

<sup>1</sup> **Fixed-Parameter Tractability of**  
<sup>2</sup> **Learning Small Decision Trees**  
<sup>3</sup> **(full paper)**

<sup>4</sup> **Anonymous author**

<sup>5</sup> **Anonymous affiliation**

<sup>6</sup> —— **Abstract** ——

---

<sup>7</sup> We consider the NP-hard problem of finding a smallest decision tree which represents a given partially  
<sup>8</sup> defined Boolean formula. We establish fixed-parameter tractability of the problem with respect to  
<sup>9</sup> the NLC-width of the instance. We formulate a dynamic programming procedure which utilizes  
<sup>10</sup> the NLC-decomposition of the instance. For this to work, we establish a succinct representation  
<sup>11</sup> of partial solutions, so that the space and time requirements of each dynamic programming step  
<sup>12</sup> remain bounded in terms of the NLC-width.

<sup>13</sup> **2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms →  
<sup>14</sup> Parameterized complexity and exact algorithms → Fixed parameter tractability

<sup>15</sup> **Keywords and phrases** parameterized complexity, NLC-width, rank-width, decision trees, partially  
<sup>16</sup> defined Boolean formulas

17    **1    Introduction**

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

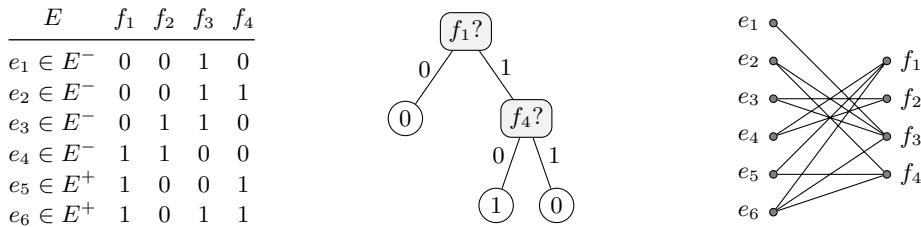
30    Because of their simplicity, decision trees are particularly attractive for providing in-  
 31    terpretable models of the underlying CI, an aspect whose importance has been strongly  
 32    emphasized over the recent years [10, 12, 15, 19, 21]. In this context, one prefers *small trees*,  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

37    MINIMUM DECISION TREE SIZE (DTS): given a CI  $E = E^+ \uplus E^-$  and an integer  $s$ ,  
 38    is there a decision tree with at most  $s$  nodes for  $E$ ?

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

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

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



■ **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).

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

## 67 2 Preliminaries

### 68 2.1 Parameterized Complexity

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

### 74 2.2 Graphs and NLC-width

75 We will assume that the reader is familiar with basic graph theory (see, e.g. [11, 1]). We  
 76 consider (vertex and edge labelled) undirected graphs. Let  $G = (V, E)$  be an undirected  
 77 graph. We write  $V(G) = V$  and  $E(G) = E$  for the sets of vertices and edges of  $G$ , respectively.  
 78 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  
 79  $G[V']$  denote the graph induced by the vertices in  $V'$ , i.e.  $G[V']$  has vertex set  $V'$  and edge  
 80 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  $E' \subseteq E$   
 81 of edges we let denote  $G - E'$  the graph with vertex set  $V$  and edge set  $E \setminus E'$ .

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

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

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

( $G_b, \lambda_b$ ) is obtained from the disjoint union of  $(G_l, \lambda_l)$  and  $(G_r, \lambda_r)$  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  $R : [k] \rightarrow [k]$  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  $(G_b, \lambda_b)$  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.

### 2.3 Classification Problems

An *example*  $e$  is a function  $e : \text{feat}(e) \rightarrow \{0, 1\}$  defined on a finite set  $\text{feat}(e)$  of *features*. For a set  $E$  of examples, we put  $\text{feat}(E) = \bigcup_{e \in E} \text{feat}(e)$ . We say that two examples  $e_1, e_2$  *agree* on a feature  $f$  if  $f \in \text{feat}(e_1)$ ,  $f \in \text{feat}(e_2)$  and  $e_1(f) = e_2(f)$ . If  $f \in \text{feat}(e_1)$ ,  $f \in \text{feat}(e_2)$  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  $\text{feat}(e_1) = \text{feat}(e_2)$ . 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  $F \subseteq \text{feat}(E)$  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  $(E, \text{feat}(E))$  having an edge between an example  $e \in E$  and a feature  $f \in \text{feat}(e)$  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  $\text{feat}(v)$ , 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  $\text{feat}(T) = \{v \in V(T) \mid \text{feat}(v)\}$ .

Consider a CI  $E$  and a decision tree  $T$  with  $\text{feat}(T) \subseteq \text{feat}(E)$ . 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  $e(\text{feat}(v)) = 0$  ( $e(\text{feat}(v)) = 1$ , 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  $E$ .

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

MINIMUM DECISION TREE SIZE (DTS)

142 Input: A classification instance  $E$  and an integer  $s$ .

Question: Is there a decision tree of size at most  $s$  for  $E$ ?

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

144 ▶ **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.

145 **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  $\text{feat}(f)$  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  $sC_s a^s \leq s4^s a^s \leq a^{2s+1}$  many DTs of size at most  $s$  using only features in  $A$  and those can be enumerated in  $\mathcal{O}(a^{2s+1})$  time. ◀

161 ▶ **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.

162 **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  $a^{2(2^a+1)+1} = a^{2^{a+1}+3}$ . ◀

163 ▶ **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  $\mathcal{O}((2^{|E|})^{4|E|-1})$ .

164 **Proof.** Note first that  $|\text{feat}(E)| \leq 2^{|E|}$  since we can assume that  $E$  does not contain two equivalent features. Moreover,  $E$  has a DT if and only if  $\text{feat}(E)$  is a support set, which can be checked in time  $\mathcal{O}(|E|^2 |\text{feat}(E)|)$  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. ◀

### 177 3 An FPT-Algorithm for NLC-width

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

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

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

todo: Due to proposition ...

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

### 192 3.1 Description of the Main Ideas Behind the Algorithm

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

- 197 ► **Observation 6.(1)**  $e(f) = 0$  for every forgotten example  $e \in \text{exam}(B_b) \setminus \text{exam}(b)$  and  
198 future feature  $f \in \text{feat}(E) \setminus \text{feat}(B_b)$ ,  
199 (2)  $e(f) = 0$  for every future example  $e \in E \setminus \text{exam}(B_b)$  and forgotten feature  $f \in \text{feat}(B_b) \setminus$   
200  $\text{feat}(b)$ ;

todo: adjust to NLC-width

201 **Proof.** Towards showing (1), let  $e$  be an example in  $\text{exam}(B_b) \setminus \text{exam}(b)$  and let  $f$  be a  
202 feature in  $\text{feat}(E) \setminus \text{feat}(B_b)$ . We claim that because  $(T, \chi)$  is a tree decomposition of  $G_I(E)$ ,  
203 the graph  $G_I(E)$  cannot contain an edge between  $e$  and  $f$ , which implies that  $e(f) = 0$ .  
204 Suppose for a contradiction that this is not the case, i.e.  $\{e, f\} \in E(G_I(E))$ . Then, because  
205 of property (T1) of a tree decomposition, there must exist a node  $b'$  such that  $e, f \in \chi(b')$ .  
206 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  
207 that  $e \notin \chi(b')$  since otherwise  $e$  would violate property (T2) of a tree decomposition. This  
208 completes the proof for (1); the proof for (2) is analogous. ◀

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

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

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

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

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

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

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

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

- 269 ■ whether it uses a future or a forgotten feature and

270 ■ how it behaves w.r.t. the examples in  $\text{exam}(b)$ .

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

### 273 3.2 Formal Definition of Records and Preliminary Results

274 In the following, let  $E$  be a CI and let  $B$  be a  $k$ -NLC-expression tree for  $G_I(E)$ . Consider a  
 275 node  $b$  of  $B$ . Recall that  $b$  is either a leaf node associated with a  $k$ -graph  $i(v)$ , a relabelling  
 276 node with 1 child and with relabelling function  $R_b$ , or a join node with a left child, a right  
 277 child and a join matrix  $M_b$ . Moreover, recall that  $(G_b, \lambda_b)$  is the  $k$ -graph associated with  $b$   
 278 (whose unlabeled version is a subgraph of  $G$ ) and  $V_b$  is the set of vertices of  $G_b$ . Additionally,  
 279 we will use the following notation. We denote by  $\text{feat}(b)$  the set  $V_b \cap \text{feat}(E)$  of features in  
 280  $V_b$  and by  $\text{exam}(b)$  the set  $V_b \cap E$  of examples in  $V_b$ .

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

288 Let  $T$  be a decision tree and  $t \in V(T)$ . We say that a node  $t_A$  is a *left/right ancestor*  
 289 of  $t$  if  $t$  is contained in the subtree of  $T$  rooted at the left/right child of  $t_A$ . We denote by  
 290  $\text{anc}_L(t)/\text{anc}_R(t)$  the set of all left/right ancestors of  $t$  in  $T$ . We denote by  $\text{anc}(t)$  the set of  
 291 all *ancestors* of  $t$  in  $T$ , i.e.,  $\text{anc}(t) = \text{anc}_L(t) \cup \text{anc}_R(t)$ .

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

296 We say that  $T$  is a *decision tree* for  $b$ , if  $T$  is a decision tree for  $\text{exam}(b)$  that uses only  
 297 the features in  $\text{feat}(b)$ . We say that an inner node  $t \in V(T)$  is *left/right redundant* in  $T$  if  
 298  $\text{feat}(t) \in \text{feat}(\text{anc}_L(t))/\text{feat}(t) \in \text{feat}(\text{anc}_R(t))$ . We say that  $t$  is redundant if it is either left  
 299 redundant or right redundant. Intuitively, a node  $t$  is left/right redundant if all examples  
 300 that end up at  $t$ , i.e., the examples  $E_T(t)$ , go the left/right child of  $t$  in  $T$ . Therefore, if  $t$   
 301 is left/right redundant in  $T$ , then the tree obtained after left/right-contracting  $t$  is still a  
 302 decision tree.

303 We say that  $T$  is a *decision tree template* for  $b$  if  $T$  is a decision tree for  $\text{exam}(b)$  that can  
 304 additionally use the future features in  $I_{[k]}$ . Here, we assume that a future feature  $f_{L'} \in I_{[k]}$   
 305 for some  $L' \subseteq [k]$  is 1 at an example  $e \in \text{exam}(b)$  if  $\lambda_b(e) \in L'$  and otherwise it is 0. We say  
 306 that a decision tree template is *complete* if it does not use any features in  $I_{[k]}$ , otherwise  
 307 we say that it is *incomplete*. Informally, the role of the future features in a decision tree  
 308 template is provide spaceholders for the features in  $\text{feat}(E) \setminus \text{feat}(b)$ . Because all of those  
 309 features behave the same w.r.t. to examples in  $\text{exam}(b)$  having the same label, they can  
 310 be charactericed by the set of labels for which those features are 1. Let  $T$  be a decision  
 311 tree template for  $b$  and let  $t \in V(T)$ . We denote by  $A(t)$  the set of *filtered labels* for  $t$ , i.e.,  
 312  $A(t) = (\bigcap_{f_{L'} \in \text{feat}(\text{anc}_L(t)) \cap I_{[k]}} \overline{L'}) \cap (\bigcap_{f_{L'} \in \text{feat}(\text{anc}_R(t)) \cap I_{[k]}} L')$ . Informally,  $A(t)$  is the set of all  
 313 labels  $l \in [k]$  such that an example  $e$  with label  $l$  would end up at  $t$ , if only the effect of  
 314 the future features on the path to  $t$  is considered. We say that  $t$  with  $f_{L'} = \text{feat}(t) \in I_{[k]}$  is  
 315 *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

definition of new  
features

316 left-redundant or right-redundant. Intuitively,  $t$  is left/right redundant if all examples that  
 317 can reach  $t$  (considering the influence of the future features only) end up in the left/right  
 318 child of  $t$ . This also implies that if  $t$  is left/right redundant then the decision tree obtained  
 319 after left/right contracting  $t$  is equivalent with  $T$  (all examples end up in the same leaves).

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

330 Let  $T$  be a decision tree skeleton and  $t \in V(T)$ . Similarly as we did for decision tree  
 331 templates, we say that  $T$  is *complete* if it uses no future features and otherwise we say that it  
 332 is incomplete. We say that an inner node  $t$  with  $f_l = \text{feat}(t) \in F_{[k]}$  is *left/right redundant* in  
 333  $T$  if  $f_l \in \text{feat}(\text{anc}_L(t)) / f_l \in \text{feat}(\text{anc}_R(t))$ . Similarly, as for decision tree (templates), if  $t$  is  
 334 left/right redundant, then we can left/right contract  $t$  without changing the properties of  $T$ .

335 Let  $T$  be a decision tree (skeleton/template). Then, we denote by  $r(T)$  the decision tree  
 336 obtained from  $T$  after left/right contracting every left/right redundant node of  $T$ . Note that  
 337 if  $T$  is a decision tree (skeleton/template) for  $b$ , then so is  $r(T)$ .

338 ▶ **Observation 7.** *Let  $T$  be a decision tree skeleton/template for  $b$ . Then, so is  $r(T)$ .*

a short proof

339 **Proof.**

340 We say that  $T$  is *reduced* if  $r(T) = T$ .

341 ▶ **Lemma 8.** *Let  $T$  be a reduced decision tree (skeleton/template) using at most a real  
 342 features,  $b$  forgotten features, and  $c$  future features. Then,  $T$  has size at most ?.*

todo

343 **Proof.**

definition of  
344 relabelling

345 Let  $T$  be a decision tree. A *feature relabelling* for  $T$  is a function  $\alpha : F' \rightarrow \text{feat}(E) \cup$   
 346  $F_L \cup I_L$ , where  $F' \subseteq \text{feat}(T)$  and  $L$  is some set of labels (usually  $L = [k]$ ). With a  
 347 slight abuse of notation, we denote by  $\alpha(T)$ , the decision tree obtained after relabeling all  
 348 features in  $F'$  (used by  $T$ ) according to  $\alpha$ , i.e.,  $\alpha(T)$  is obtained from  $T$  after replacing the  
 349 feature assignment function  $\text{feat}_T(t)$  for  $T$  with the function  $\text{feat}_{\alpha(T)}(t)$  defined by setting  
 350  $\text{feat}_{\alpha(T)}(t) = \alpha(\text{feat}_T(t))$  if  $\text{feat}(t) \in F'$  and  $\text{feat}_{\alpha(T)}(t) = \text{feat}_T(t)$ , otherwise. We say that  
 351 two feature relabellings  $\alpha_1 : F_1 \rightarrow \text{feat}(E) \cup F_L \cup I_L$  and  $\alpha_2 : F_2 \rightarrow \text{feat}(E) \cup F_L \cup I_L$  are  
 352 *compatible* if they agree on their shared domain  $F_1 \cap F_2$ .

353 We denote by  $\alpha_b^s$  the *standard feature relabelling* for  $b$ , i.e., the function  $\alpha_b^s : \text{feat}(b) \rightarrow [k]$   
 354 defined by setting  $\alpha_b^s(f) = \lambda_b(f)$  for every  $f \in \text{feat}(b)$ .

Semantics of  
records

355 We are now ready to define the records and their semantics. A *record* for  $b$  is a pair  $(T, s)$   
 356 such that  $T$  is a reduced decision tree skeleton for  $b$  and  $s$  is a natural number. We say that a  
 357 record  $(T, s)$  is *valid* for  $b$  if  $s$  is the minimum number such that there is a (reduced) decision  
 358 tree template  $T'$  for  $b$  such that  $r(\alpha_b^s(T')) = T$  and  $s = |V(T') \setminus V(T)|$ . We denote by  $\mathcal{R}(b)$   
 the set of all valid records for  $b$ . The following corollary follows immediately from Lemma 8.

359 ▶ **Corollary 9.**  $|\mathcal{R}(b)| \leq ?$

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

362 ▶ **Lemma 10.** *Let  $T$  be a decision tree and let  $\alpha$  be a feature relabelling for  $T$ . Then,  
363  $r(\alpha(T)) = r(\alpha(r(T)))$ .*

auxiliary  
properties  
of  
feature  
relabelings  
and reductions

364 ▶ **Observation 11.** *Let  $T$  be a decision tree and let  $\alpha_1$  and  $\alpha_2$  be two compatible feature  
365 relabelling for  $T$ . Then,  $\alpha_1\alpha_2(T) = \alpha_2\alpha_1(T)$ .*

### 366 3.3 Proof to the Main Result

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

369 ▶ **Lemma 12** (leaf node). *Let  $b \in V(B)$  be a leaf node. Then  $\mathcal{R}(b)$  can be computed in time  
370 ??.*

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

377 If, on the other hand  $v$  is an example, then  $\mathcal{R}(b)$  contains all records  $(T, 0)$  such that  $T$   
378 is a reduced decision tree skeleton for  $b$  using only the features in  $I_{[k]}$  and which correctly  
379 classify  $v$ . Because of Lemma ??, those can be enumerated in time  $\mathcal{O}(?)$  and checking for  
380 each of those whether it correctly classifies  $v$  can be achieved in time  $\mathcal{O}(?)$ .

381 ◀ todo: show correctness

382 ▶ **Lemma 13** (join node). *Let  $b \in V(B)$  be a join node. Then  $\mathcal{R}(b)$  can be computed in time  
383  $\mathcal{O}(k(2k + 2^k + 2)2^{6k+1})$ .*

384 **Proof.** Let  $b_L$  and  $b_R$  be the left and right child of  $b$  in  $B$ , respectively.

385 Let  $M_b$  be the join matrix for the node  $b$ , i.e.,  $M_b$  is a  $k \times k$  binary matrix. For every  
386 label  $i \in [k]$ , let  $A_{i,*} = \{j \in [k] \mid M_b[i, j] = 1\}$  and  $A_{*,i} = \{j \in [k] \mid M_b[j, i] = 1\}$ .

387 To distinguish between forgotten features from the left and the right subtree, we introduce  
388 the left  $i_L$  and the right version  $i_R$  for every label  $i \in [k]$ . With a slight abuse of notation,  
389 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  
390 set  $\{1_R, \dots, k_R\}$  of (right) labels.

391 To compute the set  $\mathcal{R}(b)$  of valid record for  $b$ , we first enumerate all reduced DT skeletons  
392  $T$  using features in  $[k_L] \cup [k_R] \cup I_{[k]}$ . Because of Lemma 17, those can be enumerated in time  
393  $\mathcal{O}((2k + 2^k + 2)2^{3k+1})$ .

394 For every such reduced DT skeleton  $T$ , we now do the following in order to decide whether  
395  $T$  gives rise to a valid record for  $b$ . Let  $\alpha^{LR \rightarrow} : F_{[k_L]} \cup F_{[k_R]} \rightarrow F_{[k]}$  be the feature relabeling  
396 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  
397 original feature  $f_i$ .

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

402 Similarly, let  $\alpha^R : F_{[k_L]} \rightarrow I_{[k]}$  be the feature relabeling that relabels every forgotten  
403 feature  $f_{i_L} \in F_{[k_L]}$  to the future feature  $f_{A_{i,*}}$ . Let  $T_R$  be the reduced DT skeleton obtained

404 from  $T$  after applying the relabelling using  $\alpha^R$  followed by  $\alpha^{LR \rightarrow}$  and then reducing the  
 405 resulting DT skeleton, i.e.,  $T_R = r(\alpha^{LR \rightarrow}(\alpha^R(T)))$ .

406 Let  $\hat{T} = \alpha^{LR \rightarrow}(T)$  and  $\hat{s} = |V(T) \setminus V(\hat{T})|$ . We now check whether there are records  
 407  $(T_L, s_L) \in \mathcal{R}(b_L)$  and  $(T_R, s_R) \in \mathcal{R}(b_R)$ . If not we discard  $T$  and if yes, then we add the  
 408 record  $(\hat{T}, s_L + s_R + \hat{s})$  to  $\mathcal{R}(b)$ . This completes the description about how the records  
 409  $\mathcal{R}(b)$  are computed. Moreover, the run-time for computing  $\mathcal{R}(b)$  can be obtained as follows.  
 410 First, because of Lemma 17, we can enumerate all reduced DT skeletons  $T$  in time  $\mathcal{O}((2k +$   
 411  $2^k + 2)2^{3k+1})$ . Moreover, computing  $\hat{T}$  and  $\hat{s}$  can be done in time  $\mathcal{O}(|T|) = \mathcal{O}(s)$ . Finally,  
 412 computing  $T_L$  and  $T_R$  and checking the existence of the records  $(T_L, s_L) \in \mathcal{R}(b_L)$  and  
 413  $(T_R, s_R) \in \mathcal{R}(b_R)$  can be achieved in time  $\mathcal{O}(?)$ . Therefore, we obtain  $\mathcal{O}(?)$  as the total  
 414 run-time for computing  $\mathcal{R}(b)$ .

old run-time  
argument below<sup>415</sup>  
should be replaced  
above<sup>416</sup>  
417

We now show the correctness of our construction for  $\mathcal{R}(b)$ , i.e., we have to show that a record  $(T, s)$  is valid if and only if we have added such a record according to our construction above.

418 Towards showing the forward direction, suppose that  $(\hat{T}, s)$  is a valid record in  $\mathcal{R}(b)$ .  
 419 Therefore, there is a DT template  $T'$  for  $b$  such that  $\hat{T} = r(\eta_{\alpha^s_b}(T'))$  and  $s = |V(T') \setminus V(T)|$ .

420 Because  $\hat{T}$  is obtained from  $T'$  by reduction, every node in  $\hat{T}$  corresponds to a unique  
 421 node in  $T'$ . Therefore, there is an injective function  $z_H : V(\hat{T}) \rightarrow V(T')$  mapping every  
 422 node in  $\hat{T}$  to its original node in  $T'$ . Let  $T$  be the DT obtained from  $\hat{T}$  after by setting  
 423  $feat_T(t) = i_H$  if  $feat_{\hat{T}}(t) = i$  and  $feat_{T'}(t) \in feat(b_H)$  for  $H \in \{L, B\}$ .

424 Note that  $\hat{T} = \eta_{\alpha^{LR \rightarrow}}(T)$  and  $\hat{T}$  is reduced because  $(\hat{T}, s) \in \mathcal{R}(b)$ .

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

428 Note that  $T = r(\eta_{\alpha^{\rightarrow L}}(\eta_{\alpha^s_{b_L}}(\eta_{\alpha^{\rightarrow R}}(\eta_{\alpha^s_{b_R}}(T')))))$ .

429 Let  $T_L = r(\eta_{\alpha^L}(T))$  and  $T_R = r(\eta_{\alpha^R}(T))$ . It remains to show that there are  $s_L$  and  $s_R$   
 430 with  $s = s_L + s_R$  such that  $(T_L, s_L) \in \mathcal{R}(b_L)$  and  $(T_R, s_R) \in \mathcal{R}(b_R)$ .

431 Let  $T'_L = r(\eta_{\alpha^L}(\eta_{\alpha^{\rightarrow R}}(\eta_{\alpha^s_{b_L}}(T'))))$  and  $T'_R = r(\eta_{\alpha^R}(\eta_{\alpha^{\rightarrow L}}(\eta_{\alpha^s_{b_L}}(T'))))$ .

432 Note that  $T_L = r(\eta_{\alpha^s_{b_L}}(T'_L))$  because of Lemma ?? and the observation that  $\eta_{\alpha^s_{b_L}} \circ \eta_{\alpha^L} \circ$   
 433  $\eta_{\alpha^{\rightarrow R}} \circ \eta_{\alpha^s_{b_R}} = ?$ .

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

- 436 ■  $\hat{T}$  is reduced and  $\hat{T} = \eta_{\alpha^{LR \rightarrow}}(T)$ ,
- 437 ■  $T_L = r(\eta_{\alpha^L}(T))$  and  $(T_L, s_L) \in \mathcal{R}(b_L)$ ,
- 438 ■  $T_R = r(\eta_{\alpha^R}(T))$  and  $(T_R, s_R) \in \mathcal{R}(b_R)$ .

439 Let  $T'_L$  be the reduced DT template for  $b_L$  such that  $T_L = r(\eta_{\alpha^s_{b_L}}(T'_L))$  and  $s_L =$   
 440  $|V(T'_L) \setminus V(T_L)|$ , which exists because  $(T_L, s_L) \in \mathcal{R}(b_L)$ . Similarly, let  $T'_R$  be the reduced  
 441 DT template for  $b_R$  such that  $T_R = r(\eta_{\alpha^s_{b_R}}(T'_R))$  and  $s_R = |V(T'_R) \setminus V(T_R)|$ , which exists  
 442 because  $(T_R, s_R) \in \mathcal{R}(b_R)$ .

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

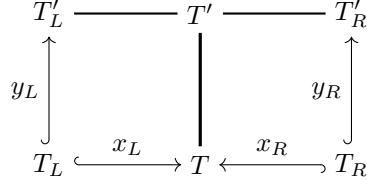
todo:show  
minimality here<sup>446</sup>  
maybe it can be  
done using the  
forward direction<sup>448</sup>

446 Suppose that there is a reduced DT template  $T'$  for  $b$  such that  $\hat{T} = r(\alpha^s_b(T'))$  and  
 447  $|V(T') \setminus V(\hat{T})| < s_L + s_R$ .

448 Informally, we obtain  $T'$  from  $T$  after reversing the relabelling and reduction operations  
 449 applied to  $T'_L$  and  $T'_R$  to obtain  $T_L$  and  $T_R$ , respectively; recall that  $T_H = r(\eta_{\alpha^s_{b_H}}(T'_H))$  for

450  $H \in \{L, R\}$ . That is, we will reverse the labelling for the nodes in  $T$  and add back the nodes  
 451 to  $T$  that have been removed from  $T'_L$  and  $T'_R$ .

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



■ **Figure 2**

457 Our first order of business is to rename all forgotten features in  $T$  to their real features  
 458 as given by  $T'_L$  and  $T'_R$ . That is, for every node  $t$  in  $T$  assigned to a forgotten feature, i.e.,  
 459  $\text{feat}(t) \in F_{[k_L]} \cup F_{[k_R]}$ , we do the following. If  $\text{feat}(t) \in F_{[k_H]}$  for  $H \in \{L, R\}$ , then  $t$  is also  
 460 in  $T_H$  and hence also in  $T'_H$ . Therefore, we can change  $\text{feat}(t)$  to the real feature assigned to  
 461  $t$  in  $T'_H$ . Let  $T^0$  be the DT obtained from  $T$  after renaming all forgotten features to real  
 462 features in this manner.

todo: explain

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

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

471 Consider an edge  $e = (p, c)$  in  $T_R$  such that  $p$  is the parent of  $c$  in  $T_R$ . Then,  $e$  corresponds  
 472 to a path  $P'_R(e)$  between  $y_R(p)$  and  $y_R(c)$  in  $T'_R$ . Similarly,  $e$  corresponds to a path  $P_R(e)$   
 473 between  $x_R(p)$  and  $x_R(c)$  in  $T^1$ . Similarly to above, we now add all nodes to  $T^1$  that have  
 474 been removed when going from  $T'_R$  to  $T_R$  (via the reduction  $r(\eta_{\alpha_{T'_R}^s}(T'_R))$ ). To achieve this,  
 475 we go over every edge  $e = (p, c)$  of  $T_R$  such that  $p$  is the parent of  $c$  in  $T_R$  and plugin the  
 476 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  
 477 obtained from  $T^1$  after doing this operation for every edge of  $T_R$ .

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

480 We start by showing that  $\hat{T} = r(\eta_{\alpha_b^s}(T'))$ . Because  $\hat{T} = \alpha_b^s(T^0)$ , it suffices to show that  
 481 the only nodes removed from  $T'$  are the ones that we added to  $T^0$  to obtain  $T'$ . Or in other  
 482 words, we need to show that only the nodes that are redundant in  $\eta_{\alpha_b^s}(T')$  are the nodes in  
 483  $V(T') \setminus V(T^0)$ .

484 Consider a node  $t \in V(T') \setminus V(T^0)$ , i.e.,  $t$  is a node that we added to  $T^0$  to obtain  $T'$ .  
 485 Then,  $t \in V(T'_H) \setminus V(T_H)$  for some  $H \in \{L, R\}$ . Because  $T_H = r(\eta_{\alpha_{b_H}^s}(T'_H))$ ,  $t$  is redundant  
 486 in  $\eta_{\alpha_{b_H}^s}(T'_H)$ , because of some node  $t' \in V(T_H)$  with  $\alpha_{b_H}^s(\text{feat}_{T'_H}(t)) = \alpha_{b_H}^s(\text{feat}_{T'_H}(t'))$ . Since  
 487  $t' \in V(T_H)$  also  $t' \in V(T')$  and therefore  $t$  is also redundant in  $\eta_{\alpha_b^s}(T')$  (because of  $t'$ ), as  
 488 required.

489 Now consider a node  $t \in V(T^0)$  and assume for a contradiction that  $t$  is redundant in  
 490  $\alpha_b^s(T')$  because of some node  $t' \in V(T')$  with  $\alpha_b^s(\text{feat}_{T'}(t)) = \alpha_b^s(\text{feat}_{T'}(t'))$ . Then, because  
 491  $\hat{T} = \alpha_b^s(T^0)$  is reduced, we obtain that  $t' \in V(T') \setminus V(T^0)$ . Therefore,  $t' \in V(T'_H) \setminus V(T_H)$   
 492 for some  $H \in \{L, R\}$ . But then,  $t'$  is redundant in  $\eta_{\alpha_b^s}(T'_H)$  because of some node  $t'' \in V(T_H)$   
 493 with  $\alpha_b^s(\text{feat}_{T'}(t'')) = \alpha_b^s(\text{feat}_{T'_H}(t'))$ , which implies that also  $t$  is redundant in  $\hat{T}$  because of  
 494  $t''$  a contradiction to our assumption that  $\hat{T}$  is reduced. This shows that  $\hat{T} = r(\eta_{\alpha_b^s}(T'))$ .  
 495 Moreover, because  $|V(T^0)| = |V(\hat{T})|$  and  $|V(T') \setminus V(T^0)| = s_L + s_R$ , it also follows that  
 496  $s_L + s_R = |V(T') \setminus V(\hat{T})|$ .

proof of  
minimality of  $T'$  497  
still missing 498

496 Moreover,  $V(T) \setminus Im(x_H)$  and  $V(T'_H) \setminus Im(y_H)$  can be partitioned into subtrees that  
 have been deleted after the application of  $r \circ p_*$ ,  $r \circ p'_*$  on  $T$  or of the standard reduction  
 499 on  $T'_H$ : let  $X_H^*$  and  $Y_H^*$  be the set of roots of the above subtrees in  $V(T) \setminus Im(x_H)$  and  
 500  $V(T'_H) \setminus Im(y_H)$  respectively. In addition, for every element  $y \in Y_H^*$ , let  $Y_y^H$  be the maximal  
 501 subtree of  $T'_H$  rooted at  $y$  with no elements from  $Im(y_H)$  and that does not contain any  
 502 vertex from  $Y_H^* \setminus \{y\}$ ; let  $(Y_y^H, S_y^H)$  the corresponding single pair. In a similar way, for every  
 503 element  $x \in X_H^*$ , let  $X_x^H$  be the maximal subtree of  $T$  rooted at  $x$  with no elements from  
 504  $Im(x_H)$  and that does not contain any vertex from  $X_H^* \setminus \{x\}$ ; let  $(X_x^H, S_x^H)$  the corresponding  
 505 single pair. Finally, for every  $y \in Y_H^*$ , let  $P_y^H$  be the shortest downwards path in  $T'_H$  that  
 506 contains  $y$  and with both endpoints in  $Im(y_H)$ , say  $y_H(t)$  and  $y_H(t')$ .

507 *Claim 1:* For every  $H \in \{L, R\}$  and for every  $y, y' \in Y_H^*$ , the paths  $P_y^H$  and  $P_{y'}^H$  are either  
 508 edge disjoint or  $P_y^H = P_{y'}^H$ .

509 *Proof.* If  $P_y^H$  and  $P_{y'}^H$  are edge disjoint, then the statement is proven immediately. Suppose  
 510  $P_y^H$  and  $P_{y'}^H$  share an edge. By minimality and the fact they are downwards paths,  $P_y^H$  and  
 511  $P_{y'}^H$  share the endpoint towards the root. If they also share the other endpoint, then the  
 512 statement is proven immediately. Suppose now their endpoints towards the leaves is different,  
 513 say  $w$  and  $w'$ , and consider the last edge those paths have in common in a root-to-leaf order,  
 514 say  $uv$ .

515 Without loss of generality, we can assume  $w$  belongs to the left branch of  $v$  and  $w'$  belongs  
 516 to the right branch of  $v$ . Note that  $v \in V(T'_H) \setminus Im(y_H)$ , or we get a contradiction due the  
 517 minimality of  $P_y^H$ . Now we get the following contradiction: by construction,  $w$  and  $w'$  are  
 518 both elements of  $Im(y_H)$  but at least one of them must be in  $V(T'_H) \setminus Im(y_H)$  since it is an  
 519 element of either  $Y_y^H$  or of  $Y_{y'}^H$ . This proves Claim 1.

520 Now for every  $y \in Y_H^*$  we consider the path  $Q_y^H$  in  $T$  having endpoints  $x_H(t)$  and  $x_H(t')$ .

521 Now we are able to describe how to obtain a witness  $T'$  of  $T$  for  $b$ . For every  $y \in Y_L^*$ , in  
 522 the last edge of path  $Q_y^L$  we plug in the single pair  $(Y_{y'}^L, S_{y'}^L)$  rooted at  $y'$ , for every internal  
 523 node  $y'$  of  $P_y^L$ , in the order the nodes  $y'$  appear in  $P_y^L$ . Note that, in the case an element  
 524 of  $Y_L^*$  is present in more than one  $P_y^L$ , we plug in the corresponding single pair only once.  
 525 Note also that whenever we plug in some single pair  $(Y_y^L, S_y^L)$  in a DT, the tree  $Y_y^L$  has real  
 526 features and future features as nodes. Call this graph  $T^*$ . Now we do the same sequence of  
 527 plug ins of the single pairs corresponding to the internal vertices of  $P_y^R$  in the last edge of  
 528 the path  $Q_y^R$ . Again, in the case an element of  $Y_R^*$  is present in more than one  $P_y^R$ , we plug  
 529 in the corresponding single pair only once. Call the tree obtained in this way  $T'$ . Node that  
 530  $T'$  contains real features from  $\text{feat}(b_L)$  and from  $\text{feat}(b_R)$  and future features with labels in  
 531  $\mathcal{P}([k])$ .

532 To conclude this part of the proof we have to show two things: (i)  $T$  is obtained from  $T'$   
 533 after removing  $s$  vertices; (ii)  $T'$  is a real DT for  $b$ . We start proving (i): by construction  $T'$   
 534 is obtained from  $T$  after adding  $s_L$  elements from  $T'_L$  and  $s_R$  elements from  $T'_R$ , and so with  
 535  $s_L + s_R = s$  more elements.

536 Before considering statement (ii), we consider the following relabelling  $p_+$  of  $T'$ : every  
 537 real feature in  $feat(b_R)$  is assigned to a feature with its label at node  $b_R$  and every other  
 538 feature is assigned to itself. The real DT  $T'_L$  can be obtained from  $T'$  by the application of  
 539 the composition  $r \circ p_* \circ p_+$ .

540 Now we consider statement (ii). We show that given an example  $e \in exam(b_L)$ ,  $e$  is  
 541 correctly classified by  $T'$  and to do so we show that  $e$  ends in a leaf of  $T'$  that corresponds  
 542 to the leaf where  $e$  ends in  $T'_L$ . Say that  $e$  goes along a path  $P$  of  $T'_L$  from the root to a  
 543 leaf  $\ell$  and let  $Q$  be the corresponding path in  $T'$ , i.e. the path from  $r$  to  $\ell$  (note that by  
 544 construction  $\ell$  is present in  $T'$  and is still a leaf). Let  $v$  be a node of  $Q$ , we can have the  
 545 following different cases.

- 546   ■  $v$  is a real feature from  $feat(b_L)$ :  $v$  is also present in  $T'_L$  as real feature;
- 547   ■  $v$  is a real feature from  $feat(b_R)$ :  $v$  might not be present in  $T'_L$  due reductions but if it is  
       present it is a future feature  $A_i$  for some  $i \in [k]$ ;
- 549   ■  $v$  is a future feature  $f_A$ :  $v$  might not be present in  $T'_L$  due reductions but if it is present  
       it is still the same future feature  $A_i$ .

551 If  $v$  is present in  $T'_L$  then the behaviour of  $v$  on  $e$  in  $T'_L$  and in  $T'$  is the same. Suppose  
 552 now  $v$  is a node of  $Q$  that is being reduced due his label and so it is not present in  $T'_L$ .  
 553 This means there is a set of ancestors of  $v$  such that their labels allows to remove  $v$  and by  
 554 construction  $v$  behaves on  $e$  like those ancestors. This proves  $e$  goes along  $Q$  and in particular  
 555 it ends at leaf  $\ell$  and so  $T'$  is a real DT for  $b_L$ . With symmetric construction, we show that  
 556  $T'$  is also a real DT for  $b_R$ .

557 Now we prove the backward direction. Let  $T$  be a reduced DT such that  $s$  is the minimum  
 558 number of elements that have been deleted from a witness  $T'$  of  $T$  for  $b$ . In particular, we  
 559 recall that  $T'$  is a real DT for  $b$  with actual feature labels in  $[k] \cup [k']$  and future feature  
 560 labels in  $\mathcal{P}([k])$ .

561 We create at real DT  $T'_L$  by the application of the composition  $r \circ p_* \circ p_+$  to  $T'$ . By  
 562 assumption  $T'$  is a real DT for  $b_L$  and by construction  $T'_L$  is a real DT for  $b_L$ . Denote  
 563 with  $T_L$  the DT template obtained from  $T'_L$  by standard reduction and denote with  $s_L$   
 564 the number of nodes that have been deleted from  $T'_L$  to obtain  $T$ . By induction we have  
 565  $(T_L, s_L) \in \mathcal{R}(b_L)$ . Now we note that  $T_L$  is obtained from  $T$  after the application of the  
 566 composition  $r \circ p_*$ . In a symmetric way, we construct  $T'_R$ ,  $T_R$  and the record  $(T_R, s_R) \in \mathcal{R}(b_R)$ .  
 567 Then  $(T, s_L + s_R) \in \mathcal{R}(b)$ . ◀

568   « « « < HEAD

569 ▶ **Lemma 14** (relabel node). *Let  $b \in V(B)$  be relabel node. Then  $\mathcal{R}(b)$  can be computed in  
 570 time  $\mathcal{O}(k(2k + 2^k + 2)2^{3k+1})$ .*

571 **Proof.** Let  $b_C$  be the unique child of  $b$  in  $B$ . Let  $R$  be the mapping of  $[k]$  to itself that  
 572 represent the node  $b$ . Moreover, since we are considering a nice NLC-expression we can  
 573 assume  $R$  is the identity mapping, i.e.  $R(\ell) = \ell$ , for all values except for a unique element  $i$   
 574 of its domain, i.e.  $R(i) = j$  for some  $j \in [k] \setminus \{i\}$ .

575 We say that a future feature  $A$  is *good* if it does not distinguish between  $i$  and  $j$ , that  
 576 is  $i \in A$  if and only if  $j \in A$ , and *bad* otherwise. Let  $(T_C, s_C)$  be an element of  $\mathcal{R}(b_C)$ . Let  
 577  $p''$  the following relabelling of the DT template  $T_C$ : every feature with label  $i$  is assigned  
 578 to label  $j$  and every future feature with label  $A$  is assigned to the future feature with label  
 579  $A \setminus \{i\}$ .

580     If  $T_C$  has a bad future feature then we do not take any other action. Suppose now  $T_C$   
 581     has only good future features; now let  $T$  be the DT template obtained from  $T_C$  after the  
 582     application of the composition  $r \circ p''$  and let  $s^*$  be the number of nodes that have been  
 583     deleted from  $T_C$  to  $T$ .

584     If there is a record in  $\mathcal{R}(b)$  of the form  $(T, s')$  for some integer  $s' \leq s_C + s^*$  then we do  
 585     not take any other action. If there is a record in  $\mathcal{R}(b)$  of the form  $(T, s')$  for some integer  
 586      $s' > s_C + s^*$  then we replace it with  $(T, s_C + s^*)$ . If there is no record in  $\mathcal{R}(b)$  of the form  
 587      $(T, s')$  for some integer  $s'$  then we add  $(T, s_C + s^*)$  to  $\mathcal{R}(b)$ .

588     Now we want to evaluate the running time of computing  $\mathcal{R}(b)$ . Consider record  $(T_C, s_C)$   
 589     in  $\mathcal{R}(b_C)$ . In  $\mathcal{O}(k)$  time we check if  $T_C$  all the future features are good. For every such DT  
 590      $T_C$ , there are at most  $2^{2k}$  paths from the root to the leaves and for every of these paths there  
 591     are at most  $k$  nodes for each of the following: feature with label  $i$  and and future feature  
 592     that contains  $i$ . This means  $r \circ p''$  can be done in  $\mathcal{O}(k)$  time. This means to compute  $\mathcal{R}(b)$   
 593     takes  $\mathcal{O}(k|\mathcal{R}(b_C)|) = \mathcal{O}(k(2k + 2^k + 2)2^{3k+1})$  time.

594     Now we have to show the correctness of the construction for  $\mathcal{R}(b)$ , i.e.  $(T, s) \in \mathcal{R}(b)$  if  
 595     and only if  $s$  is the minimum number of elements that have been deleted from a witness  $T'$   
 596     of  $T$  for  $b$ .

597     We start with the forward direction. Let  $(T, s) \in \mathcal{R}(b)$ . By construction there exists a  
 598     record  $(T_C, s_C) \in \mathcal{R}(b_C)$  such that  $T$  is obtained from  $T_C$  after the application of  $r \circ p''$  and  
 599     let  $s^* = s - s_C$ . By induction  $s_C$  is the minimum amount of nodes that have been deleted  
 600     from a witness  $T'_C$  of  $T_C$  for  $b_C$ . By construction we also know that every future feature of  
 601     both  $T'_C$  and  $T_C$  is good.

602     Denote with  $T'$  the real DT obtained  $T'_C$  after the application of  $r \circ p''$ : note that this  
 603     last reduction does not any node since every future feature of  $T'_C$  is good and there is no  
 604     feature with label  $i$ . To conclude this part of the proof we have to show two things: (i)  $T$  is  
 605     obtained from  $T'$  after removing  $s$  vertices; (ii)  $T'$  is a witness of  $T$  for  $b$ .

606     Before proving (i), we describe how  $T$  can be obtained from  $T'$ . Let  $p'''$  be the following  
 607     relabelling of  $T'$ : every real feature that contains  $j$  is assigned to the real feature  $A \cup \{i\}$   
 608     and every other feature is assigned to itself. Then the application of the composition  $p'''$ ,  
 609     the standard reduction and  $r \circ p''$  to  $T'$  is exactly the standard reduction for  $T'$  which then  
 610     result to the DT template  $T$ . By Lemma 15 the score of the standard reduction from  $T'$  to  
 611      $T$  is exactly  $s_C + s^* = s$ .

612     Now we consider statement (ii). First note that  $\text{exam}(b) = \text{exam}(b_C)$ . We show that  
 613     a given example  $e \in \text{exam}(b)$  is correctly classified by  $T'$ . Say that  $e$  goes along a path  $P$   
 614     of  $T'_C$  from the root to a leaf  $\ell$ . We show  $e$  goes along the path  $P$  in  $T'$  as well: every real  
 615     feature has not changed and so  $e$  behaves the same. Since every future feature of  $T'_C$  is good,  
 616     then  $e$  behave the same on the corresponding future feature of  $T'$ .

617     Now we prove the backward direction. Let  $T$  be a reduced DT such that  $s$  is the minimum  
 618     number of elements that have been deleted from a witness  $T'$  of  $B$  for  $b$ . In particular, we  
 619     recall that real  $T'$  is a DT for  $b$  with real features and future feature labels in  $\mathcal{P}([k] \setminus \{i\})$ .

620     We create the real DT  $T'_C$  as the application of  $r \circ p'''$  to  $T'$ , the DT template  $T_C$  as the  
 621     application of the standard reduction to  $T'_C$ . By construction we have  $(T_C, s_C) \in \mathcal{R}(b_C)$ ,  
 622     where  $s_C$  is the number of nodes that have been removed from  $T'_C$  to  $T_C$ . Note that  $T_C$  has  
 623     only good future features. Finally we note that  $T$  is obtained from  $T_C$  by the application of  
 624      $r \circ p''$ . ◀

**625    3.4   Formal Definition of Records and Preliminary Results**

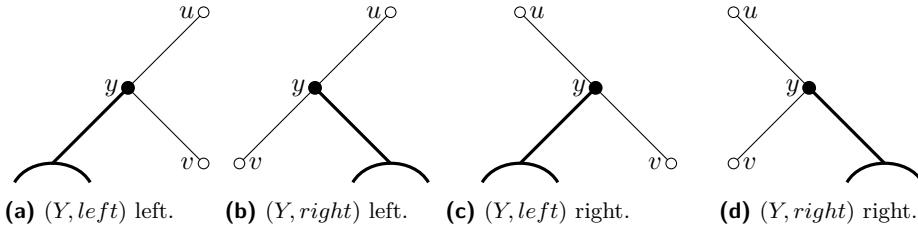
**626    =====**



627 **NLC-width**

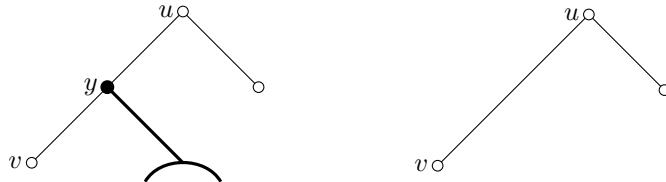
628 »»»> e150fdde332112fd1c2acb6bd85a9a5606b79547 We start off with some definitions. We  
 629 say an edge is a *left* (*right*) *edge* of a subcubic rooted tree if it connects a non-leaf node with  
 630 his left (resp. right) child. Let  $Y$  be a rooted subcubic tree and  $S \in \{\text{left}, \text{right}\}$ , then we  
 631 say the pair  $(Y, S)$  is a *single pair* if the root of  $Y$  has at most one child and the side  $S$   
 632 indicates whether the edge from the root is either a left or right edge. Moreover, we say that  
 633  $(Y, S)$  is single pair in a subcubic rooted tree  $T$  if  $Y$  is a maximal subtree of  $T$  and in  $Y$  the  
 634 root have at most the  $S$  child. Note that when tree of a single pair is made of just a node,  
 635 the side is not relevant.

636 Now we can define two operations on subcubic rooted trees and single pairs. We say that  
 637 we *plug in* a single pair  $(Y, S)$  in a left (right) edge  $uv$  as follows: we make the root  $y$  of  $Y$  the  
 638 left (right) child of  $u$ ,  $Y \setminus \{y\}$  to be the  $S$  subtree of  $y$  and  $v$  to be the  $H \in \{\text{left}, \text{right}\} \setminus S$   
 639 child of  $y$ . See Figure 3 for the corresponding drawings. Note after a plug in of a single pair  
 640 in an edge, the node  $v$  belongs in the same side of the subtree rooted at  $u$  as it was before  
 641 the plug in.



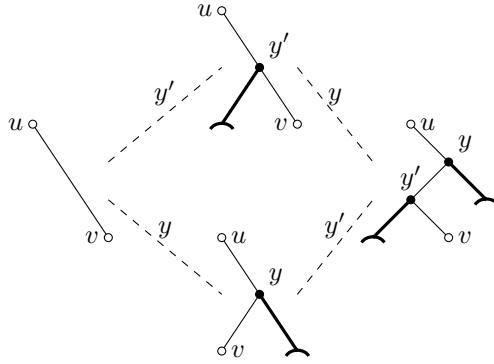
■ **Figure 3** The drawings describe the plug in operation in the different four cases. The bold part highlight the single pair  $(Y, S)$ .

642 Let  $(Y, S)$  be a single pair in a rooted subcubic tree  $T$ , then we *remove*  $(Y, S)$  from  $T$  as  
 643 follows. Let  $y$  be the root of  $Y$ . If  $y$  is the root of  $T$ , then we obtain an empty tree. If  $y$  is a  
 644 leaf node of  $T$ , then we obtain  $T - y$ . Otherwise let  $y$  be a non-root and non-leaf node, let  $u$   
 645 be the parent of  $y$  and  $v$  be the child of  $y$  that is not in  $V(Y)$ , then we consider the tree  
 646 obtained from  $T$  after replacing  $y$  with  $v$  as the child of  $u$  and deleting  $Y$ . See Figure 4 for  
 647 an example.



■ **Figure 4** The drawing describe an example of the remove operation: a single pair  $(Y, \text{right})$  is removed from a subcubic rooted tree. The bold part highlight the single pair  $(Y, S)$ .

648 It is clear from the four different plug in cases that if we want to plug in two pairs  $(Y, S)$   
 649 and  $(Y', S')$  on an edge  $uv$  such that the ancestor-descendant relationship is given, say  $y$  of  
 650  $Y$  has to be in the path from the root to  $y'$  of  $Y'$ , then we can do these plug ins in any order  
 651 but with some care. It is the same if we first plug in  $(Y, S)$  in the edge  $uv$  and then plug in  
 652  $(Y', S')$  in the edge  $vy$  or if we first plug in  $(Y', S')$  in the edge  $uv$  and then plug in  $(Y, S)$  in  
 653 the edge  $uy'$ . See Figure 5 for the an example.



■ **Figure 5** An example of plugging in two pairs  $(Y, \text{left})$  and  $(Y', \text{right})$  in a left edge  $uv$ .

654 For a subset of labels  $A \subseteq [k]$ , we define the feature template  $f_A$  by setting  $e(f_A) = 1$  if  
 655 and only if  $\text{lab}(e) \in A$  and  $e(f_A) = 0$  otherwise. With a small abuse of notation, we often  
 656 identify the feature template  $f_A$  with the corresponding subset of labels  $A$ .

657 Suppose we have a DT such that some feature label  $i$  occurs twice on a path from the  
 658 root to the leaves, say  $f_1$  is the instance closer to the root and  $f_2$  is the other instance. If  $f_2$   
 659 is in the left (resp. right) subtree of  $f_1$ , we remove  $f_2$ 's right (resp. left) subtree. In this case  
 660 we say we have done an *actual removal*.

661 Suppose we have a feature template labelled  $A$  in our decision tree. Let  $A_1, \dots, A_\ell$  be the  
 662 sequence of feature templates on the path from the root to  $A$  in order (not including  $A$ ). Let  
 663  $A'_i = A_i$  if  $A$  is in the right sub-tree of  $A_i$  and let  $A'_i = \overline{A_i}$  otherwise. If  $\overline{A} \subseteq A'_1 \cup \dots \cup A'_\ell$ ,  
 664 then we remove the subtree rooted at the left child of  $A$ . If  $A \subseteq \overline{A'_1} \cup \dots \cup \overline{A'_\ell}$ , then we  
 665 remove the subtree rooted at the right child of  $A$ . In this case we say we have done a *template*  
 666 *removal*. If this procedure has been applied to a record exhaustively, we say that the DT is  
 667 *reduced*.

668 To be short, for a DT  $T$  and a node  $v$ , we write  $v \in T$  instead of  $v \in V(T)$  and  $v \notin T$   
 669 otherwise. In a DT  $T$  we say that path  $p$  is a *downward* path if it is contained in a  
 670 path having the root as endpoint.

671 We now formally define two important operations. Given a DT  $T$ , we say that we *reduce*  
 672  $T$  if we exhaustively do actual removals and template removals. Call  $r(T)$  the resulting DT.

673 Recall that in any DT  $T$ , every non-leaf node  $v$  has one of the following three contents:  $v$   
 674 is a real feature (without label), or  $v$  is a feature with a label, or  $v$  is a future feature with  
 675 the corresponding subset of labels. A *relabelling*  $p$  for  $T$  is an assignment of contents of  $T$   
 676 as follows. Every feature is assigned to a feature with either future, real or with a label.  
 677 We say that we *relabel* the DT  $T$  via the relabelling  $p$  if for every node of  $T$  we apply the  
 678 corresponding assignment and call  $p(T)$  the resulting DT.

679 The following lemma shows that, after repeatedly applying it the necessary amount of  
 680 times, to obtain a reduced DT after a sequence of relabels, it is safe to reduce at the end.

681 ▶ **Lemma 15** (Relabelling Lemma). *Let  $T$  be a DT and  $p$  be relabelling of  $T$ . Then  $(r \circ p \circ r)(T) = (r \circ p)(T)$ .*

683 **Proof.** For every  $v \in T$ , we want to prove  $v \in (r \circ p \circ r)(T) \Leftrightarrow v \in (r \circ p)(T)$ .

684  $\Rightarrow$  Suppose there is a node  $v \notin (r \circ p)(T)$ . Since  $v \in p(T)$ , there is a set of ancestors of  $v$   
 685 in  $p(T)$  that allows to remove  $v$ . Let  $A_v$  be the union of all the minimal set of ancestors of  $v$   
 686 in  $p(T)$  that allows to remove  $v$ . If  $A_v$  is a set of ancestors of  $v$  in  $T$  that allows to reduce  $v$

687 then  $v \notin r(T)$  and so  $v \notin (r \circ p \circ r)(T)$ . Otherwise let  $A'_v$  be the subset of  $A_v$  in  $(p \circ r)(T)$ .  
 688 We conclude by noting that  $A'_v$  contains one of the minimal sets  $A_v$  is composed of and so  
 689  $v \notin (r \circ p \circ r)(T)$ .

690  $\Leftarrow$  Suppose there is a node  $v \notin (r \circ p \circ r)(T)$ . If  $v \in (p \circ r)(T)$ , there exists a set  $A_v$  of  
 691 ancestors of  $v$  in  $(p \circ r)(T)$  that allows to reduce  $v$ . Then  $A_v$  is a set of ancestors of  $v$  in  $p(T)$   
 692 that allows to reduce  $v$  and so  $v \notin (r \circ p)(T)$ . If  $v \notin (p \circ r)(T)$  then  $v \notin r(T)$ : there exists a  
 693 set  $A_v$  of ancestors of  $v$  in  $T$  that allows to remove  $v$ . This means  $A_v$  is a set of ancestors of  
 694  $v$  in  $p(T)$  that allows to remove  $v$  and so  $v \notin (r \circ p)(T)$ .  $\blacktriangleleft$

695 We say that a DT  $T$  is a *real DT* if every non-leaf node is either a real feature or a future  
 696 feature, whereas it is a *DT template* if it contains no real feature.

697 Let  $B$  be a rooted subcubic tree that corresponds to a  $k$ -NLC expression of the graph  
 698  $G_I(E)$ . For  $b \in V(B)$ , we write  $feat(b)$  and  $exam(b)$  for the sets of features and examples  
 699 introduced at node  $b$ . We say that a real DT  $T$  is a DT for the node  $b$  if every real feature of  
 700  $T$  is an element of  $feat(b)$  and every example in  $exam(b)$  is correctly classified by  $T$ , i.e. if  
 701  $e \in exam(b) \cap E^+$  then  $e$  ends in a leaf with a + label and if  $e \in exam(b) \cap E^-$  then  $e$  ends  
 702 in a leaf with a - label.

703 Given a real DT  $T$  and a node  $b \in B$ , often we want to perform a very specific composition  
 704 of operations. Let  $p_b$  be the following relabelling of  $T$ : every real feature of  $T$  is assigned to  
 705 a feature with the label given by the  $k$ -NLC expression at node  $b$  and every other feature is  
 706 assigned to itself. Then the composition  $r \circ p_b$  is called the *standard reduction* of  $T$  at node  
 707  $b$ . Given a DT  $T$  and a node  $b \in B$ , it is useful to give the following relabelling  $p'_b$ : every  
 708 feature with a label is assigned to the real feature of that node. The relabelling  $p'_b$  is called  
 709 the *real relabelling* of  $T$  at node  $b$ .

710 We say that a DT template  $T$  is a DT for the node  $b$  if there exists a real DT  $T'$  for  $b$  such  
 711 that  $T$  is the standard reduction of  $T'$ . In this case we say that  $T'$  is the witness of  $T$  for  $b$ .

712  $\blacktriangleright$  **Lemma 16.** *If there are  $\ell$  features with labels and  $2^h$  future features, then every reduced  
 713 DT template has height at most  $\ell + h$ . Furthermore, every path from the root to the leaves  
 714 contains at most  $\ell$  features with label and at most  $h - 1$  future features.*

715 **Proof.** Consider a path  $P$  of maximum length from the root to the leaves in a reduced DT  
 716 template  $T$ . By the assumptions on  $T$ , no feature with label appears more than once on  
 717 this path: the number of these feature nodes on this path is at most  $\ell$ . Consider two future  
 718 features  $f_A$  and  $f_{A'}$  that appear in  $P$ , say  $f_A$  is the instance closer to the root. Since  $T$  is  
 719 reduced, we must have that  $\emptyset \subset A' \subset A$ . Since the label of any future feature has at most  $h$   
 720 elements, there can be at most  $h - 1$  feature template nodes on this path. The path ends  
 721 with a leaf node, so this gives a total of  $\ell + h - 1 + 1 = \ell + h$  nodes, as required.  $\blacktriangleleft$

722  $\blacktriangleright$  **Lemma 17.** *If there are  $\ell$  features with label and  $2^h$  future features, then there are at  
 723 most  $(\ell + 2^k + 2)2^{\ell+k+1}$  reduced DT templates. Furthermore, these can be enumerated in  
 724  $\mathcal{O}((\ell + 2^k + 2)2^{\ell+k+1})$ -time.*

725 **Proof.** By Lemma 16, the tree has height at most  $\ell + k$ . Each node of the decision tree could  
 726 be a feature with label, a future feature, or a leaf: at most  $\ell + 2^h + 2$  different contents. Since  
 727 there are at most  $2^{\ell+h+1}$  nodes in the tree, there are at most  $(\ell + 2^h + 2)2^{\ell+h+1}$  possible  
 728 decision trees.  $\blacktriangleleft$

729 The *semantics* for a record are defined as follows. We say that a pair  $(T, s)$  is a *record* for  
 730 the node  $b \in B$  and we write  $(T, s) \in \mathcal{R}(b)$ , if  $T$  is a DT template for  $b$  and  $s$  is the minimum  
 731 number of elements that have been deleted from a witness  $T'$  of  $T$  for  $b$ .

### 732 3.5 Proof to the Main Result

733 Now, it suffices to compute  $\mathcal{R}(b)$  via leaf-to-root dynamic programming. The following  
 734 four lemmas show how this can be achieved for all of the four types of nodes in a  $k$ -NLC  
 735 expression tree  $B$ .

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

738 **Proof.** Let  $v$  be the vertex of  $G_I(E)$  that corresponds to the leaf node  $b$ . This means either  
 739  $v \in E$  or  $v \in \text{feat}(E)$ .

740 We have to enumerate all possible reduced DT templates  $T$  for  $b$ . It is enough to consider  
 741 all reduced DT templates  $T$  of height at most  $k + 1$  and discard those that are not DT  
 742 templates for  $b$ ; these can be enumerated in time  $\mathcal{O}((2^k + 3)2^{k+2})$  by Lemma 17 and the  
 743 check can be done in time  $\mathcal{O}(k)$ . We add the pair  $(T, 0)$  to the set of records  $\mathcal{R}(b)$ .

744 Now we have to show the correctness of the construction for  $\mathcal{R}(b)$ , i.e.  $(T, s) \in \mathcal{R}(b)$  if  
 745 and only if  $s$  is the minimum number of elements that have been deleted from a witness  $T'$   
 746 of  $T$  for  $b$ .

747 We start with the forward direction. Let  $(T, s) \in \mathcal{R}(b)$ . By construction, we have that  
 748  $s = 0$  and  $T$  is a DT template for  $b$  which is already reduced. Then  $T$  is trivially a witness  
 749 of  $T$  for  $b$ .

750 Now we prove the backward direction. Let  $T$  be a reduced DT template such that  $0$   
 751 is the minimum number of elements that have been deleted from a witness  $T'$  of  $T$  for  $b$ .  
 752 This means  $T'$  is obtained from  $T$  after the real relabelling at node  $b$  is applied:  $T$  is a DT  
 753 template among the considered DTs above which leads to the fact that  $(T, 0) \in \mathcal{R}(b)$ . ◀

754 ▶ **Lemma 19** (join node). *Let  $b \in V(B)$  be a join node. Then  $\mathcal{R}(b)$  can be computed in time  
 755  $\mathcal{O}(k(2k + 2^k + 2)2^{6k+1})$ .*

756 **Proof.** Let  $b_L$  and  $b_R$  be the left, resp. right, child of  $b$  in  $B$ : we may assume the labels for  
 757  $\text{feat}(b_L)$  are in  $[k]$  and the labels for  $\text{feat}(b_R)$  are in  $[k']$ . Moreover, let  $M$  be the  $k \times k$   $\{0, 1\}$   
 758 matrix that represent the node  $b$ . Finally, for every label  $i \in [k]$ , let  $A_i = \{j \in [k] \mid M_{i,j} = 1\}$ .

759 We consider every reduced DT  $T$  for  $b$  with feature labels in  $[k] \cup [k']$  and future feature  
 760 labels in  $\mathcal{P}([k])$ ; these can be enumerated in time  $\mathcal{O}((2k + 2^k + 2)2^{3k+1})$  by Lemma 17.

761 For every such DT  $T$ , we create a DT  $T_L$  as follows. Let  $p_*$  be the following relabelling:  
 762 for every  $i' \in [k']$ , every feature with label  $i'$  is assigned to the future feature  $A_i$ . Then we  
 763 apply the composition  $r \circ p_*$  to  $T$ . In a symmetrical way we create a DT  $T_R$ . Let  $p'_*$  be the  
 764 following relabelling: for every  $i \in [k]$ , every feature with label  $i$  is assigned to the future  
 765 feature  $A_{i'}$  and every future feature  $A_i$  is assigned to the future feature  $A_{i'}$ . Then we apply  
 766 the composition  $r \circ p'_*$  to  $T$ .

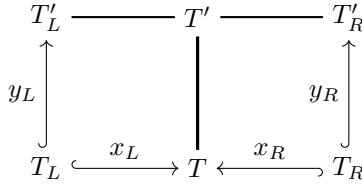
767 Now we want to understand if there is a record in  $\mathcal{R}(b_L)$  of the form  $(T_L, s_L)$  for some  
 768 positive integer  $s_L$  and if there is a record in  $\mathcal{R}(b_R)$  of the form  $(T_R, s_R)$  for some positive  
 769 integer  $s_R$ : if the answer is yes in both cases, we add a record  $(T, s_L + s_R)$  to  $\mathcal{R}(b)$ ; otherwise  
 770 we discard this option.

771 Now we want to evaluate the running time of computing  $\mathcal{R}(b)$ . Every reduced DT  $T$  can  
 772 be enumerated in time  $\mathcal{O}((2k + 2^k + 2)2^{3k+1})$  by Lemma 17. For every such DT  $T$ , there are  
 773 at most  $2^{3k}$  paths from the root to the leaves and for every of these paths there are at most  
 774  $k$  nodes for each of the following: features with label in  $[k]$ , features with label in  $[k']$  and  
 775 future features by Lemma 16. This means  $r \circ p_*$  and  $r \circ p'_*$  can be done in  $\mathcal{O}(k2^{3k})$  time.

Now we have to show the correctness of the construction for  $\mathcal{R}(b)$ . We start with the forward direction. Let  $(T, s) \in \mathcal{R}(b)$ . By construction there exist records  $(T_L, s_L) \in \mathcal{R}(b_L)$  and  $(T_R, s_R) \in \mathcal{R}(b_R)$  such that  $T_L$  and  $T_R$  are obtained by the application of  $r \circ p_*$  and  $r \circ p'_*$  respectively to  $T$  and  $s_L + s_R = s$ .

By induction, for  $H \in \{L, R\}$ , we know that  $s_H$  is the minimum number of elements that have been deleted from a witness  $T'_H$  of  $T_H$  for  $b_H$ .

For  $H \in \{L, R\}$ , we define maps  $x_H$  and  $y_H$  as follows. Let  $x_H : V(T_H) \rightarrow V(T)$  and  $y_H : V(T_H) \rightarrow V(T'_H)$  be the functions that maps every node of  $T_H$  to the corresponding node in  $T$  and in  $T'_H$  and note that by constructions both these maps are injective.



Moreover,  $V(T) \setminus Im(x_H)$  and  $V(T'_H) \setminus Im(y_H)$  can be partitioned into subtrees that have been deleted after the application of  $r \circ p_*$ ,  $r \circ p'_*$  on  $T$  or of the standard reduction on  $T'_H$ : let  $X_H^*$  and  $Y_H^*$  be the set of roots of the above subtrees in  $V(T) \setminus Im(x_H)$  and  $V(T'_H) \setminus Im(y_H)$  respectively. In addition, for every element  $y \in Y_H^*$ , let  $Y_y^H$  be the maximal subtree of  $T'_H$  rooted at  $y$  with no elements from  $Im(y_H)$  and that does not contain any vertex from  $Y_H^* \setminus \{y\}$ ; let  $(Y_y^H, S_y^H)$  the corresponding single pair. «««< HEAD ===== »»»> e150fdde332112fd1c2acb6bd85a9a5606b79547 In a similar way, for every element  $x \in X_H^*$ , let  $X_x^H$  be the maximal subtree of  $T$  rooted at  $x$  with no elements from  $Im(x_H)$  and that does not contain any vertex from  $X_H^* \setminus \{x\}$ ; let  $(X_x^H, S_x^H)$  the corresponding single pair. Finally, for every  $y \in Y_H^*$ , let  $P_y^H$  be the shortest downwards path in  $T'_H$  that contains  $y$  and with both endpoints in  $Im(y_H)$ , say  $y_H(t)$  and  $y_H(t')$ .

*Claim 1:* For every  $H \in \{L, R\}$  and for every  $y, y' \in Y_H^*$ , the paths  $P_y^H$  and  $P_{y'}^H$  are either edge disjoint or  $P_y^H = P_{y'}^H$ .

*Proof.* If  $P_y^H$  and  $P_{y'}^H$  are edge disjoint, then the statement is proven immediately. Suppose  $P_y^H$  and  $P_{y'}^H$  share an edge. By minimality and the fact they are downwards paths,  $P_y^H$  and  $P_{y'}^H$  share the endpoint towards the root. If they also share the other endpoint, then the statement is proven immediately. Suppose now their endpoints towards the leaves is different, say  $w$  and  $w'$ , and consider the last edge those paths have in common in a root-to-leaf order, say  $uv$ .

Without loss of generality, we can assume  $w$  belongs to the left branch of  $v$  and  $w'$  belongs to the right branch of  $v$ . Note that  $v \in V(T'_H) \setminus Im(y_H)$ , or we get a contradiction due the minimality of  $P_y^H$ . Now we get the following contradiction: by construction,  $w$  and  $w'$  are both elements of  $Im(y_H)$  but at least one of them must be in  $V(T'_H) \setminus Im(y_H)$  since it is an element of either  $Y_y^H$  or of  $Y_{y'}^H$ . This proves Claim 1.

Now for every  $y \in Y_H^*$  we consider the path  $Q_y^H$  in  $T$  having endpoints  $x_H(t)$  and  $x_H(t')$ .

«««< HEAD ===== *Claim 2:* For every  $H \in \{L, R\}$  and for every  $y \in Y_H^*$ , every internal vertex of  $Q_y^H$  is an element of  $X_H^*$ .

*Proof.* Suppose that  $Q_y^H$  has an internal vertex  $t \notin X_H^*$ . By definition, there exists a vertex  $v \in V(T_H)$  such that  $x_H(v) = t$ . Since  $x_H$  is injective then  $v \notin \{v_1, v_2\}$ . Since  $y_H$  is injective  $y_H(v) \notin \{y_H(v_1), y_H(v_2)\}$  and belongs to  $P_y^H$ , which contradicts the minimality of  $P_y^H$ . This proves Claim 2.

816 Before we describe how to obtain a witness  $T'$  of  $T$  for  $b$ , we must make an observation.  
 817 We note that  $Im(x_L) \cup Im(x_R) = V(T)$ : the idea is that every node of  $T$  must originate  
 818 from either  $T_L$  or  $T_R$ .

819 »»»> e150fdde332112fd1c2acb6bd85a9a5606b79547 Now we are able to describe how to  
 820 obtain a witness  $T'$  of  $T$  for  $b$ . For every  $y \in Y_L^*$ , in the last edge of path  $Q_y^L$  we plug in the  
 821 single pair  $(Y_{y'}^L, S_{y'}^L)$  rooted at  $y'$ , for every internal node  $y'$  of  $P_y^L$ , in the order the nodes  $y'$   
 822 appear in  $P_y^L$ . Note that, in the case an element of  $Y_L^*$  is present in more than one  $P_y^L$ , we  
 823 plug in the corresponding single pair only once. Note also that whenever we plug in some  
 824 single pair  $(Y_y^L, S_y^L)$  in a DT, the tree  $Y_y^L$  has real features and future features as nodes. Call  
 825 this graph  $T^*$ . Now we do the same sequence of plug ins of the single pairs corresponding to  
 826 the internal vertices of  $P_y^R$  in the last edge of the path  $Q_y^R$ . Again, in the case an element  
 827 of  $Y_R^*$  is present in more than one  $P_y^R$ , we plug in the corresponding single pair only once.  
 828 Call the tree obtained in this way  $T'$ . Node that  $T'$  contains real features from  $feat(b_L)$  and  
 829 from  $feat(b_R)$  and future features with labels in  $\mathcal{P}([k])$ .

830 To conclude this part of the proof we have to show two things: (i)  $T$  is obtained from  $T'$   
 831 after removing  $s$  vertices; (ii)  $T'$  is a real DT for  $b$ . We start proving (i): by construction  $T'$   
 832 is obtained from  $T$  after adding  $s_L$  elements from  $T'_L$  and  $s_R$  elements from  $T'_R$ , and so with  
 833  $s_L + s_R = s$  more elements.

834 Before considering statement (ii), we consider the following relabelling  $p_+$  of  $T'$ : every  
 835 real feature in  $feat(b_R)$  is assigned to a feature with its label at node  $b_R$  and every other  
 836 feature is assigned to itself. The real DT  $T'_L$  can be obtained from  $T'$  by the application of  
 837 the composition  $r \circ p_* \circ p_+$ .

838 Now we consider statement (ii). We show that given an example  $e \in exam(b_L)$ ,  $e$  is  
 839 correctly classified by  $T'$  and to do so we show that  $e$  ends in a leaf of  $T'$  that corresponds  
 840 to the leaf where  $e$  ends in  $T'_L$ . Say that  $e$  goes along a path  $P$  of  $T'_L$  from the root to a  
 841 leaf  $\ell$  and let  $Q$  be the corresponding path in  $T'$ , i.e. the path from  $r$  to  $\ell$  (note that by  
 842 construction  $\ell$  is present in  $T'$  and is still a leaf). Let  $v$  be a node of  $Q$ , we can have the  
 843 following different cases.

- 844   ■  $v$  is a real feature from  $feat(b_L)$ :  $v$  is also present in  $T'_L$  as real feature;
- 845   ■  $v$  is a real feature from  $feat(b_R)$ :  $v$  might not be present in  $T'_L$  due reductions but if it is  
   present it is a future feature  $A_i$  for some  $i \in [k]$ ;
- 847   ■  $v$  is a future feature  $f_A$ :  $v$  might not be present in  $T'_L$  due reductions but if it is present  
   it is still the same future feature  $A_i$ .

849 If  $v$  is present in  $T'_L$  then the behaviour of  $v$  on  $e$  in  $T'_L$  and in  $T'$  is the same. Suppose  
 850 now  $v$  is a node of  $Q$  that is being reduced due his label and so it is not present in  $T'_L$ .  
 851 This means there is a set of ancestors of  $v$  such that their labels allows to remove  $v$  and by  
 852 construction  $v$  behaves on  $e$  like those ancestors. This proves  $e$  goes along  $Q$  and in particular  
 853 it ends at leaf  $\ell$  and so  $T'$  is a real DT for  $b_L$ . With symmetric construction, we show that  
 854  $T'$  is also a real DT for  $b_R$ .

855 Now we prove the backward direction. Let  $T$  be a reduced DT such that  $s$  is the minimum  
 856 number of elements that have been deleted from a witness  $T'$  of  $T$  for  $b$ . In particular, we  
 857 recall that  $T'$  is a real DT for  $b$  with actual feature labels in  $[k] \cup [k']$  and future feature  
 858 labels in  $\mathcal{P}([k])$ .

859 We create a real DT  $T'_L$  by the application of the composition  $r \circ p_* \circ p_+$  to  $T'$ . By  
 860 assumption  $T'$  is a real DT for  $b_L$  and by construction  $T'_L$  is a real DT for  $b_L$ . Denote  
 861 with  $T_L$  the DT template obtained from  $T'_L$  by standard reduction and denote with  $s_L$

862 the number of nodes that have been deleted from  $T'_L$  to obtain  $T$ . By induction we have  
 863  $(T_L, s_L) \in \mathcal{R}(b_L)$ . Now we note that  $T_L$  is obtained from  $T$  after the application of the  
 864 composition  $r \circ p_*$ . In a symmetric way, we construct  $T'_R$ ,  $T_R$  and the record  $(T_R, s_R) \in \mathcal{R}(b_R)$ .  
 865 Then  $(T, s_L + s_R) \in \mathcal{R}(b)$ .  $\blacktriangleleft$

866 ▶ **Lemma 20** (relabel node). *Let  $b \in V(B)$  be relabel node. Then  $\mathcal{R}(b)$  can be computed in  
 867 time  $\mathcal{O}(k(2k + 2^k + 2)2^{3k+1})$ .*

868 **Proof.** Let  $b_C$  be the unique child of  $b$  in  $B$ . Let  $R$  be the mapping of  $[k]$  to itself that  
 869 represent the node  $b$ . Moreover, since we are considering a *nice* NLC-expression we can  
 870 assume  $R$  is the identity mapping, i.e.  $R(\ell) = \ell$ , for all values except for a unique element  $i$   
 871 of its domain, i.e.  $R(i) = j$  for some  $j \in [k] \setminus \{i\}$ .

872 We say that a future feature  $A$  is *good* if it does not distinguish between  $i$  and  $j$ , that  
 873 is  $i \in A$  if and only if  $j \in A$ , and *bad* otherwise. Let  $(T_C, s_C)$  be an element of  $\mathcal{R}(b_C)$ . Let  
 874  $p''$  the following relabelling of the DT template  $T_C$ : every feature with label  $i$  is assigned  
 875 to label  $j$  and every future feature with label  $A$  is assigned to the future feature with label  
 876  $A \setminus \{i\}$ .

877 If  $T_C$  has a bad future feature then we do not take any other action. Suppose now  $T_C$   
 878 has only good future features; now let  $T$  be the DT template obtained from  $T_C$  after the  
 879 application of the composition  $r \circ p''$  and let  $s^*$  be the number of nodes that have been  
 880 deleted from  $T_C$  to  $T$ .

881 If there is a record in  $\mathcal{R}(b)$  of the form  $(T, s')$  for some integer  $s' \leq s_C + s^*$  then we do  
 882 not take any other action. If there is a record in  $\mathcal{R}(b)$  of the form  $(T, s')$  for some integer  
 883  $s' > s_C + s^*$  then we replace it with  $(T, s_C + s^*)$ . If there is no record in  $\mathcal{R}(b)$  of the form  
 884  $(T, s')$  for some integer  $s'$  then we add  $(T, s_C + s^*)$  to  $\mathcal{R}(b)$ .

885 Now we want to evaluate the running time of computing  $\mathcal{R}(b)$ . Consider record  $(T_C, s_C)$   
 886 in  $\mathcal{R}(b_C)$ . In  $\mathcal{O}(k)$  time we check if  $T_C$  all the future features are good. For every such DT  
 887  $T_C$ , there are at most  $2^{2k}$  paths from the root to the leaves and for every of these paths there  
 888 are at most  $k$  nodes for each of the following: feature with label  $i$  and and future feature  
 889 that contains  $i$ . This means  $r \circ p''$  can be done in  $\mathcal{O}(k)$  time. This means to compute  $\mathcal{R}(b)$   
 890 takes  $\mathcal{O}(k|\mathcal{R}(b_C)|) = \mathcal{O}(k(2k + 2^k + 2)2^{3k+1})$  time.

891 Now we have to show the correctness of the construction for  $\mathcal{R}(b)$ , i.e.  $(T, s) \in \mathcal{R}(b)$  if  
 892 and only if  $s$  is the minimum number of elements that have been deleted from a witness  $T'$   
 893 of  $T$  for  $b$ .

894 We start with the forward direction. Let  $(T, s) \in \mathcal{R}(b)$ . By construction there exists a  
 895 record  $(T_C, s_C) \in \mathcal{R}(b_C)$  such that  $T$  is obtained from  $T_C$  after the application of  $r \circ p''$  and  
 896 let  $s^* = s - s_C$ . By induction  $s_C$  is the minimum amount of nodes that have been deleted  
 897 from a witness  $T'_C$  of  $T_C$  for  $b_C$ . By construction we also know that every future feature of  
 898 both  $T'_C$  and  $T_C$  is good.

899 Denote with  $T'$  the real DT obtained  $T'_C$  after the application of  $r \circ p''$ : note that this  
 900 last reduction does not any node since every future feature of  $T'_C$  is good and there is no  
 901 feature with label  $i$ . To conclude this part of the proof we have to show two things: (i)  $T$  is  
 902 obtained from  $T'$  after removing  $s$  vertices; (ii)  $T'$  is a witness of  $T$  for  $b$ .

903 Before proving (i), we describe how  $T$  can be obtained from  $T'$ . Let  $p'''$  be the following  
 904 relabelling of  $T'$ : every real feature that contains  $j$  is assigned to the real feature  $A \cup \{i\}$   
 905 and every other feature is assigned to itself. Then the application of the composition  $p'''$ ,  
 906 the standard reduction and  $r \circ p''$  to  $T'$  is exactly the standard reduction for  $T'$  which then  
 907 result to the DT template  $T$ . By Lemma 15 the score of the standard reduction from  $T'$  to  
 908  $T$  is exactly  $s_C + s^* = s$ .

909 Now we consider statement (ii). First note that  $\text{exam}(b) = \text{exam}(b_C)$ . We show that  
 910 a given example  $e \in \text{exam}(b)$  is correctly classified by  $T'$ . Say that  $e$  goes along a path  $P$   
 911 of  $T'_C$  from the root to a leaf  $\ell$ . We show  $e$  goes along the path  $P$  in  $T'$  as well: every real  
 912 feature has not changed and so  $e$  behaves the same. Since every future feature of  $T'_C$  is good,  
 913 then  $e$  behave the same on the corresponding future feature of  $T'$ .

914 Now we prove the backward direction. Let  $T$  be a reduced DT such that  $s$  is the minimum  
 915 number of elements that have been deleted from a witness  $T'$  of  $B$  for  $b$ . In particular, we  
 916 recall that real  $T'$  is a DT for  $b$  with real features and future feature labels in  $\mathcal{P}([k] \setminus \{i\})$ .

917 We create the real DT  $T'_C$  as the application of  $r \circ p'''$  to  $T'$ , the DT template  $T_C$  as the  
 918 application of the standard reduction to  $T'_C$ . By construction we have  $(T_C, s_C) \in \mathcal{R}(b_C)$ ,  
 919 where  $s_C$  is the number of nodes that have been removed from  $T'_C$  to  $T_C$ . Note that  $T_C$  has  
 920 only good future features. Finally we note that  $T$  is obtained from  $T_C$  by the application of  
 921  $r \circ p''$ . ◀

922 Now we can finally prove Theorem 4 and Theorem ??, which we restate here.

923 **Theorem 4 (restated).** *Let  $E$  be a CI, let  $(B, \chi)$  be an NLC-expression decomposition of  
 924 width  $k$  for  $G_I(E)$ , and let  $s$  be an integer. Then, deciding whether  $E$  has a DT of size at  
 925 most  $s$  is fixed-parameter tractable parameterized by  $k$ . In particular, such computation takes  
 926  $\mathcal{O}()$  time.*

927 **Proof.** We start off by computing  $\mathcal{R}(b)$  for every node  $b$  of  $B$ , via leaf-to-root dynamic  
 928 programming. An upper bound for the running time for this step is the number of nodes of  
 929  $B$  times the maximum running time to compute the record at each node which is given by  
 930 Lemmas 18, 19 and 20.

931 Now we look at the root node  $r$  of  $B$ . We go through all the records of  $\mathcal{R}(r)$  and select a  
 932 record  $(T, s) \in \mathcal{R}(r)$  such that  $|T| + s$  is minimum over all DTs with no future feature. ◀

933 **Theorem ?? (restated).** DTS is fixed-parameter tractable parameterized by NLC-width.

## 934 4 An FPT-Algorithm for bounded solution size and $\delta_{max}$ .

935 In the following, let  $E$  be a CI and  $q \notin \text{feat}(E)$ . A *pattern* is a pair  $(T, \varphi)$  where  $T = (V, A)$   
 936 is a rooted subcubic tree, every leaf-node is either a *positive* or *negative* leaf and  $\varphi$  maps every  
 937 inner-node  $v \in V(T)$  to an element of  $\text{feat}(E) \cup \{q\}$ . We say that  $\varphi$  is the *trait* function of  
 938  $T$  and  $\varphi(v)$  is the trait of node  $v \in V(T)$ . Finally we say that a node  $v \in V(T)$  is a *fixed  
 939 node* if  $\varphi(v) \in \text{feat}(E)$ .

940 A pattern  $(T, \varphi^*)$  is an *improvement* for a pattern  $(T, \varphi)$  if  $\varphi^*(v) = \varphi(v)$  for every fixed  
 941 node  $v$  of  $(T, \varphi)$ . A *complete improvement*  $(T, \varphi^*)$  for  $(T, \varphi)$  is an improvement such that  
 942  $Im(\varphi^*) \subseteq \text{feat}(E)$ . Note that any complete improvement of a pattern is a decision tree.  
 943 Given a pattern  $(T, \varphi)$ , a *threshold assignment* of  $(T, \varphi)$  is a function  $\psi$  that maps every fixed  
 944 node  $v \in V(T)$  to a rational number  $\psi(v)$ .

945 Given a threshold assignment  $\psi$  for a decision tree  $(T, \varphi)$ , for each node  $v$  of  $T$  we define  
 946 the set of examples that arrive at node  $v$ ,  $E_T(v)$  as follows:  $E_T(v)$  is the set of all examples  
 947  $e \in E$  such that for each left (right, respectively) arc  $(u, w)$  on the unique path from the  
 948 root of  $T$  to  $v$  we have  $(\varphi(u))(e) \leq \psi(u)$  ( $(\varphi(u))(e) > \psi(u)$ , respectively). A decision tree  
 949  $(T, \varphi)$  *correctly classifies* an example  $e \in E$  given  $\psi$  if  $e$  is a positive (negative) example and  
 950  $e \in E_T(v)$  for a positive (negative) leaf. We say that  $(T, \varphi)$  *classifies*  $E$  given  $\psi$  if  $T$  correctly  
 951 classifies every example  $e \in E$  given  $\psi$ .

952 We say that  $(T, \varphi)$  can classify  $E$  if there exists a complete improvement  $(T, \varphi^*)$  for  $(T, \varphi)$   
 953 and there exists a threshold assignment  $\psi$  for  $(T, \varphi^*)$  such that  $(T, \varphi^*)$  classifies  $E$  given  $\psi$ .

#### 954 4.1 Preprocess

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

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

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

- 964   ■ let  $th_L(v)$  be the maximum value in  $D_E(\varphi(v))$  such that  $(T_{c_\ell}, \varphi)$  can classify every  
 965 example in  $E_v[\varphi(v) \leq th_L(v)]$ ;
- 966   ■ let  $th_R(v)$  be the minimum value in  $D_E(\varphi(v))$  such that  $(T_{c_r}, \varphi)$  can classify every example  
 967 in  $E_v[\varphi(v) > th_R(v)]$ .

968 Before formally proving in Lemma 21 that we are able to compute  $E_v$  and  $th_L(v)$ ,  $th_R(v)$   
 969 (when  $v$  is a fixed node) for every  $v \in V(T)$ , we want to describe the role of  $E_v$  in the proof  
 970 of Lemma 22.

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

976 Let us recall a definition. Let  $(T, \varphi)$  be a pattern and  $v \in V(T)$  be an inner node of  
 977  $T$  with left child  $\ell$ , right child  $r$ , and parent  $p$ . We say that  $T'$  is obtained from  $T$  after  
 978 *left/right-contracting*  $v$  if  $T'$  is a rooted subcubic tree obtained from  $T$  after removing  $v$   
 979 together with all nodes in  $T_r/T_\ell$  and adding the edge between  $p$  and  $\ell/r$ ; if  $v$  has no parent  
 980 then no edge is added.

981 In order to argue that a pattern  $(T, \varphi)$  can classify  $E$ , we have first to compute a complete  
 982 improvement (or a series of improvements that ends up in a complete improvement) of  
 983  $(T, \varphi)$ . Let  $v$  be a non-fixed node of  $T$ . If the pattern  $(T', \varphi)$ , where  $T'$  is obtained after  
 984 left/right-contracting  $v$ , can classify every example in  $E_v$ , then we discard the pattern  $(T, \varphi)$ :  
 985  $(T', \varphi)$  is equivalent to  $(T, \varphi)$  but of smaller size. TO EXPAND.

986 **Claim 1.** *Let  $(T, \varphi)$  be a pattern. For every node  $v \in V(T)$ ,  $(T_v, \varphi)$  can not classify every  
 987 example in  $E_v$ .*

988 We prove this claim by induction on the height of  $T$ .

989 Let us first consider the case in which  $v$  is the root of  $T$ . Here we have that  $T_v = T$  and  
 990  $E_v = E$  and so the claim directly follows from the assumption that  $T$  can not be extended  
 991 to a DT for  $E$ . Now let  $v$  be the child of an unknown feature node  $v_p$ . Recall that, by  
 992 assumption,  $v_p$  does not distinguish any pair of examples and so it is safe to assume that  
 993  $E_{v_p}$  is a set of examples that can not be classified by extending  $T_v$ .

994 Finally suppose now that  $v$  is the left child of a supported feature node  $v_p$ . By the  
 995 maximality of  $th_L(v_p)$ , the set  $E_v = E_{v_p}[f_p \leq th_L(v_p) + 1]$  necessarily contains examples

996 that can not be classified by extending  $T_v$ . In a similar way, the minimality of  $th_R(v)$  shows  
 997 that the claim holds when  $v$  is the right child of a supported feature node. This proves the  
 998 Claim 1.

999 Thanks to Claim 1, it follows that for every node  $v$  that has a supported feature,  $th_L(v) <$   
 1000  $th_R(v)$ . Suppose that for some node  $v$  with supported feature  $f$  we have  $th_L(v) \geq th_R(v)$ .  
 1001 Setting  $(feat(v) = th_L(v))$  generates a DT for  $E_v$ , which is a contradiction.

1002 ▶ **Lemma 21.** *Let  $E$  be a CI, let  $T$  be a partial DT of depth at most  $d$  and let  $\bar{\alpha}$  be a  
 1003  $S$ -feature assignment of  $T$  for a support set  $S$ . Then there is an algorithm that runs in time  
 1004  $\mathcal{O}(2^{d^2/2}n^{1+o(1)}\log n)$  and computes the set  $E_v$  and thresholds  $th_L(v)$  and  $th_R(v)$  for every  
 1005 node  $v \in V(T)$ .*

1006 **Proof.** The idea is to use the recursive algorithm **findLR** illustrated in Algorithm 1. That  
 1007 is, given  $E$ ,  $T$ ,  $\bar{\alpha}$ , the algorithm **findLR** attempts to find the triples  $(E_v, th_L(v), th_R(v))$   
 1008 for every node  $v \in V(T)$ . Lines 3 to 4: if  $T$  consists of a leaf node, the algorithm just  
 1009 report  $(E, \text{nil}, \text{nil})$ . Let  $c_\ell$  and  $c_r$  be the left, resp. right, child of the root  $v$ . Lines 6  
 1010 to 11: if the root of  $T$  has an unknown feature, the algorithm calls itself recursively to  
 1011 compute the triple for  $(E, T_{c_\ell}, \alpha)$  and  $(E, T_{c_r}, \alpha)$ . Lines 13 to 15: if the root of  $T$  has a  
 1012 supported feature  $f$ , the algorithm computes the pair  $(t_\ell, t_r)$  for the root using the algorithm  
 1013 **binarySearch** and then calls itself recursively to compute the triple for  $(E[f \leq t_\ell + 1], T_{c_\ell}, \alpha)$   
 1014 and  $(E[f > t_r - 1], T_{c_r}, \alpha)$ .

1015 A key element for the correctness of **findLR** is the algorithm **binarySearch** illustrated  
 1016 in Algorithm 2. Given  $E$ ,  $T$ ,  $\alpha$ ,  $f$ ,  $c_\ell$  and  $c_r$ , this algorithm computes the pair  $(t_\ell, t_r)$  for  
 1017 the root of  $T$ . This sub-routine performs a standard binary search procedure on the array  
 1018  $D$  containing all the values in  $D_E(f)$  in ascending order to find maximum  $t_\ell$  and minimum  
 1019  $t_r$  such that  $(T_{c_\ell}, \alpha)$  and  $(T_{c_r}, \alpha)$  can be extended to DT for  $E[f \leq t_\ell]$  and for  $E[f > t_r]$   
 1020 respectively. To achieve this, the sub-routine makes at most  $\log|E|$  calls to **findTH**; note  
 1021 that each of those calls is made for a tree of smaller depth. Lines 3 to 12: the algorithm  
 1022 finds the maximum  $t_\ell$  by calling algorithm **findTH** in Line 6 repeatedly. Lines 13 to 22: the  
 1023 algorithm finds the minimum  $t_r$  by calling algorithm **findTH** in Line 16 repeatedly.

1024 A sub-routine used for **binarySearch** is the algorithm **findTH** illustrated in Algorithm 3.  
 1025 This algorithm is very similar to Algorithm 1 but the output is some way much simpler.

1026 The running time of Algorithm 1 can now be obtained by multiplying the number of  
 1027 recursive calls to **findLR** with the time required for one recursive call. To obtain the number  
 1028 of recursive calls first note that if **findLR** is called with a pseudo DT of depth  $d$ , then it  
 1029 makes at most  $(2\log n) + 2$  recursive calls to **findLR** with a pseudo DT of depth at most  
 1030  $d - 1$ , where  $n = |E|$ . Therefore the number  $T(n, d)$  of recursive calls for a pseudo DT of  
 1031 depth  $d$  is given by the recursion relation  $T(n, d) = (2(\log n) + 2)T(n, d - 1)$  starting with  
 1032  $T(n, 0) = 0$ . This implies that  $T(n, d) \in \mathcal{O}((\log n)^d)$ . Finally, the runtime for one recursive  
 1033 call is easily seen to be at most  $\mathcal{O}(n \log n)$ . Hence, the total runtime of the algorithm is at  
 1034 most  $\mathcal{O}((\log n)^d n \log n)$ , which because (see also [9, Exercise 3.18]):

$$1035 (\log n)^d \leq 2^{d^2/2} 2^{\log \log d^2/2} = 2^{d^2/2} n^{o(1)}$$

1036 is at most  $\mathcal{O}(2^{d^2/2} n^{1+o(1)} \log n)$ . ◀

## 1037 4.2 The algorithm

1038 Now we have computed a set  $E_v$  for every node  $v \in V(T)$ , whether it is a leaf or has a  
 1039 supported or unknown feature. A *pool set* for node  $v \in V(T)$  is a set  $\Pi(v) \subseteq E_v$ , such that if  
 1040 every example of  $\Pi(v)$  arrives at node  $v$  then either

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

---

**Input:** CI  $E$ , pseudo DT  $T$ ,  $S$ -feature assignment  $\alpha$  for  $T$   
**Output:** a triple  $(E_v, th_L(v), th_R(v))$  for every node  $v \in V(T)$ .

```

1: function FINDLR( $E, T, \alpha$ )
2:    $r \leftarrow$  “root of  $T$ ”
3:   if  $r$  is a leaf then
4:     return  $(E, \text{nil}, \text{nil})$ 
5:    $c_\ell, c_r \leftarrow$  “left child and right child of  $r$ ”
6:   if  $r$  has an unknown feature then
7:      $\lambda_\ell \leftarrow$  FINDLR( $E, T_{c_\ell}, \alpha$ )
8:      $\lambda_r \leftarrow$  FINDLR( $E, T_{c_r}, \alpha$ )
9:     if  $\lambda_\ell \neq \text{nil}$  and  $\lambda_r \neq \text{nil}$  then
10:      return  $(E, \text{nil}, \text{nil}) \cup \lambda_\ell \cup \lambda_r$ 
11:    return  $\text{nil}$ 
12:     $f \leftarrow \alpha(r)$ 
13:     $(t_\ell, t_r) \leftarrow$  BINARYSEARCH( $E, T, \alpha, c_\ell, c_r, f$ )
14:     $\lambda_\ell \leftarrow$  FINDLR( $E[f \leq t_\ell + 1], T_{c_\ell}, \alpha$ )
15:     $\lambda_r \leftarrow$  FINDLR( $E[f > t_r - 1], T_{c_r}, \alpha$ )
16:    return  $(E, t_\ell, t_r) \cup \lambda_\ell \cup \lambda_r$ 
```

---

- 1041 ■  $T$  can not be extended to a DT for  $E$ , or  
1042 ■ there is an node  $u \in V(T_v)$  with unknown feature  $f_u$  and two elements  $e, e' \in \Pi(v)$  such  
1043 that  $f_u$  must distinguish  $e$  and  $e'$ .

1044 For every node  $v \in V(T)$ , we define  $\Pi(v)$  in a leaves-to-root fashion as follows:

- 1045 (a) if  $v$  is a negative leaf then  $\Pi(v) = \{e^+\}$ , where  $e^+$  is any example in  $E^+ \cap E_v$ ;  
1046 (b) if  $v$  is a positive leaf then  $\Pi(v) = \{e^-\}$ , where  $e^-$  is any example in  $E^- \cap E_v$ ;  
1047 (c) if  $v$  is a supported feature node. Let  $c_\ell$  and  $c_r$  be the left, resp. right, child of  $v$ , then  
1048  $\Pi(v) = \Pi(c_\ell) \cup \Pi(c_r)$ ;  
1049 (d) if  $t$  is an unknown feature node. Let  $c_\ell$  and  $c_r$  be the left, resp. right, child of  $v$ , then  
1050  $\Pi(v) = \Pi(c_\ell) \cup \Pi(c_r)$ .

1051 Now we want to show that the construction of  $\Pi$  is correct, that is:

1052 **Claim 2.**  $\Pi(v)$  is a pool set for  $v$  for every node  $v \in V(T)$ .

1053 We start proving (a): let  $v$  be a negative leaf and suppose there is an element  $e^+ \in E^+ \cap E_v$ ,  
1054 then there is no threshold assignment for  $T$  that would correctly classify  $e$ . The correctness  
1055 for (b) is obtained by a symmetric argument. Let us consider (c) and let  $f \in S$  be the  
1056 feature at  $v$ : thanks to the preprocessing, for every  $e_\ell \in \Pi(c_\ell)$  and  $e_r \in \Pi(c_r)$ , we know that  
1057  $f(e_\ell) < f(e_r)$ . This means that either every element of  $\Pi(c_\ell)$  is sent to  $c_\ell$  or every element of  
1058  $\Pi(c_r)$  is sent to  $c_r$ : by induction the statement is proven. Finally we consider statement (d).  
1059 If every element of  $\Pi(c_\ell)$  is sent to  $c_\ell$  or every element of  $\Pi(c_r)$  is sent to  $c_r$ , the statement  
1060 is proven by induction. Otherwise, there is an example  $e_\ell \in \Pi(c_\ell)$  that ends in  $c_r$  and an  
1061 example  $e_r \in \Pi(c_r)$  that ends in  $c_\ell$ . This means  $v$  is an unknown feature node in  $T_t$  that  
1062 distinguishes two examples in  $\Pi(v)$ , namely  $e_\ell$  and  $e_r$ . This proves Claim 2.

1063 In particular, let us consider the pool set  $\Pi(r)$  for the root  $r$  of  $T$ , we define  $\Pi(T) := \Pi(r)$ .  
1064 In this way given  $T$ , we are able to compute the corresponding pool set.

1065 Let  $S$  be a support set for a CI  $E$ , we stay that  $B \subseteq \text{feat}(E)$  is a *branching set* for  $S$  if  
1066 for every minimal DT  $T$  for  $E$  such that  $S \subset \text{feat}(T)$  then  $B \cap (\text{feat}(T) \setminus S) \neq \emptyset$ .

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

---

**Input:** CI  $E$ , pseudo DT  $T$ ,  $S$ -feature assignment  $\alpha$  for  $T$ , feature  $f$  of the root of  $T$ , left child  $c_\ell$  of the root of  $T$ , right child  $c_r$  of the root of  $T$

**Output:** maximum threshold  $t_\ell$  in  $D_E(f)$  for  $f$  such that  $(T_{c_\ell}, \alpha)$  can be extended to a DT for  $E[f \leq t_\ell]$  and minimum threshold  $t_r$  in  $D_E(f)$  for  $f$  such that  $(T_{c_r}, \alpha)$  can be extended to a DT for  $E[f > t_r]$

```

1: function binarySearch( $E, T, \alpha, f, c_\ell, c_r$ )
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( $E[f \leq D[m]], T_{c_\ell}, \alpha$ ) = 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( $E[f > D[m]], T_{c_r}, \alpha$ ) = 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  $(t_\ell, t_r)$ 

```

---

1067 ▶ **Lemma 22.** 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  
1068  $S$  computes a branching set  $R_0$  for  $S$  of size at most  $s^{2s+3}\delta_{\max}$ .

1069 **Proof.** Let  $E$  be a CI, a support set  $S$  for  $E$  and an integer  $s$ . We start by enumerating all  
1070 partial DTs of size at most  $s$  such that the set of supported features is exactly  $S$ . For every  
1071 such partial DT  $T$ , thanks to Lemma 21, we are able to obtain the set  $E_v$  for every node  
1072  $v \in V(T)$  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  
1073 the set  $\Pi(v)$  for every node  $v \in V(T)$  and ultimately  $\Pi(T)$ .

1074 Let  $R(T)$  be the set of all the features in  $feat(E) \setminus S$  that distinguish at least two  
1075 examples in  $\Pi(T)$ . The algorithm returns the set of features  $R_0$  obtained by considering the  
1076 union of the sets  $R(T)$  over all these partial DTs  $T$  for  $S$  of size at most  $s$ . By Lemma 1 this  
1077 algorithm runs in time  $\mathcal{O}(2^{d^2/2} s^{2s+1} n^{1+o(1)} \log n)$ .

1078 Now we show the size of  $R_0$  is bounded. By construction  $|\Pi(T)| \leq |T| \leq s$ ; for every two  
1079 distinct elements of  $\Pi(T)$ , by definition, there are at most  $\delta_{\max}$  features that distinguish  
1080 such two examples. This means that  $|R(T)| \leq s^2\delta_{\max}$  and so  $R_0$  has size at most  $s^{2s+3}\delta_{\max}$ .

1081 We are left to show that  $R_0$  is a branching set for  $S$ . Let  $T$  be a minimal DT for  $E$  such  
1082 that  $S \subset feat(T)$  and suppose by contradiction that  $R_0 \cap (feat(T) \setminus S) = \emptyset$ . In particular we  
1083 have that  $R(T) \cap (feat(T) \setminus S) = \emptyset$ . This means that every unknown feature of  $T$  does not  
1084 distinguish any two elements in  $\Pi(T)$ . By Claim 2,  $\Pi(T) = \Pi(r)$ , where  $r$  is the root of  $T$ , is  
1085 a pool set and so  $T$  can not be extended to a DT for  $E$ , which is a contradiction. ◀

■ **Algorithm 3** Algorithm to compute the threshold assignment for a pseudo DT and a feature assignment.

---

**Input:** CI  $E$ , pseudo DT  $T$ ,  $S$ -feature assignment  $\alpha$  for  $T$   
**Output:** TRUE if all examples in  $E$  can be correctly classified by  $(T, \alpha)$ , FALSE otherwise

```

1: function findTH( $E, T, \alpha$ )
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  $r$  has an unknown feature then
9:      $\lambda_\ell \leftarrow \text{FINDTH}(E, T_{c_\ell}, \alpha)$ 
10:     $\lambda_r \leftarrow \text{FINDTH}(E, T_{c_r}, \alpha)$ 
11:    if  $\lambda_\ell = \text{TRUE}$  and  $\lambda_r = \text{TRUE}$  then
12:      return TRUE
13:    return FALSE
14:    $f \leftarrow \alpha(r)$ 
15:    $t \leftarrow \text{BINARYSEARCH}(E, T, \alpha, c_\ell, f)$ 
16:    $\lambda_r \leftarrow \text{FINDTH}(E[f > t], T_{c_r}, \alpha)$ 
17:   if  $\lambda_r = \text{FALSE}$  then
18:     return FALSE
19:   return TRUE

```

---

1086 ▶ **Lemma 23** ([23]). Let  $E$  be a CI and let  $k$  be an integer. Then there is an algorithm that  
1087 in time  $\mathcal{O}(\delta_{\max}(E)^k |E|)$  enumerates all (of the at most  $\delta_{\max}(E)^k$ ) minimal support sets of  
1088 size at most  $k$  for  $E$ .

1089 ▶ **Lemma 24** ([23]). Let  $T$  be a DT of minimum size for  $E$  and let  $S$  be a support set  
1090 contained in  $\text{feat}(T)$ . Then, the set  $R = \text{feat}(T) \setminus S$  is useful.

1091 ▶ **Theorem 25.** MINIMUM DECISION TREE SIZE is fixed-parameter tractable parametrized  
1092 by  $\delta_{\max} + s$ .

1093 **Proof.** We start by presenting the algorithm for MINIMUM DECISION TREE SIZE, which is  
1094 illustrated in Algorithm 4 and Algorithm 5.

1095 Given a CI  $E$  and an integer  $s$ , the algorithm returns a DT of minimum size among all  
1096 DTs of size at most  $s$  if such a DT exists and otherwise the algorithm returns **nil**. The  
1097 algorithm **minDT** starts by computing the set  $\mathcal{S}$  of all minimal support sets for  $E$  of size  
1098 at most  $s$ , which because of Lemma 23 results in a set  $\mathcal{S}$  of size at most  $\binom{|E|}{s}$ . In Line 4  
1099 the algorithm then iterates over all sets  $S$  in  $\mathcal{S}$  and calls the function **minDTS** given in  
1100 Algorithm 5 for  $E$ ,  $s$ , and  $S$ , which returns a DT of minimum size among all DTs  $T$  for  $E$   
1101 of size at most  $s$  such that  $S \subseteq \text{feat}(T)$ . It then updates the currently best decision tree  $B$   
1102 if necessary with the DT found by the function **minDTS**. Moreover, if the best DT found  
1103 after going through all sets in  $\mathcal{S}$  has size at most  $s$ , it is returned (in Line 9), otherwise  
1104 the algorithm returns **nil**. Finally, the function **minDTS** given in Algorithm 5 does the  
1105 following. It first computes a DT  $T$  of minimum size that uses exactly the features in  $S$   
1106 using Lemma ???. It then tries to improve upon  $T$  with the help of useful sets. That is, it  
1107 uses Lemma 22 to compute the branching set  $R_0$ . It then iterates over all (of the at most  
1108  $\binom{|S|}{1}$ ) features  $f \in R_0$  (using the for-loop in Line 4), and calls itself recursively on the feature  
1109 set  $S \cup \{f\}$ . If this call finds a smaller DT, then the current best DT  $B$  is updated. Finally,  
1110 after the for-loop the algorithm either returns  $B$  if its size is less than  $s$  or **nil** otherwise.

1111 Towards showing the correctness of Algorithm 4, consider the case that  $E$  has a DT  
 1112 of size at most  $s$  and let  $T$  be a such a DT of minimum size. Because of Observation ??,  
 1113  $\text{feat}(T)$  is a support set for  $E$  and therefore  $\text{feat}(T)$  contains a minimal support set  $S$  of size  
 1114 at most  $s$ . Because the for-loop in Line 4 of Algorithm 4 iterates over all minimal support  
 1115 sets of size at most  $s$  for  $E$ , it follows that Algorithm 5 is called with parameters  $E$ ,  $s$ , and  
 1116  $S$ . If  $\text{feat}(T) = S$ , then  $B$  is set to a DT for  $E$  of size  $|T|$  in Line 2 of Algorithm 5 and the  
 1117 algorithm will output a DT of size at most  $|T|$  for  $E$ . If, on the other hand,  $\text{feat}(T) \setminus S \neq \emptyset$ ,  
 1118 then because  $T$  has minimum size and  $S$  is a support set for  $E$  with  $S \subseteq \text{feat}(T)$ , we obtain  
 1119 from Lemma 24 that the set  $R = \text{feat}(T) \setminus S$  is useful for  $S$ . Therefore, because of Lemma 22,  
 1120  $R$  has to contain a feature  $f$  from the set  $R_0$  computed in Line 3. It follows that Algorithm 5  
 1121 is called with parameters  $E$ ,  $s$ , and  $S \cup \{v\}$ . From now onwards the argument repeats and  
 1122 since  $R_0 \neq \emptyset$  the process stops after at most  $s - |S|$  recursive calls after which a DT for  $E$  of  
 1123 size at most  $|T|$  will be computed in Line 2 of Algorithm 5. Finally, it is easy to see that if  
 1124 Algorithm 4 outputs a DT  $T$ , then it is a valid solution. This is because,  $T$  must have been  
 1125 computed in Line 2 of Algorithm 5, which implies that  $T$  is a DT for  $E$ . Moreover,  $T$  has  
 1126 size at most  $s$ , because of Line 8 in Algorithm 4.

1127 To analyse the run-time of the algorithm, we first remark that the whole algorithm can  
 1128 be seen as a bounded-depth search tree algorithm, i.e., a branching algorithm with small  
 1129 recursion depth and few branches at every node. In particular, every recursive call adds at  
 1130 least one feature to the set of features bounding the recursion depth to at most  $s$ . Moreover,  
 1131 every feature that is added is either added in Line 2 of Algorithm 4, when enumerating  
 1132 all minimal support sets, in which case there are at most  $\delta_{\max}(E)$  branches or the feature  
 1133 is added in Line 5 of Algorithm 5, in which case there are at most  $|R_0| \leq s^{2s+3}\delta_{\max}(E)$   
 1134 branches. It follows that the algorithm can be seen as a branching algorithm of depth  
 1135 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  
 1136 step. Therefore, the total run-time of the algorithm is at most the number of nodes in  
 1137 the branching tree, i.e., at most  $(s^{2s+3}\delta_{\max}(E))^s$ , times the maximum time required in  
 1138 one recursive call. Now the maximum time required for one recursive call is dominated  
 1139 by the time spend in Line 2 of Algorithm 5, i.e., the time required to compute a DT of  
 1140 minimum size using exactly the features in  $S$  with the help of Theorem ??, which is at  
 1141 most  $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$ . Therefore, we obtain  $(s^{2s+3}\delta_{\max}(E))^s 2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$   
 1142 as the total run-time of the algorithm, which shows that DTS is fixed-parameter tractable  
 1143 parameterized by  $s + \delta_{\max}(E)$ .  $\blacktriangleleft$

#### 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:    $S \leftarrow$  "set of all minimal support sets for  $E$  of size at most  $s$  using Lemma 23"
3:    $B \leftarrow$  nil
4:   for  $S \in S$  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
  
```

---

**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  $S \subseteq \text{feat}(T)$ ; 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 22”
4:   for  $f \in R_0$  do
5:      $T \leftarrow \text{MINDTS}(E, s, S \cup \{f\})$ 
6:     if  $T \neq \text{nil}$  and  $|T| < |B|$  then
7:        $B \leftarrow T$ 
8:     if  $|B| \leq s$  then
9:       return  $B$ 
10:    return nil

```

---

1144 **5 Conclusion**

1145 We have initiated the study of the parameterized complexity of learning DTs from data. Our  
1146 main tractability result provides novel insights into the structure of DTs and is based on  
1147 the NLC-width parameter that seems to be well suited to measure the complexity of input  
1148 instances for the problem.

1149 The problem of learning DTs comes in many variants and flavors, which opens up a wide  
1150 range of new research directions to explore. For instance:

- 1151   ■ What other (structural) parameters can be exploited to efficiently learn DTs? Is learning  
1152   DTs of small size fixed-parameter tractable parameterized by the rank-width of  $G_I(E)$ ?
- 1153   ■ Instead of learning DTs of small size, one often wants to learn DTs of small height.  
1154   Therefore, it is natural to ask whether our approach can be also used in this setting.  
1155   While one can adapt our approach to obtain an XP-algorithm for learning DTs of small  
1156   height parameterized by NLC-width, it is not clear to us whether the problem also allows  
1157   for an fpt-algorithm.
- 1158   ■ Can we extend our approach to CIs, where features range over an arbitrary domain? In  
1159   this case, one usually still uses DTs that make binary decisions (i.e. whether a feature is  
1160   smaller equal or larger than a given threshold). While it is relatively easy to see that our  
1161   approach can be extended if the domain’s size (for every feature) is bounded or used as  
1162   an additional parameter, it is not clear what happens if the size of the domain is allowed  
1163   to grow arbitrarily.

1164 **References**

---

- 1165   1 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics.  
1166   Springer-Verlag London Ltd., London, second edition, 2009.
- 1167   2 Christian Bessiere, Emmanuel Hebrard, and Barry O’Sullivan. Minimising decision tree size  
1168   as combinatorial optimisation. In Ian P. Gent, editor, *Principles and Practice of Constraint  
1169   Programming - CP 2009*, pages 173–187, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 1170   3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth.  
1171   *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 1172   4 Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, and  
1173   Kazuhisa Makino. Logical analysis of data: classification with justification. *Ann. Oper. Res.*,  
1174   188(1):33–61, 2011.

- 1175 5 Endre Boros, Vladimir Gurvich, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan.  
1176 Decomposability of partially defined Boolean functions. *Discr. Appl. Math.*, 62(1-3):51–75,  
1177 1995.
- 1178 6 Endre Boros, Takashi Horiyama, Toshihide Ibaraki, Kazuhisa Makino, and Mutsunori Yagiura.  
1179 Finding essential attributes from binary data. *Ann. Math. Artif. Intell.*, 39:223–257, 11 2003.  
1180 doi:10.1023/A:1024653703689.
- 1181 7 Endre Boros, Toshihide Ibaraki, and Kazuhisa Makino. Variations on extending partially  
1182 defined Boolean functions with missing bits. *Information and Computation*, 180(1):53–70,  
1183 2003.
- 1184 8 Yves Crama, Peter L. Hammer, and Toshihide Ibaraki. Cause-effect relationships and partially  
1185 defined Boolean function. *Ann. Oper. Res.*, 16:299–326, 1988.
- 1186 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin  
1187 Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.  
1188 doi:10.1007/978-3-319-21275-3.
- 1189 10 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In Giuseppe De  
1190 Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín,  
1191 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 -  
1193 Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*,  
1194 volume 325 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2020.
- 1195 11 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer  
1196 Verlag, New York, 2nd edition, 2000.
- 1197 12 Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *CoRR*,  
1198 abs/1702.08608, 2017. URL: <http://arxiv.org/abs/1702.08608>, arXiv:1702.08608.
- 1199 13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts  
1200 in Computer Science. Springer Verlag, 2013.
- 1201 14 Wolfgang Espelage, Frank Gurski, and Egon Wanke. Deciding clique-width for graphs of  
1202 bounded tree-width. *J. Graph Algorithms Appl.*, 7(2):141–180, 2003.
- 1203 15 Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-  
1204 making and a “right to explanation”. *AI Magazine*, 38(3):50–57, 2017.
- 1205 16 Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete.  
1206 *Information Processing Letters*, 5(1):15–17, 1976.
- 1207 17 Toshihide Ibaraki, Yves Crama, and Peter L. Hammer. *Partially defined Boolean functions*,  
1208 page 511–563. Encyclopedia of Mathematics and its Applications. Cambridge University Press,  
1209 2011.
- 1210 18 Daniel T. Larose. *Discovering knowledge in data*. Wiley-Interscience [John Wiley & Sons],  
1211 Hoboken, NJ, 2005. An introduction to data mining.
- 1212 19 Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*,  
1213 61(10):36–43, 2018.
- 1214 20 E.J. McCluskey. *Introduction to the Theory of Switching Circuits*. Electrical and electronic  
1215 engineering series. Princeton University series. McGraw-Hill, 1965.
- 1216 21 Don Monroe. AI, explain yourself. *AI Communications*, 61(11):11–13, 2018. doi:10.1145/  
1217 3276742.
- 1218 22 Seerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary  
1219 survey. *Data Min. Knowl. Discov.*, 2(4):345–389, 1998. doi:10.1023/A:1009744630224.
- 1220 23 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learn-  
1221 ing. In *Proceedings of AAAI'21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*,  
1222 pages 6454–6462. AAAI Press, 2021.
- 1223 24 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on  
1224 Algorithms*, 5(1), 2008.
- 1225 25 J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi:  
1226 10.1023/A:1022643204877.

- 1227   **26** Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster  
1228   parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt,  
1229   and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International  
1230   Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume  
1231   8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.  
1232   **27** Richard Stanley and Eric W. Weisstein. Catalan number, from mathworld—a wolfram web  
1233   resource, 2015.  
1234   **28** Egon Wanke.  $k$ -NLC graphs and polynomial algorithms. *Discr. Appl. Math.*, 54(2-3):251–266,  
1235   1994. Efficient algorithms and partial  $k$ -trees.