Using MaxSAT for Efficient Explanations of Tree Ensembles

Alexey Ignatiev[1], **Yacine Izza**[2], **Peter J. Stuckey**[1], **Joao Marques-Silva**[3]

[1] Monash University, Melbourne, Australia
[2] University of Toulouse, France
[3] IRIT, CNRS, Toulouse, France
alexey.ignatiev@monash.edu, yacine.izza@univ-toulouse.fr, peter.stuckey@monash.edu, joao.marques-silva@irit.fr

Abstract

Tree ensembles (TEs) denote a prevalent machine learning model that do not offer guarantees of interpretability, that represent a challenge from the perspective of explainable artificial intelligence. Besides model agnostic approaches, recent work proposed to explain TEs with formally-defined explanations, which are computed with oracles for propositional satisfiability (SAT) and satisfiability modulo theories. The computation of explanations for TEs involves linear constraints to express the prediction. In practice, this deteriorates scalability of the underlying reasoners. Motivated by the inherent propositional nature of TEs, this paper proposes to circumvent the need for linear constraints and instead employ an optimization engine for pure propositional logic to efficiently handle the prediction. Concretely, the paper proposes to use a MaxSAT solver and exploit the objective function to determine a winning class. This is achieved by devising a propositional encoding for computing explanations of TEs. Furthermore, the paper proposes additional heuristics to improve the underlying MaxSAT solving procedure. Experimental results obtained on a wide range of publicly available datasets demonstrate that the proposed MaxSAT-based approach is either on par or outperforms the existing reasoning-based explainers, thus representing a robust and efficient alternative for computing formal explanations for TEs.

1 Introduction

Tree ensembles (Zhou 2012), including random forests (Breiman 2001) and gradient boosted trees (Friedman 2001), are one of the most successful machine learning (ML) approaches. Unfortunately, although often succinct in representation, their decisions are not necessarily easy to explain, since they involve aggregating information over many trees.

Besides applying intrinsically interpretable ML models (Rudin 2018; Molnar 2020), there are two major approaches to explaining ML models on demand (Miller 2019; Guidotti et al. 2019): *model-agnostic* approaches do not make any use of the ML model representation that they are explaining, relying on querying the ML model (Chen and Guestrin 2016; Lundberg and Lee 2017; Ribeiro, Singh, and Guestrin 2018); and *model-precise* (or *rigorous*, or *formal*) explanations, which make use of the structure of the ML model and formally capture the model's semantics (Shih, Choi, and Darwiche 2018; Ignatiev, Narodytska, and Marques-Silva 2019a; Barceló et al. 2020; Marques-Silva and Ignatiev 2022). Despite a number of advantages, model-agnostic approaches are by definition inexact with respect to the actual ML model since they only examine a small proportion of the possible outputs (Ignatiev, Narodytska, and Marques-Silva 2019c; Narodytska et al. 2019; Ignatiev 2020; Shrotri et al. 2022). Rigorous explanations, in contrast, can make accurate statements about the ML model being explained, although their operation relies on formal reasoning and often suffers from scalability issues (Shih, Choi, and Darwiche 2018; Ignatiev, Narodytska, and Marques-Silva 2019a).

In this paper, we give explanations of tree ensembles by applying a formal approach, which returns provably minimal explanations for Boosted Tree classifications. This is achieved by encoding the tree ensemble in propositional logic. One of the key difficulties in reasoning about tree ensembles is reasoning about the aggregation over the many trees in the tree ensemble. Previous approaches to tackling the large linear constraints that arise in reasoning over the aggregation of the many trees in the ensemble either used SAT Modulo Theory (SMT) solvers (Ignatiev, Narodytska, and Marques-Silva 2019c) or Mixed Integer Programming (MIP) solvers (Chen et al. 2019; Kanamori et al. 2021; Parmentier and Vidal 2021) to directly handle large linear constraints, or encoded the linear constraints to SAT (Izza and Marques-Silva 2021), which can be costly.

In this paper, we use the optimisation capabilities of modern core-guided MaxSAT solvers (Bacchus, Järvisalo, and Martins 2021) to avoid encoding the linear constraint, instead mapping the aggregation reasoning to a suite of MaxSAT queries. This new approach is much more efficient than competing approaches since the underlying (OLL[1]) MaxSAT optimisation procedure introduces new literals that allow us to learn about the large linear constraints effectively (Morgado, Dodaro, and Marques-Silva 2014; Ignatiev, Morgado, and Marques-Silva 2019).

[1] The OLL algorithm originates from the work on solving optimization problems in ASP (Andres et al. 2012) and was then adapted to MaxSAT (Morgado, Dodaro, and Marques-Silva 2014; Ignatiev, Morgado, and Marques-Silva 2019).

2 Preliminaries

SAT and MaxSAT. We assume standard definitions for propositional satisfiability (SAT) and maximum satisfiability (MaxSAT) solving (Biere et al. 2021). A propositional formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *clause* is a disjunction of literals. A *literal* is either a Boolean variable or its negation. Whenever convenient, clauses are treated as sets of literals. A truth assignment maps each variable to $\{0, 1\}$. Given a truth assignment, a clause is satisfied if at least one of its literals is assigned value 1; otherwise, it is falsified. A formula is satisfied if all of its clauses are satisfied; otherwise, it is falsified. If there exists no assignment that satisfies a CNF formula, then the formula is *unsatisfiable*.

In the context of unsatisfiable formulas, the maximum satisfiability (MaxSAT) problem is to find a truth assignment that maximizes the number of satisfied clauses. A number of variants of MaxSAT exist (Biere et al. 2021, Chapters 23 and 24). Hereinafter, we will be mostly interested in Partial Weighted MaxSAT, which can be formulated as follows. The formula ϕ is represented as a conjunction of *hard* clauses $\mathcal{H}$, which must be satisfied, and *soft* clauses $\mathcal{S}$, each with a weight, which represent a preference to satisfy those clauses, i.e. $\phi = \mathcal{H} \wedge \mathcal{S}$. Whenever convenient, a soft clause c with weight w will be denoted by (c, w). The Partial Weighted MaxSAT problem consists in finding an assignment that satisfies all the hard clauses and maximizes the total weight of satisfied soft clauses.

Classification problems. We consider a classification problem, characterized by a set of features $\mathcal{F} = \{1, \ldots, m\}$, and by a set of classes $\mathcal{K} = \{c_1, \ldots, c_K\}$. Each feature $j \in \mathcal{F}$ is characterized by a domain D_j. As a result, feature space is defined as $\mathbb{F} = D_1 \times D_2 \times \ldots \times D_m$. A specific point in feature space represented by $\mathbf{v} = (v_1, \ldots, v_m)$ denotes an *instance* (or an *example*). Also, we use $\mathbf{x} = (x_1, \ldots, x_m)$ to denote an arbitrary point in feature space. In general, when referring to the value of a feature $j \in \mathcal{F}$, we will use a variable x_j, with x_j taking values from D_j. A classifier implements a *total classification function* $\tau : \mathbb{F} \to \mathcal{K}$.

Tree ensembles. Decision trees (Hyafil and Rivest 1976; Breiman et al. 1984; Quinlan 1986, 1993) are one of the oldest forms of ML model, and are very widely used, but they are known to suffer from bias if the tree is small, and tend to overfit the data if the tree is large. They are commonly deemed to have a significant advantage over most ML models in that their decisions appear easy to explain to a human, although Izza, Ignatiev, and Marques-Silva (2020) show that this does not hold in general.

In order to overcome the weaknesses of decision trees, *tree ensembles* were introduced, which generate many decisions trees and rely on aggregating their decisions to come to an overall decision. Two popular ensemble methods are *random forests* (RFs) (Breiman 2001) and *gradient boosted trees* (BTs) (Friedman 2001). Here, we focus on BT models trained with the use of the XGBoost algorithm (Chen and Guestrin 2016). In general, BTs are a tree ensemble that relies on building a sequence of small decision trees. In each stage, new trees are built to compensate for the inaccuracies of the already constructed tree ensemble. Consider multi-class classification problems, i.e. $K > 2$, and let a BT be an ensemble $\mathfrak{E}$ of decision trees. Hereinafter, we say that ensemble $\mathfrak{E}$ computes a classification function $\tau(\mathbf{x})$. Each class $i \in [K]$ in $\mathfrak{E}$ is represented by $n \in \mathbb{N}_{>0}$ trees T_{Kj+i}, $j \in \{0, \ldots, n-1\}$. Therefore, each decision tree is attached to a particular class i and contributes to the weight class i. The overall decision of a BT is made by evaluating the total weight assigned by the trees for each class and selecting the class with largest total weight. Let $w_i : \mathbb{F} \to \mathbb{R}$, $i \in [K]$, denote the total weight of class i computed by ensemble $\mathfrak{E}$ for instance $\mathbf{x} \in \mathbb{F}$. Similarly, each individual tree T_{Kj+i} can also be seen as computing a function $\mathcal{T}_{Kj+i} : \mathbb{F} \to \mathbb{R}$ s.t. the total weight of a class i is computed as

$$w_i(\mathbf{x}) = \sum\nolimits_{j \in \{0, \ldots, n-1\}} \mathcal{T}_{Kj+i}(\mathbf{x}), \ \forall_{i \in [K]} \tag{1}$$

When the context does not explicitly refer to a specific class, we use $T_i \in \mathfrak{E}$ to denote an *arbitrary* decision tree. Also, whenever convenient, we will use notation $\mathcal{P}_i$ to denote the set of all paths in tree T_i, and $P_r \in \mathcal{P}_i$ to denote an individual path, which can be associated with the corresponding leaf (or terminal) node $\mathfrak{t}_r \in T_i$. Similarly, a non-terminal node of a tree is denoted by $\mathfrak{n} \in T_i$, and each such $\mathfrak{n}$ represents a condition on the value of some feature $j \in \mathcal{F}$ in the form $x_j < d$ s.t. $d \in D_j$, where d is the *splitting threshold*.

Example 1 *An example of a BT trained by XGBoost for the Iris dataset is shown in Figure 1. The dataset includes 4 numeric features:* sepal.length, sepal.width, petal.length, petal.width, *and 3 classes:* class_1 = "setosa", class_2 = "versicolor", *and* class_3 = "virginica". *Each class $i \in [3]$ is represented in the BT by 2 trees T_{3j+i}, $j \in \{0, 1\}$. This way, given an input instance* (sepal.length = 5.1) $\wedge$ (sepal.width = 3.5) $\wedge$ (petal.length = 1.4) $\wedge$ (petal.width = 0.2)*, the scores of classes 1–3 are $w_1 = \mathcal{T}_1 + \mathcal{T}_4 = 0.72284$, $w_2 = \mathcal{T}_2 + \mathcal{T}_5 = -0.40355$, and $w_3 = \mathcal{T}_3 + \mathcal{T}_6 = -0.41645$. Hence, the model predicts class 1 ("setosa") as it has the highest score. The path taken by this instance in tree T_2 is defined as* {sepal.width $\geq$ 2.95, petal.length $<$ 3}. □

Interpretability & explanations. Interpretability is generally accepted to be a subjective concept, without a formal definition (Lipton 2018). In this paper we measure interpretability in terms of the overall succinctness of the information provided by an ML model to justify a given prediction. Moreover, and building on earlier work, we equate explanations with the so-called *abductive explanations* (AXps) (Shih, Choi, and Darwiche 2018; Ignatiev, Narodytska, and Marques-Silva 2019a,b; Darwiche and Hirth 2020; Izza, Ignatiev, and Marques-Silva 2020; Ignatiev et al. 2020; Barceló et al. 2020; Marques-Silva et al. 2020, 2021; Huang et al. 2021; Marques-Silva and Ignatiev 2022; Huang et al. 2022), i.e. subset-minimal sets of feature-value pairs that are sufficient for the prediction. More formally, given an instance $\mathbf{v} \in \mathbb{F}$, with prediction $c \in \mathcal{K}$, i.e. $\tau(\mathbf{v}) = c$, an AXp is a minimal subset $\mathcal{X} \subseteq \mathcal{F}$ such that,

$$\forall (\mathbf{x} \in \mathbb{F}). \bigwedge\nolimits_{j \in \mathcal{X}} (x_j = v_j) \to (\tau(\mathbf{x}) = c) \tag{2}$$

Another name for an AXp is a *prime implicant* (PI) explanation (Shih, Choi, and Darwiche 2018; Darwiche and Hirth

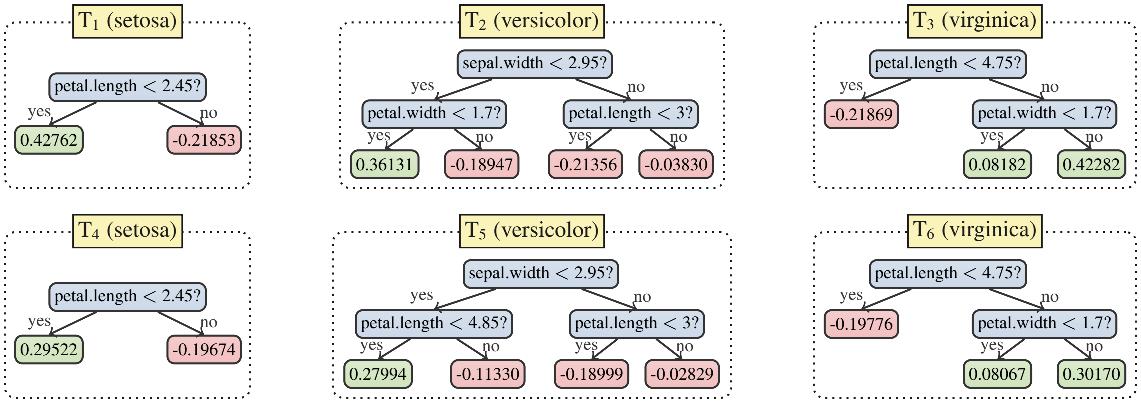


Figure 1: Example of a simple boosted tree model (Chen and Guestrin 2016) generated by XGBoost on the well-known Iris classification dataset. Here, the tree ensemble has 2 trees for each of the 3 classes with the depth of each tree being at most 2.

2020). For simplicity, we will use abductive explanation and the acronym *AXp* interchangeably.

Example 2 *Consider the BT from Example 1 and the same data instance. By examining the model, one can observe that it suffices to specify* petal.length $= 1.4$ *to guarantee that the prediction is "setosa". The weight for "setosa" will be 0.72284 as before, and the maximal weight for "versicolor" will be 0.36131 + 0.27994 = 0.64125, and the weight for "virginica" will be -0.41645 as before. Therefore, an abductive explanation for this prediction includes a single feature. Furthermore, it is the only AXp for the given instance.* □

3 MaxSAT As Entailment Oracle

3.1 Propositional Encoding of a Tree

The propositional encoding described here builds on the original SMT encoding for XGBoost models (Ignatiev, Narodytska, and Marques-Silva 2019c). However, as propositional logic cannot offer the expressive power of SMT and can deal with Boolean variables only, one has to handle the domain and the possible values of a (continuous) numeric feature $x_j \in D_j$ utilized by the classifier, with the use of some kind of additional encoding, an example of which is also detailed below.

Concretely, the encoding of a non-terminal node of a tree $T_i \in \mathfrak{E}$ is organized as follows. As each non-terminal node $\mathfrak{n} \in T_i$ checks the condition $x_j < d$, $d \in D_j$ for a feature $j \in \mathcal{F}$, we can examine the complete TE $\mathfrak{E}$ to identify all the splitting thresholds $s_{j,k} \in D_j$ used for feature j in all the trees of $\mathfrak{E}$ (in sorted order); then consider m_j+1, $m_j \in \mathbb{N}_{>1}$, disjoint intervals $I_1 \equiv [\min(D_j), s_{j,1})$, $I_2 \equiv [s_{j,1}, s_{j,2})$, …, $I_{m_j+1} \equiv [s_{j,m_j}, \max(D_j)]$. Next, we can "name" these m_j+1 intervals with the use of auxiliary Boolean variables $l_{j,k}$, $j \in \mathcal{F}$ and $k \in [m_j+1]$ s.t. $l_{j,k} = 1$ iff $x_j \in I_k$. These variables can be used to represent *concrete* feature values of a data instance $\mathbf{v} \in \mathbb{F}$, i.e. with a slight abuse of notation we can use $l[\![x_j = v_j]\!] \equiv l_{j,k}$ iff $v'_j \in I_k$. Furthermore, let us use additional Boolean variables $o_{j,k}$ for each of the threshold values $s_{j,k}$, $k \in [m_j]$, for feature j, i.e. variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$. Then the domain D_j can be expressed using the following constraints

$$o_{j,k} \rightarrow o_{j,k+1}, \ \forall_{k \in [m_j-1]} \tag{3}$$

$$l_{j,1} \leftrightarrow o_{j,1} \tag{4}$$

$$l_{j,k+1} \leftrightarrow (\neg o_{j,k} \wedge o_{j,k+1}), \ \forall_{k \in [m_j-1]} \tag{5}$$

$$l_{j,m_j+1} \leftrightarrow \neg o_{j,m_j} \tag{6}$$

$$\bigvee\nolimits_{k \in [m_j+1]} l_{j,k} \tag{7}$$

Note that in the case of one threshold value d for a feature $j \in \mathcal{F}$, *two* distinct intervals exist and it suffices to consider a single Boolean variable o_j to represent it. Also note that the encoding above can be related to the order encoding of integer domains studied in the context of mapping CSP to SAT (Ansótegui and Manyà 2004). Finally, to enforce the "yes" and "no" branches of the node $x_j < s_{j,k}$, one can use literals $o_{j,k}$ or $\neg o_{j,k}$, respectively.

Example 3 *In the BT model shown in Figure 1, there are 4 thresholds for feature* petal.length*, namely, values 2.45, 3, 4.75, and 4.85. Assuming* petal.length *is a 3rd feature, let us introduce variables $o_{3,k}$ for $k \in [4]$, and $l_{3,k}$ for $k \in [5]$. As an example, $o_{3,4} \leftrightarrow (x_3 < 4.85)$ and $l_{3,5} \leftrightarrow \neg o_{3,4} \leftrightarrow (x_j \geq 4.85)$. The following constraints can be used to express the domain of the possible values for feature 3:*

$$\begin{gathered} o_{3,1} \rightarrow o_{3,2}, \ o_{3,2} \rightarrow o_{3,3}, \ o_{3,3} \rightarrow o_{3,4} \\ l_{3,2} \leftrightarrow (\neg o_{1,1} \wedge o_{3,2}), \ l_{3,3} \leftrightarrow (\neg o_{2,2} \wedge o_{3,3}) \\ l_{3,4} \leftrightarrow (\neg o_{3,3} \wedge o_{3,4}), \ l_{3,1} \leftrightarrow o_{3,1}, \ l_{3,5} \leftrightarrow \neg o_{3,4} \\ (l_{3,1} \vee l_{3,2} \vee l_{3,3} \vee l_{3,4} \vee l_{3,5}) \end{gathered}$$

□

A leaf node $\mathfrak{t}_r \in T_i$ can be encoded by a Boolean variable t_r s.t. $t_r = 1$ iff node $\mathfrak{t}_r$ is reached. Each leaf node $\mathfrak{t}_r \in T_i$ corresponds to a path $P_r \in \mathcal{P}_i$. Thus, we can encode each path $P_r \in \mathcal{P}_i$ as a conjunction of literals over variables $o_{j,k}$ representing the conditions $x_j < d$ on the features j appearing in P_r, and relate it with the corresponding leaf node $\mathfrak{t}_r$ using the following constraint:

$$\left(\bigwedge\nolimits_{(x_j < s_{j,k}) \in P_r} o_{j,k} \wedge \bigwedge\nolimits_{(x_j \geq s_{j,k}) \in P_r} \neg o_{j,k} \right) \leftrightarrow t_r \tag{8}$$

Note that if multiple trees in an ensemble $\mathfrak{E}$ share a path P_r (this often happens for tree ensembles trained with XGBoost), a single variable t_r can be used to represent all of its occurrences.

Together with the above constraints, we must ensure that exactly one path in each tree T_i is executed at a time. This can be enforced by using constraint $\sum_{P_r\in\mathcal{P}_i} t_r = 1$.

Example 4 *As shown in Example 3, feature 3* (petal.length) *has 5 intervals and so 4 associated variables $o_{3,k}$, $k\in[4]$, e.g. $o_{3,1} = 1$ iff $x_3 < 2.45$. As tree T_1 has 2 paths and so 2 leaf nodes, they can be represented using the constraints:*

$$o_{3,1} \leftrightarrow t_1,\ \ \neg o_{3,1} \leftrightarrow t_2,\ \ t_1 + t_2 = 1$$

where the corresponding leaf nodes $\mathfrak{t}_1$ and $\mathfrak{t}_2$ have weights $w_1 = 0.42762$ and $w_2 = -0.21853$, respectively. Also, observe that variables t_1 and t_2 can be reused to encode the leaves of tree T_4. □

Now, let us denote the CNF formula encoding the paths and the leaf nodes of a tree T_i in ensemble $\mathfrak{E}$, by $\mathcal{H}_{T_i}$, following (8). Also, let us denote the CNF formula encoding the domain of each feature $j\in\mathcal{F}$ including constraints (3)–(7) by $\mathcal{H}_{D_j}$. Finally, in the following, we will use

$$\mathcal{H} = \bigwedge\nolimits_{T_i\in\mathfrak{E}} \mathcal{H}_{T_i} \wedge \bigwedge\nolimits_{j\in\mathcal{F}} \mathcal{H}_{D_j} \qquad (9)$$

to represent the hard part of the MaxSAT encoding of TE $\mathfrak{E}$.

3.2 Dealing with Classifier Predictions

In order to formally reason about TEs, one has to additionally encode the process of determining the prediction of the model obtained for an input data instance according to (1). For this, when using the language of SMT (Ignatiev, Narodytska, and Marques-Silva 2019c), it suffices to introduce a real variable for each tree T_{Kj+i} as well as for the total weight w_i of class i, and relate them with a linear constraint (1). As a result, entailment queries (2) boil down to deciding whether a misclassification is possible for the classifier $\tau(\mathbf{x})$ given a candidate explanation $\mathcal{X}$, i.e. whether there is a data instance $\mathbf{v}'$ that complies with $\mathcal{X}$ such that $w_\iota(\mathbf{v}') \geq w_i(\mathbf{v}')$ for some class $\iota \neq i$.

In contrast to SMT, in the case of propositional logic one has to represent the linear constraints (1) integrating *all* individual leaf-node variables t_r multiplied by the corresponding weight, i.e. summands $w_r\cdot t_r$, to determine the total weights of the classes. Namely, the total weight for a class i can be computed as $\sum_i = \sum_{T_{Kj+i}\in\mathfrak{E}} \sum_{\mathfrak{t}_r\in T_{Kj+i}} w_r\cdot t_r$. (This is valid as we use $\sum_{P_r\in\mathcal{P}_i} t_r = 1$ to enforce exactly one path to be taken in each tree.) This involves using either cardinality (if all the weights are 1) or pseudo-Boolean (if the weights are arbitrary real numbers) encodings, which may be quite expensive in practice. What is worse, checking if a given candidate explanation $\mathcal{X}$ for $\tau(\mathbf{v}) = c_i$ entails the prediction requires to efficiently reason about pairwise inequalities comparing the total weight of i'th class, with the total weights of all the other classes $\iota\neq i$, each computed using the above cardinality or PB constraints, i.e. $\sum_\iota \geq \sum_i$.

Example 5 *For our running example, consider class 1 ("setosa"), represented by trees T_1 and T_4. Assume that the leaf nodes of T_1 are encoded as t_1 with weight 0.42762 and t_2 with weight -0.21853. Observe that variables t_1 and t_2 can be reused to encode the leaves of T_4 with weights 0.29522 and -0.19674, respectively. Hence, the total weight of class "setosa" can be expressed as $\sum_1 = 0.42762\cdot t_1 - 0.21853\cdot t_2 + 0.29522\cdot t_1 - 0.19674\cdot t_2 = 0.72284\cdot t_1 - 0.41527\cdot t_2$. By applying similar reasoning, we can assume that the total weight of class 3 ("virginica") can be computed as $\sum_3 = -0.41645\cdot t_3 + 0.16249\cdot t_4 + 0.72452\cdot t_5$.* □

We observe that in order to avoid the bottleneck of dealing with hard clauses $\mathcal{H}$ together with inequalities $\sum_\iota \geq \sum_i$ one can consider optimizing objective function $\sum_\iota - \sum_i$ subject to hard clauses $\mathcal{H}$ defined in (9).

Proposition 1 *The optimal value of the objective function above is non-negative iff there is a data instance (which can be extracted from the corresponding optimal solution) that is classified by the model as class ι instead of class i.* □

Also, note that if there are $K > 2$ classes, one has to deal with $K-1$ optimization problems. In general, in the context of computing an AXp for TE models, the following holds:

Corollary 1 *Given a data instance $\mathbf{v}\in\mathbb{F}$ s.t. $\tau(\mathbf{v}) = c_i$, a subset of features $\mathcal{X}\subseteq\mathcal{F}$ is an AXp for $\mathbf{v}$ iff the optimal values of the objective functions of the optimization problems for all classes $c_\iota\in\mathcal{K}$, $c_\iota\neq c_i$:*

$$\text{maximize} \qquad \mathcal{S}_{i,\iota} = \sum\nolimits_\iota - \sum\nolimits_i \qquad (10)$$

$$\text{subject to} \qquad \mathcal{H}\wedge\bigwedge\nolimits_{j\in\mathcal{X}} l[\![x_j = v_j]\!] \qquad (11)$$

are negative. □

Example 6 *The objective function to replace inequality $\sum_3 \geq \sum_1$ from Example 5 is $\sum_3 - \sum_1 = -0.41645\cdot t_3 + 0.16249\cdot t_4 + 0.72452\cdot t_5 - 0.72284\cdot t_1 + 0.41527\cdot t_2$. In MaxSAT, it can be written as a set of weighted soft clauses:*

$$\mathcal{S}_{1,3} = \left\{ \begin{array}{l} (\neg t_1, 0.72284), (t_2, 0.41527), \\ (\neg t_3, 0.41645), (t_4, 0.16249), (t_5, 0.72452) \end{array} \right\}$$

Remark 1 *To represent a negative summand $-w_r\cdot t_r$ of $\sum_\iota - \sum_i$ as a Boolean literal, one has to use a constraint $-w_r\cdot t_r = w_r\cdot\neg t_r - w_r$. Observe that constants $-w_r$ should be taken into account when determining the sign of the optimal value of objective function $\sum_\iota - \sum_i$.*

3.3 Computing AXps with MaxSAT

As soon as the formulas (10)–(11) are constructed for all classes $c_\iota\neq c_i$, a MaxSAT solver can be applied to check if feature subset $\mathcal{X}$ is an AXp for $\tau(\mathbf{v}) = c_i$. This entailment check is detailed in Algorithm 1. Concretely, given a candidate explanation $\mathcal{X}$ for $\tau(\mathbf{v}) = c_i$ made by a TE $\mathfrak{E}$ encoded into the set $\mathcal{H}$ of hard clauses and a number of sets of weighted soft clauses $\mathcal{S}$, the procedure iterates over all the relevant objective functions (see line 1), each represented by a set $\mathcal{S}_{i,\iota}$ of soft clauses, and checks if another class c_ι can get a higher total score given the literals enforcing $\bigwedge_{j\in\mathcal{X}}(x_j = v_j)$. This is done by making a single MaxSAT

```
Algorithm 1: Entailment check with MaxSAT
Function EntCheck ($\langle\mathcal{H}, \mathcal{S}\rangle$, $\mathbf{v}$, $c_i$, $\mathcal{X}$)
    Input: $\mathcal{H}$: Hard clauses,   $\mathcal{S}$: Objective functions,
           $\mathbf{v}$: Input instance,  $c_i$: Prediction, i.e. $\tau(\mathbf{v}) = c_i$,
           $\mathcal{X}$: Candidate explanation
    Output: true or false
1   foreach $\mathcal{S}_{i,\iota} \in \mathcal{S}$:        # all relevant objective functions for $c_i$
2       $\mu \leftarrow \text{MaxSAT}(\mathcal{H} \wedge \bigwedge_{j\in\mathcal{X}} l[\![x_j = v_j]\!], \mathcal{S}_{i,\iota})$
3       if ObjValue($\mu$) $\geq 0$:                  # non-negative objective?
4           return false                        # misclassification reached
5   return true                           # $\mathcal{X}$ indeed entails prediction $c_i$
```

```
Algorithm 2: Standard deletion-based AXp extraction
Function ExtractAXp ($\mathfrak{E}$, $\mathbf{v}$, $c_i$)
    Input: $\mathfrak{E}$: TE computing $\tau(\mathbf{x})$,  $\mathbf{v}$: Input instance,
           $c_i$: Prediction, i.e. $\tau(\mathbf{v}) = c_i$
    Output: $\mathcal{X}$: abductive explanation
1   $\langle\mathcal{H}, \mathcal{S}\rangle \leftarrow$ Encode($\mathfrak{E}$)  # MaxSAT encoding s.t. $\mathcal{S} \triangleq \cup\mathcal{S}_{i,\iota}$
2   $\mathcal{X} \leftarrow \mathcal{F}$                              # $\mathcal{X}$ is over-approximation
3   foreach $j \in \mathcal{X}$:
4       if EntCheck($\langle\mathcal{H}, \mathcal{S}\rangle, \mathbf{v}, c_i, \mathcal{X} \setminus \{j\}$): # $j$ unneeded?
5           $\mathcal{X} \leftarrow \mathcal{X} \setminus \{j\}$                  # If so, drop it
6   return $\mathcal{X}$                                    # $\mathcal{X}$ is AXp
```

```
Algorithm 3: Incremental core-guided MaxSAT solver
Function MaxSAT ($\phi$, $\mathcal{A}$)
    Input: $\phi$: Partial CNF formula (hard and soft clauses),
           $\mathcal{A}$: Set of assumption literals
    Output: $\mu$: MaxSAT model
1   cost $\leftarrow 0$                                   # initially, cost is 0
2   $\mathcal{C} \leftarrow$ ValidCores($\phi, \mathcal{A}$)     # get valid unsatisfiable cores
3   foreach $\kappa \in \mathcal{C}$:                   # iterate over known cores $\kappa$
4       cost $\leftarrow$ cost + CoreWt($\kappa$)     # add its weight to cost
5       $\phi \leftarrow$ Process($\phi, \kappa$)          # process $\kappa$ and update $\phi$
6   while SAT($\phi$, $\mathcal{A}$) = false: # iterate until $\phi$ gets satisfiable
7       $\kappa \leftarrow$ GetCore($\phi$)              # new unsatisfiable core
8       cost $\leftarrow$ cost + CoreWt($\kappa$)     # add its weight to cost
9       $\phi \leftarrow$ Process($\phi, \kappa$)          # process $\kappa$ and update $\phi$
10      Record($\phi, \mathcal{A}, \kappa$)              # record $\kappa$ for the future
11  return GetModel($\phi$)                       # $\phi$ is now satisfiable
```

call. If such a misclassification is possible, i.e. when the optimal value of the objective function is non-negative, $\mathcal{X}$ does not entail prediction c_i. Otherwise, it does. The overall explanation procedure is depicted in Algorithm 2 and follows the standard deletion-based AXp extraction (Ignatiev, Narodytska, and Marques-Silva 2019a). It receives a TE model $\mathfrak{E}$ computing function $\tau(\mathbf{x})$, a concrete data instance $\mathbf{v}$ and the corresponding prediction $c_i = \tau(\mathbf{v})$. First, Algorithm 2 encodes the classifier (see line 1), as described in Section 3.1. Then it iterates through the features $j \in \mathcal{X}$ (initially set to $\mathcal{F}$) and checks if $\mathcal{X} \setminus \{j\}$ entails the prediction c_i by invoking Algorithm 1. If it does, feature j is irrelevant and is removed. Otherwise, it must be kept in $\mathcal{X}$. Algorithm 2 proceeds until all features are tested, and reports the updated $\mathcal{X}$.

An immediate observation to make here is that instead of a simple deletion-based algorithm for AXp extraction, one could apply the ideas behind the QuickXplain and Progression algorithms (Junker 2004; Marques-Silva, Janota, and Belov 2013), which in some cases may help reducing the number of MaxSAT oracle calls.

Incremental MaxSAT with assumptions. Although any *off-the-shelf* MaxSAT solver (Morgado et al. 2013; Ansótegui, Bonet, and Levy 2013; Davies and Bacchus 2011) can be exploited in Algorithm 1 as is (assuming that the chosen MaxSAT solver can cope with Partial Weighted CNF formulas with real weights), computing a single AXp may involve quite a large number of oracle calls, and invoking a new MaxSAT solver from scratch at each iteration of Algorithm 2 may be computationally expensive. An alternative is to create all the necessary MaxSAT oracles once, at the initialization stage of the explanation approach, and reuse them *incrementally* as needed with varying sets of assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$ — in a way similar to the well-known MiniSat-like incremental assumption-based interface used in SAT solvers (Eén and Sörensson 2003).

A possible setup of a generic incremental *core-guided* MaxSAT solver is shown in Algorithm 3 where the flow of a (non-incremental) core-guided solver is *augmented* with several highlighted parts, which briefly overview the changes needed for incrementality. (Please ignore lines 2–5 and 10 for now.) In general, a core-guided solver operates as a loop iterating as long as the formula it is dealing with is unsatisfiable (line 6).[2] (At the beginning, the solver aims at satisfying all the hard and soft clauses of the input formula ϕ.) Each iteration of the loop extracts an unsatisfiable core κ (line 7) and processes it by relaxing the clauses in κ and adding new constraints allowing some of the clauses in κ to be falsified. The *weight* of κ contributes to the total cost of the MaxSAT solution. As soon as the formula gets satisfiable, the solver stops and reports its satisfying assignment.

Given the above, incrementality in a core-guided solver with the use of the set $\mathcal{A}$ of assumptions can be made to work by handing $\mathcal{A}$ over to the underlying SAT solver at every iteration of the MaxSAT algorithm (see the highlighted modification made in line 6). This way all the clauses learnt by the SAT solver during the entire MaxSAT oracle call can be reused later in the following MaxSAT calls.

Moreover, as each call to Algorithm 1 "equates" to an entailment query (2), where a reasoner aims at refuting a given input formula, one may want to extract a subset of assumptions in $\mathcal{A}$ that are deemed responsible for the entailment (2) to hold. This can be done by aggregating all the subsets $\mathcal{A}' \subseteq \mathcal{A}$ appearing in individual unsatisfiable cores κ extracted by the underlying SAT solver (see line 7), i.e. $\mathcal{A}' = \kappa \cap \mathcal{A}$, such that the union of alls such subsets $\mathcal{A}'$,

[2] The reader is referred to (Morgado et al. 2013; Ansótegui, Bonet, and Levy 2013) for details on core-guided MaxSAT.

i.e. $\mathcal{A}^* = \cup\mathcal{A}'$, is guaranteed to be responsible for the entailment (2) to hold. Notice that this is similar to the *core extraction mechanism* of SAT solvers. It can be used in practice to avoid iterating over all features in $\mathcal{F}$ (see line 3 of Algorithm 2) and instead focus on those features that "matter".

Reusing unsatisfiable cores. To further improve the MaxSAT oracle, we can *reuse unsatisfiable cores* detected in the previous MaxSAT calls, similar to core caching studied in the context of *minimal correction subset* (MCS) enumeration (Previti et al. 2017). Indeed, Algorithm 2 provides all the MaxSAT solvers dealing with objective $\mathcal{S}_{i,\iota}$ with exactly the same formula $\phi \equiv \mathcal{H} \wedge \mathcal{S}_{i,\iota}$ but a varying set of assumptions $\mathcal{A}$. Hence, some of the cores extracted from ϕ for assumptions $\mathcal{A}_1$ may be reused with assumptions $\mathcal{A}_2$, s.t. $\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset$, thus saving a potentially large number of SAT oracle calls in Algorithm 3.

Although implementations of core reuse may vary, a high-level view on the corresponding changes in the solver are shown in lines 2–5 and 10 of Algorithm 3.[3] Namely, each core κ is *recorded* for future use (line 10). Next time Algorithm 3 is called, it starts by examining the set of recorded cores and determining, which of them may be reused in the current call (see line 2). All of such cores are processed the standard way (lines 3–5), each saving a single SAT call. Afterwards, the MaxSAT solver proceeds as normal.

Let us recall that each core κ_i found at iteration i of a MaxSAT algorithm comprises a subset $\mathcal{S}'$ of soft clauses $\mathcal{S}$ that are unsatisfiable together with the hard clauses $\mathcal{H} \subseteq \phi$. In our incremental MaxSAT solver, κ_i may also include a subset $\mathcal{A}'$ of the assumption literals $\mathcal{A}$. Additionally, in the case of the algorithms based on *soft cardinality constraints* (Morgado, Dodaro, and Marques-Silva 2014), κ_i may include literals "representing" some other (previously found) unsatisfiable cores κ_j, $j < i$. This way, each core κ_i can be seen as *depending* on (1) the corresponding subset $\mathcal{S}'$ of soft clauses, (2) assumptions $\mathcal{A}'$, and (3) cores κ_j, $j < i$. As a result, thanks to the fact that each of these are represented by Boolean literals,[4] it is this dependency of κ_i that can be recorded for future use. Namely, let $\mathcal{D}_i$ be the set of all such literals that κ_i depends on. Then we can use a *separate* SAT solver to store the dependency in the form of a clause $\left(\bigwedge_{l \in \mathcal{D}_i} l\right) \rightarrow \zeta_i$, where ζ_i is a Boolean variable introduced to represent κ_i. This way, detecting which cores can be reused for formula ϕ under assumption literals $\mathcal{A}$, i.e. a call to $\textsc{ValidCores}(\phi, \mathcal{A})$, can be done by applying unit propagation in the external SAT solver given all the soft clauses $\mathcal{S} \subseteq \phi$ and assumptions $\mathcal{A}$. Note that there is no need to make a full-blown SAT call here as literals ζ_i for all reusable cores κ_i will be propagated in the right order, i.e. ζ_j will precede ζ_i for $j < i$, given $\mathcal{S}$ and $\mathcal{A}$.

Distance-based stratification. In practice, weighted formulas are challenging for core-guided MaxSAT algorithms as they often suffer from making too many iterations 6–10 due to the effect of *clause weight splitting*. As a result, a number of heuristics were proposed to address this problem, namely *Boolean lexicographic optimization* (BLO) (Marques-Silva et al. 2011) and *diversity-based stratification* (Ansótegui et al. 2013). Both techniques aim at splitting the set of soft clauses into multiple levels, based on their weight, and solving a series of MaxSAT problems, each time adding into consideration the clauses of the next (smaller weighed) level. Although powerful in general, both techniques often fail in the setting of TE explainability. [5]

As a result, we developed an alternative heuristic to stratify soft clauses referred to as *distance-based stratification*. The idea is to (1) construct a stratum $\mathcal{L} \subseteq \mathcal{S}$ by traversing the weights from highest to lowest and (2) associate weight w with the current stratum $\mathcal{L}$ if it is closer to the mean of clause weights already in $\mathcal{L}$ than to the mean of clause weights smaller than w, i.e. when

$$\left| w - \frac{\sum_{(c_i,w_i)\in\mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j,w_j)\in\mathcal{S}\wedge w_j<w} w_j}{|\{(c_j,w_j)\in\mathcal{S}\wedge w_j<w\}|} \right|$$

Example 7 *Consider the set $\mathcal{S}_{1,3}$ of soft clauses from Example 6. Diversity-based heuristic (Ansótegui et al. 2013) fails to stratify them since all of them have unique weights. Distance-based stratification makes 3 strata: $\mathcal{L}_1 = \{(t_5, 0.72452), (\neg t_1, 0.72284)\}$, $\mathcal{L}_2 = \{(\neg t_3, 0.41645), (t_2, 0.41527)\}$, and $\mathcal{L}_3 = \{(t_4, 0.16249)\}$.* □

Early termination. Although core-guided MaxSAT solvers are quite effective in practice, the proposed approach may invoke a MaxSAT engine a significant number of times. As solving optimization problems to optimality is often expensive, it makes approximate solutions to these problems a viable and efficient alternative. Motivated by these observations, each MaxSAT call in the proposed approach can be terminated *before* an exact optimal solution is computed.

Consider again Algorithm 3 and recall that the objective function in each MaxSAT call is of the form $\sum_\iota - \sum_i$. This means that as soon as the *cost* of the solution is updated in line 4 or 8 and its new value exceeds the maximum value of $\sum_\iota$, the optimal value of the objective function is guaranteed to be negative, i.e. the entailment (2) holds. In this case, the MaxSAT solver can terminate immediately. On the other hand, recall that our solver applies stratification to handle weighed soft clauses efficiently. Therefore, each stratification level once solved is finished with the working formula ϕ being satisfiable (this corresponds to line 11 in Algorithm 3). Its satisfying assignment can serve to calculate an under-approximation of the value of the objective function. If the approximate solution is non-negative, the MaxSAT solver can terminate immediately because this solution is evidence that the entailment (2) does not hold.

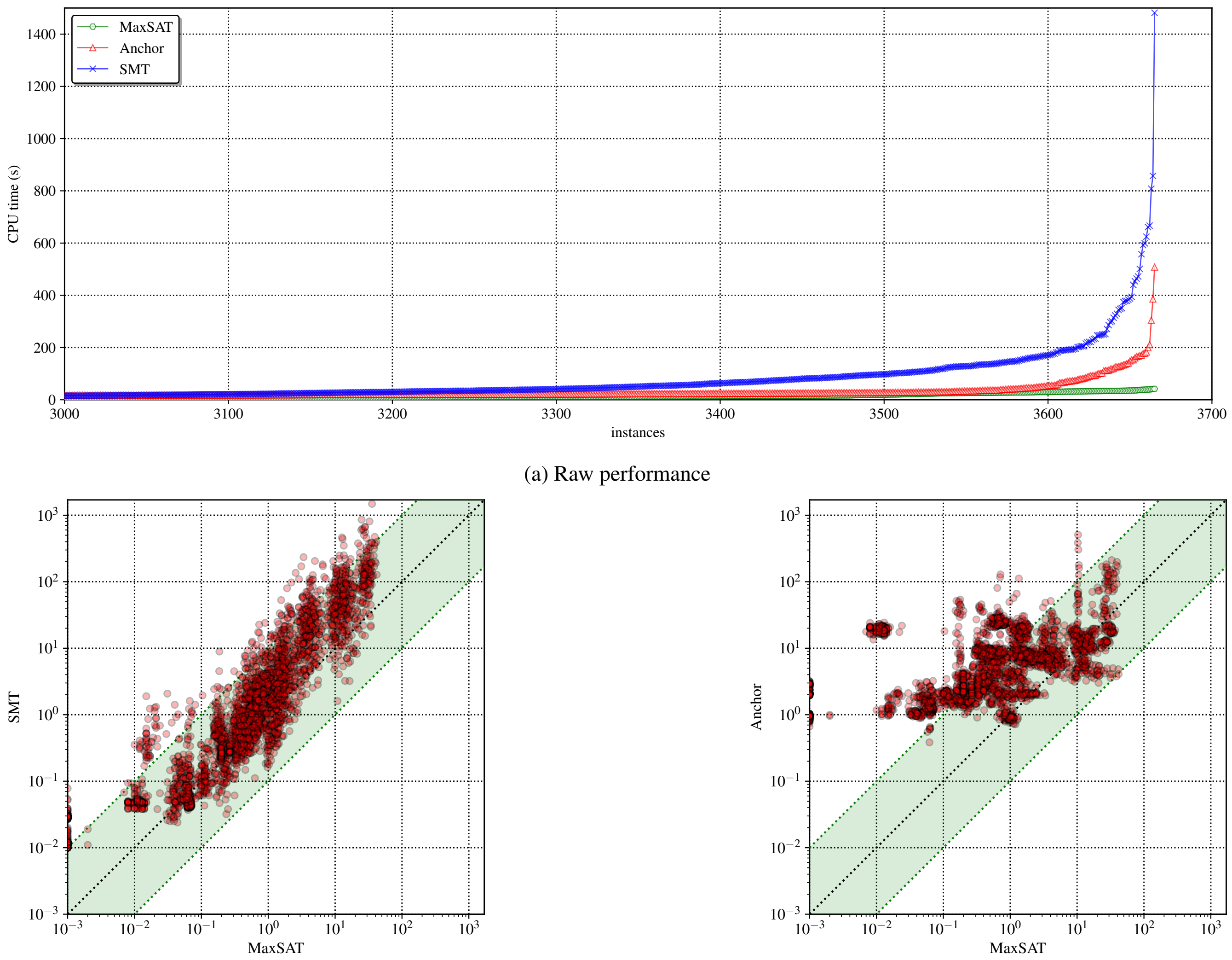
4 Experimental Results

Here we report on the experimental results obtained when testing the proposed ideas in practice in the context of computing explanations for some of the widely studied datasets.

[3] The pseudocode and the description of core reuse omit the low-level details and provide a basic overview of the general idea.

[4] Practically, each soft clause $c \in \mathcal{S}$ is represented by a unique selector literal s, i.e. we use clause $c \vee \neg s$ instead of c.

[5] The BLO condition is too strict while diversity-based stratification fails because the diversity of the weights of soft clauses is too high, i.e. each soft clause often has a *unique* real-valued weight.



(a) Raw performance

(b) Detailed runtime comparison vs. SMT

(c) Detailed runtime comparison vs. Anchor

Figure 2: Scalability of MaxSAT- and SMT-based AXp computation and of Anchor, on the considered benchmarks.

Experimental setup. The experiments are performed on a MacBook Pro with a Dual-Core Intel Core i5 2.3GHz CPU with 8GByte RAM running macOS Big Sur. The results reported do not impose any time or memory limit.

Prototype implementation. A prototype implementation[6] of the proposed approach was developed as a Python script. It builds on (Ignatiev, Narodytska, and Marques-Silva 2019c) and makes heavy use of the latest versions of the PySMT and PySAT toolkits (Gario and Micheli 2015; Ignatiev, Morgado, and Marques-Silva 2018). To our best knowledge, only two MaxSAT solvers support formulas with real weights: LMHS (Saikko, Berg, and Järvisalo 2016) and RC2 (Ignatiev, Morgado, and Marques-Silva 2019), and only the latter is core-guided. Thus the prototype of the MaxSAT-based explainer builds on Glucose 3 assisted (Audemard, Lagniez, and Simon 2013) RC2 and augments it with all the proposed heuristics. The SMT-based counterpart taken directly from (Ignatiev, Narodytska, and Marques-Silva 2019c) and uses Z3 (de Moura and Bjørner 2008) as the underlying SMT engine. (Note that we additionally tested CPLEX as an oracle for the original SMT encoding. As it was significantly outperformed by the Z3-based solution, MIP is excluded from consideration below.) Both competitors support the computation of one AXp and also their enumeration (Ignatiev et al. 2020). Although both explainers support QuickXplain-like AXp extraction (Junker 2004), it did not prove helpful in our initial results, and so the standard deletion-based Algorithm 2 was applied instead.

Additionally to the aforementioned SMT- and MaxSAT-based AXp computation, we compared these formal rigorous explanation approaches to the computation of a heuristic explanation with the use of Anchor (Ribeiro, Singh, and Guestrin 2018). Although heuristic explanations are known to suffer from a number of issues (Ignatiev, Narodytska, and

[6] Available at https://github.com/alexeyignatiev/xreason/.

Dataset	(#F	#C	#I)	BTs		Anchor				SMT						MaxSAT						
				D	%A	max	min	avg	%w	Enc		max	min	avg	%w	Enc		max	min	avg	#c	%w
ann-thyroid	(21	3	200)	3	100	23.46	0.38	2.78	0	(213	504)	2.78	0.04	0.13	80	(566	2780)	0.49	0.06	0.11	3	20
appendicitis	(7	2	106)	3	91	3.06	0.89	1.33	0	(157	428)	0.44	0.02	0.10	17	(475	2348)	0.11	0.03	0.04	7	80
biodegradation	(41	2	200)	3	87	65.99	4.43	8.47	0	(352	914)	234.08	0.33	14.70	31	(1331	5990)	5.28	1.02	2.31	29	69
divorce	(54	2	150)	3	100	24.26	13.49	19.27	0	(132	186)	0.39	0.04	0.05	0	(107	602)	0.02	0.01	0.01	8	100
ecoli	(7	5	200)	3	85	2.59	0.72	1.41	20	(445	1499)	8.68	0.16	1.39	35	(2189	11 404)	1.30	0.33	0.84	6	46
glass2	(9	2	162)	4	88	5.18	1.01	2.08	0	(217	699)	2.39	0.03	0.35	41	(927	4550)	0.63	0.13	0.22	8	59
ionosphere	(34	2	200)	3	93	25.23	2.86	7.51	0	(267	650)	22.65	0.28	2.20	17	(886	3932)	1.04	0.28	0.62	25	82
pendigits	(16	10	110)	3	99	51.36	4.05	13.92	1	(1470	4499)	125.78	6.48	29.45	0	(8037	60 198)	5.57	2.75	3.99	17	99
promoters	(58	2	106)	3	100	3.20	1.94	2.42	0	(104	129)	0.08	0.03	0.03	0	(4	304)	0.00	0.00	0.00	1	100
segmentation	(19	7	200)	3	95	23.62	1.57	4.86	0	(592	1175)	22.98	0.45	3.55	0	(1420	10 187)	1.62	0.15	0.67	16	100
shuttle	(9	7	200)	3	100	53.88	0.91	7.85	0	(509	1358)	4.52	0.20	0.58	22	(1498	10 081)	0.47	0.10	0.30	6	78
sonar	(60	2	200)	3	86	117.02	20.34	26.19	0	(295	631)	12.50	0.31	2.01	4	(847	3652)	1.03	0.46	0.69	33	96
spambase	(57	2	200)	4	96	27.26	3.02	6.74	48	(489	1338)	439.87	1.61	42.78	10	(2015	9652)	41.87	3.03	10.13	41	42
texture	(40	11	200)	3	98	507.89	9.96	51.46	48	(2073	4576)	1481.71	12.77	150.86	1	(8093	82 625)	40.57	9.77	24.68	37	52
threeOf9	(9	2	200)	3	100	1.03	0.67	0.95	0	(108	153)	0.02	0.01	0.01	0	(10	392)	0.00	0.00	0.00	1	100
twonorm	(20	2	200)	3	97	30.43	6.07	14.88	0	(463	1135)	52.86	0.20	4.27	30	(1750	7844)	1.97	0.96	1.49	20	70
vowel	(13	11	200)	4	93	21.42	3.68	11.85	56	(2292	5771)	464.07	6.53	60.50	2	(10 183	102 611)	17.62	6.31	11.94	13	42
wdbc	(30	2	200)	4	97	15.62	4.66	8.55	0	(269	654)	2.85	0.20	0.66	20	(894	4060)	0.58	0.29	0.42	22	80
wine-recognition	(13	3	178)	3	97	3.45	0.55	1.75	0	(241	454)	0.43	0.04	0.11	42	(491	2468)	0.14	0.05	0.09	9	54
wpbc	(33	2	194)	4	74	128.65	1.06	6.33	11	(302	784)	33.54	0.34	5.24	14	(1101	5090)	5.15	0.44	1.69	25	75
zoo	(16	7	59)	4	83	2.38	0.96	1.54	0	(386	651)	2.07	0.20	0.63	0	(196	2157)	0.08	0.01	0.02	8	100

Table 1: Detailed performance evaluation of computing AXps and Anchors for BTs.

Marques-Silva 2019c; Narodytska et al. 2019; Slack et al. 2020; Lakkaraju and Bastani 2020), it is still of interest to see how the proposed approach stands against one of the best heuristic explainers from the scalability perspective. (Note that all Anchor's parameters are set to their default values.)

Benchmarks. The experiments consider a selection of 21 publicly available datasets, which originate from UCI Machine Learning Repository (UCI) and Penn Machine Learning Benchmarks (PennML). The number of features (data instances, resp.) in the benchmark suite vary from 7 to 60 (59 to 58000, resp.) with the average being 26.95 (4684.47, resp.). All BT models are trained with XGBoost (Chen and Guestrin 2016) having 50 trees per class, each of depth at most 3–4 and using 80% of the dataset samples (20% used for test data). Also, the number of variables and constraints in the SMT (MaxSAT, resp.) encoding vary from 104 to 2292 variables and 129 to 5771 assertions (4 to 10183 variables and 304 to 102611 clauses, resp.).

For each of the considered datasets, we randomly picked 200 instances to explain (if a dataset has a fewer number of instances, we used all available instances). Both formal explainers are set to compute a single AXp per data instance from the selected set of instances. (Note that both deal with the same data instances, and compute the same AXp given an instance.) Enumeration of explanations turned out to be too time consuming for the SMT-based reasoner to complete with a reasonable amount of computing resources. Finally, Anchor is set to compute an explanation for exactly the same instances.

Results. Figure 2 shows the results of the conducted experiment. As can be observed in the cactus plot of Figure 2a, the MaxSAT-based explainer outperforms both the SMT approach and Anchor by a large margin. In particular, it is able to explain each of the considered instances with the runtime per instance varying from 0.001 to 41.843 seconds and the average runtime being 3.183 seconds. The runtime of the SMT-based approach varies from 0.01 to 1481.708 seconds with the average runtime being 16.672 seconds. Anchor takes from 0.384 to 507.886 seconds per instance to explain, with the average runtime of 10.225 seconds. The detailed instance-by-instance performance comparison of the two formal competitors is provided in the scatter plot shown in Figure 2b while the detailed comparison of the proposed MaxSAT approach against Anchor is depicted in Figure 2c. We should note that on average the MaxSAT approach is 5–10 times faster than its SMT-based rival, which makes it a viable and scalable alternative to the state of the art. Another observation we made is that in general the harder the benchmarks get, the larger the gap becomes between the performance of the two *formal* approaches, e.g. texture dataset has 11 classes and we observe that the SMT method may take 24 minutes to deliver an explanation while MaxSAT terminates in less than 1 minute for all tested instances. This may be found somewhat surprising given that the MaxSAT reasoner has to deal with much larger encoding (in terms of the number of clauses and variables) and has to make a much larger number of *decision* oracle calls. Finally, the MaxSAT approach proves to be more robust in terms of the average performance whilst the performance of the SMT-based explainer is somewhat unstable, even when explaining the predictions made by the same classifier for different instances of the same dataset. As for the comparison against Anchor, the proposed MaxSAT approach is 2–10 times faster on average. Observe that in many cases Anchor is outperformed by up to 2 orders of magnitude. We emphasize that this performance difference is especially remarkable given the soundness and minimality guarantees provided by AXps and the lack of such guarantees in the case of Anchor.

All the details on the benchmarks used as well as additional encoding and runtime statistics per benchmark is shown in Table 1. Columns *#F*, *#C*, and *#I* report the number of features, the number of classes, and the number of tested instances per dataset, respectively. Columns *D* and *%A* report the maximum tree depth and test accuracy of the trained BT. Sub-columns *max*, *min*, and *avg* of column *Anchor* (also of columns *SMT* and *MaxSAT*) show, respectively, the maximum, minimum, and average time in seconds to find an explanation. Given an approach (either Anchor, SMT, or MaxSAT), the percentage of instances won by this approach (over the competitor approaches) is given as *%w*. Additionally, for the SMT and MaxSAT approaches the column *Enc* details the number of variables and clauses introduced by the corresponding encoder. Finally, sub-column *#c* reports the average number of entailment oracle calls made by the MaxSAT approach.

5 Conclusions

This paper proposes a novel MaxSAT approach to explaining boosted tree classifications. It avoids directly encoding the large linear constraints required to determine the model's prediction, by mapping them to optimisation questions. This significantly improves the state-of-the-art compared to competing model-precise approaches. In order to achieve these results, we need to make use of incremental MaxSAT, where we reuse cores from one question to another, develop a new approach to stratification in MaxSAT, and extend our approach to recognise when we can terminate early.

Several lines of future work can be envisioned. First of all, applicability of the proposed ideas to other ML models admitting propositional encodings should be investigated. Second, the proposed heuristics, which aim at improving core-guided MaxSAT solving, may turn out to be beneficial in other settings where (incremental) MaxSAT engines are of use. Finally, our work exemplifies the need for a generic and efficient incremental interface for MaxSAT solvers similar to what has been done and widely used in the area of SAT.

Acknowledgments

This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004, and by the H2020-ICT38 project COALA "Cognitive Assisted agile manufacturing for a Labor force supported by trustworthy Artificial intelligence".

References

Andres, B.; Kaufmann, B.; Matheis, O.; and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In *ICLP*, 211–221.

Ansótegui, C.; Bonet, M. L.; Gabàs, J.; and Levy, J. 2013. Improving WPM2 for (Weighted) Partial MaxSAT. In *CP*, 117–132.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196: 77–105.

Ansótegui, C.; and Manyà, F. 2004. Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables. In *SAT*, 1–15.

Audemard, G.; Lagniez, J.; and Simon, L. 2013. Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction. In *SAT*, 309–317.

Bacchus, F.; Järvisalo, M.; and Martins, R. 2021. Maximum Satisfiability. In *Handbook of Satisfiability: Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 929–991. IOS Press.

Barceló, P.; Monet, M.; Pérez, J.; and Subercaseaux, B. 2020. Model Interpretability through the lens of Computational Complexity. In *NeurIPS*.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Breiman, L. 2001. Random Forests. *Mach. Learn.*, 45(1): 5–32.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth. ISBN 0-534-98053-8.

Chen, H.; Zhang, H.; Si, S.; Li, Y.; Boning, D. S.; and Hsieh, C. 2019. Robustness Verification of Tree-based Models. In *NeurIPS*, 12317–12328.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*, 785–794.

Darwiche, A.; and Hirth, A. 2020. On the Reasons Behind Decisions. In *ECAI*, 712–720.

Davies, J.; and Bacchus, F. 2011. Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In *CP*, 225–239.

de Moura, L. M.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*, 337–340.

Eén, N.; and Sörensson, N. 2003. Temporal induction by incremental SAT solving. *Electron. Notes Theor. Comput. Sci.*, 89(4): 543–560.

Friedman, J. H. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5): 1189–1232.

Gario, M.; and Micheli, A. 2015. PySMT: a Solver-Agnostic Library for Fast Prototyping of SMT-Based Algorithms. In *SMT Workshop*.

Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; and Pedreschi, D. 2019. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.*, 51(5): 93:1–93:42.

Huang, X.; Izza, Y.; Ignatiev, A.; Cooper, M. C.; Asher, N.; and Marques-Silva, J. 2022. Tractable Explanations for d-DNNF Classifiers. In *AAAI*.

Huang, X.; Izza, Y.; Ignatiev, A.; and Marques-Silva, J. 2021. On Efficiently Explaining Graph-Based Classifiers. In *KR*, 356–367.

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

Ignatiev, A. 2020. Towards Trustable Explainable AI. In *IJCAI*, 5154–5158.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2019. RC2: an Efficient MaxSAT Solver. *J. Satisf. Boolean Model. Comput.*, 11(1): 53–64.

Ignatiev, A.; Narodytska, N.; Asher, N.; and Marques-Silva, J. 2020. From Contrastive to Abductive Explanations and Back Again. In *AI*IA*, 335–355.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019a. Abduction-Based Explanations for Machine Learning Models. In *AAAI*, 1511–1519.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019b. On Relating Explanations and Adversarial Examples. In *NeurIPS*, 15857–15867.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019c. On Validating, Repairing and Refining Heuristic ML Explanations. *CoRR*, abs/1907.02509.

Izza, Y.; Ignatiev, A.; and Marques-Silva, J. 2020. On Explaining Decision Trees. *CoRR*, abs/2010.11034.

Izza, Y.; and Marques-Silva, J. 2021. On Explaining Random Forests with SAT. In *IJCAI*, 2584–2591.

Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *AAAI*, 167–172.

Kanamori, K.; Takagi, T.; Kobayashi, K.; Ike, Y.; Uemura, K.; and Arimura, H. 2021. Ordered Counterfactual Explanation by Mixed-Integer Linear Optimization. In *AAAI*, 11564–11574.

Lakkaraju, H.; and Bastani, O. 2020. "How do I fool you?": Manipulating User Trust via Misleading Black Box Explanations. In *AIES*, 79–85.

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

Lundberg, S. M.; and Lee, S. 2017. A Unified Approach to Interpreting Model Predictions. In *NIPS*, 4765–4774.

Marques-Silva, J.; Argelich, J.; Graça, A.; and Lynce, I. 2011. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4): 317–343.

Marques-Silva, J.; Gerspacher, T.; Cooper, M. C.; Ignatiev, A.; and Narodytska, N. 2020. Explaining Naive Bayes and Other Linear Classifiers with Polynomial Time and Delay. In *NeurIPS*.

Marques-Silva, J.; Gerspacher, T.; Cooper, M. C.; Ignatiev, A.; and Narodytska, N. 2021. Explanations for Monotonic Classifiers. In *ICML*, 7469–7479.

Marques-Silva, J.; and Ignatiev, A. 2022. Delivering Trustworthy AI through formal XAI. In *AAAI*.

Marques-Silva, J.; Janota, M.; and Belov, A. 2013. Minimal Sets over Monotone Predicates in Boolean Formulae. In *CAV*, 592–607.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267: 1–38.

Molnar, C. 2020. Interpretable Machine Learning. https://christophm.github.io/interpretable-ml-book/.

Morgado, A.; Dodaro, C.; and Marques-Silva, J. 2014. Core-Guided MaxSAT with Soft Cardinality Constraints. In *CP*, 564–573.

Morgado, A.; Heras, F.; Liffiton, M. H.; Planes, J.; and Marques-Silva, J. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints An Int. J.*, 18(4): 478–534.

Narodytska, N.; Shrotri, A. A.; Meel, K. S.; Ignatiev, A.; and Marques-Silva, J. 2019. Assessing Heuristic Machine Learning Explanations with Model Counting. In *SAT*, 267–278.

Parmentier, A.; and Vidal, T. 2021. Optimal Counterfactual Explanations in Tree Ensembles. In Meila, M.; and Zhang, T., eds., *ICML*, 8422–8431.

PennML. 2021. Penn Machine Learning Benchmarks. https://github.com/EpistasisLab/penn-ml-benchmarks.

Previti, A.; Mencía, C.; Järvisalo, M.; and Marques-Silva, J. 2017. Improving MCS Enumeration via Caching. In *SAT*, 184–194.

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

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *AAAI*, 1527–1535.

Rudin, C. 2018. Please Stop Explaining Black Box Models for High Stakes Decisions. *CoRR*, abs/1811.10154.

Saikko, P.; Berg, J.; and Järvisalo, M. 2016. LMHS: A SAT-IP Hybrid MaxSAT Solver. In *SAT*, 539–546.

Shih, A.; Choi, A.; and Darwiche, A. 2018. A Symbolic Approach to Explaining Bayesian Network Classifiers. In *IJCAI*, 5103–5111.

Shrotri, A. A.; Narodytska, N.; Ignatiev, A.; Meel, K.; Marques-Silva, J.; and Vardi, M. 2022. Constraint-Driven Explanations of Black-Box ML Models. In *AAAI*.

Slack, D.; Hilgard, S.; Jia, E.; Singh, S.; and Lakkaraju, H. 2020. Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. In *AIES*, 180–186.

UCI. 2021. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml.

Zhou, Z.-H. 2012. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.