A General Theoretical Framework for Learning Smallest Interpretable Models

Sebastian Ordyniak[1], Giacomo Paesani[1], Mateusz Rychlicki[1], Stefan Szeider[2]
[1]University of Leeds, Leeds, UK,
[2]TU Wien, Vienna, Austria
{sordyniak,giacomopaesani,mkrychlicki}@gmail.com, sz@ac.tuwien.ac.at

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.

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 (Gunning 2019). 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 (Rudin 2019; Molnar 2022). However, also symbolic models become increasingly opaque if their size increases. Hence one is interested in finding smallest models. This task is typically NP-hard; thus, practitioners have utilized powerful tools like SAT, MIP, and CP solvers to compute small or even smallest models that fit the data (Ignatiev and Marques-Silva 2021; Narodytska et al. 2018; Schidler and Szeider 2021; Shati, Cohen, and McIlraith 2021). 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 (Ordyniak and Szeider 2021; Eiben et al. 2023; Kobourov et al. 2023; Komusiewicz et al. 2023) 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.

[1]Empirical investigations show that δ is reasonably small for real-world data sets (Ordyniak and Szeider 2021).

	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`	-	$\delta 3^{\mathcal{O}(s)}$	$\delta 3^{\mathcal{O}(s)}$

Table 1: The number of branches required by our framework for `DS`s, `DL`s, `DT`s, and `BDD`s, both for learning simple models and for learning ensemble models. All model types except `BDD`s 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 $\mathcal{O}^*(b^2)$.

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 computation of a smallest model, 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* (`DT`s) to *decision sets* (`DS`s), *decision lists* (`DL`s), *binary decision diagrams* (`BDD`s), 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 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 `DS`s, `DL`s, and `DT`s, we can employ an adapted version of the annotation approach that has recently been developed for DTs (Komusiewicz et al. 2023). 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. (2023), 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 (Komusiewicz et al. 2023). 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 (2021) 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 $\mathcal{O}^*(b^s)$ ($\mathcal{O}^*$ 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 $\delta = 2$.

Preliminaries

Classification Instances. A *(binary) classification instance (CI)* is a triple $C = (E, F, \tau)$, where E is a set of examples over a set of binary features F and τ is a classification function $\tau : E \rightarrow \{0, 1\}$. We commonly say that an example e is a 0-example, or negative example, (1-example, or positive example) if $\tau(e) = 0$ ($\tau(e) = 1$) 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 $\tau : F' \rightarrow \{0, 1\}$, 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 $\tau_1 : F_1 \rightarrow \{0, 1\}$ and $\tau_2 : F_2 \rightarrow \{0, 1\}$, where $F_1, F_2 \subseteq F$ and $F_1 \cap F_2 = \emptyset$, we denote by $\tau_1 \cup \tau_2$ the assignment $\tau : F_1 \cup F_2 \rightarrow \{0, 1\}$ 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 $\tau(e) + \tau(e') = 1$.

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

Models and Support Sets. In the following we will define models of different types; 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 $M : E \rightarrow \{0, 1, \boldsymbol{u}\}$ 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 $M(e) = \tau(e)$ 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 $C = (E, F, \tau)$. Then, $F(M)$ is a support set for C.*

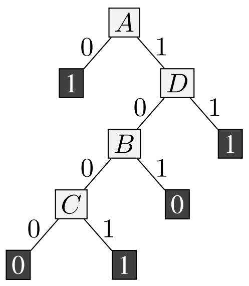
Decision Sets. A *term* t over C is a set of *literals* with each literal being of the form $(f = z)$ where $f \in F$ and $z \in \{0, 1\}$. A *rule* r is a pair (t, c) where t is a term and $c \in \{0, 1\}$. We say that a rule (t, c) is a *c-rule*. We say that a term t (or rule (t, c)) *applies to (or agrees with)* an example e if $e(f) = z$ for every element $(f = z)$ of t. Note that the empty 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 $M : E \rightarrow \{0, 1\}$ 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 $L : E \rightarrow \{0, 1\}$ 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 and $\lambda : V(T) \rightarrow F \cup \{0, 1\}$ 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 $M : E \rightarrow \{0, 1\}$ 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

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

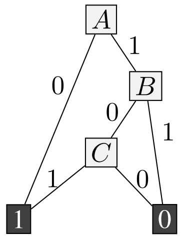


Figure 1: A classification instance C and four models that classify C: a decision tree, a decision set, a decision list, and a binary decision diagram (from left to right).

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, τ). 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 $T \in \{\texttt{DS}, \texttt{DL}, \texttt{DT}, \texttt{BDD}\}$. 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 $T \in \{\texttt{DS}, \texttt{DL}, \texttt{DT}, \texttt{BDD}\}$. We consider the following problems.

T-MINIMUM MODEL SIZE (T-MMS)

INSTANCE:	A CI $C = (E, F, \tau)$ and an integer s.
QUESTION:	Find a model of type T and size at most s for C or report correctly that no such model exists.

T-MINIMUM ENSEMBLE MODEL SIZE (T-MEMS)

INSTANCE:	A CI $C = (E, F, \tau)$ and an integer s.
QUESTION:	Find an ensemble model of type T 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.

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. 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 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 $C = (E, F, \tau)$ and if not providing an example $e \in E$ that is not correctly classified by M can be achieved in time $\mathcal{O}(|E|\|M\|)$.

(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.

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 $C = (E, F, \tau)$ and let $e \in E$ be an example not correctly classified by M. A *full set of strict extensions for a 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 $\mathcal{O}(g(|M|, |C|))$ 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 T-MMS for any strongly extendable model type T.

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

Algorithm 1: Generic Algorithm for finding a minimum model of size at most s for any strongly extendable model-type T.

```
Input: CI $C = (E, F, \tau)$ and integer $s$.
Output: return a minimum model for $C$ of type $T$ 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 $|M| \geq s$ then
7:          return nil
8:      $e \leftarrow$ any example not correctly classified by $M$
9:      $\mathcal{M} \leftarrow$ FINDSTRICTEXTSSTR($C$, $(M, A)$, $e$)
10:     $B \leftarrow$ nil
11:     for $(M', A') \in \mathcal{M}$ do
12:         if $|M'| \leq s$ then
13:             $A \leftarrow$ FINDOPTEXTSTR($C$, $s$, $(M', A')$)
14:             if $A \neq$ nil and ($B =$ nil or $|B| > |A|$) then
15:                 $B \leftarrow A$
16:     return $B$
```

Proof. We solve T-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 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 T 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 T 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 $\mathcal{O}(g(|M|, |C|))$ and returns at most $f(|M|, \delta) \leq f(s, \delta)$ strict 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 8, 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 $\mathcal{O}(|E|s)$. Therefore, the total run-time of the algorithm is at most $\mathcal{O}((f(s, \delta))^s(g(s, |C|) + |E|s))$. ☐

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

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

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 T 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 $\mathcal{O}(g(|M|, |C|))$ that given a CI C and a partial model M of type T 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.

Algorithm 2: Generic Algorithm for finding a minimum model of size at most s for any extendable model-type T.

```
Input: CI $C = (E, F, \tau)$ and integer $s$.
Output: return a minimum model for $C$ of type $T$ 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 $|M| \geq s$ then
 7:         return nil
 8:     $\mathcal{M} \leftarrow$ FINDSTRICTEXTS($C$, $s$, $(M, A)$)
 9:     $B \leftarrow$ nil
10:     for $(M', A') \in \mathcal{M}$ do
11:         if $|M'| \leq s$ then
12:             $A \leftarrow$ FINDOPTEXT($C$, $s$, $(M', A')$)
13:             if $A \neq$ nil and ($B =$ nil or $|B| > |A|$) then
14:                 $B \leftarrow A$
15:     return $B$
```

Furthermore, the extension of the framework to ensembles of models 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 T-MMS for any extendable model type T. The bounded-depth search algorithm behind the theorem is illustrated in Algorithm 2.

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

Decision Sets, Decision Lists, and Decision Trees

Due to space reasons, we are only able to illustrate our approach for DLs and leave the description of the corresponding results for DSs and DTs to the full version of the paper.

In the case of DLs an annotated DL is simple a pair (L, A), where L is a decision list and $A : L \rightarrow E$ 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 $\alpha : L \rightarrow L'$ 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 $\alpha : L \rightarrow L'$ 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 $\alpha : L \rightarrow L'$ such that for every $r = (t, b) \in L$ with $r' = (t', b') = \alpha(r)$, it holds that $b' = b$, Algorithm 3: Algorithm for finding a full set of strict extensions for DLs.

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

$t \subseteq t'$, 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 5. *Decision Lists are strongly $(\delta + |L| + 1, |L| + \delta)$-extendable.*

Proof. We claim that Algorithm 3 shows the result, i.e., we need to show that Algorithm 3 runs in time $\mathcal{O}(|L| + \delta)$ and given a CI $C = (E, F, \tau)$, 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 3 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 $(\emptyset, \tau(e))$ annotated by e at any possible position in L. Otherwise, let $r' = (t', 1 - \tau(e))$ 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 $r = (\emptyset, \tau(e))$ 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 $f \in \delta(e', e)$.

Towards showing the correctness of Algorithm 3, 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 $r' = (t', 1 - \tau(e))$ 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. ☐

Combining Lemma 5 with Theorem 2, we obtain the following.

Corollary 6. DL-MMS *can be solved in time $\mathcal{O}((\delta + s + 1)^s |E| s)$ and is therefore fixed-parameter tractable parameterized by $s + \delta$.*

Combining Lemma 5 with Theorem 3, we obtain the following.

Corollary 7. DL-MEMS *can be solved in time $\mathcal{O}((1 + s + s\delta)^s |E| s)$ and is therefore fixed-parameter tractable parameterized by $s + \delta$.*

Binary Decision Diagrams

For space reasons, we will only illustrate the main ideas behind the proof for BDDs and leave the full proof as well as the proof for BDD-ensemble to the full version of the paper.

Let $C = (E, F, \tau)$ 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, 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 $\mathit{SExt}(B, F')$ 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 $\mathit{SExt}(B) = \mathit{SExt}(B, F)$, then we could also bound the size of $\mathit{SExt}(B)$.

Lemma 8. *Let $C = (E, F, \tau)$ be a CI, let $B = (D, \rho)$ be a partial* BDD, *and let $F' \subseteq F$. Then, $\mathit{SExt}(B, F')$ can be computed in time $\mathcal{O}(|F'|(\|B\| + 1)^2 3^{\|B\|-2})$ and $|\mathit{SExt}(B, F')| \leq 2|F'|(\|B\| + 1)^2 3^{\|B\|-2}$.*

Using an approach that is very similar to the approach employed by Ordyniak and Szeider (2021), in particular the notion of useful sets, we are now able to bound the number of features that have to be considered for the computation of $\mathit{SExt}(B)$ and obtain the following lemma.

Lemma 9. *Let $C = (E, F, \tau)$ 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 $\mathit{SExt}(B, F')$.*

Lemma 10. BDD*s are* $(2(s + 2^s 2\delta)(s + 1)^2 3^{s-2}, 2^{\mathcal{O}(s)} n^{\mathcal{O}(1)})$*-extendable.*

Proof. Let $B = (D, \rho)$ be a partial BDD of size at most s. Then, because of Lemma 9, the set $\mathit{SExt}(B, F')$ is a full set of strict extensions for B, where F' is the set of features of size at most $s + 2^s 2\delta$ that can be computed in polynomial-time. Moreover, it follows from Lemma 8 that given B and F', the set $\mathit{SExt}(B, F')$ has size at most $2|F'|(\|B\| + 1)^2 3^{\|B\|-2}$ and can be computed in time $\mathcal{O}(|F'|(\|B\| + 1)^2 3^{\|B\|-2}) = 2^{\mathcal{O}(s)}$. Therefore, $\mathit{SExt}(B)$ has size at most $2(s + 2^s 2\delta)(s + 1)^2 3^{s-2}$ and can be computed in time $2^{\mathcal{O}(s)} n^{\mathcal{O}(1)}$, which shows that BDDs are $(2(s + 2^s 2\delta)(s + 1)^2 3^{s-2}, 2^{\mathcal{O}(s)} n^{\mathcal{O}(1)})$-extendable. □

Combining Lemma 10 with Theorem 4, we obtain the following.

Corollary 11. BDD-MMS *and is fixed-parameter tractable parameterized by $s + \delta$.*

We also obtain the following result for BDD-ensembles.

Lemma 12. BDD*-ensembles are* $(2(s+2^s 2\delta)(s+1)^2 3^{s-2} + 3, 2^{\mathcal{O}(s)} n^{\mathcal{O}(1)})$*-extendable.*

Combining Lemma 12 with Theorem 4, we obtain the following.

Corollary 13. BDD-MEMS *is fixed-parameter tractable parameterized by $s + \delta$.*

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, which has previously been employed by Ordyniak and Szeider (2021) to show hardness results for DTs.

Theorem 14. T-MMS *is* NP*-hard and* W[2]*-hard parameterized by s for every $T \in \{$DS, DL, DT, BDD$\}$.*

Theorem 15. *Given a CI $C = (E, F, \tau)$ 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.*

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 (Eiben et al. 2023) 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

Eiben, E.; Ordyniak, S.; Paesani, G.; and Szeider, S. 2023. Learning Small Decision Trees with Large Domain. *Proc. IJCAI 2023*, to appear.

Gunning, D. 2019. DARPA's explainable artificial intelligence (XAI) program. *Proc. IUI 2019*.

Ignatiev, A.; and Marques-Silva, J. 2021. SAT-Based Rigorous Explanations for Decision Lists. *Proc. SAT 2021*, 12831: 251–269.

Kobourov, S. G.; Löffler, M.; Montecchiani, F.; Pilipczuk, M.; Rutter, I.; Seidel, R.; Sorge, M.; and Wulms, J. 2023. The Influence of Dimensions on the Complexity of Computing Decision Trees. *Proc. AAAI 2023*, 8343–8350.

Komusiewicz, C.; Kunz, P.; Sommer, F.; and Sorge, M. 2023. On Computing Optimal Tree Ensembles. *Proc. ICML 2023*, 202: 17364–17374.

Molnar, C. 2022. *Interpretable Machine Learning*. 2 edition.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning Optimal Decision Trees with SAT. *Proc. IJCAI 2018*, 1362–1368.

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

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5): 206–215.

Schidler, A.; and Szeider, S. 2021. SAT-based Decision Tree Learning for Large Data Sets. *Proc. AAAI 2021*, 3904–3912.

Shati, P.; Cohen, E.; and McIlraith, S. A. 2021. SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features. *Proc. CP 2021*, 210(50): 1–16.