

<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].

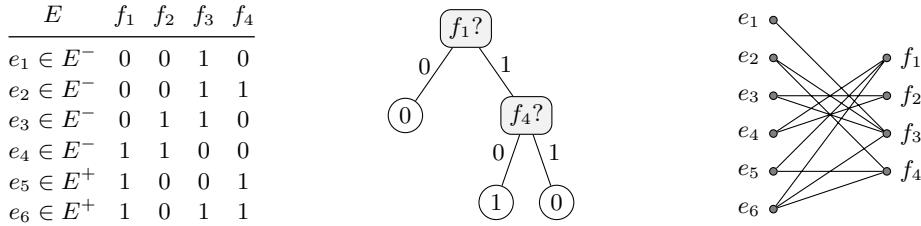
30    Because of their simplicity, decision trees are particularly attractive for providing in-  
 31    terpretable models of the underlying CI, an aspect whose importance has been strongly  
 32    emphasized over the recent years [10, 12, 15, 19, 21]. In this context, one prefers *small trees*,  
 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 *relabelling 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,

96         $(G_b, \lambda_b)$  is obtained from the disjoint union of  $(G_l, \lambda_l)$  and  $(G_r, \lambda_r)$  after adding an edge  
 97        from all vertices labeled  $i$  in  $G_l$  to all vertices labeled  $j$  in  $G_r$  whenever  $M_b[i, j] = 1$ .

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

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

105       Let  $b$  be a node in a  $k$ -NLC-expression tree of a graph  $G$ . We denote by  $V_b$  the set of  
 106      vertices of  $G_b$ . By the definition of a  $k$ -NLC-expression tree, if  $u, v \in V_b$  have the same label  
   107      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.

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

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

## 114      2.3 Classification Problems

115       An *example*  $e$  is a function  $e : \text{feat}(e) \rightarrow \{0, 1\}$  defined on a finite set  $\text{feat}(e)$  of *features*. For  
 116      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*  
   117      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  $f \in \text{feat}(e_1)$ ,  $f \in \text{feat}(e_2)$   
   118      but  $e_1(f) \neq e_2(f)$ , we say that the examples *disagree on*  $f$ .

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

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

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

## 130      2.4 Decision Trees

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

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

<sup>140</sup> a DT for  $E$ ) if  $T$  correctly classifies every example  $e \in E$ . See Figure 1 for an illustration of  
<sup>141</sup> a CI, its incidence graph, and a DT that classifies  $E$ .

<sup>142</sup> The size of  $T$  is its number of nodes, i.e.  $|V(T)|$ . We consider the following problem.

MINIMUM DECISION TREE SIZE (DTS)

<sup>143</sup> Input: A classification instance  $E$  and an integer  $s$ .  
<sup>144</sup> Question: Is there a decision tree of size at most  $s$  for  $E$ ?

<sup>145</sup> We now give some simple auxiliary lemmas that are required by our algorithm.

<sup>146</sup> ▶ **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.*

<sup>147</sup> **Proof.** We start by counting the number of trees  $T$  with  $n$  nodes that can potentially underlie  
<sup>148</sup> a DT with  $n$  nodes. Note that there is one-to-one correspondence between trees  $T$  that  
<sup>149</sup> underlie a DT with  $n$  nodes and unlabelled rooted ordered binary trees with  $n$  nodes (where  
<sup>150</sup> ordered refers to an ordering of the at most 2 child nodes). Since it is known that the number  
<sup>151</sup> of unlabelled rooted ordered binary trees with  $n$  nodes is equal to the  $n$ -th Catalan number  
<sup>152</sup>  $C_n$  and that those trees can be enumerated in  $\mathcal{O}(C_n)$  time [27], we already obtain that we  
<sup>153</sup> can enumerate all of the at most  $C_n$  possible trees  $T$  underlying a DT of size  $n$  in  $\mathcal{O}(C_n)$   
<sup>154</sup> time. Therefore, there are at most  $sC_s$  possible trees of size at most  $s$  that can underlie a  
<sup>155</sup> DT with at most  $s$  nodes and those can be enumerated in  $\mathcal{O}(sC_s)$  time. It now remains  
<sup>156</sup> to bound the number of possible feature assignments  $feat(f)$  for these trees as well as the  
<sup>157</sup> number of possibilities for the leave nodes that can be either labelled positive or negative.  
<sup>158</sup> Since we can assume that  $a \geq 2$ , we obtain that the number of possible feature assignments  
<sup>159</sup> (and labellings of leaf-nodes) of a tree  $T$  with  $n$  nodes is at most  $a^n$ . Taking everything  
<sup>160</sup> 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  
<sup>161</sup>  $s$  using only features in  $A$  and those can be enumerated in  $\mathcal{O}(a^{2s+1})$  time. ◀

<sup>162</sup> ▶ **Lemma 2.** *Let  $A$  be a set of features of size  $a$ . There are at most  $a^{2^{a+1}+3}$  inclusion-wise  
<sup>163</sup> minimal DTs using only features in  $A$  and these can be enumerated in  $\mathcal{O}(a^{2^{a+1}+3})$  time.*

<sup>164</sup> **Proof.** Note that an inclusion-wise minimal DT  $T$  that uses only features in  $A$  has at most  
<sup>165</sup>  $2^a + 1$  nodes; this is because every feature appears at most once on every path  $T$ . Therefore, we  
<sup>166</sup> obtain from Lemma 1 that the number of choices for  $T$  is at most  $a^{2(2^a+1)+1} = a^{2^{a+1}+3}$ . ◀

<sup>167</sup> ▶ **Lemma 3.** *Let  $E$  be a CI. Then one can decide whether  $E$  has a DT and if so output a  
<sup>168</sup> DT of minimum size for  $E$  in time  $\mathcal{O}((2^{|E|})^{4|E|-1})$ .*

<sup>169</sup> **Proof.** Note first that  $|feat(E)| \leq 2^{|E|}$  since we can assume that  $E$  does not contain two  
<sup>170</sup> equivalent features. Moreover,  $E$  has a DT if and only if  $feat(E)$  is a support set, which can be  
<sup>171</sup> checked in time  $\mathcal{O}(|E|^2 |feat(E)|)$  by checking, for every pair of positive and negative examples  
<sup>172</sup> in  $E$ , whether there is a feature that distinguishes them. If this is not the case, we output **NO**,  
<sup>173</sup> so assume that  $E$  has a DT. Note that any inclusion-wise minimal DT for  $E$  has at most  $|E|$   
<sup>174</sup> leaves and therefore size at most  $2|E| - 1$ . We can therefore employ Lemma 1 to enumerate  
<sup>175</sup> all inclusion-wise minimal potential DTs for  $E$  in time  $\mathcal{O}((2^{|E|})^{2(2|E|-1)+1}) \in \mathcal{O}((2^{|E|})^{4|E|-1})$ .  
<sup>176</sup> For every such tree we then check whether it is indeed a DT for  $E$  and return a DT for  $E$  of  
<sup>177</sup> minimum size found during this process. ◀

178 **3 An FPT-Algorithm for NLC-width**

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

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

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

todo: Due to  
proposition ...

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

193 **3.1 Description of the Main Ideas Behind the Algorithm**

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

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

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

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

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

todo: adjust to  
NLC-width

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

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

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

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

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

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

266 the nodes of  $T$  that use a feature in  $\text{feat}(b)$ . For all other nodes, i.e. nodes that use either  
 267 forgotten or future features, we only need to remember such a node, if at least one example  
 268 in  $\text{exam}(b)$  ends up in its right child. Moreover, even if this is the case, we only need to  
 269 remember the following for such nodes:

- 270 ■ whether it uses a future or a forgotten feature and  
 271 ■ how it behaves w.r.t. the examples in  $\text{exam}(b)$ .

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

### 274 3.2 Formal Definition of Records and Preliminary Results

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

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

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

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

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

304 We say that  $T$  is a *DT template* for  $b$  if  $T$  is a DT for  $\text{exam}(b)$  that can additionally  
 305 use the future features in  $I_{[k]}$ . Here, we assume that a future feature  $f_{L'} \in I_{[k]}$  for some  
 306  $L' \subseteq [k]$  is 1 at an example  $e \in \text{exam}(b)$  if  $\lambda_b(e) \in L'$  and otherwise it is 0. We say that  
 307 a DT template is *complete* if it does not use any features in  $I_{[k]}$ , otherwise we say that it  
 308 is *incomplete*. Informally, the role of the future features in a DT template is to provide  
 309 spaceholders for the features in  $\text{feat}(E) \setminus \text{feat}(b)$ . Because all of those features behave the  
 310 same w.r.t. examples in  $\text{exam}(b)$  having the same label, they can be characterised by the set  
 311 of labels for which those features are 1. Let  $T$  be a DT template for  $b$  and let  $t \in V(T)$ . We

312 denote by  $A_T(t)$  (or short  $A(t)$ ) if  $T$  is clear from the context) the set of *filtered labels* for  $t$ ,  
 313 i.e.,  $A(t) = (\bigcap_{f_{L'} \in \text{feat}(\text{anc}_L(t)) \cap I_{[k]}} \overline{L'}) \cap (\bigcap_{f_{L'} \in \text{feat}(\text{anc}_R(t)) \cap I_{[k]}} L')$ . Informally,  $A(t)$  is the set  
 314 of all labels  $\ell \in [k]$  such that an example  $e$  with label  $\ell$  would end up at  $t$ , if only the effect of  
 315 the future features on the path to  $t$  is considered. We say that  $t$  with  $f_{L'} = \text{feat}(t) \in I_{[k]}$  is  
 316 *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  
 317 left redundant or right redundant. Intuitively,  $t$  is left (right) redundant if all examples that  
 318 can reach  $t$  (considering the influence of the future features only) end up in the left (right)  
 319 child of  $t$ . This also implies that if  $t$  is left (right) redundant, then the DT template obtained  
 320 after left (right) contracting  $t$  is equivalent with  $T$  (all examples end up in the same leaves).  
 321 Finally, let us extend the definition  $E_T(t)$  from DTs to DT templates. That is, for a DT  
 322 template  $T$  for a node  $b$ , a node  $t \in V(T)$ , and a set of examples  $E' \subseteq \text{exam}(b)$ , we denote  
 323 by  $E_T(E', t)$  (or  $E_T(t)$  if  $E' = \text{exam}(b)$ ) the set of examples  $e \in E'$  with  $\lambda_b(e) \in A(t)$  and  
 324  $e \in E'[\tau(t)]$ , where  $\tau(t)$  is the assignment of the features in  $\text{feat}(b)$  along the path from the  
 325 root of  $T$  to  $t$ .

define  $\tau$  in prelims

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

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

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

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

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

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

360 361 362 363 364  $T'$ . This clearly holds if  $\text{feat}(t)$  is not a future feature. So suppose that  $\text{feat}(t) = f_L$ . Then, because  $t$  is left (right) redundant in  $T$  (because otherwise  $t'$  would have been removed from  $T$  when contracting  $t$ ), we have that  $A_T(t) \subseteq \bar{L}$  ( $A_T(t) \subseteq L$ ). Therefore,  $A_T(t) = A_T(t) \cap \bar{L}$  ( $A_T(t) = A_T(t) \cap L$ ), which shows that  $t$  has no influence on  $A_T(t')$  and therefore implies that  $A_T(t') = A_{T'}(t')$ .  $\blacktriangleleft$

365 366 We now show that  $r(T)$  shares certain properties with  $T$ . In particular, the first observation shows that if  $T$  is a DT template for  $b$ , then so is  $r(T)$ .

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

368 369 370 371 372 373 374 **Proof.** It suffices to show that if  $t$  is left (right) redundant in  $T$  and  $e$  is in  $E_T(t)$ , then  $e$  goes to the left (right) child of  $t$  in  $T$ . If  $\text{feat}(t) \in \text{feat}(b)$ , then  $t$  is left (right) redundant because of some left (right) ancestor  $t'$  with  $\text{feat}(t') = \text{feat}(t)$ . Moreover, because  $e \in E_T(t)$ ,  $e$  went to the left (right) child of  $t'$  and therefore  $e$  goes to the left (right) child of  $t$  (because  $\text{feat}(t) = \text{feat}(t')$ ). If, on the other hand,  $\text{feat}(t)$  is some future feature  $f_L$ , then  $A(t) \subseteq \bar{L}$  ( $A(t) \subseteq L$ ) and because  $e \in E_T(t)$ , also  $\lambda_b(e) \in A(t)$ . Therefore,  $e$  goes to the left (right) child of  $t$ .  $\blacktriangleleft$

375 376 The second observation shows the similarity in behaviour of  $T$  and  $r(T)$  with respect to future examples in  $E \setminus \text{exam}(b)$ .

377 378 **► Observation 9.** *Let  $T$  be a DT (skeleton/template) for  $b$ , and let  $e$  be an example in  $E \setminus \text{exam}(b)$  that is correctly classified by  $T$ . Then,  $e$  is also correctly classified by  $r(T)$ .*

379 380 381 382 383 384 385 386 **Proof.** The proof is very similar to the proof of Observation 8. That is, again it suffices to show that if  $t$  is left (right) redundant in  $T$  and  $e$  goes to  $t$ , then  $e$  goes to the left (right) child of  $t$  in  $T$ . The proof is essentially the same as the proof in Observation 8 for the case that  $\text{feat}(t)$  is a real feature or a future feature. Moreover, if  $\text{feat}(t)$  is a forgotten feature  $f_\ell$ , then  $t$  is left (right) redundant because of some left (right) ancestor  $t'$  with  $\text{feat}(t') = \text{feat}(t) = f_\ell$ . Moreover, because  $e$  goes to  $t$ ,  $e$  went to the left (right) child of  $t'$  and therefore  $e$  goes to the left (right) child of  $t$  (because  $e$  behaves in the same way w.r.t. every feature in  $V_b$  that has the same label).  $\blacktriangleleft$

387 388 389 Before we define our records and their semantics, we first show a bound on the number of DT skeletons (and the time to enumerate those) as this will allow us to obtain a similar bound for the number of records. We say that  $T$  is *reduced* if  $r(T) = T$ .

390 391 392 **► Observation 10.** *Let  $T$  be a reduced DT skeleton whose forgotten features use a set of at most  $k_F$  labels and whose future features use a set of at most  $k_I$  labels. Then,  $T$  has height at most  $k_F + k_I + 1$  and size at most  $2^{k_F+k_I+1}$ .*

393 394 395 396 397 **Proof.** Consider a root-to-leaf path  $P$  in  $T$ . Then, every forgotten feature appears at most once on  $P$ ; because the second occurrence of such a feature would necessarily be redundant. Therefore,  $P$  can contain at most  $k_F$  forgotten features. Similarly,  $P$  can contain at most  $k_I$  future features, since otherwise one of the future features on  $P$  would be redundant. Therefore,  $T$  has height at most  $k_F + k_I + 1$  and therefore size at most  $2^{k_F+k_I+1}$ .  $\blacktriangleleft$

398 We obtain the following corollary as a special case.

399 400 **► Corollary 11.** *Let  $T$  be a reduced DT skeleton for a node  $b \in V(B)$ . Then,  $T$  has height at most  $2k + 1$  and size at most  $2^{2k+1}$ .*

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

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

407 We obtain the following corollary as a special case.

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

410 Let  $T$  be a DT (template/skeleton) using only features in  $\text{feat}(E) \cup F_L \cup I_L$  for some set  
411  $L$  of labels (usually  $L = [k]$ ). A *feature relabelling* is a function  $\alpha : \text{feat}(E) \cup F_L \rightarrow F_{L'} \cup I_{L'}$ ,  
412 where  $L'$  is some set of labels (usually  $L' = L$ ). With a slight abuse of notation, we denote  
413 by  $\alpha(T)$ , the decision tree obtained after relabelling all features used by  $T$  according to  $\alpha$ ,  
414 i.e.,  $\alpha(T)$  is obtained from  $T$  after replacing the feature assignment function  $\text{feat}_T(t)$  for  $T$   
415 with the function  $\text{feat}_{\alpha(T)}(t)$  defined by setting  $\text{feat}_{\alpha(T)}(t) = \alpha(\text{feat}_T(t))$  if  $\alpha$  is defined for  
416  $\text{feat}(t)$  and  $\text{feat}_{\alpha(T)}(t) = \text{feat}_T(t)$ , otherwise. We say that two feature relabellings  $\alpha_1$  and  $\alpha_2$   
417 are *compatible* if they agree on their shared domain.

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

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

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

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

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

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

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

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

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

### 3.3 Proof to the Main Result

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

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

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

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

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

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

The significance of the plug in operation comes from the fact that it allows us to reverse the reduction that has been applied to a DT (template/skeleton). For instance, consider a node  $b$  of  $B$  and let  $T$  be a DT skeleton for  $b$  and let  $T'$  be a DT template for  $b$  such that  $T = r(\alpha_b^s(T'))$ . Then, we can use the plug in operation to reverse the direction or in other words obtain  $T'$  from  $T$  as follows. Let  $z : V(T) \rightarrow V(T')$  be the injective function mapping every node in  $T$  to its original node in  $T'$ . Then, we first use  $z$  to reverse the relabelling

given by  $\alpha_b^s(T')$ , i.e., let  $T^0$  be the DT obtained from  $T$  by setting  $\text{feat}_{T^0}(t) = \text{feat}_{T'}(z_H(t))$  for every  $t \in V(T^0)$ . We now add back the nodes in  $V(T') \setminus V(T)$  with the help of our plug in operation. In particular, for every edge  $e = (p, c)$  in  $T^0$ , where  $p$  is the parent of  $c$  in  $T^0$ , let  $P(e)$  be the path in  $T'$  between  $z(p)$  and  $z(c)$ . Let  $T^1$  be the DT template obtained from  $T^0$  after plugging in the path  $P(e)$  into  $T^0$  at edge  $e = (p, c)$  of  $T^0$ . Then, it is easy to see that  $T^1 = T'$ .

todo: simplify the run-time expression

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

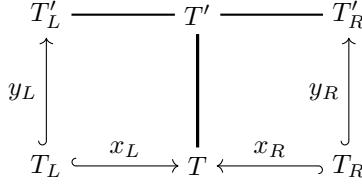
568 Informally, we obtain  $T'$  from  $T$  after reversing the relabelling and reduction operations  
 569 applied to  $T'_L$  and  $T'_R$  to obtain  $T_L$  and  $T_R$ , respectively; recall that  $T_H = r(\alpha_{b_H}^s(T'_H))$  for  
 570  $H \in \{L, R\}$ . That is, we will reverse the labelling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

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

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

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

maybe a longer explanation

**Figure 2**

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

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

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

594 Consider an edge  $e = (p, c)$  in  $T_R$  such that  $p$  is the parent of  $c$  in  $T_R$ . Then,  $e$  corresponds  
595 to a path  $P'_R(e)$  between  $y_R(p)$  and  $y_R(c)$  in  $T'_R$ . Similarly,  $e$  corresponds to a path  $P_R(e)$   
596 between  $x_R(p)$  and  $x_R(c)$  in  $T^1$ . Similarly to above, we now add all nodes to  $T^1$  that have  
597 been removed when going from  $T'_R$  to  $T_R$  (via the reduction  $r(\alpha_{b_R}^s(T'_R))$ ). To achieve this,  
598 we go over every edge  $e = (p, c)$  of  $T_R$  such that  $p$  is the parent of  $c$  in  $T_R$  and plug in the  
599 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  
600 obtained from  $T^1$  after doing this operation for every edge of  $T_R$ .

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

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

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

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

624 also the non-redundant child of  $t$  in  $\alpha_L(T)$  and therefore  $e$  is send to the child of  $t$  on  $P$ , as  
625 required.

626 In the latter case, i.e., the case that  $t$  is in  $V(T'_H) \setminus V(T_H)$ ,  $t$  is redundant in  $\alpha_{b_H}^s(T'_H)$   
627 because of some ancestor  $t' \in V(T_H)$  with  $\alpha_{b_H}^s(feat_{T'}(t)) = \alpha_{b_H}^s(feat_{T'}(t'))$ . Therefore,  
628  $feat_{T'}(t')$  behaves in the same manner towards  $e$  as  $feat_{T'}(t)$ , which because  $t'$  is on  $Q$   
629 (because  $t' \in V(T_H)$ ) implies that  $e$  is send to the (non-redundant) child of  $t$  on  $P$ .

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

712 A DT pattern  $T'$  is an *improvement* for a DT pattern  $T$  if  $T' = T$  as rooted trees and  
 713  $\text{feat}_{T'}(v) = \text{feat}_T(v)$  for every fixed node  $v$  of  $T$ . A *complete improvement*  $T'$  of  $T$  is an  
 714 improvement such that  $\text{feat}(T') \subseteq \text{feat}(E)$ . A *threshold assignment* for a DT pattern  $T$  is a

717 function  $th$  that maps every fixed node  $v \in V(T)$  to a natural number  $th(v)$ . Note that any  
 718 complete improvement  $T'$  of a DT pattern  $T$  can be made to a decision tree with a threshold  
 719 assignment.

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

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

## 732 4.1 Preprocess

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

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

739 Note that the definition of  $E_v$  is strictly related with the following: if  $v$  is a fixed node,  
 740 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  
 741 follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

804 The running time of Algorithm 1 can now be obtained by multiplying the number of recursive calls to **findLR** with the time required for one recursive call. To obtain the number

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

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

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

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

**Input:** CI  $E$ , DT pattern  $T$

**Output:** a triple  $(E_v, th_L(v), th_R(v))$  for every node  $v \in V(T)$ .

```

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

```

## 4.2 The algorithm

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

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

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

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

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

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

---

**Input:** CI  $E$ , DT pattern  $T$ , feature  $f$  of the root of  $T$ , left child  $c_\ell$  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 classify every example in  $E[f \leq t_\ell]$  and minimum threshold  $t_r$  in  $D_E(f)$  for  $f$  such that  $(T_{c_r}, \alpha)$  can classify  $E[f > t_r]$

```

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

```

---

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

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

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

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

### Algorithm 3

---

**Input:** CI  $E$ , pattern  $T$

**Output:** TRUE if  $T$  can classify all examples in  $E$ , FALSE otherwise

```

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

```

---

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

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

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

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

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

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

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

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

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

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

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

883 ▶ **Theorem 29.** MINIMUM DECISION TREE SIZE is fixed-parameter tractable parametrized  
 884 by  $\delta_{\max} + s$ .

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

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

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

917 computed in Line 2 of Algorithm 5, which implies that  $T$  is a DT for  $E$ . Moreover,  $T$  has  
 918 size at most  $s$ , because of Line 8 in Algorithm 4.

919 To analyse the run-time of the algorithm, we first remark that the whole algorithm can  
 920 be seen as a bounded-depth search tree algorithm, i.e., a branching algorithm with small  
 921 recursion depth and few branches at every node. In particular, every recursive call adds at  
 922 least one feature to the set of features bounding the recursion depth to at most  $s$ . Moreover,  
 923 every feature that is added is either added in Line 2 of Algorithm 4, when enumerating  
 924 all minimal support sets, in which case there are at most  $\delta_{\max}(E)$  branches or the feature  
 925 is added in Line 5 of Algorithm 5, in which case there are at most  $|R_0| \leq s^{2s+3}\delta_{\max}(E)$   
 926 branches. It follows that the algorithm can be seen as a branching algorithm of depth  
 927 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  
 928 step. Therefore, the total run-time of the algorithm is at most the number of nodes in  
 929 the branching tree, i.e., at most  $(s^{2s+3}\delta_{\max}(E))^s$ , times the maximum time required in  
 930 one recursive call. Now the maximum time required for one recursive call is dominated  
 931 by the time spend in Line 2 of Algorithm 5, i.e., the time required to compute a DT of  
 932 minimum size using exactly the features in  $S$  with the help of Theorem 28, which is at  
 933 most  $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$ . Therefore, we obtain  $(s^{2s+3}\delta_{\max}(E))^s 2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$   
 934 as the total run-time of the algorithm, which shows that DTS is fixed-parameter tractable  
 935 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  
 $\text{nil}$

```

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

## 936 5 Conclusion

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

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

- 943   ■ What other (structural) parameters can be exploited to efficiently learn DTs? Is learning  
 944     DTs of small size fixed-parameter tractable parameterized by the rank-width of  $G_I(E)$ ?
- 945   ■ Instead of learning DTs of small size, one often wants to learn DTs of small height.  
 946     Therefore, it is natural to ask whether our approach can be also used in this setting.  
 947     While one can adapt our approach to obtain an XP-algorithm for learning DTs of small

■ **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 24”
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

```

---

948 height parameterized by NLC-width, it is not clear to us whether the problem also allows  
949 for an fpt-algorithm.

950 ■ Can we extend our approach to CIs, where features range over an arbitrary domain? In  
951 this case, one usually still uses DTs that make binary decisions (i.e. whether a feature is  
952 smaller equal or larger than a given threshold). While it is relatively easy to see that our  
953 approach can be extended if the domain’s size (for every feature) is bounded or used as  
954 an additional parameter, it is not clear what happens if the size of the domain is allowed  
955 to grow arbitrarily.

956 ————— **References** —————

- 957 1 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics.  
958 Springer-Verlag London Ltd., London, second edition, 2009.
- 959 2 Christian Bessiere, Emmanuel Hebrard, and Barry O’Sullivan. Minimising decision tree size  
960 as combinatorial optimisation. In Ian P. Gent, editor, *Principles and Practice of Constraint  
961 Programming - CP 2009*, pages 173–187, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 962 3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth.  
963 *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 964 4 Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, and  
965 Kazuhisa Makino. Logical analysis of data: classification with justification. *Ann. Oper. Res.*,  
966 188(1):33–61, 2011.
- 967 5 Endre Boros, Vladimir Gurvich, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan.  
968 Decomposability of partially defined Boolean functions. *Discr. Appl. Math.*, 62(1-3):51–75,  
969 1995.
- 970 6 Endre Boros, Takashi Horiyama, Toshihide Ibaraki, Kazuhisa Makino, and Mutsunori Yagiura.  
971 Finding essential attributes from binary data. *Ann. Math. Artif. Intell.*, 39:223–257, 11 2003.  
972 doi:10.1023/A:1024653703689.
- 973 7 Endre Boros, Toshihide Ibaraki, and Kazuhisa Makino. Variations on extending partially  
974 defined Boolean functions with missing bits. *Information and Computation*, 180(1):53–70,  
975 2003.
- 976 8 Yves Crama, Peter L. Hammer, and Toshihide Ibaraki. Cause-effect relationships and partially  
977 defined Boolean function. *Ann. Oper. Res.*, 16:299–326, 1988.
- 978 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin  
979 Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.  
980 doi:10.1007/978-3-319-21275-3.

- 981 10 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In Giuseppe De  
982 Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín,  
983 and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence,  
984 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 -  
985 Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*,  
986 volume 325 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2020.
- 987 11 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer  
988 Verlag, New York, 2nd edition, 2000.
- 989 12 Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *CoRR*,  
990 abs/1702.08608, 2017. URL: <http://arxiv.org/abs/1702.08608>, arXiv:1702.08608.
- 991 13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts  
992 in Computer Science. Springer Verlag, 2013.
- 993 14 Wolfgang Espelage, Frank Gurski, and Egon Wanke. Deciding clique-width for graphs of  
994 bounded tree-width. *J. Graph Algorithms Appl.*, 7(2):141–180, 2003.
- 995 15 Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-  
996 making and a “right to explanation”. *AI Magazine*, 38(3):50–57, 2017.
- 997 16 Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete.  
998 *Information Processing Letters*, 5(1):15–17, 1976.
- 999 17 Toshihide Ibaraki, Yves Crama, and Peter L. Hammer. *Partially defined Boolean functions*,  
1000 page 511–563. Encyclopedia of Mathematics and its Applications. Cambridge University Press,  
1001 2011.
- 1002 18 Daniel T. Larose. *Discovering knowledge in data*. Wiley-Interscience [John Wiley & Sons],  
1003 Hoboken, NJ, 2005. An introduction to data mining.
- 1004 19 Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*,  
1005 61(10):36–43, 2018.
- 1006 20 E.J. McCluskey. *Introduction to the Theory of Switching Circuits*. Electrical and electronic  
1007 engineering series. Princeton University series. McGraw-Hill, 1965.
- 1008 21 Don Monroe. AI, explain yourself. *AI Communications*, 61(11):11–13, 2018. doi:10.1145/  
1009 3276742.
- 1010 22 Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary  
1011 survey. *Data Min. Knowl. Discov.*, 2(4):345–389, 1998. doi:10.1023/A:1009744630224.
- 1012 23 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learn-  
1013 ing. In *Proceedings of AAAI'21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*,  
1014 pages 6454–6462. AAAI Press, 2021.
- 1015 24 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on  
1016 Algorithms*, 5(1), 2008.
- 1017 25 J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi:  
1018 10.1023/A:1022643204877.
- 1019 26 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster  
1020 parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt,  
1021 and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International  
1022 Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume  
1023 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.
- 1024 27 Richard Stanley and Eric W. Weisstein. Catalan number, from mathworld—a wolfram web  
1025 resource, 2015.
- 1026 28 Egon Wanke.  $k$ -NLC graphs and polynomial algorithms. *Discr. Appl. Math.*, 54(2-3):251–266,  
1027 1994. Efficient algorithms and partial  $k$ -trees.