Proceedings Paper:
Dabrowski, K.K., Eiben, E., Ordyniak, S. et al. (2024) Learning Small Decision Trees for Data of Low Rank-Width. In: Wooldridge, M., Dy, J. and Natarajan, S., (eds.) Proceedings of the 38th AAAI Conference on Artificial Intelligence. Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24), 20-27 Feb 2024, Vancouver, Canada. AAAI Press, Washington, DC, USA, pp. 10476-10483. ISBN: 978-1-57735-887-9. ISSN: 2159-5399. EISSN: 2374-3468.

https://doi.org/10.1609/aaai.v38i9.28916

Learning Small Decision Trees for Data of Low Rank-Width

Konrad K. Dabrowski[1], **Eduard Eiben**[2], **Sebastian Ordyniak**[3], **Giacomo Paesani**[3], **Stefan Szeider**[4]

[1]Newcastle University,
[2]Royal Holloway, University of London,
[3]University of Leeds,
[4]TU Wien

konrad.dabrowski@newcastle.ac.uk, eduard.eiben@rhul.ac.uk, {giacomopaesani,sordyniak}@gmail.com, sz@ac.tuwien.ac.at

Abstract

We consider the NP-hard problem of finding a smallest decision tree representing a classification instance in terms of a partially defined Boolean function. Small decision trees are desirable to provide an interpretable model for the given data. We show that the problem is fixed-parameter tractable when parameterized by the rank-width of the incidence graph of the given classification instance. Our algorithm proceeds by dynamic programming using an NLC decomposition obtained from a rank-width decomposition. The key to the algorithm is a succinct representation of partial solutions. This allows us to limit the space and time requirements for each dynamic programming step in terms of the parameter.

Introduction

Decision trees (DTs) are extremely useful tools for describing, classifying, and generalizing data (Larose and Larose 2014; Murthy 1998; Quinlan 1986). In this paper, we consider DTs for *classification instances (CIs)*, consisting of a finite set E of *examples*, also called *feature vectors*, over a finite set *feat*(E) of *features*. Each example $e \in E$ is a function $e : \mathit{feat}(E) \to \{0, 1\}$, determining 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 or not they have been diagnosed with a certain disease. CIs are also called *partially defined Boolean functions* (Boros et al. 1995), as we can consider the features as Boolean variables and examples as truth assignments that evaluate to 0 (for negative examples) or 1 (for positive examples). CIs have been studied as a key concept for logical data analysis and switching theory (Boros et al. 2011, 2003; Boros, Ibaraki, and Makino 2003; Crama, Hammer, and Ibaraki 1988; Ibaraki, Crama, and Hammer 2011; McCluskey 1965).

Because of their simplicity, DTs are particularly attractive for providing interpretable models of the underlying CI. Given the impact and widespread use of machine learning methods, the importance of transparency has been strongly emphasized in recent years (Darwiche and Hirth 2023; Doshi-Velez and Kim 2017; Goodman and Flaxman 2017; Ignatiev et al. 2021; Lipton 2018; Monroe 2018). In this context, one prefers *small* DTs, as they are easier to interpret and require fewer tests to classify (a DT is smaller than another if it has fewer nodes). Small trees are also preferred because of the parsimony principle (Occam's Razor), since small trees are expected to generalize better to new data (Bessiere, Hebrard, and O'Sullivan 2009). However, given a CI $E = E^+ \uplus E^-$ and an integer s, deciding whether E has a DT with at most s nodes is NP-complete (Hyafil and Rivest 1976).

This complexity barrier motivates the study of the problem under the parameterized complexity paradigm. Ordyniak and Szeider (2021) made the first approach in this direction, parameterizing the problem by solution size in terms of the number of nodes or the depth of the computed DT. In this paper, we parameterize the problem to exploit the hidden structure of the given CI E. We capture the hidden structure of E in terms of small *rank-width* of the *incidence graph*, which is the bipartite graph $G(E)$ whose vertices are the examples in one part and the features in the other, where an example e is adjacent to a feature f if and only if $e(f) = 1$. Figure 1 shows a CI and a smallest DT for it, and the incidence graph. Rank-width is a well-known graph invariant that generalizes treewidth in the sense that all graph classes of bounded treewidth have bounded rank-width, but there are classes of dense graphs of bounded rank-width and unbounded treewidth. We denote the rank-width and the treewidth of a graph G by *rw*(G) and *tw*(G), respectively, and use *rw*$(G(E))$ and *tw*$(G(E))$ to denote the rank-width and treewidth of E, respectively.

We can state our main algorithmic result as follows (for a formal statement, see Corollary 0.4).

> *Computing a smallest DT for a given CI E is fixed-parameter tractable parameterized by the rank-width of E.*

Rank-width is an exceptionally well-suited parameter for this problem because of its robustness and generality. Borrowing from a similar notion for propositional CNF formulas (Lewis 1978), we define a *renaming* $r_X(E)$ of a CI E with respect to a set $X \subseteq \mathit{feat}(E)$ to be the CI containing all the examples e_X for $e \in E$ with $e_X(f) = 1 - e(f)$ if $f \in X$ and $e_X(f) = e(f)$ if $f \notin X$. Clearly, the size of a smallest DT is the same for E and all its renamings, and a robust parameter should not depend on a particular chosen renaming. Let us define the *renamable rank-width* of E as *rw*$^*(G(E)) =$

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

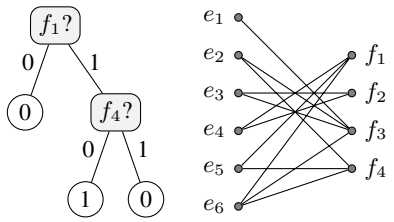

Figure 1: A CI $E = E^+ \uplus E^-$ with six examples and four features (left), a DT with 5 nodes that classifies E (middle), the incidence graph $G(E)$ (right).

$\min_{X \subseteq feat(E)} rw(r_X(G(E)))$ and the *renamable treewidth* of E as $tw^*(G(E)) = \min_{X \subseteq feat(E)} tw(r_X(G(E)))$. For two integer-valued parameters p and q, we say that p *dominates* q if, for every class of instances for which q is bounded, p is also bounded; p *strictly dominates* q if p dominates q, but q does not dominate p. If p and q dominate each other, they are *domination-equivalent*. The following statements (proof omitted) certify the robustness and generality of rank-width for the DT problem.

- *Rank-width and renamable rank-width are domination-equivalent.*
- *Rank-width strictly dominates renamable treewidth.*
- *Renamable treewidth strictly dominates treewidth.*

The main ingredient of our algorithm for rank-width is a dynamic programming (DP) leaf-to-root algorithm along a k-NLC-expression tree of the incidence graph of the CI E. Note that we use NLC-width and NLC-expression trees, a width parameter well known to be domination-equivalent to rank-width (Wanke 1994), instead of rank-width and rank-decompositions directly. This allows us to simplify the presentation of our algorithm.

The main challenge we had to overcome was to devise a succinct and correct definition of the records computed for every node of the given NLC-expression tree B and develop and introduce the correct notions and operations to allow us to construct such records. In particular, a (valid) record for a node b of B needs to provide a compact representation of an equivalence class of solutions, i.e. DTs for the whole instance E, relative to the current subinstance containing only the features and examples within the subtree of B rooted at b. We overcome this obstacle by introducing various variants of DTs (such as DT templates and skeletons) that allow us to represent and argue about these equivalence classes compactly. We also introduce various novel operations that work for DTs as well as our abstractions of DTs, such as feature relabellings, contractions and reductions, which are interesting in their own right and may be helpful in future algorithmic applications of DTs.

DP algorithms using decompositions are well understood and can often be easily designed for problems on graphs or, more generally, for problems whose solutions can be represented in terms of the graph for which the decomposition is given. However, a DP approach can become challenging for problems whose solutions have no or only minor resemblance to the graph for which the decomposition is given, as is the case for DTs; this also prevents a simple application of Courcelle's Theorem (Courcelle 1990). A prominent example is the celebrated result by Bodlaender (1996), where he uses a 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 a graph's exact tree-depth (Reidl et al. 2014) or clique-width (Espelage, Gurski, and Wanke 2003) using an optimal tree decomposition.

Preliminaries

Graphs and NLC-width We will assume that the reader is familiar with basic graph theory (see e.g. Bang-Jensen and Gutin 2009; Diestel 2017). We consider (vertex and edge labelled) undirected graphs. Let $G = (V, E)$ be a 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 $v \in V$ by $\{u, v\}$. 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 $G - E'$ denote the graph with vertex set V and edge set $E \setminus E'$.

A *k-graph* is a pair (G, λ), where $G = (V, E)$ is a graph and $\lambda : V \to [k]$ is a *vertex label mapping* that labels every vertex $v \in V$ with a label $\lambda(v)$ from $[k] = \{1, \ldots, k\}$. An *initial k-graph* is a k-graph consisting of exactly one vertex v (say, with label i) and is denoted by $i(v)$.

Node label control-width (*NLC-width*) is a graph parameter defined as follows (Wanke 1994): let $k \in \mathbb{N}$ be a positive integer. A *k-NLC-expression tree* of a graph $G = (V, E)$ is a subcubic tree B, where every node b of B is associated with a k-graph (denoted by (G_b, λ_b)), such that:

1. Every leaf represents an initial k-graph $i(v)$ with $i \in [k]$ and $v \in V$.
2. Every non-leaf node b with one child c is a *relabelling node* and has an associated relabelling function $R_b : [k] \to [k]$. Moreover, (G_b, λ_b) is obtained from (G_c, λ_c) by relabelling all vertices of G_c with label i to label $R_b(i)$ for every $i \in [k]$.
3. Every non-leaf node b with two children, i.e. a left child l and a right child r, is a *join node* and has an associated *join matrix*, i.e. a binary $k \times k$ matrix M_b. Moreover, (G_b, λ_b) is obtained from the disjoint union of (G_l, λ_l) and (G_r, λ_r) by adding an edge from all vertices labelled i in G_l to all vertices labelled j in G_r whenever $M_b[i, j] = 1$.
4. G is equal to G_r for the root node r of B.

The NLC-width of a graph G, denoted by *nlcw*(G), is the minimum k for which G has a k-NLC-expression tree. A k-NLC-expression tree is *nice* if every relabelling node has a relabelling function $R : [k] \to [k]$ such that for some $i, j \in [k]$, $R(i) = j$ and $R(\ell) = \ell$ for all $\ell \in [k] \setminus \{i\}$. Let b be a node in a k-NLC-expression tree of a graph G. We let V_b denote the set of vertices of G_b. By the definition of a

k-NLC-expression tree, if $u, v \in V_b$ have the same label in (G_b, λ_b) and $w \in V(G) \setminus V_b$, then u is adjacent to w in G if and only if v is. Computing the NLC-width of a graph is NP-hard (Gurski and Wanke 2005). However, the following result holds.

Proposition 0.1. *Let G be a graph and k be an integer. There is an fpt-algorithm for parameter k which either correctly concludes that G has NLC-width larger than k or outputs a nice 2^k-NLC-expression tree for G with $\mathcal{O}(2^k n)$ nodes.*

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 let $\mathit{feat}(E) = \bigcup_{e \in E} \mathit{feat}(e)$. 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* (Ibaraki, Crama, and Hammer 2011), $E = E^+ \uplus E^-$ 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 CI E is an NP-hard task (Ibaraki, Crama, and Hammer 2011, Theorem 12.2).

We define the *incidence graph* of E, denoted by $G(E)$, to be 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 and only if $f(e) = 1$.

Decision Trees A *decision tree* (DT) is a rooted binary tree T with vertex set $V(T)$ and edge set $E(T)$ such that each leaf node is either a *positive* or a *negative* leaf and the following holds for each non-leaf t of T:

- t is labelled with a feature denoted by $\mathit{feat}_T(t)$ or simply $\mathit{feat}(t)$ if T is clear from the context,
- t has two children, i.e. a left child and a right child.

We write $\mathit{feat}(T) = \{\, \mathit{feat}(t) \mid t \in V(T) \,\}$ to denote the set of all features used by T. The *size* of T is its number of nodes i.e. $|V(T)|$.

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

Let E be a CI and let T be a DT with $\mathit{feat}(T) \subseteq \mathit{feat}(E)$. For each node t of T, we let τ_t denote the partial feature assignment $\tau_t : \mathit{feat}(\mathrm{anc}(t)) \to \{0, 1\}$ defined by setting $\tau_t(f) = 0$ if $f \in \mathit{feat}(\mathrm{anc}^L(t))$ and $\tau_t(f) = 1$ if $f \in \mathit{feat}(\mathrm{anc}^R(t))$. We let $E_T(t)$, or simply $E(t)$ if T is clear from the context, denote the set $E[\tau(t)]$ of examples, where $E[\tau(t)]$ is the set of all examples e in E with $e(f) = \tau_t(f)$ for every feature $f \in \mathrm{anc}(t)$. We say that T *classifies* an example $e \in E$ if e is a positive (negative) example and $e \in E_T(l)$ for a positive (negative) leaf l of T. We say that T *is a DT for* E if T classifies all examples in E. See Figure 1 for an illustration of a CI, its incidence graph, and a DT that classifies E.

We will consider the following optimisation problem.

MINIMUM DECISION TREE SIZE (DTS)	
Input:	A CI E.
Question:	Find a DT of smallest size that classifies E.

We will need the following auxiliary lemma.

Lemma 0.2. *Let A be a set of features. Then, the number of DTs of size at most s that use only features in A is at most $|A|^{2s+1}$ and these can be enumerated in $\mathcal{O}(|A|^{2s+1})$ time.*

An Algorithm for Rank-width

In this section, we show that DTS parameterized by the rank-width of the incidence graph of the CI is fixed-parameter tractable. To simplify the proof and its description, we will provide the result for NLC-width. Because of Proposition 0.1, it suffices to show the result for the case when we are provided with a nice k-NLC-expression tree.

Theorem 0.3. *Let E be a CI and let B be a nice k-NLC-expression tree for $G(E)$. Then, finding a DT of smallest size that classifies E is fixed-parameter tractable parameterized by k.*

Corollary 0.4. DTS *is fixed-parameter tractable parameterized by NLC-width.*

The remainder of this section is devoted to a proof of Theorem 0.3. In principle, we will use a dynamic programming algorithm along the k-NLC-expression tree B of $G(E)$ that computes a set $\mathcal{R}(b)$ of valid records for every node b of B in a bottom-up (leaf-to-root) manner. The crucial part will be the correct definition of the records. To simplify the presentation, we will write $\mathit{feat}(b)$, $\mathit{exam}(b)$, and E_b for the sets $\mathit{feat}(E) \cap V_b$, $E \cap V_b$, and the subinstance of E induced by the features in $\mathit{feat}(b)$ and the examples in $\mathit{exam}(b)$. We will also say that the features and examples in $\mathit{feat}(b) \cup \mathit{exam}(b)$ and $(\mathit{feat}(E) \setminus \mathit{feat}(b)) \cup (E \setminus \mathit{exam}(b))$ are *old* and *novel*, respectively.

Informal Description of the Algorithm

Here, we will informally describe the main ideas behind the definition of the records for a node b of B. Intuitively, a (valid) record for b will represent a compact representation of an equivalence class of solutions (DTs) for the whole instance from the perspective of the subinstance E_b. Therefore, to illustrate the main ideas and motivations for the later definition of our records, we need to understand how a solution for the whole instance, i.e. a DT T that classifies E, behaves from the perspective of the subinstance E_b and what is the information that one needs to store about T.

We start by analysing the role of novel features used by the nodes in T. In particular, let f be a novel feature used by T, and let $e \in \mathit{exam}(b)$ be an old example. Then, because

B is a k-NLC expression tree for $G(E)$, the value $e(f)$ of f for e only depends on the current label $\lambda_b(e)$ of the example e in (G_b, λ_b). Therefore, the role of f (and consequently the role of all novel features) with respect to the old examples in $\mathit{exam}(b)$ can be described by the set of labels $L \subseteq [k]$ such that $e(f) = 1$ if and only if $\lambda_b(e) \in L$. This allows us to replace every novel feature f in T with a so-called future feature f_L, whose behaviour towards the old examples is completely described by the set $L \subseteq [k]$ of labels, i.e. $e(f_L) = 1$ if and only if $\lambda_b(e) \in L$ for every $e \in \mathit{exam}(b)$.

Let T^1 be the tree obtained from T after replacing all novel features with their corresponding future features; we will later introduce so-called feature relabellings that will allow us to relabel features of a DT in a very general manner. Then, T^1 is still a DT for $\mathit{exam}(b)$, and moreover, the future features now act as placeholders for the novel features and can be replaced later on during the algorithm. Crucially, we can now also potentially significantly reduce the size of T^1. This is because instead of keeping track of the arbitrarily many novel features in T, we now only need to keep track of the at most 2^k future features (one for every $L \subseteq [k]$).

In particular, consider a node t of T^1 with $\mathit{feat}(t) = f_L$ for some set $L \subseteq [k]$ of labels and let $A_{T^1}(t)$ be the set of *filtered labels* (see the following subsection for a formal definition of $A_{T^1}(t)$). Informally, $A_{T^1}(t)$ is the set of all labels $\ell \in [k]$ such that all examples $e \in \mathit{exam}(b)$ with label ℓ would end up at t if only the effect of the future features on the path from the root of T^1 to t is considered. Then, we say that t is *left (right) redundant* in T^1 if $A_{T^1}(t) \subseteq L$ ($A_{T^1}(t) \subseteq \overline{L}$). Intuitively, t is left (right) redundant if all examples that can reach t (considering the influence of the future features only) end up in the left (right) child of t; and therefore the other child of t is redundant because it will never receive any examples. To make use of this fact (and to remove redundant nodes), we will define the following left (right) contraction operation for left (right) redundant nodes in a DT. Let D be a DT and $d \in V(D)$ be an inner node of D with left child ℓ, right child r, and parent p. We say that D' is obtained from D after *left (right) contracting* d if D' is the DT obtained from D after removing d together with all nodes in D_r (D_ℓ) and adding the edge between p and ℓ (r); if d has no parent, then no edge is added. Therefore, if t is left (right) redundant in T^1, we can left (right) contract t in T^1 without changing the behaviour of T^1 towards the old examples in $\mathit{exam}(b)$.

Let T^2 be the tree obtained from T^1 after left (right) contracting every left (right) redundant node t assigned to a future feature. Note that T^2 is still a DT for $\mathit{exam}(b)$ (for a formal proof see Observation 0.6). Moreover, it is easy to see (and will be shown formally in Observation 0.8) that any root-to-leaf path of T^2 contains at most k nodes assigned to future features. We will later call T^2 a DT template for b.

Let us now consider the role of the old features used in T^2 and, in particular, what information we need to store about the structure (distribution) of these features in T^2. Note that if we have a record at node b, we may assume that we already verified that the record can be used to classify all old examples correctly. Therefore, we only need to store sufficient information about how the structure of the old features affects how the record can be extended when novel examples (and features) are added to the subinstance. In other words, we only need to know how the structure of the old features in T^2 affects the novel examples. In particular, let f be an old feature used by T^2 and let $e \in E \setminus \mathit{exam}(b)$ be a novel example. Because B is a k-NLC expression tree for $G(E)$, the value of $e(f)$ only depends on the current label $\lambda_b(f)$ of the feature f in (G_b, λ_b). Therefore, the role of f (and consequently the role of all old features) with respect to novel examples can be described by the label of the old feature in (G_b, λ_b). This allows us to replace every old feature f in T^2 with a so-called forgotten feature f_l, which will behave toward all novel examples like every old feature $f \in \mathit{feat}(b)$ with label l.

Let T^3 be obtained from T^2 by relabelling every old feature into its corresponding forgotten feature. Then, T^3 behaves in the same manner as T^2 with respect to all novel examples. Moreover, like T^1, T^3 now also contains potentially many nodes that are redundant with respect to their behaviour towards novel examples; in T^1 the nodes were redundant concerning their behaviour towards old examples. To reduce T^3, we say that a node t of T^3 assigned to some forgotten feature f_l is left (right) redundant if t has a left (right) ancestor t_A that is assigned to the same forgotten feature as t. Similarly to before, we can now left (right) contract every left (right) redundant node in T^3 without changing the behaviour of T^3 towards the novel examples.

Let T^4 be obtained from T^3 after left (right) contracting all left (right) redundant nodes. Note that every root-to-leaf path of T^4 can now contain at most k future features and at most k forgotten features, which implies that T^4 has bounded height and therefore also bounded size. T^4 will later be called a DT skeleton for b, and it contains all the information about the structure of T that we need to keep in the record corresponding to T. In particular, the record corresponding to T will be the pair (T^4, s), where s is the number of nodes assigned to forgotten features in T^3 that we have removed to obtain T^4, i.e. $s = |V(T^4) \setminus V(T^3)|$. Moreover, informally, we say that such a record is valid if s is the smallest number such that there is a DT template for b that reduces to T^4 (in the same manner that T^2 reduced to T^4). This completes the description of the main ideas behind our records, and we are now ready to provide formal definitions and proofs.

Formal Definition of Records and Preliminary Results

In what follows, let E be a CI and let B be a k-NLC-expression tree for $G(E)$. Consider a node b of B. Let L be a set of labels (usually $L = [k]$). For a subset $L' \subseteq L$, we let $\overline{L'}$ denote the set $L \setminus L'$. For a label $\ell \in L$, we introduce a new feature f_ℓ, which we will call a *forgotten feature*. Moreover, for a subset $L' \subseteq L$ of labels, we introduce a new feature $f_{L'}$, which we call an *future (or introduce) feature*. Let $F_L = \{ f_\ell \mid \ell \in L \}$ be the set of all forgotten features and let $I_L = \{ f_{L'} \mid L' \subseteq L \}$ be the set of all future features with respect to L. To distinguish features in $\mathit{feat}(E)$ from forgotten and future features, we will sometimes refer to them as *real features*.

Let T be a DT and $t \in V(T)$ be an inner node of T with

left child l, right child r, and parent p. We say that T' is obtained from T after *left (right) contracting* t if T' is the DT obtained from T after removing t together with all nodes in T_r (T_ℓ) and adding the edge between p and l (r); if t has no parent, then no edge is added.

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

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

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

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

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

Lemma 0.5. *Let T be a DT (skeleton/template), let $t \in V(T)$ be a left (right) redundant node in T, and let T' be the DT (skeleton/template) obtained from T after left (right) contracting t. Then, a node $t' \in V(T')$ is left (right) redundant in T' if and only if t' is left (right) redundant in T.*

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

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

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

Observation 0.6. *Let T be a DT template for b. Then $r(T)$ is also a DT template for b.*

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 *feat*$(t) \in$ *feat*(b), then t is left (right) redundant because of some left (right) ancestor t' with *feat*$(t') =$ *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 *feat*$(t) =$ *feat*(t')). If, on the other hand, *feat*(t) is some future feature f_L, then $A(t) \subseteq \overline{L}$ ($A(t) \subseteq L$) and because $e \in E_T(t)$, it follows that $\lambda_b(e) \in A(t)$. Therefore, e goes to the left (right) child of t. □

The next observation shows the similar behaviour of T and $r(T)$ with respect to future examples in $E \setminus$ *exam*(b).

Observation 0.7. *Let T be a DT (skeleton/template) for b, and let e be an example in $E \setminus$ exam(b) that is correctly classified by T. Then, e is also correctly classified by $r(T)$.*

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

Observation 0.8. *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 $2^{k_F+k_I+1}$.*

We obtain the following corollary as a special case.

Corollary 0.9. *Let T be a reduced DT skeleton for a node $b \in V(B)$. Then, T has height at most $2k + 1$ and size at most 2^{2k+1}.*

Observation 0.10. *There are at most $(k_F + 2^{k_I})^{2^{k_F+k_I+2}+1}$ reduced DT skeletons whose forgotten features use a set of at most k_F labels and whose future features use a set of at most k_I labels. Moreover, these can be enumerated in $\mathcal{O}((k_F + 2^{k_I})^{2^{k_F+k_I+2}+1})$ time.*

We obtain the following corollary as a special case.

Corollary 0.11. *There are at most $(k + 2^k)^{2^{2k+2}+1}$ reduced DT skeletons for a node $b \in V(B)$ and these can be enumerated in $\mathcal{O}((k + 2^k)^{2^{2k+2}+1})$ time.*

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

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

We now show an important property on the interchangeability of feature relabellings and reductions. That is, we show in Lemma 0.13 below that the effect of any sequence of feature relabellings and reductions that ends with the reduction operation ($r()$) is the same as the effect of the sequence that contains the same relabelling operations followed by one reduction operation at the end. To show this property, we need the following auxiliary lemma.

Lemma 0.12. *Let T be a DT (template/skeleton) for a node $b \in V(B)$ and let α be a feature relabelling. If a node $t \in V(T)$ is left (right) redundant in T, then it is also left (right) redundant in $\alpha(T)$.*

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

Lemma 0.13. *Let T be a DT (template/skeleton) and let α be a feature relabelling. Then, $r(\alpha(T)) = r(\alpha(r(T)))$.*

Proof. Let T' be the DT (template/skeleton) obtained from $\alpha(T)$ after left (right) contracting every node t that is left (right) redundant in T; note that such a node t is also left (right) redundant in $\alpha(T)$ because of Lemma 0.12. Then, $T' = \alpha(r(T))$ and, because of Lemma 0.5 (and using the fact that every node t that is left (right) redundant in T is so in $\alpha(T)$), a node $t \in V(T')$ is left (right) redundant in T' if and only if it is so in $\alpha(T)$. Therefore, a node t is left (right) redundant in $\alpha(T)$ if and only if it is left (right) redundant in T or in $\alpha(r(T)) = T'$, which shows that $r(\alpha(T)) = r(\alpha(r(T)))$. □

We are now ready to define the records and their semantics. A *record* for b is a pair (T, s) such that T is a reduced decision tree skeleton for b and s is a natural number. We say that a record (T, s) is *semi-valid* for b if there is a (reduced) DT template T' for b such that $r(\alpha_b^s(T')) = T$ and $s = |V(T') \setminus V(T)|$. We say that a record (T, s) is *valid* for b if s is the minimum number such that (T, s) is semi-valid. We let $\mathcal{R}(b)$ denote the set of all valid records for b. The following corollary follows immediately from Corollary 0.11.

Corollary 0.14. $|\mathcal{R}(b)| \leq (k + 2^k)^{2^{2k+2}+1}$.

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 and $s = |V(T)| + s'$. Therefore, given the set $\mathcal{R}(r)$ of valid records for the root r of B, we can solve DTS for E by outputting the smallest integer s such that $\mathcal{R}(r)$ contains a record (T, s') such that T is complete and $s = |V(T)| + s'$.

Proof of the Main Result

We will now show that we can compute $\mathcal{R}(b)$ for each of the three node types of a nice k-NLC expression tree provided that $\mathcal{R}(c)$ has already been computed for every child c of b. Recall that b is either a leaf node associated with a k-graph $i(v)$, a relabelling node with one child and with relabelling function R_b, or a join node with a left child, a right child and a join matrix M_b. Moreover, recall that (G_b, λ_b) is the k-graph associated with b (whose unlabelled version is a subgraph of G) and V_b is the set of vertices of G_b.

Lemma 0.15 (leaf node)**.** *Let $b \in V(B)$ be a leaf node. Then $\mathcal{R}(b)$ can be computed in $\mathcal{O}(k(1+2^k)^{2^{k+3}+1})$ time.*

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 0.10, since the time required to enumerate all these reduced DT skeletons is at most $\mathcal{O}((1+2^k)^{2^{k+3}+1})$.

If 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 classifies v. By Observation 0.10, these can be enumerated in $\mathcal{O}((1+2^k)^{2^{k+3}+1})$ time and checking for each of these whether it correctly classifies v can be achieved in $\mathcal{O}(k)$ time by Observation 0.8. □

Lemma 0.16 (join node)**.** *Let $b \in V(B)$ be a join node. Then $\mathcal{R}(b)$ can be computed in $\mathcal{O}(2^{3k+1}(2k+2^k)^{2^{3k+2}+1})$ time.*

Proof sketch. 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 $A_{i,*} = \{j \in [k] \,|\, M_b[i,j] = 1\}$ and $A_{*,i} = \{j \in [k] \,|\, M_b[j,i] = 1\}$.

To distinguish between forgotten features from the left and the right subtree, we introduce the left version i_L and the right version i_R for every label $i \in [k]$. With a slight abuse of notation, we also let $[k_L]$ denote the set $\{1_L, \ldots, k_L\}$ of (left) labels and let $[k_R]$ denote the set $\{1_R, \ldots, k_R\}$ 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 0.10, these can be enumerated in $\mathcal{O}((2k+2^k)^{2^{3k+2}+1})$ time. For every such reduced DT skeleton T, we now do the following in order to decide whether T gives rise to a valid record for b. Let $\alpha_{LR\to} : F_{[k_L]} \cup F_{[k_R]} \to F_{[k]}$ be the feature relabelling that relabels every (left/right) feature $f_{i_H} \in F_{[k_L]} \cup F_{[k_R]}$ (for some $H \in \{L, R\}$) to its original feature f_i.

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

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

Let $\hat{T} = r(\alpha_{LR\to}(T))$ and $\hat{s} = |V(T) \setminus V(\hat{T})|$. We now check whether there are records $(T_L, s_L) \in \mathcal{R}(b_L)$ and $(T_R, s_R) \in \mathcal{R}(b_R)$. If not, we discard T and if yes, then we add the record $(\hat{T}, s_L + s_R + \hat{s})$ to $\mathcal{R}(b)$. This completes the description about how the records $\mathcal{R}(b)$ are computed. We omit the proof of correctness and run-time. □

Lemma 0.17 (relabelling node)**.** *Let $b \in V(B)$ be relabelling node in B. Then $\mathcal{R}(b)$ can be computed in $\mathcal{O}(2^{2k+1}(k+2^k)^{2^{2k+2}+1})$ time.*

We are now ready to prove the main result of this section.

Proof of Theorem 0.3. Given E and B, we use Lemmas 0.15, 0.16 and 0.17 to compute the set $\mathcal{R}(b)$ of valid records for every node b of B in a leaf-to-root manner. We then go through all records in $\mathcal{R}(r)$ for the root r of B to find a record (T, s) such that $|V(T)| + s$ is minimum among all such records where T is complete. The correctness of the algorithm follows directly from the definition of the valid records together with Lemmas 0.15, 0.16 and 0.17. Moreover, we obtain the run-time of the algorithm as follows. The maximum time spent on any of the nodes b (which is achieved for a join-node) is $\mathcal{O}(2^{3k+1}(2k+2^k)^{2^{3k+2}+1})$. Because B has at most $\mathcal{O}(k|E \cup \mathit{feat}(E)|)$ (see Proposition 0.1) nodes, we obtain $\mathcal{O}(2^{3k+1}(2k+2^k)^{2^{3k+2}+1}k|E \cup \mathit{feat}(E)|)$ as the total run-time of the algorithm, which shows that DTS is fixed-parameter tractable parameterized by the NLC-width (and therefore also the rank-width) of E. □

Conclusion

We have initiated the study of the parameterized complexity of learning DTs from data with respect to structural parameters. Our main result provides novel insights into the structure of DTs and utilises succinct representations of classes of DTs. It exploits the low rank-width of the incidence graph of a given CI, which constitutes a well-suited, general and robust measure for the complexity of input data.

For instance, one often wants to learn DTs of small depth instead of learning DTs of small size. Therefore, asking whether our approach extends in this setting is natural. While one can adapt our approach to obtain an XP-algorithm for learning DTs of small depth, parameterized by rank/NLC-width, it is not clear whether this problem is fixed-parameter tractable. Another question is whether one can extend our approach to CIs where features range over an arbitrary (non-Boolean) ordered domain. In this case, one usually uses DTs that make binary decisions (i.e. whether a feature is smaller than, equal to, or larger than a given threshold). While our approach can be extended if every feature has a domain of bounded size (using the rank-width for edge-coloured graphs (Kanté and Rao 2013) for the complete bipartite incidence graph whose edges are coloured with domain values), it is not clear what happens if the domains are allowed to grow arbitrarily.

Acknowledgments

Sebastian Ordyniak acknowledges support from the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1). Stefan Szeider acknowledges support from the Austrian Science Fund (FWF, projects P36420 and P36688), and from the Vienna Science and Technology Fund (WWTF, project ICT19-065).

References

Bang-Jensen, J.; and Gutin, G. 2009. *Digraphs*. Springer Monographs in Mathematics. London: Springer-Verlag London Ltd., second edition.

Bessiere, C.; Hebrard, E.; and O'Sullivan, B. 2009. Minimising Decision Tree Size as Combinatorial Optimisation. *Proc. CP 2009*, 5732: 173–187.

Bodlaender, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6): 1305–1317.

Boros, E.; Crama, Y.; Hammer, P. L.; Ibaraki, T.; Kogan, A.; and Makino, K. 2011. Logical analysis of data: classification with justification. *Annals of Operations Research*, 188(1): 33–61.

Boros, E.; Gurvich, V.; Hammer, P. L.; Ibaraki, T.; and Kogan, A. 1995. Decomposability of Partially Defined Boolean Functions. *Discrete Applied Mathematics*, 62(1-3): 51–75.

Boros, E.; Horiyama, T.; Ibaraki, T.; Makino, K.; and Yagiura, M. 2003. Finding Essential Attributes from Binary Data. *Annals of Mathematics and Artificial Intelligence*, 39: 223–257.

Boros, E.; Ibaraki, T.; and Makino, K. 2003. Variations on extending partially defined Boolean functions with missing bits. *Information and Computation*, 180(1): 53–70.

Courcelle, B. 1990. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1): 12–75.

Crama, Y.; Hammer, P. L.; and Ibaraki, T. 1988. Cause-effect relationships and partially defined Boolean function. *Annals of Operations Research*, 16: 299–326.

Darwiche, A.; and Hirth, A. 2023. On the (Complete) Reasons Behind Decisions. *Journal of Logic, Language and Information*, 32(1): 63–88.

Diestel, R. 2017. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. New York: Springer Verlag, 5th edition.

Doshi-Velez, F.; and Kim, B. 2017. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv.org*, (1702.08608).

Espelage, W.; Gurski, F.; and Wanke, E. 2003. Deciding Clique-Width for Graphs of Bounded Tree-Width. *Journal of Graph Algorithms and Applications*, 7(2): 141–180.

Goodman, B.; and Flaxman, S. R. 2017. European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, 38(3): 50–57.

Gurski, F.; and Wanke, E. 2005. Minimizing NLC-width is NP-complete. *Proc. WG 2005*, 3787: 69–80.

Hyafil, L.; and Rivest, R. L. 1976. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, 5(1): 15–17.

Ibaraki, T.; Crama, Y.; and Hammer, P. L. 2011. *Partially defined Boolean functions (in Boolean Functions: Theory, Algorithms, and Applications)*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Ignatiev, A.; Marques-Silva, J.; Narodytska, N.; and Stuckey, P. J. 2021. Reasoning-Based Learning of Interpretable ML Models. *Proc. IJCAI 2021*, 4458–4465.

Kanté, M. M.; and Rao, M. 2013. The Rank-Width of Edge-Coloured Graphs. *Theory of Computing Systems*, 52(4): 599–644.

Larose, D. T.; and Larose, C. D. 2014. *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, 2nd edition.

Lewis, H. R. 1978. Renaming a Set of Clauses as a Horn Set. *Journal of the ACM*, 25(1): 134–135.

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

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

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

Murthy, S. K. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4): 345–389.

Ordyniak, S.; and Szeider, S. 2021. Parameterized Complexity of Small Decision Tree Learning. *Proc. AAAI 2021*, 6454–6462.

Quinlan, J. R. 1986. Induction of Decision Trees. *Machine Learning*, 1(1): 81–106.

Reidl, F.; Rossmanith, P.; Villaamil, F. S.; and Sikdar, S. 2014. A Faster Parameterized Algorithm for Treedepth. *Proc. ICALP 2014*, 8572: 931–942.

Wanke, E. 1994. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3): 251–266.