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

Anonymous author
Anonymous affiliation

Abstract

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

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

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

1 Introduction

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

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

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

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

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

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

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

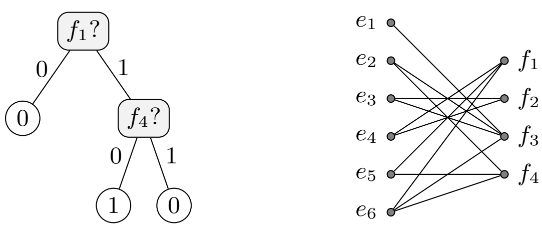


Figure 1 A CI $E = E^+ \uplus E^-$ with six examples and four features (left), a decision tree with 5 nodes that classifies E (middle), the incidence graph $G_I(E)$ (right).

DP approach on an approximate tree decomposition to compute the exact treewidth of a graph; here, the solutions are tree decompositions, which are complex structures that cannot easily be represented in terms of the graph. Other prominent examples include a DP approach to compute the exact treedepth [25] or clique-width [14] using an optimal tree decomposition. We face a similar problem, since solutions in our case are decision trees that do not bear any resemblence to the incidence graph for which we are given the tree decomposition. The main obstacle to overcome, therefore, is the design of the DP-records for our DP algorithm. That is, a record for a node b in a tree decomposition for the incidence graph of E needs to provide a compact representation of partial solutions, i.e. partial solutions in the sense that they represent the part of the solution for the whole instance E that corresponds to the sub-instance induced by all features and examples contained in the bags in the subtree of the tree decomposition rooted at the current node b. We overcome this obstacle in Section 3, where we also provide intuitive descriptions and motivation for the definition of the records (Subsection 3.1).

2 Preliminaries

2.1 Parameterized Complexity

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

2.2 Graphs and NLC-width

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

Node label control-width (NLC-width) is a graph parameter, defined as follows [27]: Let $k \in \mathbb{N}$ be a positive integer. A k-NLC-expression consists of a rooted subcubic tree such that:

1. Every leaf is labelled with a label $i \in [k]$; this corresponds to a graph with a single vertex which has a label i.
2. Every non-leaf node with one child is labelled with a function $R : [k] \to [k]$. This corresponds to taking the labelled graph corresponding to the child and for every $i \in \{1, \ldots, k\}$, if a vertex has label i, changing this label to be $R(i)$ instead.
3. Every non-leaf node with two children is labelled with a $k \times k$ $\{0, 1\}$ matrix M. This corresponds to taking the disjoint union of the graphs corresponding to its children G_1 and G_2, and then for $i, j \in [k]$, adding an edge from all vertices lablled i in G_1 to all vertices labelled j in G_2 if and only if $M_{i,j} = 1$.
4. The root node corresponds to the resulting labelled graph.

The NLC-width $NLC(G)$ of a graph G is the minimum k for which G has a k-NLC-expression. A k-NLC-expression is *nice* if every relabelling node has a function $R : [k] \to [k]$

such that for some $i, j \in [k]$, $R(i) = j$ and $R(\ell) = \ell$ for all $\ell \in [k] \setminus \{i\}$. Clearly, given a k-NLC-expression. a nice k-NLC-expression can be found in polynomial time.

Let x be a node in a k-NLC-expression tree of a graph G. We define $\chi(X)$ to be the set of vertices in $V(G)$ that correspond to leaves of the k-NLC-expression subtree rooted at x. By the definifion of a k-NLC-expression, if $s, t \in \chi(X)$ have the same label after applying node x and $u \in V(G) \setminus \chi(X)$, then s is adjacent to u if and only if t is. Furthermore, the k-NLC-expression subtree rooted at x is the graph induced by G on $\chi(X)$.

Computing the NLC-width of a graph is NP-hard [**?**]. However, it is sufficient to use the algorithm of Seymour and Oum [**?**], which returns a c-expression for some $c \leq 2^{3cw(G)+2} - 1$ in $O(n^9 \log n)$ time, or the later improvements of Oum [23] and Hliněný and Oum [**?**] that provide cubic-time algorithms which yield a c-expression for some $c \leq 8^{cw(G)} - 1$ and $c \leq 2^{cw(G)+1} - 1$, respectively.

2.3 Classification Problems

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

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

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

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

2.4 Decision Trees

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

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

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

MINIMUM DECISION TREE SIZE (DTS)	
Input:	A classification instance E and an integer s.
Question:	Is there a decision tree of size at most s for E?

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

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

Proof. We start by counting the number of trees T with n nodes that can potentially underlie a DT with n nodes. Note that there is one-to-one correspondence between trees T that underlie a DT with n nodes and unlabelled rooted ordered binary trees with n nodes (where ordered refers to an ordering of the at most 2 child nodes). Since it is known that the number of unlabelled rooted ordered binary trees with n nodes is equal to the n-th Catalan number C_n and that those trees can be enumerated in $\mathcal{O}(C_n)$ time [26], we already obtain that we can enumerate all of the at most C_n possible trees T underlying a DT of size n in $\mathcal{O}(C_n)$ time. Therefore, there are at most sC_s possible trees of size at most s that can underlie a DT with at most s nodes and those can be enumerated in $\mathcal{O}(sC_s)$ time. It now remains to bound the number of possible feature assignments $feat(f)$ for these trees as well as the number of possibilities for the leave nodes that can be either labelled positive or negative. Since we can assume that $a \geq 2$, we obtain that the number of possible feature assignments (and labellings of leaf-nodes) of a tree T with n nodes is at most a^n. Taking everything together, we obtain that there are at most $sC_sa^s \leq s4^sa^s \leq a^{2s+1}$ many DTs of size at most s using only features in A and those can be enumerated in $\mathcal{O}(a^{2s+1})$ time. ◀

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

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

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

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

3 An FPT-Algorithm for NLC-width

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

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

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

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

3.1 Description of the Main Ideas Behind the Algorithm

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

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

(2) $e(f) = 0$ for every future example $e \in E \setminus \mathit{exam}(B_b)$ and forgotten feature $f \in \mathit{feat}(B_b) \setminus \mathit{feat}(b)$;

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

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

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

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

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

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

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

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

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

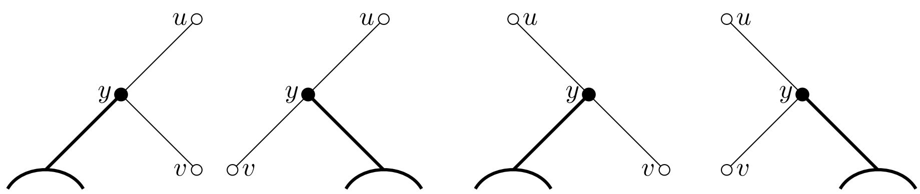
- whether it uses a future or a forgotten feature and
- how it behaves w.r.t. the examples in $exam(b)$.

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

3.2 Formal Definition of Records and Preliminary Results

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

Now we can define two operations on subcubic rooted trees and single pairs. We say that we *plug in* a single pair (Y, S) in a left (right) edge uv as follows: we make the root y of Y the left (right) child of u, $Y \setminus \{y\}$ to be the S subtree of y and v to be the $H \in \{left, right\} \setminus S$ child of y. See Figure 2 for the corresponding drawings. Note after a plug in of a single pair in an edge, the node v belongs in the same side of the subtree rooted at u as it was before the plug in.



(a) $(Y, left)$ left. **(b)** $(Y, right)$ left. **(c)** $(Y, left)$ right. **(d)** $(Y, right)$ right.

Figure 2 The drawings describe the plug in operation in the different four cases. The bold part highlight the single pair (Y, S).

Let (Y, S) be a single pair in a rooted subcubic tree T, then we *remove* (Y, S) from T as follows. Let y be the root of Y. If y is the root of T, then we obtain an empty tree. If y is a leaf node of T, then we obtain $T - y$. Otherwise let y be a non-root and non-leaf node, let u be the parent of y and v be the child of y that is not in $V(Y)$, then we consider the tree obtained from T after replacing y with v as the child of u and deleting Y. See Figure 3 for an example.

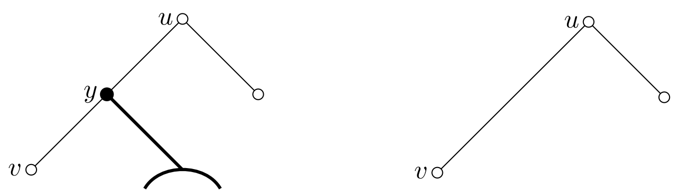


Figure 3 The drawing describe an example of the remove operation: a single pair $(Y, right)$ is removed from a subcubic rooted tree. The bold part highlight the single pair (Y, S).

It is clear from the four different plug in cases that if we want to plug in two pairs (Y, S) and (Y', S') on an edge uv such that the ancestor-descendant relationship is given, say y of Y has to be in the path from the root to y' of Y', then we can do these plug ins in any order but with some care. It is the same if we first plug in (Y, S) in the edge uv and then plug in

(Y', S') in the edge yv or if we first plug in (Y', S') in the edge uv and then plug in (Y, S) in the edge uy'. See Figure 4 for the an example.

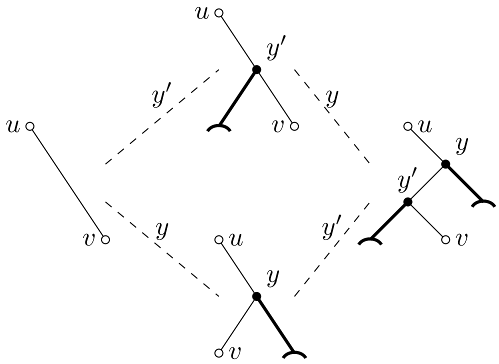


Figure 4 An example of plugging in two pairs $(Y, left)$ and $(Y', right)$ in a left edge uv.

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

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

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

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

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

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

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

▶ **Lemma 7** (Relabelling Lemma). *Let T be a DT and p be relabelling of T. Then $(r \circ p \circ r)(T) = (r \circ p)(T)$.*

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

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

in $p(T)$ that allows to remove v. If A_v is a set of ancestors of v in T that allows to reduce v then $v \notin r(T)$ and so $v \notin (r \circ p \circ r)(T)$. Otherwise let A'_v be the subset of A_v in $(p \circ r)(T)$. We conclude by noting that A'_v contains one of the minimal sets A_v is composed of and so $v \notin (r \circ p \circ r)(T)$.

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

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

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

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

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

▶ **Lemma 8.** *If there are ℓ features with labels and 2^h future features, then every reduced DT template has height at most $\ell + h$. Furthermore, every path from the root to the leaves contains at most ℓ features with label and at most $h - 1$ future features.*

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

▶ **Lemma 9.** *If there are ℓ features with label and 2^h future features, then there are at most $(\ell + 2^k + 2)2^{\ell+k+1}$ reduced DT templates. Furthermore, these can be enumerated in $\mathcal{O}((\ell + 2^k + 2)2^{\ell+k+1})$-time.*

Proof. By Lemma 8, the tree has height at most $\ell + k$. Each node of the decision tree could be a feature with label, a future feature, or a leaf: at most $\ell + 2^h + 2$ different contents. Since there are at most $2^{\ell+h+1}$ nodes in the tree, there are at most $(\ell + 2^h + 2)2^{\ell+h+1}$ possible decision trees. ◀

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

3.3 Proof to the Main Result

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

▶ **Lemma 10** (leaf node). *Let $b \in V(B)$ be a leaf node. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}(k(2^k+3)2^{k+2})$.*

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

We have to enumerate all possible reduced DT templates T for b. It is enough to consider all reduced DT templates T of height at most $k+1$ and discard those that are not DT templates for b; these can be enumerated in time $\mathcal{O}((2^k+3)2^{k+2})$ by Lemma 9 and the check can be done in time $\mathcal{O}(k)$. We add the pair $(T, 0)$ to the set of records $\mathcal{R}(b)$.

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

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

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

▶ **Lemma 11** (join node). *Let $b \in V(B)$ be a join node. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}()$.*

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

We consider every reduced DT T for b with feature labels in $[k] \cup [k']$ and future feature labels in $\mathcal{P}([k])$; these can be enumerated in time $\mathcal{O}((2k+2^k+2)2^{3k+1})$ by Lemma 9.

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

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

Now we want to evaluate the running time of computing $\mathcal{R}(b)$. Every reduced DT T can be enumerated in time $\mathcal{O}((2k+2^k+2)2^{3k+1})$ by Lemma 9. For every such tree T, there are at most 2^{3k} paths from the root to the leaves and for every of these paths there are at most k nodes for each of the following features: with label in $[k]$, with label in $[k']$ and future by Lemma 8. This means p_* and p'_* can be done in $\mathcal{O}(k2^{3k})$ time.

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

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

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

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

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

Claim 1: For every $H \in \{L, R\}$ and for every $y, y' \in Y^*_H$, the paths P^H_y and $P^H_{y'}$ are either edge disjoint or $P^H_y = P^H_{y'}$.

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

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

Now for every $y \in Y^*_H$ we consider the path Q^H_y in T having endpoints $x_H(t)$ and $x_H(t)$.

Now we are able to describe how to obtain a witness T' of T for b. For every $y \in Y^*_L$, in the last edge of path Q^L_y we plug in the single pair $(Y^L_{y'}, S^L_{y'})$ rooted at y', for every internal node y' of P^L_y, in the order the nodes y' apprear in P^L_y. Note that, in the case an element of Y^*_L is present in more than one P^L_y, we plug in the corresponding single pair only once. Note also that whenever we plug in some single pair (Y^L_y, S^L_y) in a DT, the tree Y^L_y has real features and future features as nodes. Call this graph T^*. Now we do the same sequence of plug ins of the single pairs corresponding to the internal vertices of P^R_y in the last edge of the path Q^R_y. Again, in the case an element of Y^*_R is present in more than one P^R_y, we plug in the corresponding single pair only once. Call the tree obtained in this way T'. Node that T' contains real features from $feat(b_L)$ and from $feat(b_R)$ and future features with labels in $\mathcal{P}([k])$.

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

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

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

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

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

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

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

▶ **Lemma 12** (relabel node). *Let $b \in V(B)$ be relabel node. Then $\mathcal{R}(b)$ can be computed in time $\mathcal{O}()$.*

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

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

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

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

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

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

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

Before proving (i), we describe how T can be obtained from T'. Let p''' be the following relabelling of T': every real feature that contains j is assigned to the real feature $A \cup \{i\}$ and every other feature is assigned to itself. Then the application of the composition p''', the standard reduction and $r \circ p''$ to T' is exactly the standard reduction for T' which then result to the DT template T. By Lemma 7 the score of the standard reduction from T' to T is exactly $s_C + s^* = s$.

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

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

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

Now we can finally prove Theorem 4 and Theorem 5, which we restate here.

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

Proof. We start off by computing $\mathcal{R}(b)$ for every node b of B, via leaf-to-root dynamic programming. An upper bound for the running time for this step is the number of nodes of B times the maximum running time to compute the record at each node which is given by Lemmas 10, 11 and 12.

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

Theorem 5 (restated). DTS *is fixed-parameter tractable parameterized by NLC-width.*

4 Conclusion

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

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

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

References

1. Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, second edition, 2009.
2. Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, pages 173–187, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
3. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
4. Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Logical analysis of data: classification with justification. *Ann. Oper. Res.*, 188(1):33–61, 2011.
5. Endre Boros, Vladimir Gurvich, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Decomposability of partially defined Boolean functions. *Discr. Appl. Math.*, 62(1-3):51–75, 1995.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

25 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

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

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