = f_\ell$. Moreover, because e goes to t, e went to the left (right) child of t' and therefore e goes to the left (right) child of t (because e behaves in the same way w.r.t. every feature in V_b that has the same label). ◀

Before we define our records and their semantics, we first show a bound on the number of DT skeletons (and the time to enumerate those) as this will allow us to obtain a similar bound for the number of records. We say that T is *reduced* if $r(T) = T$.

▶ **Observation 10.** *Let T be a reduced DT skeleton whose forgotten features use a set of at most k_F labels and whose future features use a set of at most k_I labels. Then, T has height at most $k_F + k_I + 1$ and size at most $2^{k_F+k_I+1}$.*

Proof. Consider a root-to-leaf path P in T. Then, every forgotten feature appears at most once on P; because the second occurrence of such a feature would necessarily be redundant. Therefore, P can contain at most k_F forgotten features. Similarly, P can contain at most k_I future features, since otherwise one of the future features on P would be redundant. Therefore, T has height at most $k_F + k_I + 1$ and therefore size at most $2^{k_F+k_I+1}$. ◀

We obtain the following corollary as a special case.

▶ **Corollary 11.** *Let T be a reduced DT skeleton for a node $b \in V(B)$. Then, T has height at most $2k + 1$ and size at most 2^{2k+1}.*

▶ **Observation 12.** *The are at most $(k_F + 2^{k_I})^{2^{k_F+k_I+2}+1}$ reduced DT skeletons whose forgotten features use a set of at most k_F labels and whose future features use a set of at most k_I labels. Moreover, those can be enumerated in time $\mathcal{O}((k_F + 2^{k_I})^{2^{k_F+k_I+2}+1})$.*

Proof. Because of Observation 10 such a DT skeleton has height at most $k_F + k_I + 1$ and size at most $2^{k_F+k_I+1}$. Therefore, the statement of the lemma follows from Lemma 1 by setting $a = k_F + 2^{k_I}$ and $s = 2^{k_F+k_I+1}$. ◀

We obtain the following corollary as a special case.

▶ **Corollary 13.** *The are at most $(k + 2^k)^{2^{2k+2}+1}$ reduced DT skeletons for a node $b \in V(B)$ and those can be enumerated in time $\mathcal{O}((k + 2^k)^{2^{2k+2}+1})$.*

Let T be a DT (template/skeleton) using only features in $\mathit{feat}(E) \cup F_L \cup I_L$ for some set L of labels (usually $L = [k]$). A *feature relabelling* is a function $\alpha : \mathit{feat}(E) \cup F_L \to F_{L'} \cup I_{L'}$, where L' is some set of labels (usually $L' = L$). With a slight abuse of notation, we denote by $\alpha(T)$, the decision tree obtained after relabelling all features used by T according to α, i.e., $\alpha(T)$ is obtained from T after replacing the feature assignment function $\mathit{feat}_T(t)$ for T with the function $\mathit{feat}_{\alpha(T)}(t)$ defined by setting $\mathit{feat}_{\alpha(T)}(t) = \alpha(\mathit{feat}_T(t))$ if α is defined for $\mathit{feat}(t)$ and $\mathit{feat}_{\alpha(T)}(t) = \mathit{feat}_T(t)$, otherwise. We say that two feature relabellings α_1 and α_2 are *compatible* if they agree on their shared domain.

We denote by α^s_b the *standard feature relabelling* for b, i.e., the function $\alpha^s_b : \mathit{feat}(b) \to F_{[k]}$ defined by setting $\alpha^s_b(f) = f_{\lambda_b(f)}$ for every $f \in \mathit{feat}(b)$.

We now show an important property on the interchangeability of feature relabellings and reductions. That is, we show in Lemma 15 below that the effect of any sequence of feature relabellings and reductions that ends with the reduction operation ($r()$) is the same as the effect of the sequence that contains the same relabelling operations followed by one reduction operation at the end. To show this property, we need the following auxiliary lemma.

▶ **Lemma 14.** *Let T be a DT (template/skeleton) for a node $b \in V(B)$ and let α be a feature relabelling. If a node $t \in V(T)$ is left (right) redundant in T, then it is also left (right) redundant in $\alpha(T)$.*

Proof. We distinguish the following two cases. If $\mathit{feat}(t) \in \mathit{feat}(b) \cup F_{[k]}$, then t is left (right) redundant in T because of some left (right) ancestor t' of t in T with $\mathit{feat}(t) = \mathit{feat}(t')$. Because $\alpha(\mathit{feat}(t)) = \alpha(\mathit{feat}(t'))$, we obtain that t is also left (right) redundant in $\alpha(T)$ because of t'. If, on the other hand, $\mathit{feat}(t) \in I_{[k]}$, then t is left (right) redundant in T because of some set A of ancestors t_A with $\mathit{feat}(t_A) \in I_{[k]}$. Because the domain of α does not include future features, it follows that α does not change the feature assignment for t nor for its ancestors in A, and therefore t is also left (right) redundant in $\alpha(T)$. ◀

▶ **Lemma 15.** *Let T be a DT (template/skeleton) and let α be a feature relabelling. Then, $r(\alpha(T)) = r(\alpha(r(T)))$.*

Proof. Let T' be the DT (template/skeleton) obtained from $\alpha(T)$ after left (right) contracting every node t that is left (right) redundant in T; note that such a node t is also left (right) redundant in $\alpha(T)$ because of Lemma 14. Then, $T' = \alpha(r(T))$ and moreover because of Lemma 7 (and using the fact that every node t that is left (right) redundant in T is so in $\alpha(T)$), a node $t \in V(T')$ is left (right) redundant in T' if and only if it is so in $\alpha(T)$. Therefore, a node t is left (right) redundant in $\alpha(T)$ if and only if it is left (right) redundant in T or in $\alpha(r(T)) = T'$, which shows that $r(\alpha(T)) = r(\alpha(r(T)))$. ◀

We are now ready to define the records and their semantics. A *record* for b is a pair (T, s) such that T is a reduced decision tree skeleton for b and s is a natural number. We say that a record (T, s) is *semi-valid* for b if there is a (reduced) DT template T' for b such that $r(\alpha_b^s(T')) = T$ and $s = |V(T') \setminus V(T)|$. We say that a record (T, s) is *valid* for b if s is the minimum number such that (T, s) is semi-valid. We denote by $\mathcal{R}(b)$ the set of all valid records for b. The following corollary follows immediately from Corollary 13.

▶ **Corollary 16.** $|\mathcal{R}(b)| \leq (k + 2^k)^{2^{2k+2}+1}$

Note that E has a DT of size at most s if and only if $\mathcal{R}(r)$ for the root r of B contains a record (T, s) such that T is complete.

3.3 Proof to the Main Result

We will now show that we can compute $\mathcal{R}(b)$ for every of the 3 node types of a nice k-NLC expression tree provided that $\mathcal{R}(c)$ has already been computed for every child c of b.

▶ **Lemma 17** (leaf node)**.** *Let $b \in V(B)$ be a leaf node. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}(k(1 + 2^k)^{2^{k+3}+1})$.*

Proof. Let $i(v)$ be the initial k-graph associated with b. If v is a feature, then $\mathcal{R}(b)$ contains all records $(T, 0)$ such that T is a reduced DT skeleton for b using only the features in $\{f_{\lambda(v)}\} \cup I_{[k]}$. The correctness in this case follows because V_b contains no examples and therefore every reduced DT skeleton constitutes a valid record for b. Moreover, the run-time follows from Observation 12, since the time required to enumerate all those reduced DT skeletons is at most $\mathcal{O}((1 + 2^k)^{2^{k+3}+1})$.

If, on the other hand v is an example, then $\mathcal{R}(b)$ contains all records $(T, 0)$ such that T is a reduced DT skeleton for b using only the features in $I_{[k]}$ and which correctly classify v. Because of Observation 12, those can be enumerated in time $\mathcal{O}((1 + 2^k)^{2^{k+3}+1})$ and checking for each of those whether it correctly classifies v can be achieved in time $\mathcal{O}(k)$ because of Observation 10. ◀

Before we present the corresponding lemmas for the join-node and the relabelling node, we first need to introduce the so-called plug in operation that allows us to reverse the reductions applied to a DT (skeleton/template).

Let T and T' be two DT (templates/skeletons). Let $P = (t, p_1, \dots, p_\ell, t')$ be the path from t to t' in T such that t is an ancestor of t' in T, for some integer ℓ. Moreover, let $e = (p, c)$ be an edge in T' such that p is the parent of c in T'. We say that the DT (template/skeleton) T'' is obtained by *pluging in the path P into T' at edge e* if T'' is obtained from T' by doing the following. For an inner vertex p_i of P, let $T(P, p_i)$ be the subtree of T rooted at the unique child c of p_i that is not on P. Let P' be the induced subtree of T containing all vertices of P plus all vertices of $T(P, p_i)$ for every i with $1 \leq i \leq \ell$. Then, T'' is obtained from T' by removing the edge $e = (p, c)$, adding P', and adding the edge from p to p_1 as well as the edge from p_ℓ to c. Moreover, T'' inherits all feature assignments as well as the left (right) child relation from T and T'.

The significance of the plug in operation comes from the fact that it allows us to reverse the reduction that has been applied to a DT (template/skeleton). For instance, consider a node b of B and let T be a DT skeleton for b and let T' be a DT template for b such that $T = r(\alpha_b^s(T'))$. Then, we can use the plug in operation to reverse the direction or in other words obtain T' from T as follows. Let $z : V(T) \to V(T')$ be the injective function mapping every node in T to its original node in T'. Then, we first use z to reverse the relabelling given by $\alpha_b^s(T')$, i.e., let T^0 be the DT obtained from T by setting $feat_{T^0}(t) = feat_{T'}(z_H(t))$ for every $t \in V(T^0)$. We now add back the nodes in $V(T') \setminus V(T)$ with the help of our plug in operation. In particular, for every edge $e = (p, c)$ in T^0, where p is the parent of c in T^0, let $P(e)$ be the path in T' between $z(p)$ and $z(c)$. Let T^1 be the DT template obtained from T^0 after pluging in the path $P(e)$ into T^0 at edge e, for every edge $e = (p, c)$ of T^0. Then, it is easy to see that $T^1 = T'$.

todo: simplify the run-time expression

▶ **Lemma 18** (join node). *Let $b \in V(B)$ be a join node. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}(2^{3k+1}(2k+2^k)^{2^{3k+2}+1})$.*

Proof. Let b_L and b_R be the left and right child of b in B, respectively. Let M_b be the join matrix for the node b, i.e., M_b is a $k \times k$ binary matrix. For every label $i \in [k]$, let $A_{i,*} = \{\, j \in [k] \mid M_b[i,j] = 1 \,\}$ and $A_{*,i} = \{\, j \in [k] \mid M_b[j,i] = 1 \,\}$.

To distinguish between forgotten features from the left and the right subtree, we introduce the left i_L and the right version i_R for every label $i \in [k]$. With a slight abuse of notation, we also denote by $[k_L]$ be the set $\{1_L, \dots, k_L\}$ of (left) labels and we denote by $[k_R]$ be the set $\{1_R, \dots, k_R\}$ of (right) labels.

To compute the set $\mathcal{R}(b)$ of valid records for b, we first enumerate all reduced DT skeletons T using features in $[k_L] \cup [k_R] \cup I_{[k]}$. Because of Observation 12, those can be enumerated in time $\mathcal{O}((2k+2^k)^{2^{3k+2}+1})$. For every such reduced DT skeleton T, we now do the following in order to decide whether T gives rise to a valid record for b. Let $\alpha_{LR\to} : F_{[k_L]} \cup F_{[k_R]} \to F_{[k]}$ be the feature relabelling that relabels every (left/right) feature $f_{i_H} \in F_{[k_L]} \cup F_{[k_R]}$ (for some $H \in \{L, R\}$) to its original feature f_i.

Let $\alpha_L : F_{[k_R]} \to I_{[k]}$ be the feature relabelling that relabels every forgotten feature $f_{i_R} \in F_{[k_R]}$ to the future feature $f_{A_{*,i}}$. Let T_L be the reduced DT skeleton obtained from T after applying the relabelling using α_L followed by $\alpha_{LR\to}$ and then reducing the resulting DT skeleton, i.e., $T_L = r(\alpha_{LR\to}(\alpha_L(T)))$.

Similarly, let $\alpha_R : F_{[k]_L} \to I_{[k]}$ be the feature relabelling that relabels every forgotten feature $f_{i_L} \in F_{[k_L]}$ to the future feature $f_{A_{i,*}}$. Let T_R be the reduced DT skeleton obtained from T after applying the relabelling using α_R followed by $\alpha_{LR\to}$ and then reducing the resulting DT skeleton, i.e., $T_R = r(\alpha_{LR\to}(\alpha_R(T)))$.

Let $\hat{T} = r(\alpha_{LR\to}(T))$ and $\hat{s} = |V(T) \setminus V(\hat{T})|$. We now check whether there are records $(T_L, s_L) \in \mathcal{R}(b_L)$ and $(T_R, s_R) \in \mathcal{R}(b_R)$. If not we discard T and if yes, then we add the record $(\hat{T}, s_L + s_R + \hat{s})$ to $\mathcal{R}(b)$. This completes the description about how the records $\mathcal{R}(b)$ are computed. Moreover, the run-time for computing $\mathcal{R}(b)$ can be obtained as follows. First, because of Observation 12, we can enumerate all reduced DT skeletons T in time $\mathcal{O}((2k+2^k)^{2^{3k+2}+1})$. Moreover, computing $\hat{T}$ and $\hat{s}$ can be done in time $\mathcal{O}(|T|) = \mathcal{O}(2^{3k+1})$ (using Observation 10). Finally, computing T_L and T_R and checking the existence of the records $(T_L, s_L) \in \mathcal{R}(b_L)$ and $(T_R, s_R) \in \mathcal{R}(b_R)$ can be achieved in time $\mathcal{O}(|T|) = \mathcal{O}(2^{3k+1})$; here we assume that the records in $\mathcal{R}(b)$ are stored in an array whose key is $\hat{T}$. Therefore, we obtain $\mathcal{O}(|T|(2k+2^k)^{2^{3k+2}+1}) = \mathcal{O}(2^{3k+1}(2k+2^k)^{2^{3k+2}+1})$ as the total run-time for computing $\mathcal{R}(b)$.

We now show the correctness of our construction for $\mathcal{R}(b)$, i.e., we have to show that a record is valid if and only if we have added such a record according to our construction above. For this it suffices to show that a record is semi-valid if and only if we have added such a record according to our construction above. This is because, a valid record (T, s) can be obtained from the set of all semi-valid records (T, s'), where s is the minimum s' among all semi-valid records for T.

Towards showing the forward direction, suppose that $(\hat{T}, s)$ is a semi-valid record for b. Therefore, there is a DT template T' for b such that $\hat{T} = r(\alpha_b^s(T'))$ and $s = |V(T') \setminus V(T)|$.

Let $\alpha_{\to R} : F_{[k]} \to F_{[k_R]}$ ($\alpha_{\to L} : F_{[k]} \to F_{[k_L]}$) be the feature relabelling that relabels every forgotten feature $f_i \in F_{[k]}$ to its corresponding forgotten feature in $[k_R]$ ($[k_L]$), i.e., $\alpha_{\to R}(i) = i_R$ ($\alpha_{\to L}(i) = i_L$) for every $i \in [k]$.

Let $T = r(\alpha_{\to R}(\alpha^s_{b_R}(\alpha_{\to L}(\alpha^s_{b_L}(T')))))$ and let $\hat{s} = |V(T) \setminus V(\hat{T})|$. Because $\alpha^s_b = \alpha_{LR\to} \circ \alpha_{\to R} \circ \alpha^s_{b_R} \circ \alpha_{\to L} \circ \alpha^s_{b_L}$, we obtain from Lemma 15 that $\hat{T} = r(\alpha_{LR\to}(T))$.

Let $T_L = r(\alpha_{LR\to}(\alpha_L(T)))$ and $T_R = r(\alpha_{LR\to}(\alpha_R(T)))$. It remains to show that there are s_L and s_R with $s = s_L + s_R + \hat{s}$ such that $(T_L, s_L) \in \mathcal{R}(b_L)$ and $(T_R, s_R) \in \mathcal{R}(b_R)$.

Let $T'_L = r(\alpha_L(\alpha_{\to R}(\alpha^s_{b_R}(T'))))$ and $T'_R = r(\alpha_R(\alpha_{\to L}(\alpha^s_{b_L}(T'))))$. Note that T'_H is a DT template for b_H because so is T'.

Note that $T_L = r(\alpha^s_{b_L}(T'_L))$ because of Lemma 15 and the observation that the sequence $\alpha_{LR\to} \circ \alpha_L \circ \alpha_{\to R} \circ \alpha^s_{b_R} \circ \alpha_{\to L} \circ \alpha^s_{b_L}$ of relabellings to obtain T_L via T has the same total effect as the sequence $\alpha^s_{b_L} \circ \alpha_L \circ \alpha_{\to R} \circ \alpha^s_{b_R}$ of relabellings to obtain T_L via T'_L. Using a similar argument, we obtain that $T_R = r(\alpha^s_{b_R}(T'_R))$. Let $s_H = |V(T'_H) \setminus V(T_H)|$ for every $H \in \{L, R\}$. Then, T'_H shows that (T_H, s_H) is a semi-valid record for b_H.

maybe a longer explanation

It remains to show that $s_L + s_R + \hat{s} = s$. Note first that $s = |V(T') \setminus V(\hat{T})| = |V(T') \setminus V(T)| + |V(T) \setminus V(\hat{T})| = |V(T') \setminus V(T)| + \hat{s}$ and it therefore suffices to show that $s_L + s_R = |V(T') \setminus V(T)|$. Towards showing this, let t be a node in $|V(T') \setminus V(T)|$. First note that $\mathit{feat}_{T'}(t) \in \mathit{feat}(b_H)$ for some $H \in \{L, R\}$, because all nodes with future features in T' are also in T. Therefore, t is in $V(T'_H) \setminus V(T_H)$, which shows that t is either in $V(T'_L) \setminus V(T_L)$ or in $V(T'_R) \setminus V(T_R)$, as required.

Towards showing the reverse direction, suppose that our construction adds the record $(\hat{T}, s_L + s_R + \hat{s})$ and let T, T_L, and T_R be as defined in the construction. Recall that:

- T is reduced and $\hat{T} = r(\alpha_{LR\to}(T))$,
- $T_L = r(\alpha_L(T))$ and (T_L, s_L) is semi-valid for b_L,
- $T_R = r(\alpha_R(T))$ and (T_R, s_R) is semi-valid for b_R.

Let T'_L be the reduced DT template for b_L such that $T_L = r(\alpha^s_{b_L}(T'_L))$ and $s_L = |V(T'_L) \setminus V(T_L)|$, which exists because (T_L, s_L) is semi-valid for b_L. Similarly, let T'_R be the reduced DT template for b_R such that $T_R = r(\alpha^s_{b_R}(T'_R))$ and $s_R = |V(T'_R) \setminus V(T_R)|$, which exists because (T_R, s_R) is semi-valid for b_R.

We now show how to construct a witness T' (from T, T'_L, and T'_R) for the semi-validity of the record $(\hat{T}, s_L + s_R + \hat{s})$, i.e., T' is a reduced DT template for b such that $\hat{T} = r(\alpha^s_b(T'))$ and $s_L + s_R + \hat{s} = |V(T') \setminus V(\hat{T})|$.

Informally, we obtain T' from T after reversing the relabelling and reduction operations applied to T'_L and T'_R to obtain T_L and T_R, respectively; recall that $T_H = r(\alpha^s_{b_H}(T'_H))$ for $H \in \{L, R\}$. That is, we will reverse the labelling for the nodes in T and add back the nodes to T that have been removed from T'_L and T'_R.

Let $H \in \{L, R\}$. Because T_H is obtained from T by reduction, every node in T_H corresponds to a unique node in T. Therefore, there is an injective function $x_H : V(T_H) \to V(T)$ mapping every node in T_H to its original node in T. Similarly, because T_H is obtained from T'_H by reduction, there is an injective function $y_H : V(T_H) \to V(T'_H)$ mapping every node in T_H to its original node in T'_H. See also Figure 2 for an illustration of these mappings.

In order to obtain T' from T, we will essentially need to be able to reverse the reduction operation $T_H = r(\alpha^s_{b_H}(T'_H)$ that has been applied to T'_H to obtain T_H for every $H \in \{L, B\}$. To do so we will make use of the plug in operation.

Our first order of business is to rename all forgotten features in T to their real features as given by T'_L and T'_R. That is, for every node t in T assigned to a forgotten feature, i.e., $\mathit{feat}_T(t) \in F_{[k_L]} \cup F_{[k_R]}$, we do the following. If $\mathit{feat}_T(t) \in F_{[k_H]}$ for $H \in \{L, R\}$, then t is also in T_H and hence also in T'_H. Therefore, we can change $\mathit{feat}_T(t)$ to the real feature assigned

$$\begin{tikzcd}
T'_L \arrow[r, dash] & T' \arrow[r, dash] \arrow[d, dash] & T'_R \\
T_L \arrow[u, hook, "y_L"] \arrow[r, hook, "x_L"] & T & T_R \arrow[u, hook', "y_R"] \arrow[l, hook', "x_R"']
\end{tikzcd}$$

Figure 2

to t in T'_H. Let T^0 be the DT obtained from T after renaming all forgotten features to real features in this manner.

Consider an edge $e = (p, c)$ in T_L such that p is the parent of c in T_L. Then, e corresponds to a path $P'_L(e)$ between $y_L(p)$ and $y_L(c)$ in T'_L. Similarly, e corresponds to a path $P_L(e)$ between $x_L(p)$ and $x_L(c)$ in T^0.

Our next order of business is now to add all nodes to T^0 that have been removed when going from T'_L to T_L (via the reduction $r(\alpha^s_{b_L}(T'_L)))$. To achieve this, we go over every edge $e = (p, c)$ of T_L such that p is the parent of c in T_L and plug in the path $P'_L(e)$ (from T'_L) into the last edge on the path $P_L(e)$ (from T^0). Let T^1 be the tree obtained from T^0 after doing this operation for every edge of T_L.

Consider an edge $e = (p, c)$ in T_R such that p is the parent of c in T_R. Then, e corresponds to a path $P'_R(e)$ between $y_R(p)$ and $y_R(c)$ in T'_R. Similarly, e corresponds to a path $P_R(e)$ between $x_R(p)$ and $x_R(c)$ in T^1. Similarly to above, we now add all nodes to T^1 that have been removed when going from T'_R to T_R (via the reduction $r(\alpha^s_{b_R}(T'_R)))$. To achieve this, we go over every edge $e = (p, c)$ of T_R such that p is the parent of c in T_R and plug in the path $P'_R(e)$ (from T'_R) into the last edge on the path $P_R(e)$ (from T^1). Let T' be the tree obtained from T^1 after doing this operation for every edge of T_R.

We now show that T' is indeed a witness for the semi-validity of the record $(\hat{T}, s_L+s_R+\hat{s})$, i.e., T' is a reduced DT template for b such that $\hat{T} = r(\alpha^s_b(T'))$ and $s_L+s_R+\hat{s} = |V(T')\setminus V(\hat{T})|$.

We start by showing that T' is reduced. First note that because T is reduced so is T^0. Consider a node $t \in V(T')$. If $\mathit{feat}_{T'}(t) \in \mathit{feat}(b_H)$ for some $H \in \{L, R\}$, then t is also in $V(T'_H)$. Therefore, if t were redundant in T', it would also be redundant in T'_H, which cannot be the case because T'_H is reduced. Moreover, if on the other hand, $\mathit{feat}_{T'}(t) \in I_{[k]}$, then t is in T^0 and therefore cannot be redundant because T^0 is reduced. Therefore, T' is reduced and it obviously only uses features in $\mathit{feat}(b) \cup F_{[k]}$. We show next that T' is a DT template for b, i.e., T' classifies all examples in $\mathit{exam}(b)$ correctly. Towards showing this, let $e \in \mathit{exam}(b)$, then $e \in \mathit{exam}(b_H)$ for some $H \in \{L, R\}$. Because T'_H is a DT template for b_H, we know that e is correctly classified by T'_H. Let ℓ be the leaf in T'_H that contains e, i.e., $e \in E_{T'_H}(\ell)$ and let Q be the path from the root of T'_H to ℓ. Then, ℓ also exists in T' and moreover the path P from the root of T' to ℓ contains all nodes of Q. Note furthermore that if a node t in Q has its left/right child on Q, then the same holds on P. We will show that e follows along the path P in T' and therefore ends up in ℓ, which shows that e is correctly classified by T'.

Let t be a node of P. If t is also in Q, then e will be send to the child of t in P. Otherwise, t is either in $V(T) \setminus V(T_H)$ or t is in $T'_{\overline{H}} \setminus T_{\overline{H}}$, where $\overline{H} = L$ if $H = R$ and $\overline{H} = R$ otherwise.

In the former case, $\mathit{feat}_{T'}(t) \in I_{[k]}$ or $\mathit{feat}_{T'}(t) \in \mathit{feat}(b_{\overline{H}})$, which implies that t behaves towards e in the same manner as some future feature $f_L \in I_{[k]}$, i.e., if $\mathit{feat}_{T'}(t) \in I_{[k]}$, then $f_L = \mathit{feat}_{T'}(t)$ and if $\mathit{feat}_{T'}(t) \in \mathit{feat}(b_{\overline{H}})$, then $f_L = \alpha_L(\mathit{feat}_T(t))$. Moreover, t is redundant in $\alpha_L(T)$ because of its ancestors in T_H, i.e., either $A_{\alpha_L T}(t) \subseteq L$ or $A_{\alpha_L T}(t) \subseteq \overline{L}$. Because all these ancestors are in T_H and therefore on Q, $\lambda_{b_L}(e) \in A_{\alpha_L T}(t)$, which implies that e is send to the non-redundant child of t. Finally, since P contains ℓ it follows that P contains also the non-redundant child of t in $\alpha_L(T)$ and therefore e is send to the child of t on P, as required.

In the latter case, i.e., the case that t is in $V(T'_{\overline{H}}) \setminus V(T_{\overline{H}})$, t is redundant in $\alpha^s_{b_{\overline{H}}}(T'_{\overline{H}})$ because of some ancestor $t' \in V(T_{\overline{H}})$ with $\alpha^s_{b_{\overline{H}}}(feat_{T'}(t)) = \alpha^s_{b_{\overline{H}}}(feat_{T'}(t'))$. Therefore, $feat_{T'}(t')$ behaves in the same manner towards e as $feat_{T'}(t)$, which because t' is on Q (because $t' \in V(T_{\overline{H}})$) implies that e is send to the (non-redundant) child of t on P.

It remains to show that $\hat{T} = r(\alpha^s_b(T'))$ and $s_L + s_R + \hat{s} = |V(T') \setminus V(\hat{T})|$. Towards showing this, we first show that $T = r(\alpha_{T' \to T}(T'))$, where $\alpha_{T' \to T} = \alpha_{\to L} \circ \alpha^s_{b_R} \circ \alpha_{\to L} \circ \alpha^s_{b_L}$. In other words, we need to show that the set of redundant nodes in $\alpha_{T' \to T}(T')$ is equal to $V(T') \setminus V(T) = V(T') \setminus V(T^0)$. Because, as shown above T' is reduced, it follows that if a node t is redundant $\alpha_{T' \to T}(T')$, then $t \in feat_{T'}(b_H)$ for some $H\{L, R\}$. Because all such nodes, i.e., nodes t in T' with $t \in feat_{T'}(b_H)$ are also in T'_H, we obtain that t is redundant in $\alpha_{T' \to T}(T')$ if and only if it is redundant in $\alpha^s_{b_H}(T'_H)$. Therefore, $\bigcup_{H \in \{L,R\}} V(T'_H) \setminus V(T_H)$ is the set of all redundant nodes in $\alpha_{T' \to T}(T')$, which is equal to $V(T') \setminus V(T^0)$ by construction of T', as required. Note that $|V(T') \setminus V(T^0)| = s_L + s_R$ because of the construction of T'. Now, because $\hat{T} = r(\alpha_{LR\to}(T))$ and $\alpha^s_b = \alpha_{LR\to} \circ \alpha_{T' \to T}$, we obtain from Lemma 15 that $\hat{T} = r(\alpha^s_b(T'))$. Finally, because $|V(T') \setminus V(T^0)| = s_L + s_R$ and $|V(T^0) \setminus V(\hat{T})| = \hat{s}$, it follows that $|V(T') \setminus V(\hat{T})| = s_L + s_R + \hat{s}$, as required. ◀

▶ **Lemma 19** (relabel node). *Let $b \in V(B)$ be relabelling node in B. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}(2^{2k+1}(k + 2^k)^{2^{2k+2}+1})$.*

Proof. Let c be the unique child of b in B and let $R_b : [k] \to [k]$ be the relabelling function associated with b. Because B is nice, it holds that there are labels i and j with $i \neq j$ such that $R(i) = j$ and $R(\ell) = \ell$ for every $\ell \in [k] \setminus \{i\}$.

We say that a future feature $f_L \in I_{[k]}$ is *good* if it does not distinguish between i and j, i.e., $i \in L$ if and only if $j \in L$, and *bad* otherwise. For a bad future feature f_L, we denote by $g(f_L)$ the good future feature $f_{g(L)}$, where $g(L) = L \cup \{i\}$ if $j \in L$ and $g(L) = L \setminus \{i\}$, otherwise, i.e., informally, $g(f_L)$ is the good feature corresponding to f_L that sends all examples with label i to the same side as f_L sends all examples with label j.

To obtain the set $\mathcal{R}(b)$ of valid records for b, we first enumerate all reduced DT skeletons T for b. Let $\alpha^I_{i\to j} : I_{[k]} \to I_{[k]}$ be the function defined by setting $\alpha^I_{i\to j}(f_L) = g(f_L)$ for every bad future feature $f_L \in I_{[k]}$, i.e., $\alpha^I_{i\to j}$ relabels every bad feature f_L to its corresponding good feature $g(f_L)$. Let $T^c = r(\alpha^I_{i\to j}(T))$. We now check whether $\mathcal{R}(c)$ contains a record of the form (T^c, s^c). If not, then we disregard T. Otherwise, let $\alpha^F_{i\to j}$ be the feature relabelling that relabels the forgotten feature f_i to the forgotten feature f_j. Let $\hat{T} = r(\alpha^F_{i\to j}(T))$ and $\hat{s} = |V(T) \setminus V(\hat{T})|$. We now distinguish two cases. If we have not yet added any record of the form $(\hat{T}, s')$ to $\mathcal{R}(b)$, then we add the record $(\hat{T}, s^c + \hat{s})$ to $\mathcal{R}(b)$. Otherwise we replace the unique existing record $(\hat{T}, s')$ with the record $(\hat{T}, \min\{s', s^c + \hat{s}\})$. This completes the construction of the set $\mathcal{R}(b)$ of valid records.

Note that computing $\mathcal{R}(b)$ in this manner can be achieved in the stated run-time. This is because due to Corollary 13 we can enumerate all possible choices for T in time $\mathcal{O}((k + 2^k)^{2^{2k+2}+1})$ and for every such choice T we can compute T^c and $\hat{T}$ and check the existence of a record (T^c, s) in $\mathcal{R}(c)$ in time at most $\mathcal{O}(|T|) = \mathcal{O}(2^{2k+1})$ (because of Corollary 11).

It remains to show the correctness of our construction for $\mathcal{R}(b)$, i.e., we have to show that a record is valid for b if and only if we have added such a record according to our construction above. For this it suffices to show that a record is semi-valid for b if and only if we have added such a record according to our construction above. This is because, a valid record

(T, s) can be obtained from the set of all semi-valid records (T, s'), where s is the minimum s' among all semi-valid records for T.

Towards showing the forward direction, suppose that the record $(\hat{T}, s)$ is semi-valid for b. Then, there is a reduced DT template T' for b such that $\hat{T} = \alpha_b^s(T')$ and $s = |V(T') \setminus V(\hat{T})|$. Let $T = r(\alpha_c^s(T'))$. Then, $\hat{T} = r(\alpha_{i\to j}^F(T))$ because of Lemma 15 together with the observation that $\alpha_{i\to j}^F \circ \alpha_c^s = \alpha_b^s$. Note that T corresponds to the reduced DT skeleton considered by our construction. Let $T^c = r(\alpha_{i\to j}^I(T))$, let $\hat{s} = |V(T) \setminus V(\hat{T})|$, and let $s^c = s - \hat{s}$. It remains to show that the record (T^c, s^c) is semi-valid for c. Let $T'' = r(\alpha_{i\to j}^I(T'))$. Then, T'' is a reduced DT template for c, because so is T' for b and moreover all examples, in particular those with label i, in *exam*(c) end up in the same leaf in T'' as they do in T'; because of the relabelling $\alpha_{i\to j}^I$ that relabelled all bad future features in T' into their corresponding good future features in T''. Moreover, $T^c = r(\alpha_c^s(T'')$ because of Lemma 15 and furthermore $s^c = s - \hat{s} = |V(T'') \setminus V(T^c)|$. Therefore, T'' shows that (T^c, s^c) is semi-valid for c.

Towards showing the reverse direction, suppose that we have added the record $(\hat{T}, s^c + \hat{s})$ using our construction. Then, there is a DT skeleton T for b with $\hat{T} = r(\alpha_{i\to j}^F(\hat{T}))$ and $\hat{s} = |V(T) \setminus V(\hat{T})|$ and a record $(T^c, s^c) \in \mathcal{R}(c)$ with $T^c = r(\alpha_{i\to j}^I(T))$.

We have to show that the record $(\hat{T}, s^c + \hat{s})$ is semi-valid for b. Because $(T^c, s^c) \in \mathcal{R}(c)$, there is a reduced DT template T' for c such that $T^c = r(\alpha_c^s(T')$ and $s^c = |V(T') \setminus V(T^c)|$. Informally, we now construct a witness T''' for the semi-validity of $(\hat{T}, s^c + \hat{s})$ for b from T' by reversing the reduction $T^c = r(\alpha_{i\to j}^I(T))$.

Let $a : V(T^c) \to V(T')$ be the injective function that maps every node in T^c to its corresponding node in T'; which exists because $T^c = r(\alpha_c^s(T'))$. First Let $b : V(T^c) \to V(T)$ be the injective function that maps every node in T^c to its corresponding node in T; which exists because $T^c = r(\alpha_{i\to j}^I(T))$. First we relabel every future feature in T' in to its corresponding future feature. Let T'' be the DT template obtained from T' by setting $feat_{T''}(a(b^{-1}(t))) = feat_T(t)$ for every node $t \in V(T^c)$ with $feat_T(t) \in I_{[k]}$ and $feat_{T''}(t) = feat_{T'}(t)$ otherwise. Moreover, let T''' be the DT template obtained from T'' by doing the following for every edge $e = (p, c)$ in T^c, where p is the parent of c in T^c. Let $P(e)$ be the path from $b(p)$ to $b(c)$ in T. Then, we plug in the path $P(e)$ into T'' at the edge $(p', a(c))$, where p' is the parent of $a(c)$ in T''.

there is one problem remaining: T''' could now have some forgotten features from T; those should be replaced by arbitrary real features with the same label; but only if such examples also exist; maybe there is a better way??

Then, T''' is a DT template for b, because T' is a DT template for c and we only changed where examples with label i go, which are not present in *exam*(b). Moreover, $T = r(\alpha_c^s(T'''))$ and therefore $\hat{T} = r(\alpha_b^s(T'''))$. Finally, because $|V(T''') \setminus V(T)| = |V(T') \setminus V(T^c)| = s^c$, it holds that $|V(T''') \setminus V(\hat{T})| = s^c + \hat{s}$, which shows that the record $(\hat{T}, s^c + \hat{s})$ is semi-valid for b.

◀

4 An FPT-Algorithm for bounded solution size and δ_{max}.

In the following, let E be a CI and $q \notin feat(E)$. A *decision tree pattern*, or simply a *DT pattern*, T is a rooted subcubic tree, where every leaf node is either a *positive* or *negative* leaf and every non-leaf node is labelled with a feature in $feat(E) \cup \{q\}$. For every node v of a DT pattern T, we indicate with $feat_T(v)$ the label associated to that node. Finally we say that an innder node $v \in V(T)$ is a *fixed node* if $feat_T(v) \in feat(E)$ and *non-fixed* otherwise.

A DT pattern T' is an *improvement* for a DT pattern T if $T' = T$ as rooted trees and $feat_{T'}(v) = feat_T(v)$ for every fixed node v of T. A *complete improvement* T' of T is an improvement such that $feat(T') \subseteq feat(E)$. A *threshold assignment* for a DT pattern T is a function th that maps every fixed node $v \in V(T)$ to a natural number $th(v)$. Note that any complete improvement T' of a DT pattern T can be made to a decision tree with a threshold assignment.

Let T be a DT pattern and th be a threshold assignment for T, for each node v of T we define the set of examples that arrive at node v, $E_T(v)$ as follows: $E_T(v)$ is the set of all examples $e \in E$ such that for each left (right, respectively) arc (u, w) on the unique path from the root of T to v either u is a fixed node and $(feat(u))(e) \leq th(u)$ $((feat(u))(e) > th(u)$, respectively) or u is a non-fixed node. A DT pattern T is *valid* for a set of examples $E' \subseteq E$ if there is threshold assignment for the fixed nodes such that for every positive (negative) example e, $e \in E_T(v)$ for a positive (negative) leaf v.

The definition of $E_T(v)$ is an indication of the behaviour of feature q and of non-fixed nodes. Informally, if any example reaches at a non-fixed node of T then it reach both his children. While no feature in $feat(E)$ can simulate such behaviour for any threshold, q simultaneously cover the two cases a feature with his threshold does not distinguish any two examples.

4.1 Preprocess

Let E be a CI and T be a DT pattern. For every $v \in V(T)$, we define the set of *expected examples* E_v as follows:

- if v is the root, then $E_v = E$;
- if v is the left child of a fixed node v_p, then $E_v = E_{v_p}[feat(v_p) \leq th_L(v_p) + 1]$;
- if v is the right child of a fixed node v_p, then $E_v = E_{v_p}[feat(v_p) > th_R(v_p) - 1]$;
- if v is a child of a non-fixed node v_p, then $E_v = E_{v_p}$.

Node that the definition of E_v is strictly related with the following: if v is a fixed node, let c_ℓ and c_r be the left, risp. right, child of v, we define two values $th_L(v)$ and $th_R(v)$ as follows:

- let $th_L(v)$ be the maximum value in $D_E(feat(v))$ such that T_{c_ℓ} is valid for $E_v[feat(v) \leq th_L(v)]$;
- let $th_R(v)$ be the minimum value in $D_E(feat(v))$ such that T_{c_r} is valid for $E_v[feat(v) > th_R(v)]$.

Before formally proving in Lemma 22 that we are able to compute E_v and $th_L(v)$, $th_R(v)$ (when v is a fixed node) for every $v \in V(T)$, we want to describe the role of E_v in the proof of Lemma 24.

Let us consider the following situation. Suppose we are trying to find a DT of minimum size for a CI E using at least the features in a given support set S. The first step would be to compute a minimum size DT T^* for E such that $feat(T^*) = S$. Next we analyse the case an optimal DT for E uses not only every feature from S but some additional feature: for this reason we consider DT patterns T of size at most s and such that $feat(T) = S \cup \{q\}$.

Let E be a CI, S be a support set for E and T be a DT pattern of size at most s such that $feat(T) = S \cup \{q\}$. If T is a valid DT pattern for E, then T, and every T' obtained after left/right-contracting every non-fixed node v of T, can be easily extended to a solution.

The following two lemmas cover the case T is not a valid DT pattern for E.

▶ **Lemma 20.** *Let T be a DT pattern that is not valid for E. For every node v of T it holds that T_v is not valid for E_v.*

Proof. Let T be a DT pattern that is not valid for E. We show this statement in a root-to-leaves fashion: first we show the statement holds for the root; then we prove it holds for every other node, given the fact it holds for each of its ancestors (or its parent). Let r be the root of T. By definition $E_r = E$ and $T_r = T$ and so the statement follows directly from the assumption.

Let v be the left child of a fixed node v_p. By the definition of $th_L(v_p)$, the DT pattern T_v is not valid for $E_v = E_{v_p}[feat(v_p) \leq th_L(v_p) + 1]$. Similarly if v is the right child of a fixed node v_p, the DT pattern T_v is not valid for $E_v = E_{v_p}[feat(v_p) > th_R(v_p) - 1]$.

Let v be a child of a non-fixed node v_p. Suppose by contradiction that T_v is valid for E_v. We show that T_{v_p} is valid for E_{v_p} and consequently reaching a contradiction with the assumption: any threshold assignment for the fixed nodes of T_v that is a witness of the validity of T_v for E_v is also threshold assignment for the fixed nodes of T_{v_p} that is a witness of the validity of T_{v_p} for $E_{v_p} = E_v$; note this is true because v_p is a non-fixed node. ◀

▶ **Lemma 21.** *Let T be a DT pattern that is not valid for E. For every fixed node v of T it holds that $th_L(v) < th_R(v)$.*

Proof. Let T be a DT pattern that is not valid for E. Suppose by contradiction that there is a fixed node v^* such that $th_L(v^*) \geq th_R(v^*)$. Let c_ℓ and c_r be the left and right child of v^*. We can set the threshold for $feat(v^*)$ as $th_L(v^*)$ and note that, by definition and the assumption, T_{c_ℓ} is valid for E_{c_ℓ} and T_{c_r} is valid for E_{c_r}. This is a contradiction with Lemma 20 as for every node $v \in V(T)$, T_v is not valid for E_v. ◀

Now we are finally ready to prove we can efficiently compute E_v, $th_L(v)$ and $th_R(v)$ for every node $v \in V(T)$.

▶ **Lemma 22.** *Let E be a CI, let T be a DT pattern of depth at most d. Then there is an algorithm that runs in time $\mathcal{O}(2^{d^2/2} n^{1+o(1)} \log n)$ and computes the set E_v and thresholds $th_L(v)$ and $th_R(v)$ for every node $v \in V(T)$.*

Proof. The idea is to use the recursive algorithm **findLR** illustrated in Algorithm 1. That is, given E, T, the algorithm **findLR** attempts to find the triple $(E_v, th_L(v), th_R(v))$ for every node $v \in V(T)$. Lines 3 to 4: if T consists of a leaf node, the algorithm just report $(E, \texttt{nil}, \texttt{nil})$. Let c_ℓ and c_r be the left, risp. right, child of the root v. Lines 6 to 11: if the root of T is a non-fixed node, the algorithm calls itself recursively to compute on (E, T_{c_ℓ}) and (E, T_{c_r}). Lines 13 to 15: if the root of T is a fixed node v, the algorithm computes the pair (t_ℓ, t_r) for the root using the algorithm **binarySearch** and then calls itself recutsively to compute the triple for $(E[feat(v) \leq t_\ell + 1], T_{c_\ell})$ and $(E[feat(v) > t_r - 1], T_{c_r})$.

A key element for the correctness of **findLR** is the algorithm **binarySearch** illustrated in Algorithm 2. Given E, T, f, c_ℓ and c_r, this algorithm computes the pair (t_ℓ, t_r) for the root of T that has feature f. This sub-routine performs a standard binary search procedure on the array D containing all the values in $D_E(f)$ in ascending order to find maximum t_ℓ and minimum t_r such that T_{c_ℓ} and T_{c_r} can be extended to DT for $E[f \leq t_\ell]$ and for $E[f > t_r]$ respectively. To achieve this, the sub-routine makes at most $log|E|$ calls to **findTH**; note that each of those calls is made for a tree of smaller depth. Lines 3 to 12: the algorithm finds the maximum t_ℓ by calling algorithm **findTH** in Line 6 repeatedly. Lines 13 to 22: the algorithm finds the minimum t_r by calling algorithm **findTH** in Line 16 repeatedly.

A sub-routine used for **binarySearch** is the algorithm **findTH** illustrated in Algorithm 3. This algorithm is very similar to Algorithm 1 but the output is some way much simpler.

The running time of Algorithm 1 can now be obtained by multiplying the number of recursive calls to **findLR** with the time required for one recursive call. To obtain the number of recursive calls first note that if **findLR** is called with DT pattern of depth d, then it makes at most $(2\log n)+2$ recursive calls to **findLR** with a pattern of depth at most $d-1$, where $n=|E|$. Therefore the number $T(n,d)$ of recursive calls for a pattern of depth d is given by the recursion relation $T(n,d)=(2(\log n)+2)T(n,d-1)$ starting with $T(n,0)=0$. This implies that $T(n,d)\in\mathcal{O}((\log n)^d)$. Finally, the runtime for one recursive call is easily seen to be at most $\mathcal{O}(n\log n)$. Hence, the total runtime of the algorithm is at most $\mathcal{O}((\log n)^d n\log n)$, which because (see also [9, Exercise 3.18]):

$$(\log n)^d \le 2^{d^2/2}2^{\log\log d^2/2} = 2^{d^2/2}n^{o(1)}$$

is at most $\mathcal{O}(2^{d^2/2}n^{1+o(1)}\log n)$. ◀

Algorithm 1 Algorithm to compute the triple $(E_v, th_L(v), th_R(v))$ for every node $v\in V(T)$.

```
Input: CI $E$, DT pattern $T$
Output: a triple $(E_v, th_L(v), th_R(v))$ for every node $v \in V(T)$.
 1: function findLR($E$, $T$)
 2:     $r \leftarrow$ “root of $T$”
 3:     if $r$ is a leaf then
 4:         return $(E, \mathtt{nil}, \mathtt{nil})$
 5:     $c_\ell, c_r \leftarrow$ “left child and right child of $r$”
 6:     if $r$ is a non-fixed node then
 7:         $\lambda_\ell \leftarrow$ FINDLR($E$, $T_{c_\ell}$)
 8:         $\lambda_r \leftarrow$ FINDLR($E$, $T_{c_r}$)
 9:         if $\lambda_\ell \neq \mathtt{nil}$ and $\lambda_r \neq \mathtt{nil}$ then
10:             return $(E, \mathtt{nil}, \mathtt{nil}) \cup \lambda_\ell \cup \lambda_r$
11:         return nil
12:     $f \leftarrow feat(r)$
13:     $(t_\ell, t_r) \leftarrow$ BINARYSEARCH($E$, $T$, $f$, $c_\ell$, $c_r$)
14:     $\lambda_\ell \leftarrow$ FINDLR($E[f \le t_\ell + 1]$, $T_{c_\ell}$)
15:     $\lambda_r \leftarrow$ FINDLR($E[f > t_r - 1]$, $T_{c_r}$)
16:     return $(E, t_\ell, t_r) \cup \lambda_\ell \cup \lambda_r$
```

4.2 The algorithm

Now we have computed a set E_v for every node $v\in V(T)$, whether it is a leaf, fixed or non-fixed node. A *pool set* for node $v\in V(T)$ is a set $\Pi(v)\subseteq E_v$, such that if $\Pi(v)\subseteq E_T(v)$ then either

- T_v is not valid for E_v, or
- for any complete improvement T'_v for T_v that that is valid for E_v, there are two elements $e, e'\in\Pi(v)$ and there is a non-fixed node u for T such that $feat_{T'}(u)$ must distinguish e and e'.

For every node $v\in V(T)$, we define $\Pi(v)$ in a leaves-to-root fashion as follows. If v is a negative leaf then $\Pi(v)=\{e^+\}$, where e^+ is any example in $E^+\cap E_v$; similarly, if v is a positive leaf then $\Pi(v)=\{e^-\}$, where e^- is any example in $E^-\cap E_v$. Let c_ℓ and c_r be the left, resp. right, child of v, then $\Pi(v)=\Pi(c_\ell)\cup\Pi(c_r)$.

Now we want to show that the construction of Π is correct, that is:

▶ **Lemma 23.** *$\Pi(v)$ is a pool set for v for every node $v\in V(T)$.*

Algorithm 2 Algorithm to compute the pair $(th_L(r), th_R(r))$ for the root r of T

Input: CI E, DT pattern T, feature f of the root of T, left child c_ℓ of the root of T, right child c_r of the root of T

Output: maximum threshold t_ℓ in $D_E(f)$ for f such that (T_{c_ℓ}, α) can classify every example in $E[f \leq t_\ell]$ and minimum threshold t_r in $D_E(f)$ for f such that (T_{c_r}, α) can classify $E[f > t_r]$

```
 1: function binarySearch($E$, $T$, $f$, $c_\ell$, $c_r$)
 2:     $D \leftarrow$ "array containing all elements in $D_E(f)$ in
            ascending order"
 3:     $L \leftarrow 0$; $R \leftarrow |D_E(f)| - 1$; $b \leftarrow 0$
 4:     while $L \leq R$ do
 5:         $m \leftarrow \lfloor (L + R)/2 \rfloor$
 6:         if FINDTH($E[f \leq D[m]]$, $T_{c_\ell}$) = TRUE then
 7:             $L \leftarrow m + 1$; $b \leftarrow 1$
 8:         else
 9:             $R \leftarrow m - 1$; $b \leftarrow 0$
10:     if $b = 1$ then
11:         $t_\ell \leftarrow D[m]$
12:     $t_\ell \leftarrow D[m - 1]$                ▷ assuming that $D[-1] = D[0] - 1$
13:     $L \leftarrow 0$; $R \leftarrow |D_E(f)| - 1$; $b \leftarrow 0$
14:     while $L \leq R$ do
15:         $m \leftarrow \lfloor (L + R)/2 \rfloor$
16:         if FINDTH($E[f > D[m]]$, $T_{c_r}$) = TRUE then
17:             $R \leftarrow m - 1$; $b \leftarrow 1$
18:         else
19:             $L \leftarrow m + 1$; $b \leftarrow 0$
20:     if $b = 1$ then
21:         $t_r \leftarrow D[m]$
22:     $t_r \leftarrow D[m + 1]$        ▷ assuming that $D[|D_E(f)|] = D[|D_E(f)| - 1] + 1$
23:     return $(t_r, t_r)$
```

Proof. We show this by induction on the depth of T and let v be the root of T. Since $E_T(v) = E$ it is trival to note that $\Pi(v) \subseteq E_T(v)$. We start proving the base case: let T be a pattern of depth 0. Suppose v is negative leaf. Since $E_v = E$ is not uniform, there is an example $e^+ \in E^+ \cap E_v$. The case where v is a positive leaf can be proved in a symmetrical manner.

Now, let T be a pattern of depth at least one and let c_ℓ and c_r be the left and right child of v. Suppose first that v is a fixed node and let $f = feat(v)$. Thanks to Lemma 20, for every $e_\ell \in \Pi(c_\ell)$ and $e_r \in \Pi(c_r)$, we know that $f(e_\ell) < f(e_r)$. This means that either $\Pi(c_\ell) \subseteq E_T(c_\ell)$ or $\Pi(c_r) \subseteq E_T(c_r)$, say that $\Pi(c_i) \subseteq E_T(c_i)$, for $i \in \{\ell, r\}$. Since T_{c_i} has depth smaller than $T_v = T$, by the inductive hypothesis $\Pi(c_i)$ is a pool set for c_i.

Finally suppose v is a non-fixed node. Let us consider any complete improvement T'_v for T_v. For any threshold assignment for T'_v, we have one of the following three cases: either $\Pi(c_\ell) \subseteq E_{T'}(c_\ell)$ or $\Pi(c_r) \subseteq E_{T'}(c_r)$ or there is an example $e_\ell \in \Pi(c_\ell)$ and an example $e_r \in \Pi(c_r)$ such that $e_\ell \in E_{T'}(c_r)$ and $e_r \in E_{T'}(c_\ell)$. In the first two cases the statement is again proven thanks to the inductive hypothesis since T_{c_ℓ} and T_{c_r} have depth smaller than T_v. In the third case, v is a non-fixed node for T such that $feat_{T'}(v)$ distinguishes e_ℓ and e_r. ◀

In particular, let us consider the pool set $\Pi(r)$ for the root r of T, we define $\Pi(T) := \Pi(r)$. In this way given T, we are able to compute the corresponding pool set.

Algorithm 3

```
Input: CI $E$, pattern $T$
Output: TRUE if $T$ can classify all examples in $E$, FALSE otherwise
 1: function findTH($E$, $T$)
 2:     $r \leftarrow$ “root of $T$”
 3:     if $r$ is a leaf then
 4:         if $E$ is not uniform then
 5:             return FALSE
 6:         return TRUE
 7:     $c_\ell, c_r \leftarrow$ “left child and right child of $r$”
 8:     if $r$ is a non-fixed then
 9:         $\lambda_\ell \leftarrow$ FINDTH($E$, $T_{c_\ell}$)
10:         $\lambda_r \leftarrow$ FINDTH($E$, $T_{c_r}$)
11:         if $\lambda_\ell =$ TRUE and $\lambda_r =$ TRUE then
12:             return TRUE
13:         return FALSE
14:     $f \leftarrow feat(r)$
15:     $t \leftarrow$ BINARYSEARCH($E$, $T$, $f$, $c_\ell$, $c_r$)
16:     $\lambda_\ell \leftarrow$ FINDLR($E[f \leq t_\ell + 1]$, $T_{c_\ell}$)
17:     $\lambda_r \leftarrow$ FINDLR($E[f > t_r - 1]$, $T_{c_r}$)
18:     if $\lambda_r =$ FALSE then
19:         return FALSE
20:     return TRUE
```

Let S be a support set for a CI E, we stay that $B \subseteq feat(E)$ is a *branching set* for S if for every minimal DT T for E such that $S \subset feat(T)$ then $B \cap (feat(T) \setminus S) \neq \emptyset$.

▶ **Lemma 24.** *There is a $\mathcal{O}(2^{d^2/2}s^{2s+1}n^{1+o(1)}\log n)$ time algorithm that given a support set S computes a branching set R_0 for S of size at most $s^{2s+3}\delta_{\max}$.*

Proof. Let E be a CI, a support set S for E and an integer s. We start by enumerating all DT patterns T of size at most s such that $feat(T) = S \cup \{q\}$. For every such DT pattern T, thanks to Lemma 22, we are able to obtain the set E_v for every node $v \in V(T)$ in time $\mathcal{O}(2^{d^2/2}n^{1+o(1)}\log n)$. In a leaves-to-root fashion, we are able to compute the set $\Pi(v)$ for every node $v \in V(T)$ and ultimately $\Pi(T)$.

Let $R(T)$ be the set of all the features in $feat(E)\setminus S$ that distinguish at least two examples in $\Pi(T)$. The algorithm returns the set of features R_0 obtained by considering the union of the sets $R(T)$ over all these DT patterns T of size at most s. By Lemma 1 this algorithm runs in time $\mathcal{O}(2^{d^2/2}s^{2s+1}n^{1+o(1)}\log n)$.

Now we show the size of R_0 is bounded. By construction $|\Pi(T)| \leq |T| \leq s$; for every two distinct elements of $\Pi(T)$, by definition, there are at most $\delta_{\max}$ features that distinguish such two examples. This means that $|R(T)| \leq s^2\delta_{\max}$ and so R_0 has size at most $s^{2s+3}\delta_{\max}$.

We are left to show that R_0 is a branching set for S. Let T be a minimal DT for E such that $S \subset feat(T)$ and suppose by contradiction that $R_0 \cap (feat(T) \setminus S) = \emptyset$. In particular we have that $R(T) \cap (feat(T) \setminus S) = \emptyset$. This means that for every feature f of T that does not belong to S, f does not distinguish any two elements in $\Pi(T)$. By Lemma 23, $\Pi(T) = \Pi(r)$, where r is the root of T, is a pool set and so T is not valid for E, which is a contradiction. ◀

▶ **Lemma 25** ([23]). *Let E be a CI and let k be an integer. Then there is an algorithm that in time $\mathcal{O}(\delta_{\max}(E)^k|E|)$ enumerates all (of the at most $\delta_{\max}(E)^k$) minimal support sets of size at most k for E.*

▶ **Lemma 26** ([23]). *Let T be a DT of minimum size for E and let S be a support set contained in $feat(T)$. Then, the set $R = feat(T) \setminus S$ is useful.*

▶ **Observation 27** ([23]). *Let T be a DT for a CI E, then $feat(T)$ is a support set of E.*

Proof. Suppose for a contradiction that this is not the case and there is an example $e^+ \in E^+$ and an example $e^- \in E^-$ such that e^+ and e^- agree on all features in $feat(T)$. Therefore, e^+ and e^- are contained in the same leaf node of T, contradicting our assumption that T is a DT. ◀

▶ **Theorem 28** ([23]). *Let E be a CI, $S \subseteq feat(E)$ be a support set for E, and let s and d be integers. Then, there is an algorithm that runs in time $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)} \log \|E\|$ and computes a DT of minimum size among all DTs T with $feat(T) = S$ and $size(T) \leq s$ if such a DT exists; otherwise* `nil` *is returned.*

▶ **Theorem 29.** Minimum Decision Tree Size *is fixed-parameter tractable prarametrized by $\delta_{\max} + s$.*

Proof. We start by presenting the algorithm for Minimum Decision Tree Size, which is illustrated in Algorithm 4 and Algorithm 5.

Given a CI E and an integer s, the algorithm returns a DT of minimum size among all DTs of size at most s if such a DT exists and otherwise the algorithm returns `nil`. The algorithm **minDT** starts by computing the set $\mathcal{S}$ of all minimal support sets for E of size at most s, which because of Lemma 25 results in a set $\mathcal{S}$ of size at most (). In Line 4 the algorithm then interates over all sets S in $\mathcal{S}$ and calls the function **minDTS** given in Algorithm 5 for E, s, and S, which returns a DT of minimum size among all DTs T for E of size at most s such that $S \subseteq feat(T)$. It then updates the currently best decision tree B if necessary with the DT found by the function **minDTS**. Moreover, if the best DT found after going through all sets in $\mathcal{S}$ has size at most s, it is returned (in Line 9), otherwise the algorithm returns `nil`. Finally, the function **minDTS** given in Algorithm 5 does the following. It first computes a DT T of minimum size that uses exactly the features in S using Lemma 28. It then tries to improve upon T with the help of useful sets. That is, it uses Lemma 24 to compute the branching set R_0. It then interates over all (of the at most ()) features $f \in R_0$ (using the for-loop in Line 4), and calls itself recursively on the support set $S \cup \{f\}$. If this call finds a smaller DT, then the current best DT is updated. Finally, after the for-loop the algorithm either returns a solution if its size is less then s or `nil` otherwise.

Towards showing the correctness of Algorithm 4, consider the case that E has a DT of size at most s and let T be a such a DT of minimum size. Because of Observation 27, $feat(T)$ is a support set for E and therefore $feat(T)$ contains a minimal support set S of size at most s. Because the for-loop in Line 4 of Algorithm 4 interates over all minimal support sets of size at most s for E, it follows that Algorithm 5 is called with parameters E, s, and S. If $feat(T) = S$, then B is set to a DT for E of size $|T|$ in Line 2 of Algorithm 5 and the algorithm will output a DT of size at most $|T|$ for E. If, on the other hand, $feat(T) \setminus S \neq \emptyset$, then because T has minimum size and S is a support set for E with $S \subseteq feat(T)$, we obtain from Lemma 26 that the set $R = feat(T) \setminus S$ is useful for S. Therefore, because of Lemma 24, R has to contain a feature f from the set R_0 computed in Line 3. It follows that Algorithm 5 is called with parameters E, s, and $S \cup \{v\}$. From now onwards the argument repeats and since $R_0 \neq \emptyset$ the process stops after at most $s - |S|$ recursive calls after which a DT for E of size at most $|T|$ will be computed in Line 2 of Algorithm 5. Finally, it is easy to see that if Algorithm 4 outputs a DT T, then it is a valid solution. This is because, T must have been computed in Line 2 of Algorithm 5, which implies that T is a DT for E. Moreover, T has size at most s, because of Line 8 in Algorithm 4.

To analyse the run-time of the algorithm, we first remark that the whole algorithm can be seen as a bounded-depth search tree algorithm, i.e., a branching algorithm with small recursion depth and few branches at every node. In particular, every recursive call adds at least one feature to the set of features bounding the recursion depth to at most s. Moreover, every feature that is added is either added in Line 2 of Algorithm 4, when enumerating all minimal support sets, in which case there are at most $\delta_{\max}(E)$ branches or the feature is added in Line 5 of Algorithm 5, in which case there are at most $|R_0| \leq s^{2s+3}\delta_{\max}(E)$ branches. It follows that the algorithm can be seen as a branching algorithm of depth at most s with at most $s^{2s+3}\delta_{\max}(E) = \max\{s^{2s+3}\delta_{\max}(E), \delta_{\max}(E)\}$ branches at every step. Therefore, the total run-time of the algorithm is at most the number of nodes in the branching tree, i.e., at most $(s^{2s+3}\delta_{\max}(E))^s$, times the maximum time required in one recursive call. Now the maximum time required for one recursive call is dominated by the time spend in Line 2 of Algorithm 5, i.e., the time required to compute a DT of minimum size using exactly the features in S with the help of Theorem 28, which is at most $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)} \log \|E\|$. Therefore, we obtain $(s^{2s+3}\delta_{\max}(E))^s 2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)} \log \|E\|$ as the total run-time of the algorithm, which shows that DTS is fixed-parameter tractable parameterized by $s + \delta_{\max}(E)$. ◀

Algorithm 4 Main method for finding a DT of minimum size.

```
Input: CI $E$ and integer $s$
Output: DT for $E$ of minimum size (among all DTs of size at most $s$) if such a DT exists, otherwise
    nil
 1: function minDT($E$, $s$)
 2:     $\mathcal{S} \leftarrow$ "set of all minimal support sets for $E$ of size at most $s$ using Lemma 25”
 3:     $B \leftarrow$ nil
 4:     for $S \in \mathcal{S}$ do
 5:         $T \leftarrow$ MINDTS($E$, $s$, $S$)
 6:         if ($T \neq$ nil) and ($B =$ nil or $|B| > |T|$) then
 7:             $B \leftarrow T$
 8:     if $B \neq$ nil and $|B| \leq s$ then
 9:         return $B$
10:     return nil
```

5 Approximation

For every natural number $k \geq 1$, we can define a CI E_k as follows. Let E_k be the CI with exactly 2^k examples $\{e_1, \dots, e_{2^k}\}$ on k binary features $\{f_1, \dots, f_k\}$: there is exactly one example for every of the 2^k feature assignments. An example $e \in exam(E_k)$ is a positive example if $|\{f \in feat(E_k) \mid f(e) = 1\}|$ is even and negative otherwise.

Let D_k be the set of all the examples e in $exam(E_k)$ such that $f_i(e) = 1$ for every $i \in [k-2]$ and denote by $\overline{D_k}$ the set $exam(E_k) \setminus D_k$. Now we are ready to define a new feature f^* as follows: $f^*(e) = 1$ if e is a positive example or $e \in D_k$ and $f^*(e) = 0$ otherwise.

6 Conclusion

We have initiated the study of the parametrized complexity of learning DTs from data. Our main tractability result provides novel insights into the structure of DTs and is based on

Algorithm 5 Method for finding a DT of minimum size using at least the features in a given support set S.

```
Input: CI $E$, integer $s$, support set $S$ for $E$ with $|S| \leq s$
Output: DT of minimum size among all DTs $T$ for $E$ of size at most $s$ such that $S \subseteq feat(T)$; if no
    such DT exists, nil
 1: function minDTS($E$, $s$, $S$)
 2:     $B \leftarrow$ "compute a DT of minimum size for $E$ using exactly the features in $S$ using Theorem ??"
 3:     $R_0 \leftarrow$ "compute the branching set $R_0$ for $S$ using Lemma 24"
 4:     for $f \in R_0$ do
 5:         $T \leftarrow$ minDTS($E$, $s$, $S \cup \{f\}$)
 6:         if $T \neq$ nil and $|T| < |B|$ then
 7:             $B \leftarrow T$
 8:     if $|B| \leq s$ then
 9:         return $B$
10:     return nil
```

the NLC-width parameter that seems to be well suited to measure the complexity of input instances for the problem.

The problem of learning DTs comes in many variants and flavors, which opens up a wide range of new research directions to explore. For instance:

- What other (structural) parameters can be exploited to efficiently learn DTs? Is learning DTs of small size fixed-parameter tractable parameterized by the rank-width of $G_I(E)$?
- Instead of learning DTs of small size, one often wants to learn DTs of small height. Therefore, it is natural to ask whether our approach can be also used in this setting. While one can adapt our approach to obtain an XP-algorithm for learning DTs of small height parameterized by NLC-width, it is not clear to us whether the problem also allows for an fpt-algorithm.
- Can we extend our approach to CIs, where features range over an arbitrary domain? In this case, one usually still uses DTs that make binary decisions (i.e. whether a feature is smaller equal or larger than a given threshold). While it is relatively easy to see that our approach can be extended if the domain's size (for every feature) is bounded or used as an additional parameter, it is not clear what happens if the size of the domain is allowed to grow arbitrarily.

References

1 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, second edition, 2009.

2 Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, pages 173–187, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Logical analysis of data: classification with justification. *Ann. Oper. Res.*, 188(1):33–61, 2011.

5 Endre Boros, Vladimir Gurvich, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Decomposability of partially defined Boolean functions. *Discr. Appl. Math.*, 62(1-3):51–75, 1995.

6 Endre Boros, Takashi Horiyama, Toshihide Ibaraki, Kazuhisa Makino, and Mutsunori Yagiura. Finding essential attributes from binary data. *Ann. Math. Artif. Intell.*, 39:223–257, 11 2003. doi:10.1023/A:1024653703689.

7 Endre Boros, Toshihide Ibaraki, and Kazuhisa Makino. Variations on extending partially defined Boolean functions with missing bits. *Information and Computation*, 180(1):53–70, 2003.

8 Yves Crama, Peter L. Hammer, and Toshihide Ibaraki. Cause-effect relationships and partially defined Boolean function. *Ann. Oper. Res.*, 16:299–326, 1988.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2020.

11 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.

12 Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017. URL: http://arxiv.org/abs/1702.08608, arXiv:1702.08608.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer Verlag, 2013.

14 Wolfgang Espelage, Frank Gurski, and Egon Wanke. Deciding clique-width for graphs of bounded tree-width. *J. Graph Algorithms Appl.*, 7(2):141–180, 2003.

15 Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-making and a “right to explanation”. *AI Magazine*, 38(3):50–57, 2017.

16 Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

17 Toshihide Ibaraki, Yves Crama, and Peter L. Hammer. *Partially defined Boolean functions*, page 511–563. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.

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

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

20 E.J. McCluskey. *Introduction to the Theory of Switching Circuits*. Electrical and electronic engineering series. Princeton University series. McGraw-Hill, 1965.

21 Don Monroe. AI, explain yourself. *AI Communications*, 61(11):11–13, 2018. doi:10.1145/3276742.

22 Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Min. Knowl. Discov.*, 2(4):345–389, 1998. doi:10.1023/A:1009744630224.

23 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learning. In *Proceedings of AAAI'21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 6454–6462. AAAI Press, 2021.

24 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1), 2008.

25 J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi:10.1023/A:1022643204877.

26 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International*

Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

27 Richard Stanley and Eric W. Weisstein. Catalan number, from mathworld–a wolfram web resource, 2015.

28 Egon Wanke. k-NLC graphs and polynomial algorithms. *Discr. Appl. Math.*, 54(2-3):251–266, 1994. Efficient algorithms and partial k-trees.