A General Theoretical Framework for Learning Smallest Interpretable Models

Sebastian Ordyniak[a], Giacomo Paesani[b], Mateusz Rychlicki[a], Stefan Szeider[c]

[a]*University of Leeds, 183 Woodhouse Lane, Leeds, LS2 9HD, UK*
[b]*Sapienza University of Rome, Piazzale Aldo Moro 5, Rome, 00185, Italy*
[c]*Algorithms and Complexity Group, TU Wien, Favoritenstrasse 9–11, Wien, 1040, Austria*

Abstract

We develop a general algorithmic framework that allows us to obtain fixed-parameter tractability for computing smallest symbolic models that represent given data. Our framework applies to all ML model types that admit a certain extension property. By establishing this extension property for decision trees, decision sets, decision lists, and binary decision diagrams, we obtain that minimizing these fundamental model types is fixed-parameter tractable. Our framework even applies to ensembles, which combine individual models by majority decision.

1. Introduction

The modern highly successful subsymbolic Machine Learning models like neural networks can exhibit lack of robustness, display bias, and their operation is invariably inscrutable for human decision makers [11]. Hence, symbolic models such as decision trees, decision lists and sets, and binary decision diagrams have recently received new attention as they are easier to analyze and control and more interpretable [24, 21]. However, also symbolic models become increasingly opaque if their size increases. Hence one is interested in finding a model of smallest size that still classifies all examples correctly. This task is typically NP-hard (see [14] for the case of DTs and our hardness results in Section 9 for the remaining models); thus, practitioners have utilized powerful tools like SAT, MIP, and CP solvers to compute small or even smallest models that fit the data [15, 22, 25, 26]. Also, from a theoretical perspective, the parameterized complexity of finding smallest decision trees or ensembles of decision trees has become the subject of intensive research [23, 7, 17, 18] which revealed that the problem is fixed-parameter tractable when parameterized by the solution size plus a bound δ on the number of features any two examples differ[1]. Such studies are still pending for other symbolic ML model types like decision lists, decision sets, binary decision diagrams, and ensembles.

In this paper, we significantly extend this line of research to all the mentioned symbolic model types. However, instead of developing specialized algorithms for all the model types, we develop a general algorithmic framework that applies to all of them and any further model type that admits one of two natural extension properties: *strong extendability* and (weak) *extendability*. Therefore, to show that a particular type of model admits the efficient[2] computation of a smallest model,

[1]Empirical investigations show that δ is reasonably small for real-world data sets [23].
[2]Throughout the paper, our notion of efficiency includes fixed-parameter tractable algorithms.

	strongly extendable?	# branches	
		single	ensemble
DS	✓	δ	$2 + s\delta$
DL	✓	$\delta + s + 1$	$1 + s(1 + \delta)$
DT	✓	$\delta(s + 1)$	$2 + 2s\delta$
BDD	-	$\delta 3^{O(s)}$	$\delta 3^{O(s)}$

Table 1: The number of branches required by our framework for DSs, DLs, DTs, and BDDs, both for learning simple models and for learning ensemble models. All model types except BDDs are strongly extendable and allow for polynomial number of branches. The run-time of our algorithm to learn a small model (ignoring polynomial factors) only depends on the number of branches b and is given by $O^*(b^2)$.

one only needs to show that the model type admits one of the two extension properties. This way, we generalize the fixed-parameter tractability results from *decision trees* (DTs) to *decision sets* (DSs), *decision lists* (DLs), *binary decision diagrams* (BDDs), and even *ensembles* of all these model types.

Our framework uses a bounded-depth branching algorithm that, starting from the empty model, exhaustively branches into all "important" extensions, i.e., all extensions that could potentially be part of an optimal model, of the current model until either a small model is found or the framework correctly returns that no model of the required size (in the following denoted by s) exists. The main challenge behind the algorithm is to restrict the number of important extensions of the current model that need to be considered at every step. In particular, it is crucial to bound the number of additional features whose addition to the model is required for an exhaustive enumeration of all minimum models. We employ two main approaches which lead to strong and (weak) extendability, respectively. For DSs, DLs, and DTs, we can employ an adapted version of the annotation approach that has recently been developed for DTs [18]. Here, parts of the model are annotated with examples that allow us to guide the selection of important features. While the approach is similar to the approach of Komusiewicz et al. [18], we show that it can also be applied to DSs and DLs (as well as ensembles thereof), and we also manage to simplify the approach significantly for the case of DTs. Indeed, we can simplify the annotation by showing that it suffices to annotate only with already correctly classified examples. We also simplify their correctness proof by defining extension in a declarative manner instead of explicitly in terms of operations (that are required to obtain the extension). This allows us not to have to ensure that the operations are invariant under reordering, which led to an unnecessarily technical proof used in previous work [18]. Surprisingly, the annotation approach does not seem to apply to BDDs. In this case, we are, however, able to adapt the ideas (most notably the notion of "useful" sets of features) behind the first algorithm for DTs given by Ordyniak and Szeider [23] to show that BDDs, as well as their ensembles, can be learned efficiently. Our algorithmic results are summarized in Table 1. There we state the number of branches that our framework requires to extend the current model for each of the considered model types, since this is the main parameter that influences the run-time of our framework for learning a smallest model of size at most s, which is then given by $O^*(b^s)$ (O^* suppresses polynomial factors). Interestingly, the number of branches dramatically differs between the different model types and their ensembles, particularly between the strongly extendable models using the first approach and the (weakly) extendable models using the second approach.

We complement our algorithmic results with hardness results. First, we show that similar to decision trees, also for the other model types, the parameter δ is indispensable; when parameterized by solution size alone, we obtain W[2]-hardness. Second, we show that for decision sets and decision lists, we cannot replace the parameter solution size with either the total number of terms or the maximum size of a term by showing that, in this case, the problems remain NP-hard even for $\delta = 2$.

2. Related Work

Recent years have seen significant advances in exact methods for learning optimal interpretable models. This section reviews the key developments for different model types.

Decision Trees. Ordyniak and Szeider [23] initiated the line of research investigating the parameterized complexity of learning optimal models, showing that finding smallest decision trees is fixed-parameter tractable when parameterized by the solution size and a data diversity measure; this was later generalized from Boolean features to features having an arbitrary large domain [7]. Following this foundational work, Komusiewicz et al. [18] extended these methods to tree ensembles, while Kobourov et al. [17] explored the influence of dimensions on computational complexity. Moreover, Dabrowski et al. [4] showed that finding smallest decision trees is also fixed-parameter tractable parameterized by the rank-width of the input data. Recently, Harmender and Meirav [9] investigated the parameterized complexity of finding smallest decision trees that allow for a given number of falsily classified examples.

For the practical computation of optimal decision trees, Narodytska et al. [22] pioneered SAT encodings that find the smallest trees consistent with given data. This approach was subsequently improved by Avellaneda [3] through an efficient inference strategy and by Schidler and Szeider [25] with better scalability for larger datasets. In practise one often wants to find a decision tree of a given maximum size that minimizes the number of classification errors; in contrast to finding a smallest decision tree without classification errors. In this case, alternative approaches exists and include constraint programming methods by Verhaeghe et al. [27], which use AND/OR search structures, and MILP formulations by Verwer and Zhang [28], which handle numeric features through a binary encoding of threshold decisions.

Specialized search algorithms have also been developed, including Optimal Sparse Decision Trees (OSDT) by Hu et al. [13], which uses analytical lower bounds and bitvector-based computations, and its extension GOSDT by Lin et al. [19], which addresses imbalanced data and continuous features. Dynamic programming approaches include DL8.5 by Aglin et al. [1] and MurTree by Demirović et al. [6], which significantly improve efficiency through caching and specialized solvers for subtrees. Recent work by van der Linden et al. [20] has also incorporated fairness constraints into optimal decision tree learning.

Decision Lists and Sets. For decision lists, Angelino et al. [2] introduced CORELS, a branch-and-bound algorithm for learning optimal rule lists using a regularized objective. Yu et al. [29, 30] developed a SAT-based approach with their DLSAT algorithm, which finds minimum-size decision lists that perfectly classify training data. Ignatiev and Marques-Silva [15] contributed rigorous explanation methods for decision lists, enhancing their interpretability.

For decision sets, Ignatiev et al. [16] proposed SAT-based methods for learning explainable decision sets, which Yu et al. [30] later unified with decision lists in a MaxSAT framework. Other approaches include MaxSAT-based methods by Ghosh and Meel [10] and column generation techniques by Dash et al. [5], which learn Boolean rule sets through integer programming.

Binary Decision Diagrams. Research on optimal BDDs is more recent but showing promising results. Hu et al. [12] proposed the first complete approach using MaxSAT to learn optimal BDDs, including techniques to merge compatible subtrees for further compression. Florio et al. [8] developed a flexible MILP model for Optimal Decision Diagrams that can incorporate various constraints, while Shati et al. [26] extended SAT-based search to handle non-binary features in BDDs.

Comparative Insights. Experimental comparisons have revealed interesting trade-offs between model types. BDDs often achieve higher accuracy than decision trees of comparable complexity [12, 8] and show greater stability due to their ability to share substructures. Meanwhile, decision sets can sometimes provide shorter explanations for individual instances than decision lists [30], though decision lists offer more structured reasoning.

3. Preliminaries

Classification Instances. A *(binary) classification instance (CI)* is a triple $C = (E, F, \mu)$, where E is a set of examples over a set of binary features F and μ is a classification function $\mu : E \to \{0, 1\}$. We commonly say that an example e is a 0-example, or negative example, (1-example, or positive example) if $\mu(e) = 0$ ($\mu(e) = 1$) and we denote by $e(f)$ the value of the example $e \in E$ on the feature $f \in F$. The size of a C is given by $\|C\| = |E| \cdot |F|$.

We say that two examples e and e' agree (don't agree) on a feature f if $e(f) = e'(f)$ ($e(f) \neq e'(f)$) and denote with $\delta(e, e')$ the set of features on which e and e' disagree on. For a (partial) assignment $\tau : F' \to \{0, 1\}$, where $F' \subseteq F$, we denote by $E[\tau]$ the set of all examples in E that agree with τ, i.e., all examples e with $e(f) = \tau(f)$ for every feature $f \in F'$. For two partial assignments $\tau_1 : F_1 \to \{0, 1\}$ and $\tau_2 : F_2 \to \{0, 1\}$, where $F_1, F_2 \subseteq F$ and $F_1 \cap F_2 = \emptyset$, we denote by $\tau_1 \cup \tau_2$ the assignment $\tau : F_1 \cup F_2 \to \{0, 1\}$ defined by setting $\tau(f) = \tau_1(f)$ if $f \in F_1$ and $\tau(f) = \tau_2(f)$ if $f \in F_2$. Finally, $\delta(C)$, or simply δ if C is clear from the context, denotes the maximum size of $\delta(e, e')$ over all pairs of examples (e, e'), where $\mu(e) + \mu(e') = 1$.

In the following, let $C = (E, F, \mu)$ be a CI.

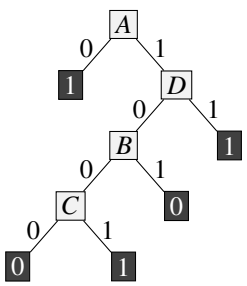
Models and Support Sets. In the following we will define models of different types (such as decision trees, decision sets, and decision lists); some of these are illustrated in Figure 1. Here, we will introduce some notation that applies to all models. Let M be a model. We denote by $F(M)$ the set of all features used by M. Moreover, we will denote by $M : E \to \{0, 1, \boldsymbol{u}\}$ the *classification function* defined by M, which *classifies* every example $e \in E$ as either 0, 1, or $\boldsymbol{u}$ (which means undefined). We say that M *classifies* e *correctly* if $M(e) = \mu(e)$ and we will say that M is a *model for* C if M classifies all examples of C correctly.

A set $S \subseteq F$ of features is a *support set* of C if it contains at least one feature from $\delta(e, e')$ for every pair (e, e') of 0-example e and 1-example e'. The following observation follows immediately from the fact that a model for a CI needs to at least be able to distinguish every 0-example from every 1-example.

Observation 1. *Let M be a model for a CI $C = (E, F, \mu)$. Then, $F(M)$ is a support set for C.*

Decision Sets. A *term* t over C is a set of *literals* with each literal being of the form $(f = z)$ where $f \in F$ and $z \in \{0, 1\}$. A *rule* r is a pair (t, c) where t is a term and $c \in \{0, 1\}$. We say that a rule (t, c) is a *c-rule*. We say that a term t (or rule (t, c)) *applies to (or agrees with)* an example

A	B	C	D	out
1	0	1	1	1
1	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	0	1	0	1
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1



if $A = 0$ then 1
if $A = 1 \wedge B = 0 \wedge C = 1$ then 1
default 0

if $A = 0$ then 1
elseif $B = 1$ then 0
elseif $C = 1$ then 1
elseif true then 0

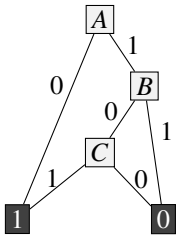


Figure 1: A classification instance C and four models that classify C: a decision tree, a decision set (top), a decision list (bottom), and a binary decision diagram (from left to right).

e if $e(f) = z$ for every element $(f = z)$ of t. Note that the empty term (or rule) applies to any example.

A *decision set* M is a pair (T, b), where T is a set of terms and $b \in \{0, 1\}$ is the classification of the default rule $r_D = (\emptyset, b)$. We denote by $\|M\|$ the size of M which is equal to $(\sum_{t \in T} |t|) + 1$; the $+1$ is for the default rule. The classification function $M : E \to \{0, 1\}$ of a DS $M = (T, b)$ is defined by setting $M(e) = b$ for every example $e \in E$ such that no term in T applies to e and otherwise we set $M(e) = 1 - b$.

Decision Lists. A *decision list* L is a sequence of rules $(r_1 = (t_1, c_1), \ldots, r_\ell = (t_\ell, c_\ell))$, for some $\ell \geq 0$. The size of a DL L, denoted by $\|L\|$, is equal to $\sum_{i=1}^{\ell}(|t_i| + 1)$. The classification function $L : E \to \{0, 1\}$ of a DL L is defined by setting $L(e) = b$ if the first rule in L that applies to e is a b-rule. For convenience, we set $L(e) = \boldsymbol{u}$ if no rule in L applies to e.

Decision Trees. A *decision tree* M is a pair (T, λ) such that T is a rooted binary tree with vertex set $V(T)$ and $\lambda : V(T) \to F \cup \{0, 1\}$ is a function that assigns a feature in F to every inner node of T and either 0 or 1 to every leaf node of T. Every inner node of T has exactly 2 children, one left child (or 0-child) and one right-child (or 1-child). The classification function $M : E \to \{0, 1\}$ of a DT $M = (T, \lambda)$ is defined as follows for an example $e \in E$. Starting at the root of T one does the following at every inner node t of T. If $e(\lambda(t)) = 0$ one continues with the 0-child of t and if $e(\lambda(t)) = 1$ one continues with the 1-child of t until one eventually ends up at a leaf node l at which e is classified as $\lambda(l)$.

For every node t of T, we denote by $E_M(t)$ the set of examples that reach t from the root of T. $E_M(t)$ can be defined recursively as follows: Set $E_M(r) = E$ for the root r of T and if t is an x-child of some (inner) node p, then we set $E_M(t) = E_M(p) \cap \{e \mid e(\lambda(p)) = x\}$ for every $x \in \{0, 1\}$. We denote by $\|M\|$ ($h(M) = h(T)$) the size (height) of a DT, which is equal to the number of leaves of T (the length of a longest root-to-leaf path in T).

Binary Decision Diagrams. A *binary decision diagram (BDD)* B is a pair (D, ρ) where D is a directed acyclic graph with three special vertices $\{s, t_0, t_1\}$ such that:

- s is a source vertex that can (but does not have to) be equal to t_0 or t_1,
- t_0 and t_1 are the only sink vertices of D,
- every non-sink vertex has exactly two outgoing neighbors, which we call the 0-neighbor and the 1-neighbor and

- $\rho : V(D) \setminus \{t_0, t_1\} \to F$ is a function that associates to every non-sink node of D a feature of E

For an example $e \in E$, we denote by $P_B(e)$ (or $P(e)$ if B is clear from the context), the unique path from s to either t_0 or t_1 followed by e in B. That is starting at s and ending at either t_0 or t_1, $P(e)$ is iteratively defined as follows. Initially, we set $P(e) = (s)$, moreover, if $P(e)$ ends in a vertex v other than t_0 or t_1, then we extend $P(e)$ by the $e(\rho(v))$-neighbor of v in D. Let B be a BDD and e an example of a CI (E, F, μ). The classification function $B : E \to \{0, 1\}$ of B is given by setting $B(e) = b$ if $P_B(e)$ ends in t_b. We denote by $\|B\|$ the size of B, which is equal to $|V(D)|$.

Ensembles. An *T-ensemble* $\mathcal{E}$ is a set of models of type T, where $T \in \{\mathtt{DS}, \mathtt{DL}, \mathtt{DT}, \mathtt{BDD}\}$. We say that $\mathcal{E}$ classifies an example $e \in E$ as b if so do the majority of models in $\mathcal{E}$, i.e., if there are at least $\lfloor |\mathcal{E}|/2 \rfloor + 1$ models in $\mathcal{E}$ that classify e as b. We denote by $\|\mathcal{E}\|$ the size of $\mathcal{E}$, which is equal to $\sum_{M \in \mathcal{E}} \|M\|$.

Considered Problems. Let $\Gamma \in \{\mathtt{DS}, \mathtt{DL}, \mathtt{DT}, \mathtt{BDD}\}$. We consider the following problems.

Γ-Minimum Model Size (Γ-MMS)	
Instance:	A CI $C = (E, F, \mu)$ and an integer s.
Question:	Find a model of type Γ and size at most s for C or report correctly that no such model exists.

Γ-Minimum Ensemble Model Size (Γ-MEMS)	
Instance:	A CI $C = (E, F, \mu)$ and an integer s.
Question:	Find an ensemble model of type Γ and size at most s for C or report correctly that no such model exists.

Note that all our (tractability) results still apply to the corresponding optimisation variants of the above problems, i.e., our algorithms are able to find a smallest (ensemble) model in fpt-time if the parameter s is replaced by the size of an optimal model.

4. The Framework

In this section we develop the framework for learning models as well as ensembles of models for different model-types. In the following sections, we will then show how this framework can be employed to learn (ensembles) of decision sets, decision lists, decision trees, and any form of binary decision diagrams. At its core the framework uses a bounded-depth branching algorithm that starting from an empty model branches over all simple extensions of the current model that could potentially be part of an optimum model until either an optimum model is found or it is shown that no optimal model of the required size exists. The framework can therefore be applied to all types of models, where the number of simple extensions of a model that can potentially lead to an optimum model is bounded by our parameters $s + \delta$ and those extensions can be computed efficiently[3]. Designing such a procedure, i.e., an efficient algorithm that given a model computes a small but complete set of extensions that can be part of an optimum model, for every model

[3]Note that parameterizing a problem by a set S of parameters is equivalent to parameterizing the problem by the sum of the parameters in S.

type is the main challenge of our approach. In particular, to achieve this one needs to design the update procedure in such a way that at every step only a small set of novel features need to be considered that must potentially be added to the current model.

The framework will need to deal with so-called partial models and so-called annotated models. That is, a partial model can be thought of as an incomplete model that by itself is not yet a model but can be completed into one and an annotated model can be thought of as a pair (M, A), where M is a model and A is an annotation of the model with examples that will help guide the search for possible extensions. While the exact definitions of these notions will depend on the particular type of model, for the purposes of presenting our framework we merely require them to satisfy the following natural properties.

(P0) Deciding whether a model M is a model for a CI $C = (E, F, \mu)$ and if not providing an example $e \in E$ that is not correctly classified by M can be achieved in time $O(|E|||M||)$.

(P1) Any model M is an extension of the empty partial/annotated model, denoted by `nil`.

(P2) If a partial/annotated model is a strict extension of another partial/annotated model, then the former is larger than the latter.

To make our framework work for all of our model types, we need to distinguish between two forms of "extendability", i.e., strong and (weak) extendability that we will introduce in the next two subsections.

4.1. Strong Extendability

In this section, we will introduce and develop our framework for strong extendable models, which as we will see later include `DS`s, `DL`s, and `DT`s. For all these model types, we will later introduce so-called annotated models, which can be thought of as a pair (M, A), where M is a model and A is an annotation of the model with examples that will help guide the search for possible simple extensions. The main advantage of strong extendability versus (weak) extendability is that model types that are strongly extendable automatically allow also for an efficient algorithm to learn ensembles of that model type.

We are now ready to provide a formal definition of strong extendability. Let (M, A) be an annotated model such that M is not a model for the CI $C = (E, F, \mu)$ and let $e \in E$ be an example not correctly classified by M. A *full set of strict extensions for an annotated model* (M, A) *and example* e is a set $\mathcal{E}$ of strict extensions of (M, A) such that every model M' that correctly classifies e and is an extension of (M, A) is also an extension of some annotated model in $\mathcal{E}$.

We say that a model-type T is *strongly* $(f(|M|, \delta), g(|M|, |C|))$*-extendable* for some computable functions $f(|M|, \delta)$ and $g(|M|, |C|)$ if there is an algorithm running in time $O(g(|M|, |C|))$ that given a CI C, an annotated model (M, A) of type T, and an example $e \in E$ that is not correctly classified by M computes a full set $\mathcal{E}$ of strict extensions for (M, A) and e with $|\mathcal{E}| \leq f(|M|, \delta)$.

The following theorem now provides an algorithm for Γ-MMS for any strongly extendable model type Γ.

Theorem 2. *Let* Γ *be a* strongly $(f(|M|, \delta), g(|M|, |C|))$-extendable *model-type. Then,* Γ-MMS *can be solved in time* $O((f(s, \delta))^s(g(s, |C|) + |E|s))$.

Proof. We solve Γ-MMS by the bounded-depth branching algorithm illustrated in Algorithm 1. The main part of the algorithm is the function FINDOPTEXTSTR(C, s, (M, A)), which, when called initially with the empty annotated model (`nil`) returns the required result. In general, the

Algorithm 1: Generic Algorithm for finding a minimum model of size at most s for any strongly extendable model-type Γ. Note that the only part of the algorithm that depends on the particular model type Γ is provided by the function FINDSTRICTEXTSSTR$(C, (M, A), e)$, whose existence is guaranteed because Γ is strongly extendable.

Input: CI $C = (E, F, \mu)$ and integer s.
Output: return a minimum model for C of type Γ and size at most s (or `nil` if no such model exists).

```
1:  function FINDOPTMODELSTR(C, s)
2:      return FINDOPTEXTSTR(C, s, nil)
3:  function FINDOPTEXTSTR(C, s, (M, A))
4:      if M is a model for C then
5:          return M
6:      if |M| ≥ s then
7:          return nil
8:      e ← any example not correctly classified by M
9:      𝓜 ← FINDSTRICTEXTSSTR(C, (M, A), e)
10:     B ← nil
11:     for (M', A') ∈ 𝓜 do
12:         if |M'| ≤ s then
13:             A ← FINDOPTEXTSTR(C, s, (M', A'))
14:             if A ≠ nil and (B = nil or |B| > |A|) then
15:                 B ← A
16:     return B
```

function FINDOPTEXTSTR$(C, s, (M, A))$ does the following: given a CI C, an integer s, and an annotated model (M, A), it outputs a minimum model M' for C of size at most s that extends (M, A) if such a model exists; otherwise it will output `nil`. To achieve this, the function first checks whether M is already a model for C and if so returns M. Otherwise, it checks whether M is already too large to be extended, i.e., if $|M| \geq s$, and if so returns `nil`. If this is not the case, the function takes any example $e \in E$ that is not correctly classified by M and calls the model-type specific function FINDSTRICTEXTSTR$(C, (M, A), e)$ to obtain a full set $\mathcal{E}$ of strict extensions for (M, A) and e. Finally, the function then calls itself recursively for every annotated model (M', A') in $\mathcal{E}$ and returns the best model found for any such strict extension.

Towards showing the correctness of the algorithm first note that if the algorithm returns a model M, then this is indeed a model for C (because of Line 4 of the algorithm) of size at most s (because of Line 12 of the algorithm).

So suppose that there is indeed a model M' for C of type Γ and size at most s. We will show that the algorithm considers every such model and therefore returns one of those models of minimum size. To achieve this it suffices to show that M' extends `nil` and whenever M' extends the current annotated model (M, A), then it will also extend one of the strict extensions in $\mathcal{E}$ computed in Line 9 of the algorithm. The former clearly holds because of (P1). Towards showing the latter, let e be the example assigned in Line 8 of the algorithm. Then, M does not correctly classify e but M' does (since it is a model for C), and therefore it holds that M' is an extension of M that correctly classifies e. Therefore, M' is an extension of some annotated model in the full set of strict extensions $\mathcal{E}$ for (M, A) and e, as required.

Let us now consider the running-time of the algorithm. Because Γ is *strongly* $(f(|M|, \delta), g(|M|, |C|))$*-extendable*, it holds that the function FINDSTRICTEXTSSTR$(C, (M, A), e)$ called in Line 9 requires time at most $O(g(|M|, |C|))$ and returns at most $f(|M|, \delta) \leq f(s, \delta)$ strict

Algorithm 2: Generic Algorithm for finding a minimum ensemble model of size at most s for any strongly extendable model-type Γ. Note that the only part of the algorithm that depends on the particular model type Γ is provided by the function FINDSTRICTEXTSSTR$(C, (M, A), e)$, whose existence is guaranteed because Γ is strongly extendable.

Input: CI $C = (E, F, \mu)$ and integer s.
Output: return a minimum ensemble model for C of type Γ and size at most s (or `nil` if no such model exists).

```
1: function FINDOPTENSMODELSTR(C, s)
2:     return FINDOPTENSEXTSTR(C, s, nil)
3: function FINDOPTENSEXTSTR(C, s, E)
4:     if E is an ensemble model for C then
5:         return E
6:     if |E| ≥ s then
7:         return nil
8:     e ← any example not classified correctly by E
9:     B ← nil
10:    for (M, A) ∈ E do
11:        if (M, A) does not correctly classify e then
12:            M ← FINDSTRICTEXTSSTR(C, (M, A), e)
13:            for (M', A') ∈ M do
14:                E' ← E \ {(M, A)} ∪ {(M', A')}
15:                if |E'| > s then
16:                    break
17:                A ← FINDOPTENSEXTSTR(C, s, E')
18:                if A ≠ nil and (B = nil or |B| > |A|) then
19:                    B ← A
20:    M ← FINDSTRICTEXTSSTR(C, nil, e)
21:    for (M', A') ∈ M do
22:        E' ← E ∪ {(M', A')}
23:        if |E'| ≤ s then
24:            A ← FINDOPTENSEXTSTR(C, s, E')
25:            if A ≠ nil and (B = nil or |B| > |A|) then
26:                B ← A
27:    return B
```

extensions. Therefore, the branching factor of the algorithm is at most $f(s, \delta)$ and since the size of the considered partial annotated models increases by at least one (P2) in each recursive call, we obtain that the recursion depth of the algorithm is at most s. Therefore, the algorithm does at most $(f(s, \delta))^s$ recursive calls. Moreover, the time required for each recursive call is dominated by the call to FINDSTRICTEXTSSTR$(C, (M, A), e)$ in Line 9, which is $g(|M|, |C|) \leq g(s, |C|)$, the check whether M is already a model in Line 4, and finding an example e that is not correctly classified by M in Line 8, which because of (P0) can be achieved in time $O(|E|s)$. Therefore, the total run-time of the algorithm is at most $O((f(s, \delta))^s(g(s, |C|) + |E|s))$. □

We show next that strong extendability is even sufficient to efficiently learn ensembles.

Theorem 3. *Let Γ be a* strongly $(f(|M|, \delta), g(|M|, |C|))$-extendable *model-type. Then,* Γ-MEMS *can be solved in time $O(b^s s(g(s, \delta) + |E|))$, where $b = f(0, \delta) + \sum_{(M,A)\in\mathcal{E}} f(|M|, \delta)$.*

Proof. We solve Γ-MEMS by the bounded-depth branching algorithm illustrated in Algorithm 2. The main part of the algorithm is the function FINDOPTENSEXTSTR(C, s, $\mathcal{E}$), which when called initially with the empty annotated ensemble model (`nil`) returns the required result. In general, the function FINDOPTEXTSTR(C, s, $\mathcal{E}$) does the following: Given a CI C, an integer s, and an annotated ensemble model $\mathcal{E}$, it outputs a minimum ensemble model $\mathcal{E}'$ for C of size at most s that extends $\mathcal{E}$ if such a model exists; otherwise it will output `nil`. To achieve this, the function first checks whether $\mathcal{E}$ is already a model for C and if so returns $\mathcal{E}$. Otherwise, it checks whether $\mathcal{E}$ is already too large to be extended, i.e., if $|\mathcal{E}| \geq s$, and if so returns `nil`. If this is not the case, the function takes any example $e \in E$ that is not correctly classified by $\mathcal{E}$ and proceeds as follows. For every annotated model $(M, A) \in \mathcal{E}$ that does not correctly classify e the algorithm calls the (model-type specific) function FINDSTRICTEXTSTR(C, (M, A), e) to obtain a full set $\mathcal{M}$ of strict extensions for (M, A) and e. For every strict extension $(M', A') \in \mathcal{M}$ the algorithm then calls itself recursively for the the ensemble model $\mathcal{E}'$ that is obtained from $\mathcal{E}$ after replacing (M, A) with (M', A'). If any of those recursive calls returns a better ensemble model than was already found it stores the best currently found ensemble model in the variable $\mathcal{B}$. Finally, the algorithm considers the case of adding a new model to the ensemble $\mathcal{E}$. To do that the algorithm first calls FINDSTRICTEXTSTR(C, `nil`, e) to obtain a full set $\mathcal{M}$ of strict extensions for the empty model `nil` and e. It then class itself recursively for the ensemble model $\mathcal{E}' = \mathcal{E} \cup (M', A')$ for every $(M', A') \in \mathcal{M}$. If any of those recursive calls returns a better ensemble model than was already found it stores the best currently found ensemble model in the variable $\mathcal{B}$.

Towards showing the correctness of the algorithm first note that if the algorithm returns an ensemble model $\mathcal{E}$, then this is indeed an ensemble model for C (because of Line 4 of the algorithm) of size at most s (because of Lines 15 and 23 of the algorithm).

So suppose that there is indeed an ensemble model $\mathcal{E}_0$ for C of type Γ and size at most s. We will show that the algorithm considers every such ensemble model and therefore returns one of those ensemble models of minimum size. To achieve this it suffices to show that $\mathcal{E}_0$ extends `nil` and whenever $\mathcal{E}_0$ extends the current annotated ensemble $\mathcal{E}$, then it will also extend one of the annotated ensembles $\mathcal{E}'$ computed in Lines 14 and 22 of the algorithm. The former clearly holds because of (P1).

Towards showing the latter, let e be the example assigned in Line 8 of the algorithm. Then, $\mathcal{E}$ does not correctly classify e but $\mathcal{E}_0$ does (since it is a model for C). Therefore, $\mathcal{E}_0$ contains a model M_0 that correctly classifies e such that either M_0 extends an annotated model (M, A) in $\mathcal{E}$ that does not correctly classify e or M_0 is not an extension of any annotated model in $\mathcal{E}$. In the former case, M_0 is an extension of some strict extension (M', A') in the full set of strict extensions for (M, A) and e, which is considered in Line 13 of the algorithm. Therefore, $\mathcal{E}_0$ is an extension of $\mathcal{E}' = \mathcal{E} \setminus \{(M, A)\} \cup \{(M', A')\}$, which is considered by the algorithm in Line 14, as required. In the latter case, M_0 is an extension of some strict (M', A') in the full set of strict extensions for `nil` and e, which is considered in Line 21 of the algorithm. Therefore, $\mathcal{E}_0$ is an extension of $\mathcal{E}' = \mathcal{E} \setminus \{(M, A)\} \cup \{(M', A')\}$, which is considered by the algorithm in Line 22, as required.

Let us now consider the running-time of the algorithm. Because Γ is *strongly* $(f(|M|, \delta), g(|M|, |C|))$-*extendable*, it holds that the function FINDSTRICTEXTSSTR(C, (M, A), e) requires time at most $O(g(|M|, |C|))$ and returns at most $f(|M|, \delta)$ strict extensions. Since the algorithm calls itself recursively for every strict extension for (M, A) and e for every $(M, A) \in \mathcal{E}$ (in Line 12) as well as every strict extension of `nil` and e (in Line 20), the branching factor of the algorithm is at most $b = f(0, \delta) + \sum_{(M,A)\in\mathcal{E}} f(|M|, \delta)$. Moreover, since the size of the considered annotated ensemble models increases by at least one (P2) in each recursive call, we obtain that

the recursion depth of the algorithm is at most s. Therefore, the algorithm does at most b^s recursive calls. Moreover, the time for each recursive call is dominated by $|\mathcal{E}| + 1$ calls to the function FINDSTRICTEXTSSTR(C, (M, A), e) (Lines 12 and 20) plus the time required to check whether $\mathcal{E}$ is already a model for C (Line 4) and for finding an example e that is not correctly classified by $\mathcal{E}$ (Line 8). Since the former can be achieved in time $O(s\dot{g}(s, \delta)$ and the latter two can be achieved in time $O(|E|s)$, because of (P0), we obtain $O(b^s s(g(s, \delta) + |E|))$ as the total run-time of the algorithm. □

4.2. Extendability

In this section, we will introduce and develop our framework for extendable models, which as we will see later include all forms of BDDs as well as BDD-ensembles. In contrast to strong extendable models, it will no longer be necessary to annotate the models but instead we need to be able to deal with "partial" models, which for the purposes of the framework can be thought of incomplete models that can be extended to a model.

We are now ready to provide a formal definition of extendable models. A *full set of strict extensions for a partial model M* is a set $\mathcal{E}$ of strict extensions of M such that for every model M' of minimum size for C that is an extension of M it holds that M' is also an extension of some partial model in $\mathcal{E}$. We say that a model-type Γ is $(f(|M|, \delta), g(|M|, |C|))$-*extendable* for some computable functions $f(|M|, \delta)$ and $g(|M|, |C|)$ if there is an algorithm running in time $O(g(|M|, |C|))$ that given a CI C and a partial model M of type Γ such that M is not a model for C and $|M|$ computes a full set $\mathcal{E}$ of strict extensions for M with $|\mathcal{E}| \leq f(|M|, \delta)$.

Note that the main difference to the case of strongly extendable models is that the full set of strict extensions does no longer need to address a specific example, but instead it is merely required that one of the strict extensions is part of some optimum model. While this makes the framework applicable to more general models (such as BDDs), it also comes with a cost. Indeed, without the example as a guide to further restrict the number of possible extensions, the number of strict extensions becomes potentially much larger; we will later see that this indeed applies in the case of BDDs. Furthermore, the extension of the framework to ensembles of models (using Algorithm 2)seems no longer possible as it specifically requires to find extensions that can be used to classify a specific example and more importantly the models in an ensemble are not required to be models for the CI. Nevertheless, we will show that the latter disadvantage can be overcome by showing that also BDD-ensemble models are extendable.

The following theorem now provides an algorithm for Γ-MMS for any extendable model type Γ.

Theorem 4. *Let* Γ *be a* $(f(|M|, \delta), g(|M|, |C|))$-extendable *model-type. Then,* Γ-MMS *can be solved in time* $O((f(s, \delta))^s(g(s, |C|) + |E|s))$.

Proof. As mentioned above, we solve Γ-MMS by the simple bounded depth branching algorithm illustrated in Algorithm 3. The main part of the algorithm is the function FINDOPTEXT(C, s, (M, A)), which when called initially with the empty partial annotated model (nil) returns the required result. In general, the function FINDOPTEXT(C, s, (M, A)) does the following: Given a CI C, an integer s, and a partial annotated model (M, A), it outputs a minimum model M' for C of size at most s that extends (M, A) if such a model exists; otherwise it will output nil. To achieve this, the function first checks whether M is already a model for C and if so returns M. Otherwise, it checks whether M is already too large to be extended, i.e., if $|M| \geq s$, and if so returns nil. If this is not the case, the function calls the (model-type specific) function FINDSTRICTEXTS(C,

Algorithm 3: Generic Algorithm for finding a minimum model of size at most s for any extendable model-type Γ.

Input: CI $C = (E, F, \mu)$ and integer s.
Output: return a minimum model for C of type Γ and size at most s (or `nil` if no such model exists).

```
1:  function FINDOPTMODEL(C, s)
2:      return FINDOPTEXT(C, s, nil)
3:  function FINDOPTEXT(C, s, (M, A))
4:      if M is a model for C then
5:          return M
6:      if |M| ≥ s then
7:          return nil
8:      M ← FINDSTRICTEXTS(C, s, (M, A))
9:      B ← nil
10:     for (M', A') ∈ M do
11:         if |M'| ≤ s then
12:             A ← FINDOPTEXT(C, s, (M', A'))
13:             if A ≠ nil and (B = nil or |B| > |A|) then
14:                 B ← A
15:     return B
```

s, (M, A)) to obtain a full set $\mathcal{E}$ of strict extensions for (M, A) and the function then calls itself recursively for every partial annotated model (M', A') in $\mathcal{E}$ and returns the best model found for any such strict extension.

Towards showing the correctness of the algorithm first note that if the algorithm returns a model M, then this is indeed a model for C (because of Line 4 of the algorithm) of size at most s (because of Line 11 of the algorithm).

So suppose that there is indeed a model M' for C of type Γ and size at most s. We will show that the algorithm considers every such model and therefore returns one of those models of minimum size. To achieve this it suffices to show that M' extends `nil` and whenever M' extends the current partial annotated model (M, A), then it will also extend one of the strict extensions in $\mathcal{E}$ computed in Line 8 of the algorithm. The former clearly holds because of (P1) and the latter holds because $\mathcal{E}$ is a full set of strict extensions for (M, A).

Let us now consider the running-time of the algorithm. Because Γ is $(f(|M|, \delta), g(|M|, |C|))$-*extendable*, it holds that the function FINDSTRICTEXTS$(C, (M, A))$ called in Line 8 requires time at most $O(g(|M|, |C|))$ and returns at most $f(|M|, \delta)$ strict extensions. Therefore, the branching factor of the algorithm is at most $f(|M|, \delta)$ and since the size of the considered partial annotated models increases by at least one (P2) in each recursive call, we obtain that the recursion depth of the algorithm is at most s. Therefore, the algorithm does at most $(f(|M|, \delta))^s$ recursive calls. Moreover, the time required for each recursive call is dominated by the call to FINDSTRICTEXTS(C, s, (M, A)) in Line 8, which is $g(|M|, |C|)$ and the check whether M is already a model in Line 4, which because of (P0) can be achieved in time $O(|E|s)$. Therefore, the total run-time of the algorithm is at most $O((f(|M|, \delta))^s(g(|M|, |C|) + |E|s)) = O((f(s, \delta))^s(g(s, |C|) + |E|s))$. □

5. Decision Sets

In the case of `DS`s an annotated `DS` is simple a pair (M, A), where $M = (T, b)$ is a decision set and $A : T \cup \{r_D\} \to E$ is the annotation function that assigns one example to every term

Algorithm 4: Algorithm for finding a full set of strict extensions for DSs.

Input: A CI $C = (E, F, \mu)$, an annotated DS $(M = (T, b), A)$, and an example $e \in E$ that is not correctly classified by M.

Output: An full set of strict extensions for $(M = (T, b), A)$ and e.

1: **function** FINDSTRICTEXTSSTR(C, $(M = (T, b), A)$, e)
2: **if** $(M, A) = \texttt{nil}$ **then**
3: $e_1 \leftarrow$ some 1-example in E
4: $A_1 \leftarrow$ function assigning e_1 to the default rule r_D.
5: $e_0 \leftarrow$ some 0-example in E
6: $A_0 \leftarrow$ function assigning e_0 to the default rule r_D.
7: **return** $\{((\emptyset, 0), A_0), ((\emptyset, 1), A_1)\}$
8: $\mathcal{X} \leftarrow \emptyset$
9: **if** $b \neq \mu(e)$ **then**
10: $e' \leftarrow A(r_D)$
11: **for** $f \in \delta(e', e)$ **do**
12: $t \leftarrow \{(f = e(f))\}$
13: $A' \leftarrow A$ with the additional assignment $A'(t) = e$
14: $\mathcal{X} \leftarrow \mathcal{X} \cup \{((T \cup \{t\}, b), A')\}$
15: **return** $\mathcal{X}$
16: $t \leftarrow$ term in T that applies to e
17: $e' \leftarrow A(t)$
18: **for** $f \in \delta(e', e)$ **do**
19: $t' \leftarrow t \cup \{(f = e'(f))\}$
20: $T' \leftarrow$ obtained from T after replacing t with t'
21: $\mathcal{X} \leftarrow \mathcal{X} \cup \{((T', b), A)\}$
22: **return** $\mathcal{X}$

(including the default rule r_D) of T. The idea of the annotation is that if a term is annotated by an example, then we only consider extensions of the term that agree with the example.

We say that a DS (T', b') is an *extension* of a DS (T, b) if either $(T, b) = \texttt{nil}$ or it holds that $b' = b$ and there is an injective function $\alpha : T \to T'$ such that $t \subseteq \alpha(t)$ for every $t \in T$. We say that a DS (T', b') is an *extension* of an annotated DS $((T, b), A)$ if (T', b') is an extension of (T, b) and additionally $A(t)$ agrees with $\alpha(t)$ for every $t \in T$. We say that an annotated DS $((T', b'), A')$ is an *extension* of an annotated DS $((T, b), A)$ if either $((T, b), A) = \texttt{nil}$ or it holds that $b' = b$ and there is an injective function $\alpha : T \to T'$ such that for every $t \in T$, it holds that $t \subseteq \alpha(t)$ and $A(t) = A'(\alpha(t))$. Moreover, we say that an annotated DS $((T', b'), A')$ is a *strict extension* of an annotated DS $((T, b), A)$ if it is an extension and additionally $((T', b'), A') \neq ((T, b), A)$; note that this also implies that $|T'| > |T|$ if we define the size of nil as 0.

Lemma 5. *Decision Sets are strongly* $(\delta, \delta + \|T\|)$*-extendable.*

Proof. We claim that Algorithm 4 shows the result, i.e., we need to show that Algorithm 4 runs in time $O(|E|s + \delta(E))$ and given a CI $C = (E, F, \mu)$, an integer s, an annotated DS $(M = (T, b), A)$, and an example e such that M does not correctly classify e computes a full set of strict extensions $\mathcal{E}$ for (M, A) and e with $|\mathcal{E}| \leq \max\{2, \delta\}$.

The main ideas behind Algorithm 4 are as follows. If $(M, A) = \texttt{nil}$, then the algorithm creates two trivial annotated DSs (in Lines 2– 7) corresponding to the two classifications of the default rule. That is, the algorithm takes an arbitrary 1-example e_1 and creates the annotated

model $((\emptyset, 1), A_1)$, where A_1 is the function assigning e_1 to the default rule. Similarly, the algorithm takes an arbitrary 0-example e_0 and creates the annotated model $((\emptyset, 1), A_0)$, where A_0 is the function assigning e_0 to the default rule. Moreover, if on the other hand $(M = (T, b), A) \neq \texttt{nil}$, then the algorithm distinguishes two cases. If $b \neq \mu e$ (Lines 9– 15 of the algorithm) and therefore no term of T applies to e, then any extension of (M, A) that correctly classifies e must contain a new term that applies to e. The algorithm therefore creates a new term t and annotates it by e. Moreover, it then ensures that t does not apply to the annotated example $e' = A(r_D)$ of the default rule r_D by adding one literal $f = e(f)$ to t, where $f \in \delta(e', e)$. It then returns one strict extension $((T \cup \{t\}), b), A')$ of (M, A) for every feature in $f \in \delta(e', e)$, where t is the term containing only the literal $f = e(f)$. Finally, if $b = \mu(e)$ (Lines 16– 22 of the algorithm), then because e is not correctly classified by M, there is a term that applies to e and the algorithm chooses an arbitrary such term in Line 16. But then any extension of (M, A) that correctly classifies e must ensure that e does not apply to t. This can be achieved by adding some literal $f = e'(f)$ for every $f \in \delta(e', e)$, where $e' = A(t)$. The algorithm therefore returns a strict extension $((T', b), A)$ of (M, A) for every $f \in \delta(e', e)$, where T' is obtained from T after adding the literal $f = e'(f)$ to t.

Towards showing the correctness of Algorithm 4, we first note that the algorithm always outputs a set of strict extensions of (M, A) and that the number of those strict extensions is at most $\max\{2, \delta\} \leq \delta$; assuming that $\delta \geq 2$. This clearly holds in the case that $(M, A) \neq \texttt{nil}$ because in this case every annotated DS added to the set $\mathcal{X}$ by the algorithm extends at least one rule of M by at least one literal. Moreover, this also holds if $(M, A) = \texttt{nil}$ because in this case the size of (M, A) is equal to -1 and both annotated DS returned by the algorithm in Line 7 have size 0. It remains to show that the set of strict extensions returned by the algorithm is indeed a full set of strict extensions for (M, A) and e. To see this let M' be a DS that extends (M, A) and correctly classifies e. We distinguish the following cases. If $(M, A) = \texttt{nil}$, then M' is clearly an extension of either $(\emptyset, 0)$ or $(\emptyset, 1)$. Moreover, since the default rule applies to every 1-example respectively 0-example, M' is also an extension of the annotation returned by the algorithm.

Otherwise, we distinguish the following two cases. If $b \neq \mu(e)$, then M' must contain a term t such that e is satisfied by t, which M does not contain; in fact M does not contain any such term because it does not correctly classify e. Let e' be the example assigned by A to the default rule of M. Then, $\mu(e) \neq \mu(e')$ and therefore t has to contain a literal on a feature that distinguishes e' from e; since otherwise t would also apply to e', which would contradict our assumption that M' is an extension of (M, A). Since the algorithm adds such a term t for every $f \in \delta(e, e')$ and the algorithm also annotates this term with e, this shows that M' is an extension of some annotated DS returned by the algorithm.

If on the other hand $b = \mu(e)$, then M must contain a term t that applies to e and suppose that t is the term chosen by the algorithm in Line 16. Moreover, since M' is an extension of (M, A), M' must contain a term t' such that $t \subseteq t'$ and $e' = A(t)$ is satisfied by t'. Since M' classifies e correctly, it follows that e cannot be satisfied by t'. Therefore, t' must contain a literal of some feature f in $\delta(e, e')$, whose value is set to $e'(f)$. Since the algorithm branches through all those features and for each of them tries to add the feature with the value of e' to t this shows that M' is an extension of some annotated DS returned by the algorithm.

Finally, let us consider the run-time of the algorithm. Note first that all but one single line instruction of the algorithm can be done in constant time. The only exception is Line 16 for finding the term of T that applies to e and this can be achieved in time $O(\|T\|)$ using the natural and appropriate data structures. Therefore, the total time required by the algorithm is $O(\|T\| + \delta)$, where the δ-term is due to the two for-loops in Lines 11 and 18. □

Combining Lemma 5 with Theorem 2, we obtain.

Corollary 6. DS-MMS *can be solved in time* $O(\delta^s|E|s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Combining Lemma 5 with Theorem 3, we obtain.

Corollary 7. DS-MEMS *can be solved in time* $O((2 + s\delta)^s|E|s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Proof. It follows from Lemma 5 that DSs are strongly $(\delta, \delta + \|T\|)$-extendable. Therefore, using Theorem 3, we obtain that DS-MEMS can be solved in time $O(b^s s(\delta + s + |E|))$, where $b = f(0, \delta) + \sum_{(M,A)\in\mathcal{E}} f(|M|, \delta) \leq 2 + s\delta$. Here we assume for simplicity that $\delta + s \leq |E|$. □

6. Decision Lists

In the case of DLs an annotated DL is simple a pair (L, A), where L is a decision list and $A : L \to E$ is the annotation function that assigns one example to every rule in L. The idea of the annotation is that if a rule is annotated by an example, then we only consider extensions of the rule that agree with the example.

We say that an injective function $\alpha : L \to L'$ between two DLs L and L' is *order-preserving* if for every two distinct $l, l' \in L$, it holds that l is ordered before l' in L if and only if $\alpha(l)$ is ordered before $\alpha(l')$ in L'.

We say that a DL L' is an *extension* of a DL L if there is an injective and order-preserving function $\alpha : L \to L'$ such that for every $r = (t, b) \in L$ with $(t', b') = \alpha(r)$, it holds that $b' = b$ and $t \subseteq t'$. We say that a DL L' is an *extension* of an annotated DL (L, A) if there is an injective order-preserving function $\alpha : L \to L'$ such that for every $r = (t, b) \in L$ with $r' = (t', b') = \alpha(r)$, it holds that $b' = b$, $t \subseteq t'$, and r' is the first rule in L' that agrees with $A(r)$. We say that an annotated DL (L', A') is an *extension* of an annotated DL (L, A) if there is an injective order-preserving function $\alpha : L \to L'$ such that for every $r = (t, b) \in L$ with $r' = (t', b') = \alpha(r)$, it holds that $b' = b$, $t \subseteq t'$, and $A'(r') = A(r)$. Finally, we say that an annotated DL (L', A') is a *strict extension* of an annotated DL (L, A) if it is an extension and additionally $(L', A') \neq (L, A)$; note that this also implies that $|L'| > |L|$.

Lemma 8. *Decision Lists are strongly* $(\delta + |L| + 1, |L| + \delta)$*-extendable.*

Proof. We claim that Algorithm 5 shows the result, i.e., we need to show that Algorithm 5 runs in time $O(|L| + \delta)$ and given a CI $C = (E, F, \mu)$, an annotated DL (L, A), and an example e such that L does not correctly classify e computes a full set of strict extensions $\mathcal{E}$ for (L, A) and e with $|\mathcal{E}| \leq \delta + |L| + 1$.

The main ideas behind Algorithm 5 are as follows. If no rule in L applies to e, then the algorithm considers all extensions (L', A') obtained from (L, A) after inserting a new (empty) rule $(\emptyset, \mu(e))$ annotated by e at any possible position in L. Otherwise, let $r' = (t', 1 - \mu(e))$ be the first rule that applies to e in L and let p' be its position. The algorithm then adds all extensions (L', A') obtained from (L, A) after inserting a new (empty) rule $r = (\emptyset, \mu(e))$ annotated by e at any possible position before p' in L, to the initially empty set $\mathcal{X}$ of extensions. Finally, the algorithm returns $\mathcal{X}$ after additionally adding to it all extensions of (M, A) obtained by adding the literal $f = e'(f)$ to the the term t' of rule r' for every $f \in \delta(e', e)$.

Algorithm 5: Algorithm for finding a full set of strict extensions for DLs.

Input: A CI $C = (E, F, \mu)$, an annotated DL (L, A), and an example $e \in E$ that is not correctly classified by L.

Output: An full set of strict extensions for (L, A) and e.

1: **function** FINDSTRICTEXTSSTR(C, (L, A), e)
2: $\quad\mathcal{X} \leftarrow \emptyset$
3: $\quad$**if** no rule in L applies to e **then**
4: $\quad\quad$**for** $p \in [0, |L|]$ **do**
5: $\quad\quad\quad r \leftarrow (\emptyset, \mu(e))$
6: $\quad\quad\quad A' \leftarrow$ extension of A by $A'(r) = e$
7: $\quad\quad\quad L' \leftarrow$ obtained from L after inserting r at p
8: $\quad\quad\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{(L', A')\}$
9: $\quad r' = (t', 1 - \mu(e)) \leftarrow$ first rule in L that applies to e
10: $\quad p' \leftarrow$ position of r' in L
11: $\quad r \leftarrow (\emptyset, \mu(e))$
12: $\quad A' \leftarrow$ extension of A with $\{A'(r) = e\}$
13: $\quad$**for** $p \in [0, p' - 1]$ **do**
14: $\quad\quad L' \leftarrow$ obtained from L after inserting r at p
15: $\quad\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{(L', A')\}$
16: $\quad e' \leftarrow A(r')$
17: $\quad$**for** $f \in \delta(e', e)$ **do**
18: $\quad\quad r \leftarrow (t' \cup \{(f = e'(f))\}, 1 - \mu(e))$
19: $\quad\quad L' \leftarrow$ list obtained from L after replacing r with r'
20: $\quad\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{(L', A)\}$
21: $\quad$**return** $\mathcal{X}$

Towards showing the correctness of Algorithm 5, we first note that the algorithm always outputs a set of strict extensions of (L, A), i.e., in every case L is extended by some rule or some rule of L is extended by some literal, and that the number of those strict extensions is at most $\delta + |L| + 1$. It remains to show that the set of strict extensions returned by the algorithm is indeed a full set of strict extensions for (L, A) and e. To see this let L_e be a DL that extends (L, A) and correctly classifies e. We need to show that L_e is an extension of some strict extension returned by the algorithm. We distinguish the following cases.

If no rule in L applies to e, i.e., the case corresponding to Line 3 of the algorithm, then because L_e correctly classifies e, it holds that $L_e \setminus L$ must contain a new rule, say r that can be inserted at some position $p \in \{0, |L|\}$ in L and that applies to e. W.l.o.g. we assume that r is the first rule in L_e that applies to e, which will allow us to annotate r with e. Therefore, we obtain that L_e extends (L', A'), where L' is obtained by adding the new rule r at position p to L and setting $A' = A \cup \{A(r) = e\}$. Since the algorithm considers all those cases in the for-loop in Line 4, this shows that (in the case that no rule in L applies to e) L_e is an extension of some annotated DL returned by the algorithm in Line 8.

Otherwise, let p' be the position of the first rule $r' = (t', 1 - \mu(e))$ in L that applies to e (see also Line 9 of the algorithm). Then, because L_e correctly classifies e, it holds that either $L_e \setminus L$ must contain a new rule, say r that can be inserted at some position $p \in \{0, p' - 1\}$ in L and that applies to e (and that can therefore be annotated with e) or $\alpha(r')$ does not apply to e. In the former case, we obtain that L_e is an extension of (L', A'), where L' is obtained from L after adding the new rule r annotated by e at position p and the algorithm considers all those cases in the for-loop

in Line 13. In the latter case, since L_e is an extension of (L, A), it follows that $\alpha(r')$ is the first rule of L' that applies to e'. Therefore, L_e is an extension of some (L', A), where L' is obtained after adding some literal $f = e'(f)$ to t' for some feature $f \in \delta(e', e)$ and the algorithm considers all those cases in the for-loop of Line 17.

Towards showing the correctness of the run-time of the algorithm, we first note that almost all single line operations in the algorithms can be achieved in constant time (with the natural and appropriate data structures). The only two exceptions are: (1) checking whether no rule applies to e in Line 3 and (2) finding the first rule in L that applies to e in Line 9. Since both of these exceptions can be achieved in time $O(|L|)$, we obtain that the total run-time of the algorithm is dominated by the two for-loops in Lines 13 and 17 of the algorithm, which together can be achieved in time $O(|L| + \delta)$, as required. Here, we assume that $\delta(e', e)$ has been precomputed before that start of the algorithm for every pair of 1-example and 0-example. □

Combining Lemma 8 with Theorem 2, we obtain the following.

Corollary 9. DL-MMS *can be solved in time* $O((\delta + s + 1)^s |E| s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Combining Lemma 8 with Theorem 3, we obtain the following.

Corollary 10. DL-MEMS *can be solved in time* $O((1 + s + s\delta)^s |E| s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Proof. It follows from Lemma 8 that DLs are strongly $(\delta + |L| + 1, |L| + \delta)$-extendable. Therefore, using Theorem 3, we obtain that DL-MEMS can be solved in time $O(b^s s(\delta + s + |E|))$, where $b = f(0, \delta) + \sum_{(M,A)\in\mathcal{E}} f(|M|, \delta) \leq 1 + s + s\delta$. Here we assume for simplicity that $\delta + s \leq |E|$. □

7. Decision Trees

Let $M = (T, \lambda)$ and $M' = (T', \lambda')$ be two DTs. We say that an injective function $\alpha : V(T) \to V(T')$ is *structure-preserving* from M to M' if

- For every $t \in V(T)$, it holds that $\lambda(t) = \lambda(\alpha(t))$; note that this also implies that leaves are mapped to leaves and inner nodes are mapped to inner nodes.
- If $p \in V(T)$ is a node with x-child c in T, then $\alpha(c)$ is contained in the subtree rooted at the x-child of $\alpha(p)$ in T'.

We say that a DT $M' = (T', \lambda')$ is an *extension* of a DT $M = (T, \lambda)$ if either $M =$ nil or there is a structure-preserving function from M to M'.

We say that a DT $M' = (T', \lambda')$ is an *extension* of an annotated DT $(M = (T, \lambda), A)$ if either $M =$ nil or there is a structure-preserving function from M to M' that additionally satisfies $A(l) \in E_{M'}(\alpha(l))$ for every leaf l of T.

We say that an annotated DT $(M' = (T', \lambda'), A')$ is an *extension* of an annotated DT $(M = (T, \lambda), A)$ if either $M =$ nil or there is a structure-preserving function from M to M' that additionally satisfies $A(l) \in A'(\alpha(l))$ for every leaf l of T.

Lemma 11. *Decision Trees are strongly* $(\delta(h(T) + 1), \delta h(T))$*-extendable.*

Algorithm 6: Algorithm for finding a full set of strict extensions for DTs.

Input: A CI $C = (E, F, \mu)$, an annotated DT $(M = (T, \lambda), A)$, and an example $e \in E$ that is not correctly classified by M.

Output: An full set of strict extensions for (M, A) and e.

function FINDSTRICTEXTSSTR(C, (M, A), e)
  if $(M, A) =$ nil **then**
    $M_0 = (T_0, \lambda_0) \leftarrow$ DT with one leaf labeled 0
    $M_1 = (T_1, \lambda_1) \leftarrow$ DT with one leaf labeled 1
    $l_0 \leftarrow$ the leaf of T_0
    $A_0 \leftarrow$ function assigning some 0-example to l_0
    $l_1 \leftarrow$ the leaf of T_1
    $A_1 \leftarrow$ function assigning some 1-example to l_1
    return $\{(M_0, A_0), (M_1, A_1)\}$
  $\mathcal{X} \leftarrow \emptyset$
  $l_e \leftarrow$ leaf of T with $e \in E_M(l_e)$
  $P_e \leftarrow$ path from root to l_e in T
  $e' \leftarrow A(l_e)$
  for $f \in \delta(e', e)$ **do**
    $M' = (T', \lambda') \leftarrow (f, e(f), \mu(e))$-extension of M
    $l \leftarrow$ new leaf of $V(T') \setminus V(T)$
    $A' \leftarrow$ obtained from A after setting $A'(l) = e$
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{(M', A')\}$
    for $a \in E(P_e)$ **do**
      $x \leftarrow y$ if a is y-edge in T
      $M' = (T', \lambda') \leftarrow (x, a, f, e(f), \mu(e))$-extension of M
      $l \leftarrow$ new leaf of $V(T') \setminus V(T)$
      $A' \leftarrow$ obtained from A after setting $A'(l) = e$
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{(M', A')\}$
  return $\mathcal{X}$

Proof. We claim that Algorithm 6 shows the result, i.e., we need to show that Algorithm 6 runs in time $O(\delta h(T))$ and given a CI $C = (E, F, \mu)$, an annotated DT $(M = (T, \lambda), A)$, and an example e such that M does not correctly classify e computes a full set of strict extensions $\mathcal{E}$ for (M, A) and e with $|\mathcal{E}| \leq \max\{2, \delta(h(T) + 1)\}$.

Before we proceed with the description and correctness proof of the algorithm, we need to introduce the operations employed by the algorithm. Let $f \in F$ and $y, z \in \{0, 1\}$. We say that a DT $M' = (T', \lambda')$ is an (f, y, z)-*extension* of a DT $M = (T, \lambda)$ if $M \neq$ nil and T' is obtained from T after adding a new node n together with a $(1 - y)$-edge from n to the root r of T and adding a new leaf l as the y-child of n. Moreover, λ' is obtained from λ by setting $\lambda'(w) = \lambda(w)$ for every $w \in V(T)$ and $\lambda'(n) = f$ and $\lambda'(l) = z$.

We say that a DT $M' = (T', \lambda')$ is an (x, e, f, y, z)-*extension* of a DT $M = (T, \lambda)$, where $e = pc \in E(T)$ is an x-edge of T with p being the parent of c in T, if $M \neq$ nil and T' is obtained from T after adding a new node n, replacing the edge e with an x-edge from p to n and an y-edge from n to v and adding a new leaf l as the $(1 - y)$-child of n. Moreover, λ' is obtained from λ by setting $\lambda'(w) = \lambda(w)$ for every $w \in V(T)$ and $\lambda'(n) = f$ and $\lambda'(l) = z$.

The main ideas behind Algorithm 6 are as follows. If $(M, A) =$ nil, then the algorithm returns the annotated DTs (M_0, λ_0) and (M_1, λ_1), where the former is the DT with one leaf labeled

0 and annotated by e and the latter is the DT with one leaf labeled 1 and annotated by e. Otherwise, let l_e be the leaf of T with $e \in E_M(l_e)$, let P_e be the path from the root to l_e in T, and let $e' = A(l_e)$. The algorithm then adds all $(f, e(f), \tau(e))$-extensions of M for every $f \in \delta(e', e)$, where the new leaf is annotated by e, to the initially empty set of extensions $\mathcal{X}$. Additionally, the algorithm adds all $(x, a, f, e(f), \tau(e)$-extensions for every x-edge a on P_e and every feature $f \in \delta(e', e)$, where the new leaf is annotated by e, to the set of extensions $\mathcal{X}$. The algorithm then returns the set $\mathcal{X}$.

If no rule in L applies to e, then the algorithm considers all extensions (L', A') obtained from (L, A) after inserting a new (empty) rule $(\emptyset, \mu(e))$ annotated by e at any possible position in L. Otherwise, let $r' = (t', 1-\mu(e))$ be the first rule that applies to e in L and let p' be its position. The algorithm then considers all extensions (L', A') obtained from (L, A) after inserting a new (empty) rule $r = (\emptyset, \mu(e))$ annotated by e at any possible position before p' in L. Finally, it considers all extensions of the term t' in rule r' with a literal $f = e'(f)$ for every $f \in \delta(e', e)$.

Towards showing the correctness of Algorithm 6, we first note that the algorithm always outputs a set of strict extensions of (M, A), i.e., in every case T is extended by at least two new nodes and that the number of those strict extensions is at most $\max\{2, \delta(h(T)+1)\} \leq \delta(h(T)+1)$; assuming that $\delta(h(T)+1) \geq 2$. It remains to show that the set of strict extensions returned by the algorithm is indeed a full set of strict extensions for (M, A) and e. To see this let $M_e = (T_e, \lambda_e)$ be a DT that extends (M, A) and correctly classifies e. We need to show that M_e is an extension of some strict extension returned by the algorithm. We distinguish the following cases.

If $(M, A) = \texttt{nil}$, then M_e is clearly an extension of the two trivial annotated DTs (M_0, A_0) or (M_1, A_1) returned in Line 9; this is because every DT that is not equal to nil is an extension of either (M_0, A_0) or (M_1, A_1).

Otherwise, $(M, A) \neq \texttt{nil}$. Let l_e be the leaf of T with $e \in E_M(l_e)$, let P_e be the path from the root to l_e in T, and let $e' = A(l_e)$ (as also defined in Lines 11– 13 of the algorithm).

Let $\alpha : V(T) \to V(T_e)$ be the structure-preserving function from M to M' that also satisfies $A(l) \in E_{M_e}(l)$ for every leaf l of T, which exists because M_e is an extension of (M, A). Let $P'_{e'}$ be the path from the root of T_e to $\alpha(l_e)$. Then, because M' correctly classifies e and because $e' \in E_{M'}(l_e)$, it follows that $P'_{e'} \setminus \{\alpha(v) \mid v \in V(P_e)\}$ must contain a node n with $\lambda_e(n) \in \delta(e', e)$ such that the leaf l'_e in T' that correctly classifies e is within the subtree rooted at the child of n that is not in $P'_{e'}$. We now distinguish two cases. If n occurs before any node in $\{\alpha(p) \mid p \in V(P_e)\}$ on $P'_{e'}$, then M_e is an extension of the $(f, e(f), \mu(e))$-extension M' of M. Moreover, since $e \in E_{M_e}(l'_e)$, it also follows that M_e is an extension of (M', A'), where A' is obtained from A by additionally setting $A'(l) = e$, where l is the new leaf in $V(T') \setminus V(T)$. Since, (M', A') is the strict extension added to $\mathcal{X}$ in Line 18 for the feature f, this shows that M_e is an extension of a strict extension returned by the algorithm in this case.

Otherwise, n occurs between some node in $\{\alpha(p) \mid p \in V(P_e)\}$ on $P'_{e'}$. Let $e = pc$ be the x-edge in P_e such that n occurs between $\alpha(p)$ and $\alpha(c)$ on $P'_{e'}$ and assume that p is the parent of c in T. In this case, M_e is an extension of the $(x, e, f, e(f), \mu(e))$-extension M' of M. Moreover, since $e \in E_{M_e}(l'_e)$, it also follows that M_e is an extension of (M', A'), where A' is obtained from A by additionally setting $A'(l) = e$, where l is the new leaf in $V(T') \setminus V(T)$. Since, (M', A') is the strict extension added to $\mathcal{X}$ in Line 24 for the feature f and the edge e, this shows that M_e is an extension of a strict extension returned by the algorithm in this case.

Towards showing the correctness of the run-time of the algorithm, we first note that almost all single line operations in the algorithms can be achieved in constant time (with the natural and appropriate data structures). The only exceptions are the Lines 11 and 12 to obtain the leaf l_e of T with $e \in E_M(l)$ together with the path P_e. Since both of these exceptions can be achieved in time $O(h(T))$, we obtain that the total run-time of the algorithm is dominated by the two for-loops in

Lines 13 and 17 of the algorithm, which together can be achieved in time $O(\delta(h(T)))$, as required. Here, we assume that $\delta(e', e)$ has been precomputed before that start of the algorithm for every pair of 1-example and 0-example. □

Combining Lemma 11 with Theorem 2, we obtain.

Corollary 12. DT-MMS *can be solved in time* $O((\delta(s+1))^s|E|s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Combining Lemma 11 with Theorem 3, we obtain.

Corollary 13. DT-MEMS *can be solved in time* $O((2+2s\delta)^s|E|s)$ *and is therefore fixed-parameter tractable parameterized by* $s + \delta$.

Proof. It follows from Lemma 11 that DTs are strongly $(\delta(h(T) + 1), \delta h(T))$-extendable. Therefore, using Theorem 3, we obtain that DT-MEMS can be solved in time $O(b^s s(s\delta + |E|))$, where $b = f(0, \delta) + \sum_{(M,A)\in\mathcal{E}} f(|M|, \delta) \leq 2 + 2s\delta$. Here we assume for simplicity that $s^2\delta \leq |E|$. □

8. Binary Decision Diagrams

Let $C = (E, F, \mu)$ be a CI. A *partial* BDD S is a pair $S = (D, \rho)$ where D is a directed acyclic graph with two special vertices t_0 and t_1 such that:

- t_0 and t_1 are sinks,
- every vertex except t_0 and t_1 has out-degree at most 2 and more specifically it has at most one 0-out-neighbor and at most one 1-out-neighbor.
- it can (but does not have to have) a specified root vertex, usually denoted by s, which is not allowed to have any in-neighbors,
- ρ is a function that associates a feature in F to every node except t_0 and t_1.

Informally, a partial BDD is obtained by inducing a BDD on some subset of its inner nodes. That is, a partial BDD $S = (D, \rho)$ is any pair for which there exists a BDD $B' = (D', \rho')$ such that $B'[D] = S$, where $B'[D] = (D'[V(D)], \rho'_{|V(D)})$ and $\rho'_{|V(D)}$ is equal to the function ρ' restricted to the vertices in D.

We say that a (partial) BDD $B' = (D', \rho')$ is an *extension* of a partial BDD $B = (D, \rho)$ if $B = B'[D]$. We say that B' is a *strict extension* of B if B' is an extension of B and additionally $B' \neq B$ or equivalently $|D'| > |D|$. We say that B' is a *simple extension* of B if B' is an extension of B and additionally $|D'| = |D| + 1$.

For a partial BDD $B = (D, \rho)$ and a subset $F' \subseteq F$ of features, we denote by $SExt(B, F')$ the set of all simple extensions $B' = (D', \rho')$ of B such that the unique vertex v in $V(D') \setminus V(D)$ satisfies that $\rho(v) \in F'$, i.e., we only consider simple extension of B, whose new nodes use only features from F'. The following lemma now shows that if we could bound the number of features that need to be considered by any strict extension in $SExt(B) = SExt(B, F)$, then we could also bound the size of $SExt(B)$.

Lemma 14. *Let* $C = (E, F, \mu)$ *be a CI, let* $B = (D, \rho)$ *be a partial* BDD*, and let* $F' \subseteq F$*. Then,* $SExt(B, F')$ *can be computed in time* $O(|F'|(\|B\| + 1)^2 3^{\|B\|-2})$ *and* $|SExt(B, F')| \leq 2|F'|(\|B\| + 1)^2 3^{\|B\|-2}$.

Proof. Since $B'[D] = B$ and $|D'| = |D| + 1$ for every $B' = (D', \rho') \in \mathit{SExt}(B, F')$, it holds that every simple extension $B' = (D', \rho')$ is obtained from B by:

(0) adding a new vertex v to D,

(1) choosing a subset $P_0 \subseteq V(D) \setminus \{t_0, t_1\}$ of in-neighbors for v that have a 0-arc to v,

(2) choosing a subset $P_1 \subseteq V(D) \setminus (\{t_0, t_1\} \cup P_0)$ of in-neighbors for v that have a 1-arc to v,

(3) choosing no 0-out-neighbor for v or choosing a 0-out-neighbor for v among $V(D) \setminus \{s\}$,

(4) choosing no 1-out-neighbor for v or choosing a 1-out-neighbor for v among $V(D) \setminus \{s\}$,

(5) choosing a feature in F' to assign to v,

(6) choosing whether or not v becomes the root of B'.

Since, there are $3^{|V(D)|-2}$ possibilities for (1) and (2), $(|V(D)| + 1)^2$ possibilities for (3) and (4), $|F'|$ possibilities for (5) and 2 possibilities for (6), we obtain $2|F'|(|V(D)|+1)^2 3^{|V(D)|-2}$ as the total number of possibilities for B'. The stated run-time can now be easily achieved by brute-force enumeration of all possibilities. □

A *partial* BDD*-ensemble* $\mathcal{E}$ is a set of partial BDDs. We say that a (partial) BDD-ensemble $\mathcal{E}'$ is an *extension* of a partial BDD-ensemble $\mathcal{E}$ if there is an injective function $\alpha : \mathcal{E} \to \mathcal{E}'$ such that $\alpha(B)$ is an extension of B for every $B \in \mathcal{E}$. We say that $\mathcal{E}'$ is a *strict extension* of $\mathcal{E}$ if $\mathcal{E}'$ is an extension of $\mathcal{E}$ and additionally $\mathcal{E}' \neq \mathcal{E}$ or equivalently $|\mathcal{E}'| > |\mathcal{E}|$. We say that $\mathcal{E}'$ is a *simple extension* of $\mathcal{E}$ if $\mathcal{E}'$ is an extension of $\mathcal{E}$ and either there is exactly one $B \in \mathcal{E}$ such that $\alpha(B)$ is a simple extension of B or $\alpha(B) = B$ for every $B \in \mathcal{E}$ and $\mathcal{E}' \setminus \mathcal{E}$ is a *trivial partial* BDD, i.e., a partial BDD $B = (D, \rho)$ with $V(D) = \{t_0, t_1\}$.

For a partial BDD-ensemble $\mathcal{E}$ and a subset $F' \subseteq F$ of features, we denote by $\mathit{SExt}(\mathcal{E}, F')$ the set of all simple extensions $\mathcal{E}'$ such the at most one new inner vertex v is assigned to a feature in F'.

Lemma 15. *Let $C = (E, F, \mu)$ be a CI, let $\mathcal{E}$ be a partial* BDD*-ensemble, and let $F' \subseteq F$. Then, $\mathit{SExt}(B, F')$ can be computed in time $O(|F'|(\|\mathcal{E}\| + 1)^2 3^{\|\mathcal{E}\|})$ and $|\mathit{SExt}(\mathcal{E}, F')| \leq 2|F'|(\|\mathcal{E}\| + 1)^2 3^{\|\mathcal{E}\|-2|\mathcal{E}|} + 3$.*

Proof. This follows immediately from Lemma 14 together with the fact that there are at most 3 distinct trivial partial BDDs, i.e., the trivial BDD without a root, the trivial BDD with root t_0, and the trivial BDD with root t_1. □

Let $S \subseteq F$ be a *support set* of C. We define an equivalence relation w.r.t. S by saying that two examples are *equivalent w.r.t. S* if they agree on all features in S. Note that if S is a support set every equivalence class w.r.t. S is *homogeneous*, i.e., contains either only 1-example or only 0-examples. We say that a set $U \subseteq F \setminus S$ is *useful* for a given support set S if for every assignment $\tau_U : U \to \{0, 1\}$ there is an assignment $\tau_S : S \to \{0, 1\}$ such that $E[\tau_S] \neq \emptyset$ but $E[\tau_S \cup \tau_U] = \emptyset$. Informally, U is useful if every assignment of U filters out at least one previously non-empty equivalence class (w.r.t. S) completely. In other words, U is not useful (or *useless*) if there is an assignment of U that keeps (agrees with) at least one example from every non-empty equivalence class. The following lemma is now an analogue of [23, Lemma 11] for BDD-ensembles instead of DTs that will be crucial in bounding the number of features that need to be considered for the computation of $\mathit{SExt}(B)$.

Lemma 16. *Let $\mathcal{E}$ be a* BDD*-ensemble for a CI $C = (E, F, \mu)$ of minimum size and let S be a support set contained in $F(\mathcal{E})$. Then, $F(\mathcal{E}) \setminus S$ is useful.*

Proof. Suppose for a contradiction that this is not the case and let $U = F(\mathcal{E}) \setminus S$. Then, $U \neq \emptyset$ and there is an assignment $\tau_U : U \to \{0, 1\}$ such that $E[\tau_S \cup \tau_U] \neq \emptyset$ for every assignment $\tau_S : S \to \{0, 1\}$. Informally, we will show that we can obtain a smaller BDD-ensemble $\mathcal{E}'$ for C by "contracting" all vertices u in any BDD in $\mathcal{E}$ that are assigned to a feature in U, which contradicts the minimality of $\mathcal{E}$.

To obtain $\mathcal{E}'$ we will replace every BDD $B = (D, \rho)$ in $\mathcal{E}$, with the BDD $B^* = (D^*, \rho^*)$ that is obtained from B as follows. For every vertex $u \in V(D)$ with $\rho(u) \in U$ we do the following. If u is the root of D, then we remove u and make the $\tau_U(u)$-neighbor of u the new root of B^*. Otherwise, let $p_1, \ldots, p_\ell$ be the in-neighbors of u in B and let c_0 and c_1 be the 0-neighbor respectively 1-neighbor of u in D. Then, we remove u from D (together with all its incident arcs) and add an x-arc from p_i to $c_{\tau_U(u)}$ for every $1 \leq i \leq \ell$ (assuming that the arc (p_i, u) is an x-arc in D). Finally, ρ^* is the restriction of ρ to $V(D^*)$.

Then, $\mathcal{E}'$ is clearly still a BDD-ensemble and because $U \neq \emptyset$ also $|\mathcal{E}'| < |\mathcal{E}|$. We will now show that $\mathcal{E}'$ is still a BDD-ensemble for C, which contradicts the minimality of $\mathcal{E}$. Assume for a contradiction that this is not the case and there is an example $e \in E$ that is not classified correctly by $\mathcal{E}'$. Let e^* be any example in the equivalence class of e w.r.t. S that agrees with τ_U. Note that such an example exists because $E[\tau_S] \neq \emptyset$ for the assignment $\tau_S : S \to \{0, 1\}$ that agrees with e and therefore $E[\tau_S \cup \tau_U] \neq \emptyset$. But then, $\mathcal{E}'(e^*) = \mathcal{E}'(e)$ (because e and e^* agree on every feature in S and $\mathcal{E}'$ uses only features in S) and $\mathcal{E}'(e^*) = \mathcal{E}(e^*)$ (because e^* agrees with τ_U), which implies that e^* is not correctly classified by $\mathcal{E}$ and contradicts our assumption that $\mathcal{E}$ is a BDD-ensemble for C. □

Because a BDD-ensemble of size one is simply a BDD, we obtain the following as a corollary of Lemma 16.

Corollary 17. *Let $B = (D, A)$ be a* BDD *for a CI $C = (E, F, \mu)$ of minimum size and let S be a support set contained in $F(B)$. Then, $F(B) \setminus S$ is useful.*

We say that a set U_0 is a *branching set* for a support set S if every useful set U for S contains at least one feature in U_0.

Lemma 18 ([23, Lemma 14])**.** *There is a polynomial-time algorithm that given a support set S of a CI $C = (E, F, \mu)$ computes a branching set U_0 for S of size at most $2^{|S|}2\delta(C)$.*

Lemma 19. *Let $C = (E, F, \mu)$ be a CI and let $B = (D, \rho)$ be a partial* BDD *that is not a* BDD *for C. There is a polynomial-time algorithm that given C and B outputs a set of features $F' \subseteq F$ of size at most $n + 2^n 2\delta(C)$, where $n = |F(B)|$, such that every* BDD *of minimum size for C that is an extension of B also extends some partial* BDD *in $SExt(B, F')$.*

Proof. To compute F' we distinguish the following two cases. If $F(B)$ is not a support set for C, then there is a pair (e, e') of examples with $\mu(e) \neq \mu(e')$ such that both examples agree on all features in $F(B)$. In this case any BDD for C that extends B must contain one of the features in $\delta(e, e')$. Therefore, in this case we set $F' = \delta(e, e')$. Otherwise, we use Lemma 18 to compute a branching set U_0 for $F(B)$ and set $F' = F(B) \cup U_0$. This completes the description of the algorithm to compute F', which clearly runs in polynomial-time. Furthermore, we have that in both cases $|F'| \leq n + 2^n 2\delta(C)$. It remains to show that every minimum size BDD for C that extends B is an extension of a partial BDD in $SExt(S, F')$.

Let $B' = (D', \rho')$ be a BDD for C of minimum size that extends B. Then, $B' \neq B$ because B is not a BDD for C. If $F(B)$ is not a support set for C, then $F' = \delta(e, e')$ for some pair of examples with $\mu(e) \neq \mu(e')$ such that both examples agree on all features in $F(B)$. Since B is not a BDD for C but B' is, D' must contain a vertex v (that is not in D) with $\rho'(v) \in F'$. Then, $B'' = B'[V(D) \cup \{v\}]$ is a strict extension of B in $SExt(B, F')$ such that B' is an extension of B'', as required. If on the other hand $F(B)$ is a support set for C, then $F' = F(B) \cup U_0$ for a branching set U_0 for $F(B)$. Since B' is of minimum size, we obtain from Corollary 17 that $F(B') \setminus F(B)$ is useful. Therefore, by the definition of a branching set, if $F(B') \setminus F(B) \neq \emptyset$, then B' contains a vertex v (that is not in B) with $\rho'(v) \in U_0 \subseteq F'$. It follows that $B'' = B'[V(D) \cup \{v\}]$ is a strict extension of B in $SExt(B, F')$ such that B' is an extension of B'', as required. If on the other hand $F(B') \setminus F(B) = \emptyset$, then B' contains a vertex v (that is not in B) with $\rho'(v) \in F(B) \subseteq F'$, which again implies that $B'' = B'[V(D) \cup \{v\}]$ is a strict extension of B in $SExt(B, F')$ such that B' is an extension of B'', as required. □

The following is analogue of the above lemma for BDD-ensembles.

Lemma 20. *Let $C = (E, F, \mu)$ be a CI and let $\mathcal{E}$ be a partial* BDD*-ensemble that is not a* BDD*-ensemble for C. There is a polynomial-time algorithm that given C and $\mathcal{E}$ outputs a set of features $F' \subseteq F$ of size at most $n + 2^n 2\delta(C)$, where $n = |F(\mathcal{E})|$, such that every* BDD*-ensemble of minimum size for C that is an extension of $\mathcal{E}$ also extends some partial* BDD*-ensemble in $SExt(\mathcal{E}, F')$.*

Proof. Let $F_{\mathcal{E}} = F(\mathcal{E})$ and $n = |F_{\mathcal{E}}|$. To compute F' we distinguish the following two cases. If $F_{\mathcal{E}}$ is not a support set for C, then there is a pair (e, e') of examples with $\mu(e) \neq \mu(e')$ such that both examples agree on all features in $F_{\mathcal{E}}$. In this case, it follows from Observation 1 that any BDD-ensemble for C that extends B must contain one of the features in $\delta(e, e')$. Therefore, in this case we set $F' = \delta(e, e')$. Otherwise, we use Lemma 18 to compute a branching set U_0 for $F_{\mathcal{E}}$ and set $F' = F_{\mathcal{E}} \cup U_0$. This completes the description of the algorithm to compute F', which clearly runs in polynomial-time. Furthermore, we have that in both cases $|F'| \leq n + 2^n 2\delta(C)$. It remains to show that every minimum size BDD-ensemble for C that extends $\mathcal{E}$ is an extension of a partial BDD-ensemble in $SExt(\mathcal{E}, F')$.

Let $\mathcal{E}'$ be a BDD-ensemble for C of minimum size that extends $\mathcal{E}$. Note that $\mathcal{E}'$ is a strict extension of $\mathcal{E}$ because $\mathcal{E}$ is not a BDD-ensemble for C. If $F_{\mathcal{E}}$ is not a support set for C, then $F' = \delta(e, e')$ for some pair of examples with $\mu(e) \neq \mu(e')$ such that both examples agree on all features in $F_{\mathcal{E}}$. Since $\mathcal{E}$ is not a BDD for C but $\mathcal{E}'$ is, there must exists a partial BDD $B' = (D', \rho')$ in $\mathcal{E}'$ such that D' contains a vertex v with $\rho'(v) \in F'$. Since $\mathcal{E}'$ is an extension of $\mathcal{E}$, it holds that either B' is an extension of some partial BDD $B = (D, \rho) \in \mathcal{E}$ or not (i.e., B' has no corresponding partial BDD in $\mathcal{E}$). In the former case, $\mathcal{E}'$ is an extension of $\mathcal{E} \setminus \{B\} \cup \{B'[V(D) \cup \{v\}]\} \in SExt(\mathcal{E}, F')$ and in the latter case $\mathcal{E}'$ is an extension of $\mathcal{E} \cup B'[\{t_0, t_1, v\}] \in SExt(\mathcal{E}, F')$, as required.

If on the other hand $F_{\mathcal{E}}$ is a support set for C, then $F' = F_{\mathcal{E}} \cup U_0$ for a branching set U_0 for $F_{\mathcal{E}}$. Since $\mathcal{E}'$ is of minimum size, we obtain from Lemma 16 that $F(\mathcal{E}') \setminus F_{\mathcal{E}}$ is useful. Therefore, by the definition of a branching set and the fact that $\mathcal{E}'$ is a strict extension of $\mathcal{E}$, some BDD $B' = (D', \rho')$ in $\mathcal{E}'$ must contain a vertex v that is in no BDD of $\mathcal{E}$ with $\rho'(v) \in F'$. Moreover, if B' is an extension of some partial BDD $B = (D, \rho) \in \mathcal{E}$, then $\mathcal{E}'$ is an extension of $\mathcal{E} \setminus \{B\} \cup \{B'[V(D) \cup \{v\}]\} \in SExt(\mathcal{E}, F')$ and otherwise $\mathcal{E}'$ is an extension of $\mathcal{E} \cup B'[\{t_0, t_1\}] \in SExt(\mathcal{E}, F')$. □

Lemma 21. BDD*s are*
$(2(s + 2^s 2\delta)(s + 1)^2 3^{s-2}, 2^{O(s)} n^{O(1)})$*-extendable.*

Proof. Let $B = (D, \rho)$ be a partial BDD of size at most s. Then, because of Lemma 19, the set $SExt(B, F')$ is a full set of strict extensions for B, where F' is the set of features of size at most $s+2^s2\delta$ that can be computed in polynomial-time. Moreover, it follows from Lemma 14 that given B and F', the set $SExt(B, F')$ has size at most $2|F'|(\|B\| + 1)^2 3^{\|B\|-2}$ and can be computed in time $O(|F'|(\|B\|+1)^2 3^{\|B\|-2}) = 2^{O(s)}$. Therefore, $SExt(B)$ has size at most $2(s+2^s2\delta)(s+1)^2 3^{s-2}$ and can be computed in time $2^{O(s)}n^{O(1)}$, which shows that BDDs are $(2(s + 2^s2\delta)(s + 1)^2 3^{s-2}, 2^{O(s)}n^{O(1)})$-extendable. □

Combining Lemma 21 with Theorem 4, we obtain the following.

Corollary 22. BDD-MMS *and is fixed-parameter tractable parameterized by* $s + \delta$.

We obtain the following result for BDD-ensembles.

Lemma 23. BDD*-ensembles are* $(2(s + 2^s2\delta)(s + 1)^2 3^{s-2} + 3, 2^{O(s)}n^{O(1)})$*-extendable.*

Proof. Let $\mathcal{E}$ be a partial BDD-ensemble of size at most s. Then, because of Lemma 20 the set $SExt(\mathcal{B}, F')$ is a full set of strict extensions for $\mathcal{E}$, where F' is the set of features of size at most $s + 2^s2\delta$ that can be computed in polynomial-time. Moreover, it follows from Lemma 15 that given B and F', the set $SExt(\mathcal{E}, F')$ has size at most $2|F'|(s + 1)^2 3^{s-2} + 3$ and can be computed in time $O(|F'|(s+1)^2 3^s) = 2^{O(s)}$. Therefore, $SExt$ has size at most $2(s+2^s2\delta)(s+1)^2 3^{s-2}+3$ and can be computed in time $2^{O(s)}n^{O(1)}$, which shows that BDDs are $(2(s+2^s2\delta)(s+1)^2 3^{s-2}+3, 2^{O(s)}n^{O(1)})$-extendable. □

Combining Lemma 23 with Theorem 4, we obtain the following.

Corollary 24. BDD-MEMS *is fixed-parameter tractable parameterized by* $s + \delta$.

9. Completing the Parameterized Complexity Landscape

In this section, we provide complementary hardness results. In particular, we will show that finding a minimum size model is W[2]-hard parameterized by s alone for all model types considered in this paper. This is already known in the case of DTs, but not for DSs, DLs, and BDDs. We will then consider replacing s by weaker but natural parameters. In particular, we will consider the parameters number of terms as well as the maximum size of any term as a parameter replacing size for DSs and DLs. Surprisingly, we will show that even finding a DS (or DL) with only one term (or alternatively with terms of maximum size 1) is NP-hard even if δ is equal to 2. Our hardness results are based on a simple reduction from the the well-known Hitting Set (HS) problem, where given a family $\mathcal{F}$ of sets over a universe U and an integer k, the task is to decide whether $\mathcal{F}$ has a *hitting set* of size at most k, i.e., a subset $H \subseteq U$ with $|H| \leq k$ such that $H \cap F \neq \emptyset$ for every $F \in \mathcal{F}$. Let $I = (I, \mathcal{F}, k)$ be an instance of HS. We denote by $C(I) = (E, F, \mu)$ the CI defined as follows. $C(I)$ has one feature f_u for every $u \in U$, one example e_b with $\mu(e_b) = 1$ that is 0 at all features, and one example e_F with $\mu(e_b) = 0$ for every $F \in \mathcal{F}$ that is 1 for every feature f_u with $u \in F$ and otherwise 0. Note that the reduction was already used by Ordyniak and Szeider [23] to show the result for DTs.

Theorem 25. Γ-MMS *is* NP*-hard and* W[2]*-hard parameterized by* s *for every* $\Gamma \in \{\mathtt{DS}, \mathtt{DL}, \mathtt{DT}, \mathtt{BDD}\}$.

Proof. We will show the results for the decision variant of Γ-MMS and use a parameterized reduction from HS, which is well-known to be W[2]-complete parameterized by k. That is, for a given instance $I = (U, \mathcal{F}, k)$ of HS, the CI of our instance of Γ-MMS is given by $C(I)$. We first show that $\mathcal{F}$ has a hitting set of size at most k if and only if $C(I)$ has a support set of size at most k; note that this equivalence was already shown in [23], we repeat it here to be self-contained.

Towards showing the forward direction of the claim, let H be a hitting set for $\mathcal{F}$ of size at most k. We claim that $S = \{f_u \mid u \in H\}$ is a support set of size at most k for $C(I, b)$. Suppose for a contradiction that this is not the case, then by the definition of a support set, there is an example e_F for some $F \in \mathcal{F}$ such that $S \cap \delta(e_1, e_F) = \emptyset$. However, then $H \cap F = \emptyset$, which contradicts our assumption that H is a hitting set for $\mathcal{F}$.

Towards showing the reverse direction of the claim, let S be a support set for $C(I)$ of size at most k. We claim that $H = \{u \mid f_u \in S\}$ is a hitting set of size at most k for $\mathcal{F}$. Suppose for a contradiction that this is not the case, then by the definition of a hitting set, there is a set $F \in \mathcal{F}$ such that $H \cap F = \emptyset$. However, then $S \cap \delta(e_1, e_F) = \emptyset$, which contradicts our assumption that S is a support set for $C(I)$.

It now follows from Observation 1 that the size of any DS, DL, DTor BDDfor $C(I, b)$ is at least equal to the size of a minimum support set plus 1, plus 1, plus 1 or plus 2, respectively. On the other hand, given a support set S it is easy to construct a DS of size $|S| + 1$, a DL of size $|S| + 1$, a DT of size $|S| + 1$, and a BDD of size $|S| + 2$ for $C(I)$. This implies that the reduction also shows W[2]-hardness for the decision variant of Γ-MMS for every $\Gamma \in \{\mathtt{DS}, \mathtt{DL}, \mathtt{DT}, \mathtt{BDD}\}$ parameter s and completes the proof of the theorem. □

Theorem 26. *Given a CI $C = (E, F, \mu)$ with $\delta(C) = 2$ and an integer k. It is* NP*-hard to decide whether there is a* DS/DL *for C with at most k literals that either:*

- *uses at most one term/rule (plus a default rule) or*
- *uses at most one literal per term.*

Proof. We again use the reduction from HS, which is well-known to be NP-hard even if all sets have size exactly 2. Let $I = (U, \mathcal{F}, k)$ be an instance of HS, where all sets in $\mathcal{F}$ have size exactly two.

We start by showing that $\mathcal{F}$ has a hitting set of size at most k if and only if $C(I)$ has a DS/DL of size at most k using at most one term/rule (plus the default rule). Towards showing the forward direction, let H be a hitting set for $\mathcal{F}$ of size at most k. Then, $(\{t_H\}, 0)$, where t_H is the term $\{f_u = 0 \mid u \in H\}$ is a DS for $C(I)$ and $((t_H, 1), (\emptyset, 0))$ is a DL for $C(I)$, as required. The reverse direction now follows from Observation 1 showing that the features of any DS/DL must form a support set, which we know from the proof of Lemma 25 corresponds to a hitting set for $\mathcal{F}$.

It remains to show that $\mathcal{F}$ has a hitting set of size at most k if and only if $C(I)$ has a DS/DL of size at most k using at most one literal per term/rule. Towards showing the forward direction, let H be a hitting set for $\mathcal{F}$ of size at most k. Then, $(\{f_u = 1 \mid u \in H\}, 1)$ is a DS for $C(I)$ and $((t_H, 1), (\emptyset, 0))$ is a DL for $C(I)$, as required. The reverse direction now follows from Observation 1 showing that the features of any DS/DL must form a support set, which we know from the proof of Lemma 25 corresponds to a hitting set for $\mathcal{F}$. □

10. Conclusion

We present a general framework for learning small (ensembles of) models (parameterized by $s + \delta$) and show its applicability to DSs, DLs, DTs, and BDDs. Since our algorithm enumerates

all minimum BDDs, it can also be applied to more restrictive variants of BDDs, such as free and ordered BDDs. While we provide our framework only for CIs with Boolean domain features, all our tractability results can be easily extended to features with unbounded domains as long as the domains are ordered and the maximum size of the domain is taken as an additional parameter. We leave it open, however, whether the recent tractability result for learning small DTs without domain as a parameter [7] can be extended to BDDs or even DSs or DLs, and we conjecture that this is not the case. Another interesting question is whether it is possible to show that the dependency on the parameters of our algorithms is the best possible or whether, in particular, our algorithm for BDDs can be significantly improved.

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

[1] Aglin, G., Nijssen, S., Schaus, P., 2020. Learning optimal decision trees using caching branch-and-bound search, in: AAAI, AAAI Press. pp. 3146–3153.
[2] Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M.I., Rudin, C., 2017. Learning certifiably optimal rule lists for categorical data. J. Mach. Learn. Res. 18, 234:1–234:78.
[3] Avellaneda, F., 2020. Efficient inference of optimal decision trees, in: AAAI, AAAI Press. pp. 3195–3202.
[4] Dabrowski, K.K., Eiben, E., Ordyniak, S., Paesani, G., Szeider, S., 2024. Learning small decision trees for data of low rank-width, in: AAAI, AAAI Press. pp. 10476–10483.
[5] Dash, S., Günlük, O., Wei, D., 2018. Boolean decision rules via column generation, in: NeurIPS, pp. 4660–4670.
[6] Demirovic, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., Stuckey, P.J., 2022. Murtree: Optimal decision trees via dynamic programming and search. J. Mach. Learn. Res. 23, 26:1–26:47.
[7] Eiben, E., Ordyniak, S., Paesani, G., Szeider, S., 2023. Learning small decision trees with large domain, in: IJCAI, ijcai.org. pp. 3184–3192.
[8] Florio, A.M., Martins, P., Schiffer, M., Serra, T., Vidal, T., 2023. Optimal decision diagrams for classification, in: AAAI, AAAI Press. pp. 7577–7585.
[9] Gahlawat, H., Zehavi, M., 2024. Learning small decision trees with few outliers: A parameterized perspective, in: AAAI, AAAI Press. pp. 12100–12108.
[10] Ghosh, B., Meel, K.S., 2019. IMLI: an incremental framework for maxsat-based learning of interpretable classification rules, in: AIES, ACM. pp. 203–210.
[11] Gunning, D., Aha, D.W., 2019. Darpa's explainable artificial intelligence (XAI) program. AI Mag. 40, 44–58.
[12] Hu, H., Huguet, M., Siala, M., 2022. Optimizing binary decision diagrams with maxsat for classification, in: AAAI, AAAI Press. pp. 3767–3775.
[13] Hu, X., Rudin, C., Seltzer, M.I., 2019. Optimal sparse decision trees, in: NeurIPS, pp. 7265–7273.
[14] Hyafil, L., Rivest, R.L., 1976. Constructing optimal binary decision trees is np-complete. Inf. Process. Lett. 5, 15–17.
[15] Ignatiev, A., Marques-Silva, J., 2021. Sat-based rigorous explanations for decision lists, in: SAT, Springer. pp. 251–269.
[16] Ignatiev, A., Pereira, F., Narodytska, N., Marques-Silva, J., 2018. A sat-based approach to learn explainable decision sets, in: IJCAR, Springer. pp. 627–645.
[17] Kobourov, S.G., Löffler, M., Montecchiani, F., Pilipczuk, M., Rutter, I., Seidel, R., Sorge, M., Wulms, J., 2023. The influence of dimensions on the complexity of computing decision trees, in: AAAI, AAAI Press. pp. 8343–8350.
[18] Komusiewicz, C., Kunz, P., Sommer, F., Sorge, M., 2023. On computing optimal tree ensembles, in: ICML, PMLR. pp. 17364–17374.
[19] Lin, J., Zhong, C., Hu, D., Rudin, C., Seltzer, M.I., 2020. Generalized and scalable optimal sparse decision trees, in: ICML, PMLR. pp. 6150–6160.

[20] van der Linden, J.G.M., de Weerdt, M., Demirovic, E., 2022. Fair and optimal decision trees: A dynamic programming approach, in: NeurIPS.
[21] Molnar, C., 2022. Model-agnostic interpretable machine learning. Ph.D. thesis. Ludwig Maximilian University of Munich, Germany.
[22] Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J., 2018. Learning optimal decision trees with SAT, in: IJCAI, ijcai.org. pp. 1362–1368.
[23] Ordyniak, S., Szeider, S., 2021. Parameterized complexity of small decision tree learning, in: AAAI, AAAI Press. pp. 6454–6462.
[24] Rudin, C., 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat. Mach. Intell. 1, 206–215.
[25] Schidler, A., Szeider, S., 2024. Sat-based decision tree learning for large data sets. J. Artif. Intell. Res. 80, 875–918.
[26] Shati, P., Cohen, E., McIlraith, S.A., 2021. Sat-based approach for learning optimal decision trees with non-binary features, in: CP, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 50:1–50:16.
[27] Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C., Schaus, P., 2020. Learning optimal decision trees using constraint programming. Constraints An Int. J. 25, 226–250.
[28] Verwer, S., Zhang, Y., 2019. Learning optimal classification trees using a binary linear program formulation, in: AAAI, AAAI Press. pp. 1625–1632.
[29] Yu, J., Ignatiev, A., Bodic, P.L., Stuckey, P.J., 2020. Optimal decision lists using SAT. CoRR abs/2010.09919.
[30] Yu, J., Ignatiev, A., Stuckey, P.J., Bodic, P.L., 2021. Learning optimal decision sets and lists with SAT. J. Artif. Intell. Res. 72, 1251–1279.