From Formal Boosted Tree Explanations to Interpretable Rule Sets

Jinqiang Yu
Department of Data Science and AI, Monash University, Australia
Australian Research Council OPTIMA ITTC, Australia

Alexey Ignatiev
Department of Data Science and AI, Monash University, Australia

Peter J. Stuckey
Department of Data Science and AI, Monash University, Australia
Australian Research Council OPTIMA ITTC, Australia

Abstract

The rapid rise of Artificial Intelligence (AI) and Machine Learning (ML) has invoked the need for *explainable AI* (XAI). One of the most prominent approaches to XAI is to train rule-based ML models, e.g. decision trees, lists and sets, that are deemed interpretable due to their transparent nature. Recent years have witnessed a large body of work in the area of constraints- and reasoning-based approaches to the inference of interpretable models, in particular decision sets (DSes). Despite being shown to outperform heuristic approaches in terms of accuracy, most of them suffer from scalability issues and often fail to handle large training data, in which case no solution is offered. Motivated by this limitation and the success of gradient boosted trees, we propose a novel anytime approach to producing DSes that are both accurate and interpretable. The approach makes use of the concept of a generalized formal explanation and builds on the recent advances in formal explainability of gradient boosted trees. Experimental results obtained on a wide range of datasets, demonstrate that our approach produces DSes that more accurate than those of the state-of-the-art algorithms and comparable with them in terms of explanation size.

2012 ACM Subject Classification Computing methodologies → Machine learning

Keywords and phrases Decision set; interpretable model; gradient boosted tree; BT compilation

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material *Software (Source Code)*: `https://github.com/jinqiang-yu/cpl/`

Acknowledgements This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

1 Introduction

Rapid development of Artificial Intelligence (AI) and Machine Learning (ML) have revolutionized all aspects of human lives in recent years [30, 1]. However, decisions made by most widely used ML models are hard for humans to understand hence the interest in the theory and practice of *Explainable AI* (XAI) rises.

One major approach to XAI is to compute *post-hoc* explanations for ML predictions to answer a *"why"* question [34, 44], i.e. why the prediction is made. Although heuristic approaches to post-hoc explanations prevail [34, 44, 43], they suffer from a number of weaknesses [21, 16, 49, 52]. Formal methods [48, 20, 37] provide alternative approaches to explanations that avoid these weaknesses. Another alternative approach to XAI is to compute *interpretable* ML models, i.e. logic-based models, including decision trees [40], decision lists [46], and decision sets [29]. These models enable decision makers to obtain

succinct explanations from the models directly. In this paper, we focus on the decision set (DS) models.

Decisions sets are particularly easy to explain: the rule that fired is an explanation of the decision. This led to an upsurge in interest of decision sets that are both interpretable and accurate. Recent work [50] uses propositional satisfiability (SAT) to generate minimum-size decision sets that are perfectly accurate on the training data, and demonstrates that decision sets that completely agree with the training data outperform others in terms of accuracy. A more scalable maximum satisfiability (MaxSAT) approach [18] to this problem was then proposed. Unfortunately, both of these methods are unable to provide any decision information if a dataset is not completely solved.

Motivated by these works and their limitations, this paper aims at making a bridge between formal post-hoc explainability and interpretable DS models. In particular, the paper focuses on developing a novel anytime approach to computing decision sets that are both interpretable and accurate, by compiling a gradient boosted tree model into a decision set on demand with the use of formal explanations. This is done with the use of the recent approach [17] to compute abductive explanations for gradient boosted trees using maximum satisfiability (MaxSAT). Furthermore, the paper proposes a range of post-hoc model reduction heuristics aiming at enhancing interpretability of the result models, done with MaxSAT and integer linear programming (ILP). The experimental results show that compared with other state-of-the-art methods, decision sets generated by the proposed approach are more accurate, and comparable with the competition in terms of interpretability.

2 Preliminaries

SAT and MaxSAT. The standard definitions for propositional satisfiability (SAT) and maximum satisfiability (MaxSAT) solving are assumed [3]. A propositional formula ϕ is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *clause* is a disjunction of literals, where a *literal* is either a Boolean variable b or its negation $\neg b$. A *truth* assignment μ is a mapping from the set of variables to $\{0, 1\}$. A clause is said to be *satisfied* by truth assignment μ if one of the literals in the clause is assigned value 1; otherwise, the clause is *falsified*. If all clauses in formula ϕ are satisfied by assignment μ, ϕ is satisfied; otherwise, assignment μ falsifies ϕ. A CNF formula ϕ is *unsatisfiable* if there exists no assignment satisfying ϕ.

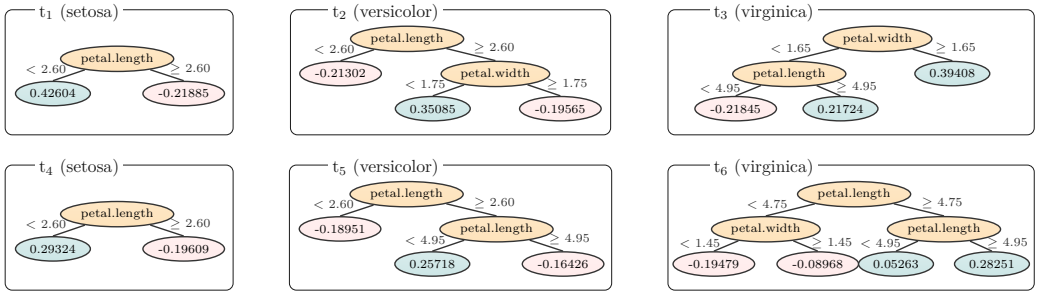
In the context of unsatisfiable formulas, the MaxSAT problem consists in finding a truth assignment that maximizes the number of satisfied clauses. Hereinafter, we use a variant of MaxSAT called Partial Weighted MaxSAT [3, Chapters 23 and 24]. The formula ϕ in this variant is represented as a conjunction of *hard* clauses $\mathcal{H}$, which must be satisfied, and *soft* clauses $\mathcal{S}$ where each of them is associated with a weight representing a preference to satisfy them, i.e. $\phi = \mathcal{H} \land \mathcal{S}$. Partial Weighted MaxSAT problems aim at finding a truth assignment μ that satisfies all hard clauses and maximizes the total weight of satisfied soft clauses.

Classification Problems. We consider classification problems with a set of classes[1] $\mathcal{K} = \{1, \ldots, k\}$, and a set of features $\mathcal{F} = \{1, \ldots, m\}$. The value of each feature $i \in \mathcal{F}$ is taken from its corresponding (numeric) domain D_i. As a result, the entire feature space is defined as $\mathbb{F} \triangleq \prod_{i=1}^{m} D_i$. A concrete point represented by $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$, s.t. each v_i is a constant

[1] Non-integer class labels can be mapped to a set $\{1, \ldots, |\mathcal{K}|\}$.

IF	“petal.length” < 2.60	THEN	class = “setosa”
IF	$2.60 \leq$ “petal.length” $< 4.95 \wedge$ “petal.width” < 1.75	THEN	class = “versicolor”
IF	“petal.length” $\geq 2.60 \wedge$ “petal.width” ≥ 1.75	THEN	class = “virginica”
IF	“petal.length” ≥ 4.95	THEN	class = “virginica”

(a) Decision set



(b) BT model [5] consisting of 2 trees per class, each of depth ≤ 2, adopted from [17]

Figure 1 Example DS and BT models computed on the well-known *Iris* classification dataset.

value taken by feature $i \in \mathcal{F}$, together with its corresponding class $c \in \mathcal{K}$, represented by a pair $(\mathbf{v}, c)$, indicate a *data instance* or *example*. With a slight abuse of notation and whenever convenient, a data point $\mathbf{v} \in \mathbb{F}$ is also referred to as an instance. Finally, $\mathbf{x} = (x_1, \ldots, x_m)$ denotes a vector of feature variables $x_i \in D_i$, $i \in \mathcal{F}$, used for reasoning over points in $\mathbb{F}$.

A classifier defines a *classification function* τ: $\mathbb{F} \to \mathcal{K}$. The objective of classification problems is to learn a function τ to generalize well on unseen data given a training dataset $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$, where each instance $e_d \in \mathcal{E}$ is a pair of $(\mathbf{v}_d, c_d)$. Classification problems are conventionally posed as an optimization problem, i.e. either to minimize the complexity of τ, or maximize its accuracy, or both.

Rules, Decision Sets and Gradient Boosted Trees. Multiple ways exist to learn classifiers given data $\mathcal{E}$. This paper focuses on arguably one of the most interpretable models, i.e. decision sets, trained by *compiling* gradient boosted trees.

A *decision rule* is in the form of “IF antecedent THEN prediction”, where the antecedent is a set of feature literals. Informally, a rule is said to classify an instance $\mathbf{v} \in \mathbb{F}$ as class $c \in \mathcal{K}$ if its antecedent is *compatible* with $\mathbf{v}$ (or *matches* $\mathbf{v}$) and its prediction is c. A *decision set* (DS) is an unordered set of decision rules $\mathcal{R}$. An instance $(\mathbf{v}, c) \in \mathcal{E}$ is misclassified by a DS if either there exists no rule in $\mathcal{R}$ matching $\mathbf{v}$, or there exists a rule classifying $\mathbf{v}$ as a class $c' \in \mathcal{K}$ s.t. $c' \neq c$.

A *gradient boosted tree* (BT) is a tree ensemble $\mathfrak{T}$ defining sets of decision trees $T_c \in \mathfrak{T}$ for each class $c \in [|\mathcal{K}|]$, where T_c comprises $N \in \mathbb{N}_{>0}$ trees t_{kz+c}, $z \in \{0, \ldots, N-1\}$, $k = |\mathcal{K}|$. Given an instance $\mathbf{v} \in \mathbb{F}$, its class is obtained by computing the sum of scores assigned by trees for each class $w(\mathbf{v}, c) = \sum_{t \in T_c} t(\mathbf{v})$ and assigning the class which has the maximum score, i.e. $\operatorname{argmax}_{c \in [|\mathcal{K}|]} w(\mathbf{v}, c)$. Whenever convenient, $\mathfrak{n} \in t$ denotes a non-terminal node, where $t \in \mathfrak{T}$ represents an arbitrary decision tree. Moreover, each such $\mathfrak{n}$ indicates a feature condition in the form of $x_i < d$, where feature $i \in \mathcal{F}$ and *splitting threshold* $d \in \mathcal{D}_i$.

▶ **Example 1.** Figure 1 shows DS and BT models trained on the Iris dataset, which has 4 numeric features and 3 classes: *“setosa”*, *“versicolor”*, and *“virginica”*. Observe that

Table 1 Several instances extracted from *Iris* dataset.

#	sepal.length	sepal.width	petal.length	petal.width	class
e_1	5.1	3.5	1.4	0.2	setosa
e_2	7.7	2.6	6.9	2.3	virginica
e_3	5.6	2.5	3.9	1.1	versicolor
e_4	6.2	2.8	4.8	1.8	virginica
e_5	5.6	2.8	4.9	2.0	virginica

instance $\mathbf{v}_1 \in e_1$ shown in Table 1 is classified as *"setosa"* by the first rule of the DS. In the BT model, each class $c \in [3]$ is represented by 2 trees t_{3z+c}, $z \in \{0, 1\}$. Thus, it also classifies $\mathbf{v}_1$ as *"setosa"*, since the score of this class $w(\mathbf{v}_1, 1) = t_1 + t_4 = 0.71928$ is higher than the score of *"versicolor"* $w(\mathbf{v}_1, 2) = t_2 + t_5 = -0.40253$ and the score of *"virginica"* $w(\mathbf{v}_1, 3) = t_3 + t_6 = -0.41324$. ◻

Interpretability and Explanations. Interpretability is not formally defined as it is considered to be a subjective concept [33]. In this paper interpretability is defined as the overall succinctness of the information offered by an ML model to justify a provided prediction. Moreover, following earlier work [48, 20], we equate explanations for ML models with *abductive explanations* (AXps), which are subset-minimal sets of features sufficient to explain a given prediction. Concretely, given an instance $\mathbf{v} \in \mathbb{F}$ and a prediction $c = \tau(\mathbf{v}) \in \mathcal{K}$, an AXp is a subset-minimal set of features $\mathcal{X} \subseteq \mathcal{F}$ such that

$$\forall(\mathbf{x} \in \mathbb{F}).\ \Big[\bigwedge\nolimits_{i \in \mathcal{X}} (x_i = v_i)\Big] \rightarrow (\tau(\mathbf{x}) = c) \tag{1}$$

▶ **Example 2.** Consider the setup of Example 1. Given instance $\mathbf{v}_1$, observe that for any instance with *"petal.length"* $= 1.4$, the BT is guaranteed to predict *"setosa"* independently of the values of other features, since the weights for *"setosa"* and *"versicolor"* are 0.71928 and -0.40253 respectively as before, and the maximal weight for *"virginica"* is $0.39408 - 0.08968 = 0.30440$. Thus, the (only) AXp $\mathcal{X}$ for the prediction for e_1 made by the BT model is {*"petal.length"*}. ◻

Explanations in BTs. Formal reasoning has been recently applied to computing AXps for BT models, with the key difficulty being how to effectively reason about the aggregation over a large number of trees in a BT model. Recent work applied satisfiability modulo theory (SMT) [21] or mixed integer linear programming (MILP) solvers [42, 27] to directly address the linear summations arising in the BT encoding. Hereinafter, we build on the recent MaxSAT approach [17], which maps the aggregation reasoning to a set of MaxSAT queries to avoid a costly encoding of the linear constraints into CNF. Also, [17] demonstrates how a MaxSAT query can be made such that (1) holds if and only if the *optimal* value of the constructed objective function is negative.[2] In general, assuming that each feature $i \in \mathcal{F}$ is numeric (continuous), the approach orders the set of splitting thresholds $\{d_{i1}, ..., d_{ih_i}\}$ in a BT $\mathfrak{T}$ for each feature i, where h_i is the total number of thresholds of feature i in $\mathfrak{T}$ and $d_{ij} \in \mathcal{D}_i$ for $j \in [h_i]$. Given an instance $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$, the above approach associates each value v_i with a single interval I'_i from the set of disjoint intervals $\mathbb{D}_i = \{ I_{i1} \equiv [\min(\mathcal{D}_i), d_{i1}),\ I_{i2} \equiv [d_{i1}, d_{i2}),\ \ldots,\ I_{ih_i+1} \equiv [d_{ih_i}, \max(\mathcal{D}_i)] \}$. Thus, AXp extraction

[2] The reader is referred to [17] for the details.

boils down to finding a subset-minimal subset $\mathcal{X} \in \mathcal{F}$ s.t.

$$\forall(\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{i \in \mathcal{X}} x_i \in I'_i\right] \rightarrow (\tau(\mathbf{x}) = c) \tag{2}$$

▶ **Example 3.** Recall Example 2 and assume *"petal.length"* and *"petal.width"* have indices 3 and 4. Note that the sets of splitting thresholds for feature *"petal.length"* $\{d_{31} = 2.60, d_{32} = 4.75, d_{33} = 4.95\}$ and for feature "petal.width" $\{d_{41} = 1.45, d_{42} = 1.65, d_{43} = 1.75\}$. Let $min(\mathcal{D}_3) = -\infty$ and $min(\mathcal{D}_4) = 0.1$. Then we can associate the values of features 3 and 4 in our instance $\mathbf{v}_1 \in e_1$ with intervals $I_{31} \equiv (-\infty, 2.60)$ and $I_{41} \equiv [0.1, 1.45)$. Hence by (2), the AXp shown in Example 2 can in fact be seen as a rule $\langle$*IF "petal.length"* $<$ 2.60 *THEN class* $=$ *"setosa"*$\rangle$. ◻

3 Related Work

Interpretable decision sets are logic-based ML models that can be traced back to the 70s and 80s [39, 15, 4, 45]. To the best of our knowledge, [6] proposed the first approach to decision sets, which were introduced as the variant of decision lists [45, 7]. The first method making use of logic and optimization to synthesize a disjunction of rules that match a given dataset was proposed in [26]. Recent work [29] argued that decision sets are more interpretable than the other logic-based models, i.e. decision lists and decision trees. This work uses smooth local search to generate a set of rules first and heuristically minimizes a linear combination of criteria afterwards, e.g. the size of a rule, their maximum number, overlap or error.

Since then a number of works proposed the use of logic reasoning and optimization procedures to train DS models [22, 36, 12, 50, 18] claiming to significantly outperform the approach of [29] in terms of accuracy and performance. Among those, the works closest to ours are [22, 50, 18]. They proposed SAT-based approaches to computing smallest-size decision sets that *perfectly* agree with the training data by minimizing either the number of rules [22, 18] or the number of literals [50, 18] used in the model. Additionally, [50] is capable of computing *sparse* decisions sets that trade off training accuracy for model size. Despite the dramatic performance increase achieved in [18], all the approaches above suffer from scalability issues.

Post-hoc explainability is one of the major approaches to XAI. Besides a plethora of heuristic sampling-based methods to post-hoc explainability [43, 34, 44], a formal reasoning based approach to computing abductive explanations [48, 20] stands out. AXps can be related with prime implicants of the decision function (hence an alternative name *prime implicant explanations*, *PI-explanations*) associated with ML predictions and are guaranteed to capture the semantics of the ML models in the entire feature space. Although hard to compute in general, AXps were shown to be effectively computable for BT models by an incremental MaxSAT-based approach [17].

Our work aims at making a bridge between interpretable DS models and AXp computation by exploiting the latter for training the former. Given a BT model, it focuses on generating decision rules that agree with the BT. Each rule represents an AXp for the prediction made by the BT model, resulting in a DS model in a way *guided* by the original BT model. The approach is shown to outperform the prior logic-based approaches to DS inference in terms of test accuracy and performance. Note that despite prior attempts to train sparse models guided by tree ensembles [38], to our best knowledge, none of the existing works have applied formal post-hoc explanations to compile interpretable models.

Finally, our approach can be related to the existing line of work on *knowledge distillation* [11, 13], where an interpretable model is trained to approximate a hard-to-interpret

black-box model, which is often seen as teacher-to-student knowledge transfer. Note that in contrast to knowledge distillation, our approach is able to *compile* a BT into an *equivalent* DS if we consider the entire feature space, as shown below.

4 Decision Sets by Boosted Tree Compilation

Based on [17], this section details a MaxSAT-based approach to compiling a BT into a DS where each rule in the DS is equivalent to a prime implicant of the BT classification function.

4.1 Rule Extraction

Recall that an AXp, as defined in (1) and (2), can be seen as an *if-then* rule. Given a hard-to-interpret BT model, the AXp extraction approach of [17] can be modified to compute an interpretable DS consisting of a *set* of AXps for the BT. However, when the features are continuous (numeric), this potential approach suffers from the following issue. Recall that an AXp $\mathcal{X} \in \mathcal{F}$ indicates a set of *concrete* feature values that are sufficient to explain a prediction $c = \tau(\mathbf{v})$ for a certain instance $\mathbf{v} \in \mathbb{F}$. Although this same AXp can explain other instances compatible with it, its applicability in general is at the mercy of expressivity of the feature literals used in the AXp, i.e. equality literals and succinct interval membership in the case of (1) and (2), respectively. Motivated by this limitation, we propose to compute AXps over the literals intrinsic to the BT model aiming at getting feature intervals that are as general as possible, as detailed below.[3]

In contrast to the work of [17], which associates each feature value $v_i \in D_i$ with a single *narrowest* interval I'_i covering the value, we exploit all the splitting points used by the BT for feature i and identify all of the corresponding literals satisfied by the feature value v_i. Note that the original MaxSAT encoding [17] introduces a single Boolean variable o_{ij} for each literal $x_i < d_{ij}$ with d_{ij} being a j'th threshold used in the BT for feature i, s.t. $o_{ij} = 1$ iff $x_i < d_{ij}$ holds true. This way, each positive o_{ij} represents an upper bound on the value of x_i while each negative $\neg o_{ij}$ represents a lower bound on x_i.

▶ **Example 4.** Feature 3 (*"petal.length"*) from Example 3 has 3 thresholds: $d_{31} = 2.60$, $d_{32} = 4.75$, $d_{33} = 4.95$. Boolean variables o_{31}, o_{32}, and o_{33} are set to true iff $x_3 < 2.60$, $x_3 < 4.75$, and $x_3 < 4.95$, respectively. Let feature 3 take value 3.9 in the instance we want to explain. Observe how we can immediately assign literals $\neg o_{31}$, o_{32}, and o_{33} to true. □

Next, given an instance $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$, let us construct a complete conjunction $\bigwedge_{i \in \mathcal{F}, j \in [h_i]} \widetilde{o}_{ij}$ of literals $\widetilde{o}_{ij}$ s.t. $\widetilde{o}_{ij}$ is to be replaced by o_{ij} if $v_i < d_{ij}$ and replaced by $\neg o_{ij}$ otherwise. By construction, this conjunction holds true for instance $\mathbf{v}$. Now, given this conjunction of literals, we can apply the existing approach of [17] to extract a subset-minimal explanation $\mathcal{Y} \subseteq \{\widetilde{o}_{ij} \,|\, i \in \mathcal{F}, j \in [h_i]\}$ for instance $\mathbf{v}$ over literals $\widetilde{o}_{ij}$ s.t.

$$\forall (\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{l \in \mathcal{Y}} l\right] \rightarrow (\tau(\mathbf{x}) = c) \tag{3}$$

Such an explanation $\mathcal{Y}$ may (or may not) define either a lower bound on feature i, an upper bound, or both, aiming to construct the *most general* interval for each feature $i \in \mathcal{Y}$. Hence, we informally refer to such explanations as *generalized* AXps or simply *rules* (hereinafter, we use both interchangeably).

[3] An alternative to our approach is *inflation* of abductive explanations, which is discussed in [23, 24]. Given an AXp, it aims at extending the set of values covered by each feature literal in the AXp while the AXp condition (1) still holds.

Algorithm 1 Deletion-based Rule Extraction

Function: RuleExtract($\mathfrak{T}, \mathbf{v}, c, \mathcal{E}$)
Input: $\mathfrak{T}$: BT defining $\tau(\mathbf{x})$, $\mathbf{v}$: Instance, c: Prediction, i.e. $c = \tau(\mathbf{v})$ $\mathcal{E}$: Training data
Output: $\mathcal{Y}$: Subset-minimal rule

1: $\langle \mathcal{H}, \mathcal{S} \rangle \leftarrow \texttt{Encode}(\mathfrak{T})$
2: $\mathcal{Y} \leftarrow \texttt{Init}(\mathfrak{T}, \mathbf{v})$
3: $\mathcal{Y} \leftarrow \texttt{Sort}(\mathcal{Y}, \mathcal{E})$
4: **for** $l \in \mathcal{Y}$ **do**
5: **if** $\texttt{EntCheck}(\langle \mathcal{H}, \mathcal{S} \rangle, c, \mathcal{Y} \setminus \{l\})$ **then**
6: $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{l\}$
7: **return** $\mathcal{Y}$

▶ **Example 5.** Consider instance $\mathbf{v}_3$ predicted as *"versicolor"* by the BT (observe that $v_3 = 3.9$ and $v_4 = 1.1$) and recall the thresholds for features 3 and 4 discussed in Example 3. We can compute a generalized AXp $\mathcal{Y} = \{\neg o_{31}, o_{33}, o_{43}\}$ representing the second rule of the DS shown in Figure 1a. The original approach of [17] would instead compute an AXp defining the narrowest intervals for features 3 and 4, representing a rule: $\langle$*IF* $2.60 \leq$ *"petal.length"* $< 4.75 \wedge$ *"petal.width"* < 1.45 *THEN class* $=$ *"versicolor"*$\rangle$, which is far less general than $\mathcal{Y}$. ◻

A possible rule extraction procedure is outlined in Algorithm 1. (Please ignore line 3 for now; feature sorting is described in Section 4.2). The input BT model $\mathfrak{T}$ is encoded into MaxSAT by applying the approach of [17]. Given an instance $\mathbf{v} \in \mathbb{F}$, the initial set of literals $\mathcal{Y} = \{\widetilde{o}_{ij} \,|\, i \in \mathcal{F}, j \in [h_i]\}$ is created. Note that any feature $i \in \mathcal{F}$ unused in the BT $\mathfrak{T}$ is excluded from $\mathcal{Y}$. The rest of the procedure implements the standard deletion-based AXp extraction [20], i.e. it iterates through all literals in $\mathcal{Y}$ one by one, and checks which of the them can be safely removed such that entailment (3) still holds.

▶ **Example 6.** Consider our running example model and instance $\mathbf{v}_2 \in e_2$ from Table 1 predicted as *"virginica"* by the BT $\mathfrak{T}$. Given the thresholds for features 3 and 4 in Example 3, set $\mathcal{Y}$ is initialized to $\{\neg o_{31}, \neg o_{32}, \neg o_{33}, \neg o_{41}, \neg o_{42}, \neg o_{43}\}$. The other two features are excluded from $\mathcal{Y}$ since they are irrelevant to the classification function in $\mathfrak{T}$. Applying Algorithm 1 results in extracting a subset-minimal generalized AXp $\mathcal{Y} = \{\neg o_{33}\}$, which represents the rule $\langle$*IF petal.length* ≥ 4.95 *THEN class* $=$ *"virginica"*$\rangle$. ◻

▶ Remark 7. Algorithm 1 relies on deciding whether formula (3) holds for each feature in explanation $\mathcal{Y}$. Here, this is done by means of a series of incremental core-guided MaxSAT oracle calls [19, 17]. One may wonder whether or not incomplete *anytime* MaxSAT solving [31, 35, 2, 32] can be applied in this setting. Although this may look plausible at first glance, time-restricted anytime MaxSAT algorithms can only *over-approximate* exact MaxSAT solutions while (3) holds *if and only if* the exact value of the objective function is negative. Therefore, an over-approximation of a MaxSAT solution is *never able* to prove the validity of (3) and so none of the features being tested can be discarded in the case of incomplete MaxSAT algorithms, which defies the purpose of Algorithm 1.

4.2 Boosted Tree Compilation

As mentioned above, generalized AXps can be seen as general decision rules that can be applied to an enormous number of instances. Therefore, it makes little sense to extract such rules for each instance in the feature space $\mathbb{F}$. Instead, one can devise an on-demand

Algorithm 2 Compile a BT into a DS

Function: Compile($\mathfrak{T}, \tau, \mathcal{C}$)
Input: $\mathfrak{T}$: BT defining $\tau(\mathbf{x})$, τ: Classification function in $\mathfrak{T}$, $\mathcal{C}$: Coverage set
Output: $\mathcal{R}$: Set of Rules

$\mathcal{R} \leftarrow \emptyset$
$\mathcal{C}_u \leftarrow \mathcal{C}$
while $\mathcal{C}_u \neq \emptyset$ **do**
  $\mathbf{v} \leftarrow \texttt{GetInst}(\mathcal{E}_u)$
  $\mathcal{Y} \leftarrow \texttt{RuleExtract}(\mathfrak{T}, \mathbf{v}, c = \tau(\mathbf{v}), \mathcal{E}_u)$
  $\mathcal{C}_c \leftarrow \texttt{GetCover}(\mathcal{Y}, \mathcal{C}_u)$
  $\mathcal{C}_u \leftarrow \mathcal{C}_u \backslash \mathcal{C}_c$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{X}'$
return $\mathcal{R}$

compilation process, i.e. given a *yet uncovered* instance $\mathbf{v} \in \mathbb{F}$, we can apply Algorithm 1 to extract a rule covering $\mathbf{v}$ (and some other instances). Clearly, *exhaustive* compilation of a BT, i.e. if the target is to cover all the instances in $\mathbb{F}$ with generalized AXps of the BT, is computationally expensive given that AXp extraction for tree ensembles is hard for D^{P} [25]. This can also lead to the large size of the resulting DSes making them hard to interpret. In practice, *local* compilation aiming at capturing the behavior of the BT on the training data only, is sufficient to generate a DS, which is both accurate and interpretable.

The proposed approach to compiling a BT $\mathfrak{T}$ into a DS $\mathcal{R}$ is shown in Algorithm 2. We initialize the set $\mathcal{C}_u$ of currently uncovered instances to be equal to $\mathcal{C}$, i.e. the set of examples we wish to cover. The algorithm represents a loop generating rules until the set of computed rules $\mathcal{R}$ covers all instances in coverage set data $\mathcal{C}_u$, i.e. until there is no uncovered instances in $\mathcal{C}$. Each iteration of the algorithm selects an instance $\mathbf{v}$ from $\mathcal{C}_u$. Afterwards, a generalized AXp $\mathcal{Y}$ for the prediction $c = \tau(\mathbf{v})$ by the BT $\mathfrak{T}$ (recall that $\mathfrak{T}$ is meant to compute classification function $\tau(\mathbf{x})$) is extracted by invoking Algorithm 1. The iteration proceeds by updating the set of rules $\mathcal{R}$ and the set of uncovered instances $\mathcal{C}_u$. The algorithm terminates when all the instances in the coverage set $\mathcal{C}$ are covered and returns a compiled DS $\mathcal{R}$.

▶ **Proposition 8.** *Let $\mathfrak{T}$ be a BT and $\mathcal{R}$ be a DS returned by Algorithm 2 for $\mathfrak{T}$. Then $\mathcal{R} \equiv \mathfrak{T}$ with respect to $\mathcal{C}$.*

We consider two usages of the algorithm: for *exhaustive compilation* the coverage set $\mathcal{C} = \mathbb{F}$ is all possible feature combinations (in practice we model this coverage set implicitly, rather than in its explicit exponential sized form), and for *training set compilation* where $\mathcal{C} = \mathcal{E}$ is the training set. Based on the properties of prime implicants, Proposition 8 states that as a generalized AXp $\mathcal{Y} \in \mathcal{R}$ is a formal explanation for a prediction made by BT $\mathfrak{T}$, a compiled DS captures the semantics of the original model $\mathfrak{T}$ on *coverage set* $\mathcal{C}$, assuming everything else is a *don't care*. Furthermore, if the process is applied subject to coverage set $\mathcal{C} = \mathbb{F}$, i.e. when we target the entire feature space $\mathbb{F}$, then $\mathcal{R}$ and $\mathfrak{T}$ behave identically, i.e. they compute the same classification function $\tau(\mathbf{x})$.

▶ **Corollary 9.** *Let Algorithm 2 return a DS $\mathcal{R}$ for a BT $\mathfrak{T}$. Then there is no instance in feature space $\mathbb{F}$ covered by two distinct rules $\mathcal{Y}_1, \mathcal{Y}_2 \in \mathcal{R}$ predicting inconsistent classes $c_1 \neq c_2$.*

As each generalized AXp for $\mathfrak{T}$ represents a prime implicant of the decision function $\tau(\mathbf{x})$

computed over literals $\widetilde{o}_{ij}$, the above corollary claims that there are no overlapping rules in the result DS $\mathcal{R}$. This contrasts with other modern approaches to DS inference, where rule overlap is known to be a problem [29, 22]. Note that this approach still suffers from another common issue of DS models: namely, if DS $\mathcal{R}$ is computed for the training data $\mathcal{E}$, there may still be instances in $\mathbb{F}$ uncovered by $\mathcal{R}$.

▶ **Example 10.** Consider the running example BT model shown in Figure 1b. Its compiled DS representation computed by Algorithm 2 is shown in Figure 1a. Observe that there is no rule overlap in the DS computed. In fact, as the DS is computed by taking into account feature space $\mathbb{F}$, it computes the same classification function as the original BT model. □

Feature Sorting. Intuitively, how general and hence how applicable a rule is depends on how frequently the features used in it appear in the training data $\mathcal{E}$ labeled with the target class. Thus, a simple heuristic to apply when extracting a rule for prediction $c = \tau(\mathbf{v})$ is to sort the initial state of $\mathcal{Y} = \{\widetilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$ based on how frequently the corresponding literals $\widetilde{o}_{ij}$ apply in examples $\mathcal{E}$ labeled with c. This feature sorting represented by line 3 in Algorithm 1 in practice (according to our experiments) results in significantly more general rules and so overall smaller DSes.

Anytime Property. Most widely used reasoning-based algorithms to infer DSes provide a solution only if the computation is completed; otherwise, no decision set is reported. In contrast to these, the proposed approach is an *anytime* algorithm, i.e. it can return a *valid* DS $\mathcal{R}$ even though the compilation process is interrupted before all the coverage set instances $\mathcal{C}$ are covered. Furthermore, it can generate a more comprehensive DS $\mathcal{R}$, which covers more instances as it keeps going, i.e. after we have covered $\mathcal{C} \subseteq \mathbb{F}$ we can continue running the algorithm for the (unseen) instances of $\mathbb{F}$.

4.3 Post-Hoc Model Reduction

The compiled DS $\mathcal{R}$ can be large (in terms of either the number of rules or the total number of literals) since each generalized AXp $\mathcal{Y} \in \mathcal{R}$ may need a significant number of literals to explain a prediction made by BT $\mathfrak{T}$, or/and many rules are required to explain all instances of $\mathcal{C}$. Once the target DS is obtained, we can apply post-hoc heuristic methods for reducing its size and so making it more interpretable. The methods below are in a way inspired by the optimization problems studied in [18, 50]. Although these ideas are applicable to any DS inference method once the result model is devised, they do not look necessary for standard DS inference algorithms as they minimize the model while training. On the contrary, no minimization is applied in the rule enumeration process described above and so post-hoc model reduction plays a vital role in our approach to reduce the size of final DS models.

Reducing the Number of Rules. Given a set of rules $\mathcal{R}$, we can compute a minimum subset $\mathcal{R}^\star \subseteq \mathcal{R}$ that is still equivalent to the BT $\mathfrak{T}$ wrt. the coverage set $\mathcal{C}$ using discrete optimization, e.g. integer-linear programming (ILP). Concretely, the approach aims at selecting the smallest-size subset $\mathcal{R}^\star \subseteq \mathcal{R}$ that covers all instances in $\mathcal{C}$, where $\mathcal{R}$ is the compiled DS from $\mathfrak{T}$. Here, the size of $\mathcal{R}^\star$ is measured as the total number of literals used. This can be done by solving the following *set cover problem* [28]. Namely, for each rule $\mathcal{Y}_j \in \mathcal{R}$, we introduce a Boolean variable u_j such that $u_j = 1$ iff $\mathcal{Y}_j$ is included in $\mathcal{R}^\star$. Additionally, a Boolean variable y_{ij} is used to indicate that $\mathcal{Y}_j$ covers $e_i \in \mathcal{C}$. As a result,

the weighted set cover problem for minimizing the total number of literals used is as follows:

$$\text{minimize} \quad \sum_{j=1}^{|\mathcal{R}|} (|\mathcal{Y}_j| + 1) \cdot u_j \tag{4}$$

$$\text{subject to} \quad \forall_{i \in [n]} \sum_{j=1}^{|\mathcal{R}|} y_{ij} \cdot u_j \geq 1 \tag{5}$$

Reducing the Number of Literals. Additionally, one can minimize the total number of literals used in the rules of $\mathcal{R}$. Given a rule $\mathcal{Y} \in \mathcal{R}$, this can be done either lexicographically by maximizing rule accuracy followed by size minimization, or by optimizing both, or trading off misclassifications for rule size – in either case, a single MaxSAT call per rule to minimize can be made. The intution is that if a rule $\mathcal{Y}$ misclassifies k instances then its optimized version $\mathcal{Y}^\star \subseteq \mathcal{Y}$ should not result in many more misclassifications on training data $\mathcal{E}$. Recall that a rule misclassifies an instance $\mathbf{v}_k \in \mathcal{C}$ if it matches $\mathbf{v}_k$ but assigns it to a wrong class.

Inspired by [18], we introduce a Boolean variable p_k, which is true iff rule $\mathcal{Y}$ covers $\mathbf{v}_k$ — this holds if $\mathcal{Y}$ does not use any literals incompatible with $\mathbf{v}_k$. If $\mathcal{Y}_{\mathbf{v}_k} = \{\widetilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$ are all the literals compatible with $\mathbf{v}_k$ then this can be modeled with constraints

$$\forall_{k \in [|\mathcal{C}|]}.\ p_k \leftrightarrow \bigwedge_{l \in \mathcal{Y} \setminus \mathcal{Y}_{\mathbf{v}_k}} \neg l \tag{6}$$

Furthermore, let rule $\mathcal{Y}$ predict $c \in \mathcal{K}$ and let $\mathcal{C}_\ominus \subseteq \mathcal{C}$ contain all instances labeled with any other class. Thus, we can apply the objective below when minimizing rule $\mathcal{Y}$:

$$\sum_{l \in \mathcal{Y}} l + \sum_{k \in [|\mathcal{C}_\ominus|]} W \cdot p_k \tag{7}$$

If W is large enough, say $|\mathcal{C}| + 1$, this lexicographically minimizes misclassifications and then literals. If W is small, e.g. $1/\lambda \cdot |\mathcal{C}|$, this trades off $\lambda \cdot |\mathcal{C}|$ misclassifications for one literal.

5 Experimental Results

This section compares the proposed approach with the state-of-the-art DS learning algorithms on a variety of publicly available datasets in terms of accuracy, scalability, model and explanation size. The experiments are performed on an Intel Xeon 8260 CPU running Ubuntu 20.04.2 LTS, with the time limit of 3600s and the memory limit of 8GByte. Our experiments contain two parts, namely, exhaustive BT compilation and training-set BT compilation.

Prototype implementation. A prototype of the compilation-based approach to generating DSes was developed as a set of Python scripts using $\mathcal{C} = \mathcal{E}$, hereinafter referred to as *cpl*. The implementation of BT compilation exploits [17] and, therefore, makes use of the RC2 MaxSAT solver [19].[4] The BTs to be compiled are computed by XGBoost [5]; the number of trees per class in a BT model is 50 and the maximum depth of each tree is 3. Post-hoc literal reduction is done again with RC2 [19]. Let cpl_l denote the implementation applying

[4] Real weights in the objective function are not conventionally supported by MaxSAT solvers; the only other solver to support real weights besides RC2 is LMHS [47].

lexicographic optimization while $cpl_{l\lambda_1}$ trades off model accuracy for the number of literals used, with $\lambda_1 = 0.005$. Let cpl_r denote the implementation with post-hoc rule reduction applied using the Gurobi ILP solver [14]. The configuration with both post-hoc lexicographic optimization and rule reduction is denoted cpl_{lr}. Finally, the proposed approach applying exhaustive compilation $\mathcal{C} = \mathbb{F}$ is referred to as cpl_f.

Competition. Our approach is compared against: *twostg* a two-stage MaxSAT approach [18] for DSes perfectly accurate on the training data; *opt* another MaxSAT approach [50] for perfectly accurate DSes; sp_{λ_1} a sparse alternative to *opt* by the same authors (with $\lambda_1 = 0.005$) optimizing like $cpl_{l\lambda_1}$; $imli_1$ and $imli_{16}$ using MaxSAT-based IMLI [12] to minimize the number of literals given a predefined number of rules (we use 1 or 16); *ids* a state-of-the-art approach [29] based on smooth local search;[5] *ripper* a popular heuristic DS algorithm RIPPER [8]; and CN2 (referred to as *cn2*) another heuristic algorithm [7, 6].[6]

Datasets. For the evaluation, 59 publicly available datasets from UCI Machine Learning Repository [9] and Penn Machine Learning Benchmarks [41] are considered. We apply 5-fold cross validation, resulting in 295 pairs of training and test (unseen) data. For the sake of a fair comparison, the datasets used are preprocessed so that each original feature $i \in \mathcal{F}$ is replaced with a number of non-intersecting feature intervals $x_i < d_{ij}$ defined by the XGBoost model (see Section 2). This guarantees that all competitors tackle the same problem instances.

5.1 Exhaustive BT Compilation

The first experiment compares exhaustive compilation, where $\mathcal{C} = \mathbb{F}$ is the entire feature space. This is impractical except for 6 small benchmarks.

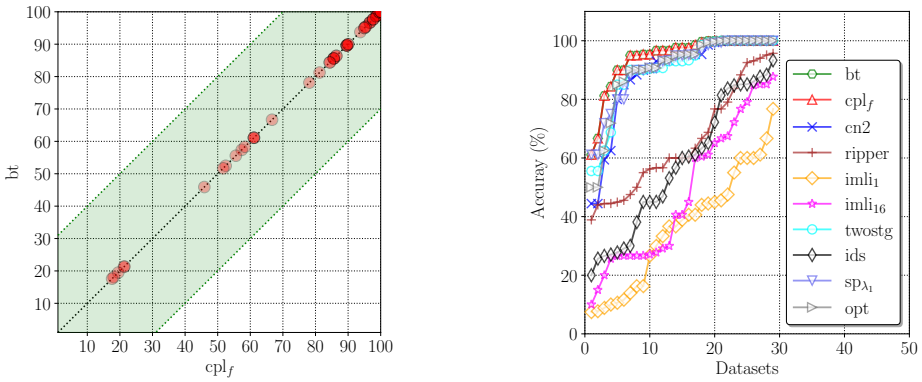
Results. Here we compare cpl_f with the competition in terms of accuracy, the total number of literals used and explanation size. We present the results as cactus plots showing the number of datasets that e.g. reach a certain accuracy, or finish in a certain runtime, for each method. These experimental results are shown in Figures 2 and 3 as well as the average results across folds are described in Table 2 where only the results of the datasets *completely* solved by compared competitors are presented. Note that cpl_f is nowhere near as scalable as the approaches described in the later experiments, but it is the *most accurate* approach to creating DSes we are aware of.

Test accuracy. An instance is considered misclassified if either there exists a rule of a wrong class that covers it, or it is not covered by any rule of the correct class. Thus, the test accuracy in this paper is calculated as $\frac{n-g}{n}$, where n is the total number of instances in the test data and g is the total number of misclassified instances. If an approach fails to train a model within the time limit, we assume its accuracy to be 0% for this dataset.

As can be seen in Figure 2b and Table 2, the best accuracy is achieved by BTs and cpl_f. In fact, these models share the same accuracy (this is also confirmed in Figure 2a), which

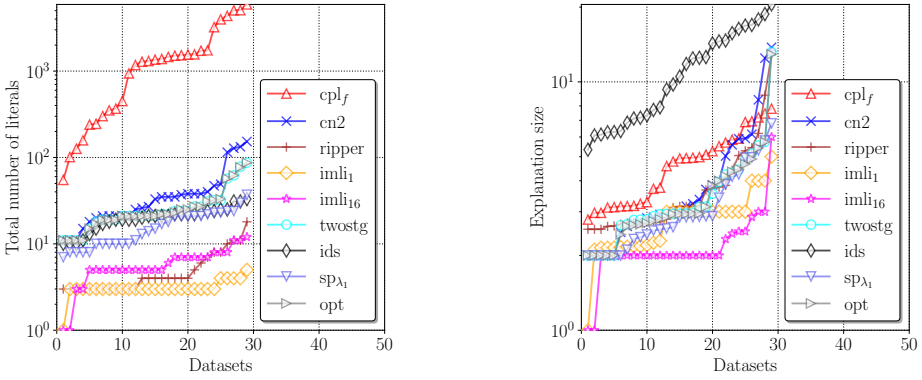
[5] Since the original implementation performs poorly [22], here we consider the new implementation of IDS [10], which is claimed to be orders of magnitude faster than the original implementation.

[6] Note that since RIPPER and IMLI compute a single class only given the training data, both of these competitors are augmented with a default rule predicting a class (1) different from the target class and (2) represented by the majority of training instances. Other algorithms, including our approach, incorporate a default rule that assigns a class based on the majority class in the training instances.



(a) Comparison with BT **(b)** Comparison with the others

Figure 2 Accuracy of exhaustive compilation. The *standard* interpretation of cactus plots is assumed, i.e. a plot sorts the datapoints for each method by the y-axis value, and then shows them in increasing order independently of other methods. Thus, the order of datasets/folds differs for different methods. Also, the order of datasets for the same method differs in different subplots.



(a) Number of literals used **(b)** Explanation size

Figure 3 Succinctness of exhaustive compilation.

should not come as a surprise given that cpl_f replicates the behavior of the BT in the entire feature space $\mathbb{F}$ (see Proposition 8).

Model Complexity. In general, complexity of a DS model can be measured by the total number of literals used in this DS. The total number of literals used in DS models is compared in Figure 3a and Table 2. Though the accuracy of DSes trained by cpl_f outperforms the other competitors, these models are significantly larger, which is no surprise given that cpl_f computes many more rules with no post-hoc reduction applied.

Explanation size. Explanation size is defined as the number of literals required to explain an instance.[7] This is arguably more important than the model size, since it defines "how hard" it is to understand an individual explanation. A small DS model tends to provide

[7] See [51] for details.

Table 2 Accuracy, number of literals used, and explanation size across folds.

Approach	Dataset					
	cardiotocography	**hayes-roth**	**iris**	**new-thyroid**	**orbit**	**zoo**
			Accuracy (%)			
bt	**100.0**	**84.38**	**96.0**	**96.74**	**99.66**	**96.0**
$\boldsymbol{cpl}_f$	**100.0**	**84.38**	**96.0**	**96.74**	**99.66**	**96.0**
sp_{λ_1}	100.0	73.44	94.0	91.63	99.43	89.05
opt	100.0	70.63	93.33	91.63	99.54	93.05
twostg	100.0	71.25	92.67	92.09	99.54	91.1
cn2	100.0	62.5	92.67	93.02	99.54	89.1
ripper	45.3	66.25	57.33	80.93	94.11	60.33
ids	27.23	43.75	58.67	76.28	85.29	40.62
$imli_{16}$	27.23	38.75	25.34	69.77	70.55	43.33
$imli_1$	45.3	39.37	32.67	26.98	8.93	60.33
			Number of literals used			
$\boldsymbol{cpl}_f$	3120.0	76.0	214.0	3614.2	729.8	1422
sp_{λ_1}	21.0	33.5	9.0	15.4	10.0	23.2
opt	21.0	63.6	19.4	23.0	11.8	30.0
twostg	21.0	64.2	19.8	22.6	11.8	29.8
cn2	21.0	116.2	27.2	36.6	13.2	40.8
ripper	**3.0**	12.8	5.0	8.2	4.0	**3.0**
ids	21.0	21.6	19.8	20.0	25.0	14.2
$imli_{16}$	5.0	**2.2**	7.4	7.4	6.4	5.0
$imli_1$	**3.0**	**2.2**	**3.0**	**4.2**	**3.0**	**3.0**
			Explanation size			
$\boldsymbol{cpl}_f$	7.26	3.76	3.02	4.9	3.18	5.4
sp_{λ_1}	**2.0**	6.31	2.45	4.13	2.86	3.64
opt	**2.0**	5.41	2.76	4.3	2.94	2.96
twostg	**2.0**	5.4	2.87	4.23	2.94	3.33
cn2	**2.0**	6.94	3.02	4.47	3.02	4.05
ripper	2.73	10.15	4.3	4.3	3.15	2.59
ids	16.08	18.23	13.06	7.74	6.23	9.28
$imli_{16}$	**2.0**	**2.2**	**2.1**	**1.97**	**2.8**	2.46
$imli_1$	2.18	**2.2**	3.0	4.0	3.0	**2.2**

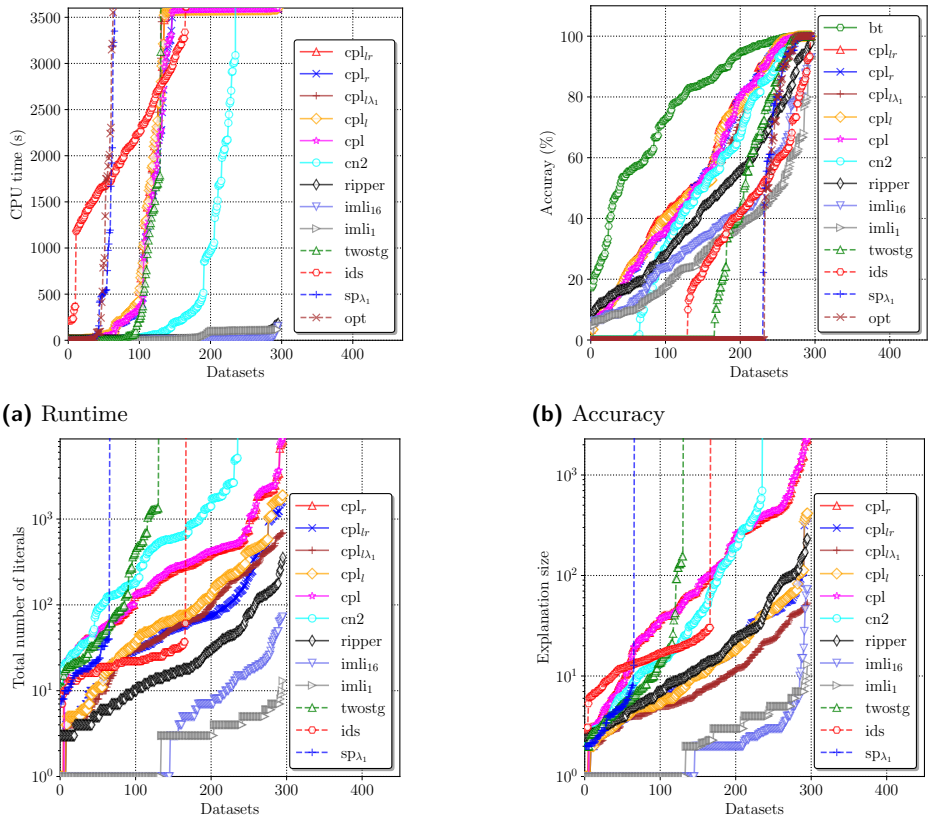
compact explanations but it is not always accurate. As can be seen in Figure 3b and Table 2 and similar to the total number of literals used in DSes, cpl_f requires more literals to explain an instance than all competitors except *ids*.

A crucial observation to make here is that we test explanation size for each of the test instances available. Although test data are meant to extrapolate the overall unseen data, such approximation of the unseen feature space is not ideal. As a result, there may be numerous instances in $\mathbb{F}$ *uncovered* by all the approaches but cpl_f, in which case it will be the *only* approach providing a user not only with a prediction but also with a succinct explanation of the prediction made.

5.2 BT Compilation Targeting Training Data

Compilation to cover the training set $\mathcal{C} = \mathcal{E}$ is much more efficient, and the main usage we expect of our algorithms.

Scalability. Figure 4a depicts scalability of all selected algorithms on the 295 considered datasets. Note that runtime of our approach includes BT training time. The best performance is demonstrated by the proposed implementation, i.e. *cpl* and cpl_*, $* \in \{l, r, lr, l\lambda_1\}$, where all selected datasets are solved within the time limit. This is not surprising since the approach is an anytime algorithm that can always return a valid DS. As for other competitors, the heuristic method *ripper* and the MaxSAT approaches $imli_1$ as well as $imli_{16}$ also solve all considered datasets. Next is the heuristic algorithm *cn2*, where 235 datasets are solved



(a) Runtime **(b)** Accuracy

(c) Number of literals used **(d)** Explanation size

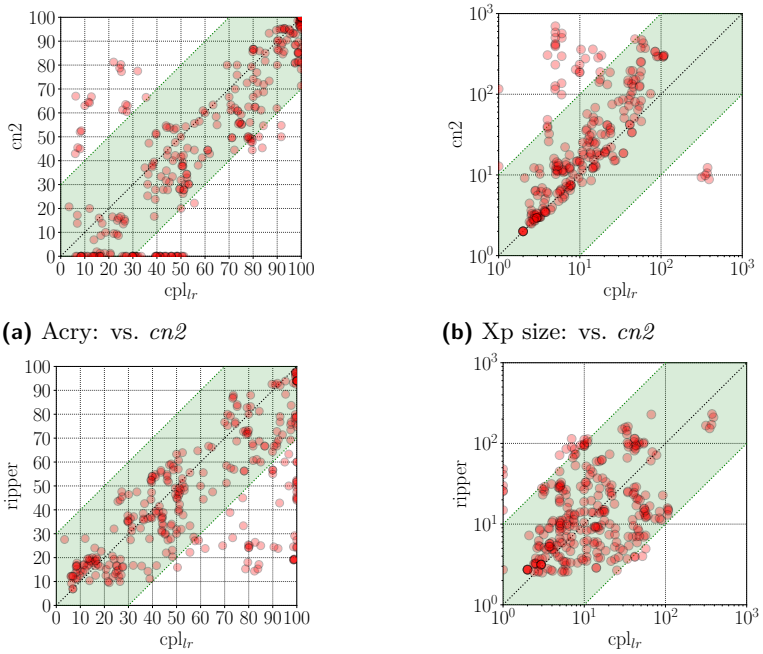
Figure 4 Summary of experimental results when the competitors aim at training a DS given training data $\mathcal{E}$ (i.e. $\mathcal{C} = \mathcal{E}$).

within the 3600s time limit. Followed by *ids*, which solves 166 considered datasets. The two-stage MaxSAT approach *twostg* successfully addresses 130 datasets, while the other MaxSAT algorithm for perfect decision sets *opt* and its sparse alternative sp_{λ_1} solve 65 and 63 datasets respectively.

Test Accuracy. The accuracy among the selected approaches is shown in Figure 4b. The average accuracy among all selected datasets for BTs is 77.34%, beating all DS approaches. The highest accuracy among DSes is achieved by all the configurations of the proposed approach, i.e. *cpl* and cpl_*, where the average accuracy ranges from 54.01% ($cpl_{l\lambda_1}$) to 57.49% (cpl_{lr}).[8] Unsurprisingly, the accuracy in $cpl_{l\lambda_1}$ is lower than the other configurations since $cpl_{l\lambda_1}$ trades off training accuracy on the number of literals in the computation process.

Next most accurate are the heuristic methods *cn2* (48.03%) followed by *ripper* (44.81%). The average accuracy of $imli_{16}$ and $imli_1$ is 35.47% and 29.7% respectively, while the average accuracy of *twostg* is 29.6% and *ids* is 26.78. Finally, the worst accuracy is demonstrated by sp_{λ_1} and *opt* (18.84% and 18.27% on average respectively) as these tools fail to provide

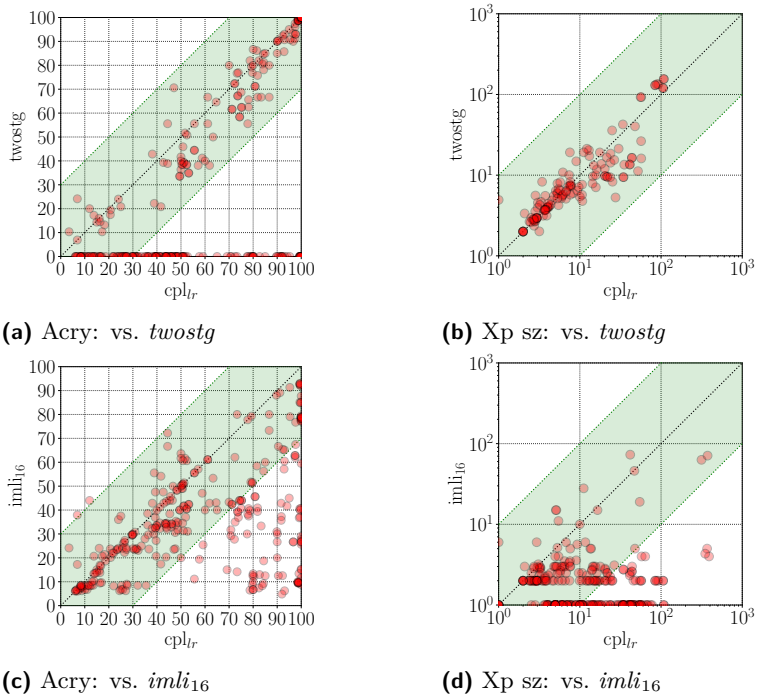
[8] Note that most datasets we used represent non-binary classification. Also, DSes are not to be compared with BTs. As Figure 4b shows (and as our work aims to demonstrate), our approach outperforms the state-of-the-art DS inference methods in terms of accuracy.



(a) Acry: vs. *cn2* **(b)** Xp size: vs. *cn2*

(c) Acry: vs. *ripper* **(d)** Xp sz: vs. *ripper*

Figure 5 Comparison of cpl_{lr} vs. *cn2* and *ripper* in terms of accuracy and explanation size.



(a) Acry: vs. *twostg* **(b)** Xp sz: vs. *twostg*

(c) Acry: vs. $imli_{16}$ **(d)** Xp sz: vs. $imli_{16}$

Figure 6 cpl_{lr} vs. $imli_{16}$ and *twostg* in terms of accuracy and explanation size.

prediction information for many datasets within the time limit. We will omit further discussion of *sp* and opt_{λ_1} since they solve so few datasets.

Model Complexity. Figure 4c illustrates the comparison among selected approaches regarding the total number of literals used in each DS solution. The average number of literals are in order: $imli_1$ (2.77), $imli_{16}$ (8.26), *ids* (21.14), *ripper* (38.47), $cpl_{l\lambda_1}$ (118.47), cpl_{lr} (157.53), cpl_l (213.27), *twostg* (265.98), cpl_r (584.39) *cpl* (620.82), *cn2* (700.49). Clearly, rule reduction and literal reduction can significantly reduce the size of the model without significantly affecting accuracy. Note how our approaches while significantly larger than the least accurate competitors, are significantly smaller than the most accurate competitor *cn2*.

Explanation Size. Figure 4d shows the explanation size for each competitor. The average explanation sizes are in order: $imli_1$ (2.61), $imli_{16}$ (3.00), $cpl_{l\lambda_1}$ (12.14), *ids* (15.28), *twostg* (17.5), cpl_{lr} (25.34), cpl_l (26.18), *ripper* (29.08), *cn2* (81.93), cpl_r (234.46), *cpl* (240.88). Figure 4d demonstrates that post-hoc literal reduction not only helps decrease the number of literals required to explain DS models, but also enables DSes to remain accurate, whereas rule reduction does not contribute to smaller explanations. With literal reduction applied our approaches are very competitive in terms of explanation size.

Detailed Comparison. While cactus plots allow us to compare many methods over a large suite of benchmarks, they do not allow direct comparison on individual benchmarks. We provide a detailed comparison of cpl_{lr} versus other decision set inference approaches in Figures 5 and 6, including *cn2*, *ripper*, *twostg*, and $imli_{16}$.[9] The scatter plots depicting explanation size are obtained for the datasets solvable by both competitors. Note that cpl_{lr} can generate more accurate DSes than the competitors. Also observe that the explanation size of DSes computed by cpl_{lr} is smaller than *cn2* and comparable with *twostg*. Although the explanation size of DSes in cpl_{lr} is larger than *ripper* and $imli_{16}$, the two approaches are less interpretable as they compute DSes representing only one class.

Summary. The experimental results were performed on various datasets, demonstrating that our approach computes DSes that outperform the state-of-the-art competitors in terms of accuracy and yield comparable explanation size to them.

6 Conclusions

This paper introduced a novel *anytime* approach to generating decision sets by means of on-demand extraction of generalized abductive explanations for boosted tree models. It can be used for exhaustive compilation of a BT model wrt. the entire feature space, or target a set of training instances. Augmented by a number of post-hoc model reduction techniques, the approach is shown to compute decision sets that are more accurate than decisions sets computed by the state-of-the-art algorithms and comparable with them in terms of explanation size.

As the proposed approach targets generating a decision set by compiling a BT, a natural line of future work is to extend the proposed approach to compile BTs into the other interpretable models, i.e. decision trees and decision lists, making use of AXp extraction for BTs. Additionally, another future work is to apply AXp extraction to compile other accurate black box models, e.g. neural networks, into decision sets.

[9] The average results across the folds are given in the appendix.

References

1. ACM. Fathers of the deep learning revolution receive ACM A.M. Turing award. http://tiny.cc/9plzpz, 2018.
2. Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *CPAIOR*, pages 39–56, 2019.
3. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2021.
4. Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
5. Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.
6. Peter Clark and Robin Boswell. Rule induction with CN2: some recent improvements. In *EWSL*, pages 151–163, 1991.
7. Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
8. William W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.
9. Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. http://archive.ics.uci.edu/ml.
10. Jiri Filip and Tomas Kliegr. Pyids-python implementation of interpretable decision sets algorithm by lakkaraju et al, 2016. In *RuleML+ RR*, 2019.
11. Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In *CEx@AI*IA*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
12. Bishwamittra Ghosh and Kuldeep S. Meel. IMLI: an incremental framework for MaxSAT-based learning of interpretable classification rules. In *AIES*, pages 203–210. ACM, 2019.
13. Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021.
14. Gurobi Optimization. Gurobi optimizer reference manual, 2022. http://www.gurobi.com/.
15. Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976.
16. Alexey Ignatiev. Towards trustable explainable AI. In *IJCAI*, pages 5154–5158, 2020.
17. Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *AAAI*, pages 3776–3785, 2022.
18. Alexey Ignatiev, Edward Lam, Peter J. Stuckey, and Joao Marques-Silva. A scalable two stage approach to computing optimal decision sets. In *AAAI*, pages 3806–3814, 2021.
19. Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an efficient MaxSAT solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.
20. Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *AAAI*, pages 1511–1519, 2019.
21. Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On validating, repairing and refining heuristic ML explanations. *CoRR*, abs/1907.02509, 2019. URL: http://arxiv.org/abs/1907.02509, arXiv:1907.02509.
22. Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva. A SAT-based approach to learn explainable decision sets. In *IJCAR*, pages 627–645, 2018.
23. Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On tackling explanation redundancy in decision trees. *J. Artif. Intell. Res.*, 75:261–321, 2022.
24. Yacine Izza, Alexey Ignatiev, Peter J. Stuckey, and Joao Marques-Silva. Delivering inflated explanations. *CoRR*, abs/2306.15272, 2023.
25. Yacine Izza and Joao Marques-Silva. On explaining random forests with SAT. In *IJCAI*, July 2021.
26. Anil P. Kamath, Narendra Karmarkar, K. G. Ramakrishnan, and Mauricio G. C. Resende. A continuous approach to inductive inference. *Math. Program.*, 57:215–238, 1992.

27 Kentaro Kanamori, Takuya Takagi, Ken Kobayashi, Yuichi Ike, Kento Uemura, and Hiroki Arimura. Ordered counterfactual explanation by mixed-integer linear optimization. In *AAAI*, pages 11564–11574. AAAI Press, 2021.

28 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

29 Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *KDD*, pages 1675–1684, 2016.

30 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

31 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In *IJCAI*, pages 1346–1352, 2018.

32 Zhendong Lei and Shaowei Cai. Nudist: An efficient local search algorithm for (weighted) partial maxsat. *Comput. J.*, 63(9):1321–1337, 2020.

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

34 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, pages 4765–4774, 2017.

35 Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artif. Intell.*, 243:26–44, 2017.

36 Dmitry Malioutov and Kuldeep S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In *CP*, pages 312–327, 2018.

37 Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pages 12342–12350, 2022.

38 Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo I. Seltzer. Fast sparse decision tree optimization via reference ensembles. In *AAAI*, pages 9604–9613, 2022.

39 Ryszard S Michalski. On the quasi-minimal solution of the general covering problem. In *International Symposium on Information Processing*, pages 125–128, 1969.

40 Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and João Marques-Silva. Learning optimal decision trees with SAT. In *IJCAI*, pages 1362–1368, 2018.

41 Randal S. Olson, William G. La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Min.*, 10(1):36:1–36:13, 2017.

42 Axel Parmentier and Thibaut Vidal. Optimal counterfactual explanations in tree ensembles. In *ICML*, volume 139 of *PMLR*, pages 8422–8431, 2021.

43 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.

44 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, pages 1527–1535, 2018.

45 Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, 1987.

46 Cynthia Rudin and Seyda Ertekin. Learning customized and optimized lists of rules with mathematical programming. *Mathematical Programming Computation*, 10:659–702, 2018.

47 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *SAT*, pages 539–546, 2016.

48 Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *IJCAI*, pages 5103–5111, 2018.

49 Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling LIME and SHAP: adversarial attacks on post hoc explanation methods. In *AIES*, pages 180–186, 2020.

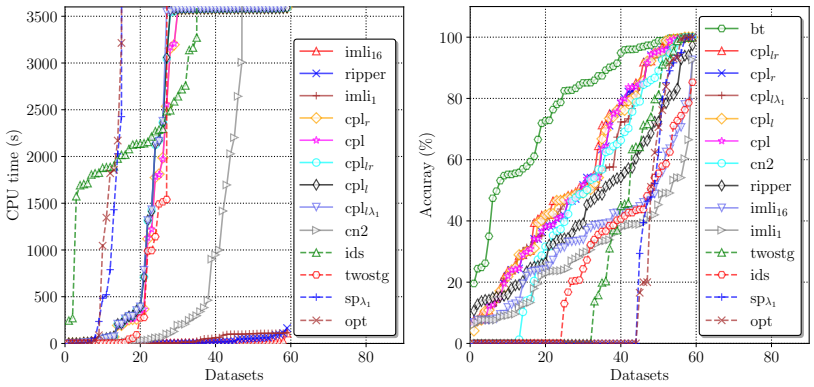
50 Jinqiang Yu, Alexey Ignatiev, Peter J. Stuckey, and Pierre Le Bodic. Computing optimal decision sets with SAT. In *CP*, pages 952–970, 2020.

51 Jinqiang Yu, Alexey Ignatiev, Peter J Stuckey, and Pierre Le Bodic. Learning optimal decision sets and lists with sat. *JAIR*, 72:1251–1279, 2021.

553 **52** Jinqiang Yu, Alexey Ignatiev, Peter J. Stuckey, Nina Narodytska, and Joao Marques-Silva.
554 Eliminating the impossible, whatever remains must be true: On extracting and applying
555 background knowledge in the context of formal explanations. In *AAAI*, 2023.

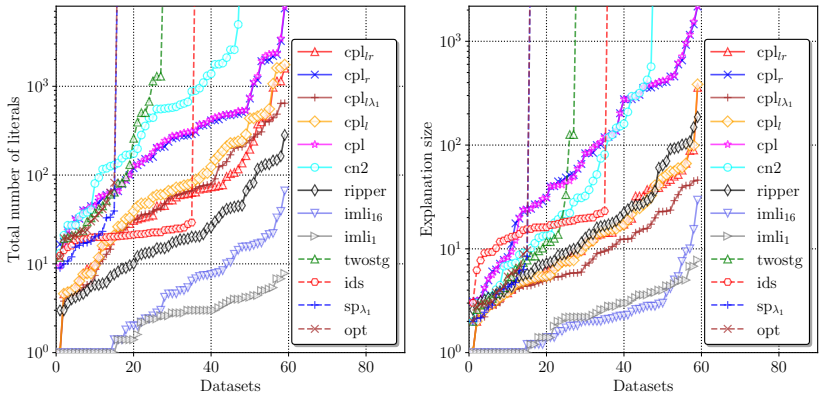
Appendices

A Summaries of Results Across Folds



(a) Runtime **(b)** Accuracy

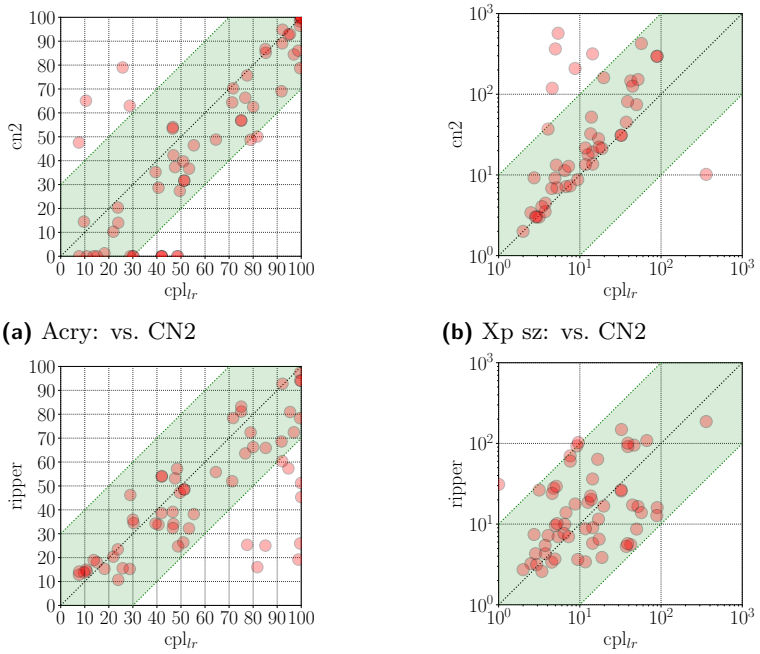
Figure 7 Experimental results of runtime and accuracy across folds.



(a) Number of literals used **(b)** Explanation Size

Figure 8 Experimental results of model complexity and explanation size across folds.

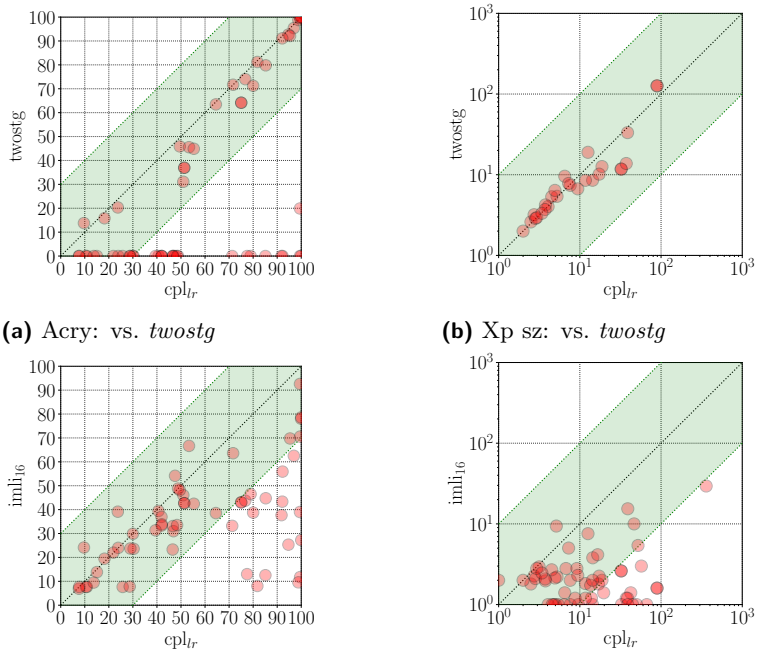
Figures 7 and 8 illustrate the average experimental results across folds regarding scalability, accuracy, model complexity, and explanation size. Since 5-fold cross validation is used, these results for each dataset are obtained from the average of 5 pairs of training and test data. Here, observations similar to those described in Section 5 can be made, i.e. the best scalability and accuracy among selected DS competitors are both demonstrated by *cpl* and cpl_*, $* \in \{l, r, lr, l\lambda_1\}$, while $imli_1$ and $imli_{16}$ show the smallest model complexity and explanation size.



(a) Acry: vs. CN2 **(b)** Xp sz: vs. CN2

(c) Acry: vs. RIPPER **(d)** Xp sz: vs. RIPPER

Figure 9 cpl_{lr} vs. CN2 and RIPPER across folds in terms of accuracy and explanation size.



(a) Acry: vs. *twostg* **(b)** Xp sz: vs. *twostg*

(c) Acry: vs. $imli_{16}$ **(d)** Xp sz: vs. $imli_{16}$

Figure 10 cpl_{lr} vs. $imli_{16}$ and *twostg* Across Folds in terms of accuracy and explanation size.

B Detailed Comparisons Across Folds

In this appendix, we provide a detailed comparison of cpl_{lr} versus other decision set inference approaches across folds.

Figure 9 and Figure 10 detail the comparisons of cpl_{lr} with CN2, RIPPER, $imli_{16}$ and *twostg* in terms of average accuracy and explanation size across folds. As can be seen in Figure 9a, the accuracy of DSes generated by cpl_{lr} is higher than the accuracy of CN2, where the average accuracy is 57.49% and 48.03%, respectively. Additionally, Figure 9b demonstrate that the explanation size of DSes produced by CN2 (81.93 on average) can be two orders of magnitude larger than the explanation size of cpl_{lr} (25.88 on average).

Figure 9c illustrate that the average accuracy in RIPPER is 44.81%, which is 12.68% lower than the accuracy in cpl_{lr}. Although Figure 9d depict that RIPPER is comparable with cpl_{lr} regarding explanation size (29.08 and 25.34 on average respectively), RIPPER is less interpretable as it computes DSes representing only one class.

As can be observed in Figure 10a, the accuracy of *twostg* (29.67% on average) is 27.82% lower than the accuracy in cpl_{lr} while Figure 10b illustrate that the explanation size is comparable between the two approaches. Finally, Figure 10c demonstrate that the accuracy of $imli_{16}$ is 22.02% lower than the accuracy of cpl_{lr} on average. However, as can be seen in Figure 10d, the explanation size of $imli_{16}$ is smaller than the explanation size of cpl_{lr} but $imli_{16}$ generates DSes targeting only a single class, which significantly diminishes the interpretability of computed DSes.