Tractable Explanations for d-DNNF Classifiers

Xuanxiang Huang[1], **Yacine Izza**[1], **Alexey Ignatiev**[2],
Martin Cooper[3], **Nicholas Asher**[4], **Joao Marques-Silva**[4]
[1] University of Toulouse, France
[2] Monash University, Melbourne, Australia
[3] IRIT, CNRS, Univ. Paul Sabatier, Toulouse, France
[4] IRIT, CNRS, Toulouse, France
xuanxiang.huang@univ-toulouse.fr, yacine.izza@univ-toulouse.fr, alexey.ignatiev@monash.edu,
martin.cooper@irit.fr, nicholas.asher@irit.fr, joao.marques-silva@irit.fr

Abstract

Compilation into propositional languages finds a growing number of practical uses, including in constraint programming, diagnosis and machine learning (ML), among others. One concrete example is the use of propositional languages as classifiers, and one natural question is how to explain the predictions made. This paper shows that for classifiers represented with some of the best-known propositional languages, different kinds of explanations can be computed in polynomial time. These languages include deterministic decomposable negation normal form (d-DNNF), and so any propositional language that is strictly less succinct than d-DNNF. Furthermore, the paper describes optimizations, specific to Sentential Decision Diagrams (SDDs), which are shown to yield more efficient algorithms in practice.

1 Introduction

The growing use of machine learning (ML) models in practical applications raises a number of concerns related with fairness, robustness, but also explainability (Lipton 2018; Weld and Bansal 2019; Monroe 2021). Recent years have witnessed a number of works on computing explanations for the predictions made by ML models[1]. Approaches to computing explanations can be broadly categorized as heuristic (Ribeiro, Singh, and Guestrin 2016; Lundberg and Lee 2017; Ribeiro, Singh, and Guestrin 2018), which offer no formal guarantees of rigor, and non-heuristic (Shih, Choi, and Darwiche 2018; Ignatiev, Narodytska, and Marques-Silva 2019a; Darwiche and Hirth 2020; Audemard, Koriche, and Marquis 2020), which in contrast offer strong guarantees of rigor. Non-heuristic explanation approaches can be further categorized into compilation-based (Shih, Choi, and Darwiche 2018, 2019; Darwiche and Hirth 2020) and oracle-based (Ignatiev, Narodytska, and Marques-Silva 2019a; Malfa et al. 2021).

Compilation-based approaches (Shih, Choi, and Darwiche 2018, 2019) compile the decision function associated with an ML classifier into some propositional language (among those covered by the knowledge compilation map (Darwiche and Marquis 2002)). As a result, more recent work studied such propositional languages from the perspective of explainability, with the purpose of understanding the complexity of computing explanations (Audemard, Koriche, and Marquis 2020; Barceló et al. 2020; Audemard et al. 2021), but also with the goal of identifying examples of queries of interest (Audemard, Koriche, and Marquis 2020; Audemard et al. 2021). Furthermore, although recent work (Audemard, Koriche, and Marquis 2020; Barceló et al. 2020) analyzed the complexity of explainability queries for classifiers represented with different propositional languages, it is also the case that it is unknown which propositional languages allow the expressible functions to be explained efficiently, and which do not. On the one hand, (Audemard, Koriche, and Marquis 2020) proposes conditions not met by most propositional languages. On the other hand (Barceló et al. 2020) studies restricted cases of propositional languages, but focusing on smallest PI-explanations. Also, since one key motivation for the use of propositional languages is the efficiency of reasoning, namely with respect to specific queries and transformations (Darwiche and Marquis 2002), a natural question is whether similar results can be obtained in the setting of explainability.

This paper studies the computational complexity of computing PI-explanations (Shih, Choi, and Darwiche 2018) and contrastive explanations (Miller 2019b) for classifiers represented with well-known propositional languages. Concretely, the paper shows that for any propositional language that implements in polynomial time the queries of consistency (**CO**) and validity (**VA**), and the transformation of conditioning (**CD**), then one PI-explanation or one contrastive explanation can be computed in polynomial time. This requirement is strictly less stringent than another one proposed in earlier work (Audemard, Koriche, and Marquis 2020), thus proving that explanations can be computed in polynomial time for a larger range of propositional languages. Concretely, the paper shows that for classifiers represented with several propositional languages, that include d-DNNF, one PI-explanation or one contrastive explanation can be computed in polynomial time. The result immediately generalizes to propositional languages less succinct than d-DNNF, e.g. OBDD, SDD, to name a few. Moreover, for the concrete

[1]There is a fast growing body of work on the explainability of ML models. Example references include (Guidotti et al. 2019; Samek et al. 2019; Samek and Müller 2019; Miller 2019b,a; Anjomshoae et al. 2019; Mittelstadt, Russell, and Wachter 2019; Xu et al. 2019; Mueller et al. 2019).

case of SDDs, the paper shows that practical optimizations lead to clear performance gains. Besides computing one explanation, one is often interested in obtaining multiple explanations, thus allowing a decision maker to get a better understanding of the reasons supporting a decision. As a result, the paper also outlines a MARCO-like (Liffiton et al. 2016) algorithm for the enumeration of both AXps and CXps. Furthermore, the paper studies the computational complexity of explaining generalizations of decision sets (Lakkaraju, Bach, and Leskovec 2016), and proposes conditions under which explanations can be computed in polynomial time.

The paper is organized as follows. Section 2 introduces the definitions and notation used throughout the paper. Section 3 shows that for several classes of propositional languages, one PI-explanation and one contrastive explanation can be computed in polynomial time. In addition, Section 3 shows how to enumerate explanations requiring one NP oracle call (in fact a SAT reasoner call) for each computed explanation. Section 4 investigates a number of generalized classifiers, which can be built from propositional languages used as building blocks. Section 5 assesses the computation of explanations of d-DNNF's and SDDs in practical settings. Section 7 concludes the paper.

2 Preliminaries

Classification problems & formal explanations. This paper considers classification problems, which are defined on a set of features (or attributes) $\mathcal{F} = \{1, \ldots, m\}$ and a set of classes $\mathcal{K} = \{c_1, c_2, \ldots, c_K\}$. Each feature $i \in \mathcal{F}$ takes values from a domain $\mathbb{D}_i$. In general, domains can be boolean, integer or real-valued, but in this paper we restrict $\mathbb{D}_i = \{0, 1\}$ and $\mathcal{K} = \{0, 1\}$. (In the context of propositional languages, we will replace 0 by $\bot$ and 1 by $\top$. This applies to domains and classes.) Feature space is defined as $\mathbb{F} = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_m = \{0,1\}^m$. The notation $\mathbf{x} = (x_1, \ldots, x_m)$ denotes an arbitrary point in feature space, where each x_i is a variable taking values from $\mathbb{D}_i$. The set of variables associated with features is $X = \{x_1, \ldots, x_m\}$. Moreover, the notation $\mathbf{v} = (v_1, \ldots, v_m)$ represents a specific point in feature space, where each v_i is a constant representing one concrete value from $\mathbb{D}_i = \{0, 1\}$. An *instance* (or example) denotes a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$. (We also use the term *instance* to refer to $\mathbf{v}$, leaving c implicit.) An ML classifier $\mathbb{C}$ is characterized by a *classification function* κ that maps feature space $\mathbb{F}$ into the set of classes $\mathcal{K}$, i.e. $\kappa : \mathbb{F} \to \mathcal{K}$. (It is assumed throughout that κ is not constant, i.e. there are at least two points $\mathbf{v}_1$ and $\mathbf{v}_2$ in feature space, where $\kappa(\mathbf{v}_1) \neq \kappa(\mathbf{v}_2)$.)

We now define formal explanations. Prime implicant (PI) explanations (Shih, Choi, and Darwiche 2018) denote a minimal set of literals (relating a feature value x_i and a constant $v_i \in \mathbb{D}_i$) that are sufficient for the prediction[2]. Formally, given $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a *weak* (or non-minimal) *abductive explanation* (weak AXp) is any subset $\mathcal{X} \subseteq \mathcal{F}$ such that,

$$\forall(\mathbf{x} \in \mathbb{F}).\left[\bigwedge\nolimits_{i\in\mathcal{X}}(x_i = v_i)\right] \to (\kappa(\mathbf{x}) = c) \qquad (1)$$

Any *subset-minimal* weak AXp is referred to as an AXp. AXps can be viewed as answering a 'Why?' question, i.e. why is some prediction made given some point in feature space. A different view of explanations is a contrastive explanation (Miller 2019b), which answers a 'Why Not?' question, i.e. which features can be changed to change the prediction. A formal definition of *contrastive explanation* (CXp) is proposed in recent work (Ignatiev et al. 2020b). Given $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a *weak* (or non-minimal) CXp is any subset $\mathcal{Y} \subseteq \mathcal{F}$ such that,

$$\exists(\mathbf{x} \in \mathbb{F}).\bigwedge\nolimits_{j\in\mathcal{F}\setminus\mathcal{Y}}(x_j = v_j) \land (\kappa(\mathbf{x}) \neq c) \qquad (2)$$

Any *subset-minimal* weak CXp is referred to as a CXp. (For simplicity, except specifically indicated, in the rest of the paper AXp will be used to denote the subset-minimal weak AXp, the same for CXp.) Building on the results of R. Reiter in model-based diagnosis (Reiter 1987), (Ignatiev et al. 2020b) proves a minimal hitting set (MHS) duality relation between AXps and CXps, i.e. AXps are MHSes of CXps and vice-versa.

Propositional languages. Following earlier work (Darwiche 2001; Darwiche and Marquis 2002; Huang and Darwiche 2007), we define negated normal form (NNF), decomposable NNF (DNNF), deterministic DNNF (d-DNNF), decision DNNF (dec-DNNF), and also smooth d-DNNF (sd-DNNF).

Definition 1 (Propositional languages (Darwiche and Marquis 2002)). *The following propositional languages are studied in the paper:*

- Negation normal form *(*NNF*) is the set of all directed acyclic graphs where each leaf node is labeled with either $\top$, $\bot$, x_i or $\neg x_i$, for $x_i \in X$. Each internal node is labeled with either $\land$ (or* AND*) or $\lor$ (or* OR*).*
- Decomposable NNF *(*DNNF*) is the set of all NNFs where for every node labeled with $\land$, $\alpha = \alpha_1 \land \cdots \land \alpha_k$, no variables are shared between the conjuncts α_j.*
- *A* d-DNNF *is a* DNNF *where for every node labeled with $\lor$, $\beta = \beta_1 \lor \cdots \lor \beta_k$, each pair β_p, β_q, with $p \neq q$, is inconsistent, i.e. $\beta_p \land \beta_q \vDash \bot$.*
- *A* dec-DNNF *(or* decision-DNNF*) is a* DNNF*, where every node labeled with $\lor$ is given by $(x_i \land \alpha) \lor (\neg x_i \land \beta)$.*
- *An* sd-DNNF *is a* d-DNNF *where for every node labeled with $\lor$, $\beta = \beta_1 \lor \cdots \lor \beta_k$, each pair β_p, β_q is defined on the same set of variables.*

The paper establishes explainability results for classifiers represented with d-DNNF. However, for simplicity of algorithms, sd-DNNF is often considered (Darwiche 2001). Furthermore, *sentential decision diagrams* (SDDs) represent a well-known subset of the d-DNNF (Darwiche 2011; Van den Broeck and Darwiche 2015; Bova 2016). SDDs are based on a strongly deterministic decomposition (Darwiche 2011), which is used to decompose a Boolean function into the

[2]PI-explanations are related with abduction, and so are also referred to as abductive explanations (AXp) (Ignatiev, Narodytska, and Marques-Silva 2019a). More recently, PI-explanations have been studied from a knowledge compilation perspective (Audemard, Koriche, and Marquis 2020; Audemard et al. 2021).

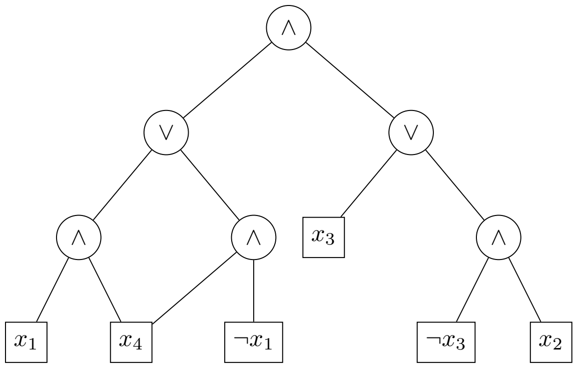

Figure 1: A d-DNNF used as a running example

form: $(p_1 \wedge s_1) \vee \cdots \vee (p_n \wedge s_n)$, where each p_i is called a *prime* and each s_i is called a *sub* (both primes and subs are sub-functions). Furthermore, the process of decomposition is governed by a variable tree (*vtree*) which stipulates the variable order (Darwiche 2011).

Throughout the paper, a term ρ denotes a conjunction of literals. A term ρ is consistent ($\rho \nvDash \bot$) if the term is satisfied in at least one point in feature space.

This paper considers exclusively the queries **CO** and **VA**, and the transformation **CD**, which are defined next. Let $\mathbf{L}$ denote a subset of NNF. Hence, we have the following standard definitions.

Definition 2 (Conditioning (Darwiche and Marquis 2002)). *Let Δ represent a propositional formula and let ρ denote a consistent term. The* conditioning *of Δ on ρ, i.e. $\Delta|_\rho$, is the formula obtained by replacing each variable x_i by $\top$ (resp. $\bot$) if x_i (resp. $\neg x_i$) is a positive (resp. negative) literal of ρ.*

Definition 3 (Queries & transformations (Darwiche and Marquis 2002)). *The following queries and transformations are used throughout with respect to a propositional language $\mathbf{L}$[3]:*

- $\mathbf{L}$ *satisfies the consistency (validity) query* **CO** *(***VA***) iff there exists a polynomial-time algorithm that maps every formula Δ from $\mathbf{L}$ to 1 if Δ is consistent (valid), and to 0 otherwise.*
- $\mathbf{L}$ *satisfies the conditioning transformation* **CD** *iff there exists a polynomial-time algorithm that maps every formula Δ from $\mathbf{L}$ and every consistent term ρ into a formula that is logically equivalent to $\Delta|_\rho$.*

Example 1. *Figure 1 shows the running example used throughout the paper. $\mathcal{F} = \{1,2,3,4\}$, $X = \{x_1,x_2,x_3,x_4\}$, and $\kappa((x_1,x_2,x_3,x_4)) = ((x_1 \wedge x_4) \vee (\neg x_1 \wedge x_4)) \wedge (x_3 \vee (\neg x_3 \wedge x_2))$. Moreover, the paper considers the concrete instance $(\mathbf{v}, c) = ((0,0,0,0),0)$.*

[3]There are additional queries and transformations of interest (Darwiche and Marquis 2002), but these are omitted for succinctness.

Algorithm 1: Finding one AXp given starting seed $\mathcal{S}$

```
    Input: Classifier $\kappa$, Seed Set $\mathcal{S}$, Instance $\mathbf{v}$,
           Class $c$, Conditioner $\varsigma_A$
    Output: AXp $\mathcal{S}$
1:  procedure findAXp($\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_A$)
2:      for all $i \in \mathcal{S}$ do
3:          $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_A(\kappa, \mathcal{S} \setminus \{i\}, \mathbf{v})$
4:          if $[c = \top \wedge \text{isValid}(\kappa|_{\mathbf{s},\mathbf{v}})]$ or
                $[c = \bot \wedge$ not $\text{isConsistent}(\kappa|_{\mathbf{s},\mathbf{v}})]$ then
5:              $\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
6:      return $\mathcal{S}$
```

3 Explanations for d-DNNF

As will be shown in this section, there is a tight connection between the definitions of AXp and CXp (see (1) and (2)) and the queries **VA**, **CO** and the transformation **CD**. Indeed, for (1) and (2), **CD** can serve to impose that the values of some features (i, represented by variable x_i) are fixed to some value v_i. In addition, **VA** (resp. **CO**) is used to decide (1), after conditioning, when $c = 1$ (resp. $c = 0$). Similarly, **VA** (resp. **CO**) is used to decide (2), again after conditioning, when $c = 1$ (resp. $c = 0$). Thus, for languages respecting the (poly-time) queries **VA** and **CO** and the (poly-time) transformation **CD**, one can compute one AXp and one CXp in polynomial time. This is detailed in the rest of this section.

Finding one AXp. We start by detailing an algorithm to find one AXp. We identify any $\mathcal{S} \subseteq \{1,\ldots,m\}$ with its corresponding bit-vector $\mathbf{s} = (s_1,\ldots,s_m)$ where $s_i = 1 \Leftrightarrow i \in \mathcal{S}$. Given vectors $\mathbf{x}, \mathbf{v}, \mathbf{s}$, we can construct the vector $\mathbf{x}^{\mathbf{s},\mathbf{v}}$ (in which $\mathbf{s}$ is a selector between the two vectors $\mathbf{x}$ and $\mathbf{v}$) such that

$$x_i^{\mathbf{s},\mathbf{v}} = (x_i \wedge \overline{s_i}) \vee (v_i \wedge s_i) \tag{3}$$

To find an AXp, i.e. a subset-minimal weak AXp, Algorithm 1 is used. (For now, seed $\mathcal{S}$ is set to $\mathcal{F}$.) (Algorithm 1 is a general greedy algorithm that is well-known and used in a wide range of settings, e.g. minimal unsatisfiable core extraction in CSPs (Chinneck and Dravnieks 1991; Bakker et al. 1993), but which is also present in the seminal work of Valiant (Valiant 1984). The novelty is the use of the same algorithm for finding AXps (and also CXps) of propositional languages that respect concrete transformations and queries of the knowledge compilation map. Possible alternatives would include the QuickXplain (Junker 2004) or the Progression (Marques-Silva, Janota, and Belov 2013) algorithms, among other options (Marques-Silva, Janota, and Mencía 2017).)

Considering $\mathbf{s}$ and $\mathbf{v}$ as constants, when $c = 1$, $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ is valid iff S is a weak AXp of $\kappa(\mathbf{v}) = c$. Furthermore, when $c = 0$, $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ is inconsistent iff S is a weak AXp of $\kappa(\mathbf{v}) = c$. We therefore have the following proposition.

Proposition 1. *For a classifier implemented with some propositional language $\mathbf{L}$, finding one AXp is polynomial-time provided the following three operations can be performed in polynomial time:*
1. construction of $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ from κ, $\mathbf{s}$ *and* $\mathbf{v}$.

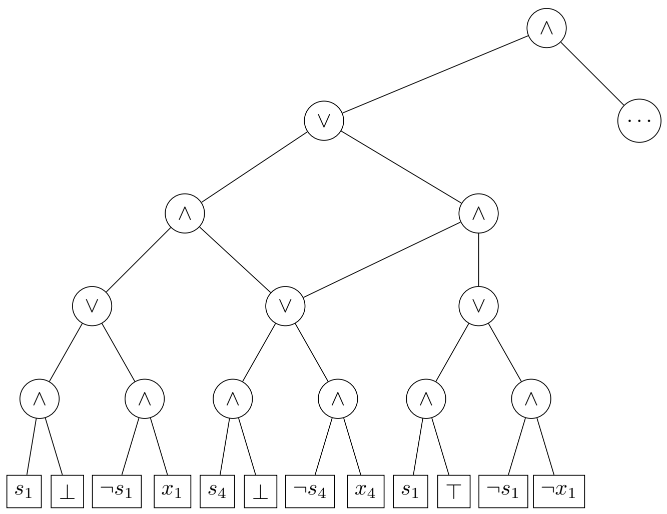

Figure 2: Partial d-DNNF for $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$, with $\mathbf{v} = (0,0,0,0)$

2. *testing validity of* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$.
3. *testing consistency of* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$.

Corollary 1. *Finding one AXp of a decision taken by a d-DNNF is polynomial-time.*

Proof. It is sufficient to show that d-DNNF's satisfy the conditions of Proposition 1. Consistency and validity on d-DNNF's is well-known to be decided in polynomial time (Darwiche and Marquis 2002). To transform a d-DNNF calculating $\kappa(\mathbf{v})$ into a d-DNNF calculating $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$, we need to replace each leaf labelled x_i by a leaf labelled $(x_i \wedge \overline{s_i}) \vee (v_i \wedge s_i)$ and each leaf labelled $\overline{x_i}$ by a leaf labelled $(\overline{x_i} \wedge \overline{s_i}) \vee (\overline{v_i} \wedge s_i)$. Note that $\mathbf{s}$ and $\mathbf{v}$ are constants during this construction. Thus, we simplify these formulas to obtain either a literal or a constant according to the different cases:

- $s_i = 0$: label $(x_i \wedge \overline{s_i}) \vee (v_i \wedge s_i)$ is x_i and label $(\overline{x_i} \wedge \overline{s_i}) \vee (\overline{v_i} \wedge s_i)$ is $\overline{x_i}$. In other words, the label of the leaf node is unchanged.
- $s_i = 1$: label $(x_i \wedge \overline{s_i}) \vee (v_i \wedge s_i)$ is the (constant) value of v_i and label $(\overline{x_i} \wedge \overline{s_i}) \vee (\overline{v_i} \wedge s_i)$ is the (constant) value of $\overline{v_i}$.

Indeed, this is just conditioning (**CD**, i.e. fixing a subset of the variables x_i, given by the set S, to v_i) and it is well known that **CD** is a polytime operation on d-DNNFs (Darwiche and Marquis 2002). □

Figure 2 illustrates the proposed transformation for part of the d-DNNF of Figure 1.

Example 2. *The operation of the algorithm for computing one AXp is illustrated for the modified d-DNNF shown in Figure 2 for the instance* $(\mathbf{v}, c) = ((0,0,0,0),0)$*. By inspection, we can observe that the value computed by the d-DNNF will be 0 as long as* $s_4 = 1$*, i.e. as long as* 4 *is part of the weak AXp. If removed from the weak AXp, one can find an assignment to* $\mathbf{x}$*, which sets* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$*. The computed AXp is* $\mathcal{S} = \{4\}$.

Algorithm 2: Finding one CXp given starting seed $\mathcal{S}$

```
    Input: Classifier $\kappa$, Seed Set $\mathcal{S}$, Instance $\mathbf{v}$,
           Class $c$, Conditioner $\varsigma_C$
    Output: CXp $\mathcal{S}$
1:  procedure findCXp($\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_C$)
2:      for all $i \in \mathcal{S}$ do
3:          $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_C(\kappa, \mathcal{S} \setminus \{i\}, \mathbf{v})$
4:          if $[c = \top \wedge$ not isValid$(\kappa|_{\mathbf{s},\mathbf{v}})]$ or
               $[c = \bot \wedge$ isConsistent$(\kappa|_{\mathbf{s},\mathbf{v}})]$ then
5:              $\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
6:      return $\mathcal{S}$
```

Finding one CXp. To compute one CXp, (2) is used. In this case, we identify any $\mathcal{S} \subseteq \{1, \ldots, m\}$ with its corresponding bit-vector $\mathbf{s}$ where $s_i = 1 \Leftrightarrow i \in \mathcal{F} \setminus \mathcal{S}$. Moreover, we adapt the approach used for computing one AXp, as shown in Algorithm 2. (As in the case of Algorithm 1, seed $\mathcal{S}$ is set to $\mathcal{F}$, for now.) (Observe that the main difference is the relationship between $\mathcal{S}$ and $\mathbf{s}$, and the test for a weak CXp. Also, recall from Section 2 that κ is assumed not to be constant, and so a CXp can always be computed.)

Proposition 2. *For a classifier implemented with some propositional language* $\mathbf{L}$*, finding one CXp is polynomial-time provided the operations of Proposition 1 can be performed in polynomial time.*

Corollary 2. *Finding one AXp/CXp of a decision taken by a classifier is polynomial-time if the classifier is given in one of the following languages: d-DNNF (Darwiche and Marquis 2002), SDD (Darwiche 2011), OBDD (Darwiche and Marquis 2002), PI (Darwiche and Marquis 2002), IP (Darwiche and Marquis 2002), renH-C (Fargier and Marquis 2008), AFF (Fargier and Marquis 2008), dFSD (Niveau, Fargier, and Pralet 2011), and EADT (Koriche et al. 2013).*

Proof. It suffices to observe that the languages listed above satisfy the conditions of Propositions 1 and 2. □

Example 3. *The operation of the algorithm for computing one CXp is illustrated for the modified d-DNNF shown in Figure 2 for the instance* $(\mathbf{v}, c) = ((0,0,0,0),0)$*. By inspection, we can observe that the value computed by the d-DNNF can be changed to 1 as long as* $s_3 = 0 \wedge s_4 = 0$*, i.e. as long as* $\{3,4\}$ *are part of the weak CXp. If removed from the weak CXp, one no longer can find an assignment to* $\mathbf{x}$ *that sets* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$*. Thus, the computed CXp is* $\mathcal{S} = \{3,4\}$.

Enumerating AXps/CXps. Finally, we outline a MARCO-like algorithm (Liffiton et al. 2016) for on-demand enumeration of AXps and CXps. For that, we use Algorithm 1 and Algorithm 2, but now allow for some initial set of features (i.e. a seed) to be specified. The seed is used for computing the next AXp or CXp, and it is picked such that repetition of explanations is disallowed. As argued below, the algorithm's organization ensures that computed explanations are not repeated. Moreover, since the algorithms for computing one AXp or one CXp run in polynomial time, then the enumeration algorithm is guaranteed to require exactly one call to an NP oracle for

Algorithm 3: Enumeration algorithm

```
Input: Feature Set $\mathcal{F}$, Classifier $\kappa$, Instance $\mathbf{v}$,
       Class $c$, Conditioners $\varsigma_A, \varsigma_C$
1:  procedure Enumerate($\mathcal{F}, \kappa, \mathbf{v}, c, \varsigma_A, \varsigma_C$)
2:      $\mathcal{H} \leftarrow \emptyset$        // $\mathcal{H}$ defined on set $P = \{p_1, \dots, p_m\}$
3:      repeat
4:          $(\mathsf{outc}, \mathbf{p}) \leftarrow \mathsf{SAT}(\mathcal{H})$
5:          if $\mathsf{outc} =$ true then
6:              $\mathcal{S} \leftarrow \{i \in \mathcal{F} \,|\, p_i = 1\}$
7:              $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_A(\kappa, \mathcal{S}, \mathbf{v})$
8:              if $[c = \top \wedge \mathsf{isValid}(\kappa|_{\mathbf{s},\mathbf{v}})]$ or
                    $[c = \bot \wedge$ not $\mathsf{isConsistent}(\kappa|_{\mathbf{s},\mathbf{v}})]$ then
9:                  $X \leftarrow \mathsf{findAXp}(\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_A)$
10:                 $\mathsf{reportAXp}(X)$
11:                 $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in X} \neg p_i)\}$
12:             else
13:                 $X \leftarrow \mathsf{findCXp}(\kappa, \mathcal{F} \setminus \mathcal{S}, \mathbf{v}, c, \varsigma_C)$
14:                 $\mathsf{reportCXp}(X)$
15:                 $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in X} p_i)\}$
16:     until $\mathsf{outc} =$ false
```

each computed explanation, in addition to procedures that run in polynomial time.

The main building blocks of the enumeration algorithm are: (1) finding one AXp given a seed (see Algorithm 1); (2) finding one CXp given a seed (see Algorithm 2); and (3) a top-level algorithm that ensures that previously computed explanations are not repeated (see Algorithm 3). The top level-algorithm invokes a SAT oracle to identify the seed which will determine whether a fresh AXp or CXp will be computed in the next iteration. As argued earlier, the algorithms for computing one AXp and one CXp use one transformation, specifically conditioning (**CD**, see line 3) and two queries, namely consistency and validity (**CO/VA**, see line 4). In the case of computing one AXp, if the prediction is $\top$, we need to check validity, i.e. for all (conditioned) assignments, the prediction is also $\top$. In contrast, if the prediction is $\bot$, then we need to check that consistency does not hold, i.e. for all (conditioned) assignments, the prediction is also $\bot$. In contrast, in the case of computing one CXp, we need to change the tests that are executed, since we seek to change the value of the prediction. It should be noted that, by changing the conditioning operation, different propositional languages can be explained. Finally, Algorithm 3 shows the proposed approach for enumerating AXps and CXps, which adapts the basic MARCO algorithm for enumerating minimal unsatisfiable cores (Liffiton et al. 2016). From the definitions, we can see that for any $\mathcal{S} \subseteq \mathcal{F}$, either $\mathcal{S}$ is a weak AXp or $\mathcal{F} \setminus \mathcal{S}$ is a weak CXp. Every set $\mathcal{S}$ calculated at line 6 of Algorithm 3 has the property that it is not a superset of any previously found AXp (due to the clauses added to $\mathcal{H}$ at line 11) and that $\mathcal{F} \setminus \mathcal{S}$ is not a superset of any previously found CXp (due to the clauses added at line 15).

Example 4. *Table 1 summarizes the main steps of enumerating the AXps and CXps of the running example (see Figure 1). It is easy to confirm that after four explanations are computed, $\mathcal{H}$ becomes inconsistent, and so the algorithm terminates. Also, one can confirm the hitting set duality between AXps and CXps (Ignatiev et al. 2020b).*

4 Generalizations

This section generalizes earlier results by considering multi-class classification, i.e. the set of classes is now $\mathcal{K} = \{c_1, \dots, c_K\}$.

Explanations for generalized decision functions. First, we consider that each class $c_j \in \mathcal{K}$ is associated with a total function $\kappa_j : \mathbb{F} \to \{0, 1\}$, such that the class c_j is picked iff $\kappa_j(\mathbf{v}) = 1$. For example, *decision sets* (Lakkaraju, Bach, and Leskovec 2016) represent one such example of multi-class classification, where each function κ_j is represented by a DNF, and a *default rule* is used to pick some class for the points $\mathbf{v}$ in feature space for which all $\kappa_j(\mathbf{v}) = 0$. Moreover, decision sets may exhibit *overlap* (Ignatiev et al. 2018), i.e. the existence of points $\mathbf{v}$ in feature space such that there exist $j_1 \neq j_2$ and $\kappa_{j_1}(\mathbf{v}) = \kappa_{j_2}(\mathbf{v}) = 1$. In practice, the existence of overlap can be addressed by randomly picking one of the classes for which $\kappa_j(\mathbf{v}) = 1$. Alternatively, DS learning can ensure that overlap is non-existing (Ignatiev et al. 2018).

Here, we consider generalized versions of DSes, by removing the restriction that each class is computed with a DNF. Hence, a *generalized decision function* (GDF) is such that each function κ_j is allowed to be an *arbitrary* boolean function. Furthermore, the following two properties of GDFs are considered:

Definition 4 (Binding GDF)**.** *A GDF is* binding *if,*

$$\forall (\mathbf{x} \in \mathbb{F}). \bigvee\nolimits_{1 \leq j \leq K} \kappa_j(\mathbf{x}) \tag{4}$$

(Thus, a binding GDF requires no default rule, since for any point $\mathbf{x}$ in feature space, there is at least one κ_j such that $\kappa_j(\mathbf{x})$ holds.)

Definition 5 (Non-overlapping GDF)**.** *A GDF is* non-overlapping *if,*

$$\forall (\mathbf{x} \in \mathbb{F}). \bigwedge\nolimits_{1 \leq j_1, j_2 \leq K, j_1 \neq j_2} (\neg \kappa_{j_1}(\mathbf{x}) \vee \neg \kappa_{j_2}(\mathbf{x})) \tag{5}$$

(Thus, a binding, non-overlapping GDF computes a total multi-class classification function.) Furthermore, we can establish conditions for a GDF to be binding and non-overlapping:

Proposition 3. *A GDF is binding and non-overlapping iff the following formula is inconsistent:*

$$\exists (\mathbf{x} \in \mathbb{F}). \kappa_1(\mathbf{x}) + \ldots + \kappa_K(\mathbf{x}) \neq 1 \tag{6}$$

Proof. Given Definition 4 and Definition 5,

1. Clearly, there exists a point $\mathbf{v} \in \mathbb{F}$ such that $\kappa_1(\mathbf{v}) + \ldots + \kappa_K(\mathbf{v}) = 0$ iff the GDF is non-binding;
2. Clearly, there exists $\mathbf{v} \in \mathbb{F}$ such that $\kappa_1(\mathbf{v}) + \ldots + \kappa_K(\mathbf{v}) \geq 2$ iff the GDF is overlapping.

Thus, the result follows. □

Remark 1. *For a GDF where each function is represented by a boolean circuit, deciding whether a GDF is binding and non-overlapping is in coNP. In practice, checking whether a GDF is binding and non-overlapping can be decided with a call to an NP oracle.*

$\mathcal{H}$	$\mathsf{SAT}(\mathcal{H})$	$\mathbf{p}$	AXp(1), CXp(0)?	$\mathcal{S}$	AXp	CXp	Block
$\emptyset$	1	$(1,1,1,1)$	1	$\{1,2,3,4\}$	$\{4\}$	—	$b_1 = (\neg p_4)$
$\{b_1\}$	1	$(1,1,1,0)$	1	$\{1,2,3\}$	$\{2,3\}$	—	$b_2 = (\neg p_2 \vee \neg p_3)$
$\{b_1, b_2\}$	1	$(1,0,1,0)$	0	$\{1,3\}$	—	$\{2,4\}$	$b_3 = (p_2 \vee p_4)$
$\{b_1, b_2, b_3\}$	1	$(1,1,0,0)$	0	$\{1,2\}$	—	$\{3,4\}$	$b_4 = (p_3 \vee p_4)$
$\{b_1, b_2, b_3, b_4\}$	0	—	—	—	—	—	—

Table 1: Example of AXp/CXp enumeration, using Algorithm 3

Proposition 4. *For a binding and non-overlapping GDF, such that each classification function is represented by a sentence of a propositional language satisfying the query* **CO** *and the transformation* **CD***, then one AXp or one CXp can be computed in polynomial time. Furthermore, enumeration of AXps/CXps can be achieved with one call to an NP oracle per computed explanation.*

Proof sketch. For computing one AXp of class c_p, one can iteratively check consistency on the remaining literals of the other functions $q \neq p$. Conditioning is used to reflect, in the classifiers, the choices made, i.e. which literals are included or not in the AXp. For a CXp a similar approach can be used. For enumeration, we can once again exploit a MARCO-like algorithm. □

Corollary 3. *For a binding non-overlapping GDF, where each κ_j is represented by a* DNNF*, one AXp and one CXp can be computed in polynomial time. Furthermore, enumeration of AXps/CXps can be achieved with one call to an NP oracle per computed explanation.*

Thus, for GDFs that are both binding and non-overlapping, even if each function is represented by the fairly succinct DNNF, one can still compute AXps and CXps efficiently. As clarified by Proposition 4, **VA** is unnecessary to find an AXp/CXp; for any GDF implemented with propositional languages satisfying the query **CO** and the transformation **CD**, an AXp/CXp can be computed in polynomial time. In addition, a MARCO-like algorithm (Liffiton et al. 2016) can be used for enumerating AXps and CXps. The results above can be generalized to the case of multi-valued classification, where binarization (one-hot-encoding) can serve for representing multi-valued (non-continuous) features.

Total congruent classifiers. We can build on the conditions for GDFs to devise relaxed conditions for poly-time explainability.

Definition 6 (Total Classifier)**.** *A classification function is total if for any point $\mathbf{v} \in \mathbb{F}$, there is a prediction $\kappa(\mathbf{v}) = c$, with $c \in \mathcal{K}$.*

Definition 7 (Congruent Classifier)**.** *A classifier is* congruent *if the tractability (i.e. whether or not solvable in polynomial time) of deciding the consistency of $\kappa(\mathbf{x}) = c$ is the same for any $c \in \mathcal{K}$.*

Similarly, we can define a total congruent proposition language, for which the query **CO** is satisfied iff deciding $\kappa(\mathbf{v}) = c$ is in polynomial time for any $c \in \mathcal{K}$. Thus, the same argument used for GDFs, can be used to prove that,

Proposition 5. *For a total congruent propositional language, which satisfies the operations of* **CO** *and* **CD***, one AXp and one CXp can be computed in polynomial time. Furthermore, enumeration of AXps/CXps can be achieved with one call to an NP oracle per computed explanation.*

5 Experimental Results

In this section, we present the experiments carried out to assess the practical effectiveness of the proposed approach. The assessment is performed on the computation of AXps and CXps for d-DNNFs and SDDs.

Experimental setup. The experiments consider a selection of 23 binary classification datasets (with binary and/or categorical features) that are publicly available and originate from the Penn Machine Learning Benchmarks (Olson et al. 2017) and the UCI Machine Learning Repository (UCI). To learn d-DNNFs (resp. SDDs), we first train Read-Once Decision Tree (RODT) models on the given datasets using Orange3 (Demšar et al. 2013) and then compile the obtained RODTs into d-DNNFs (resp. SDDs). Categorical features are binarized through one-hot-encoding and the resulting binarized features are grouped together. (A RODT is a *free* BDD (FBDD) whose underlying graph is a tree (Barceló et al. 2020; Wegener 2000), where FBDD is defined as a BDD that satisfies the *read-once property*: each variable is encountered at most once on each path from the root to a leaf node.) The compilation of RODTs to d-DNNFs can be easily done by direct mapping, since RODT is a special case of FBDDs, and FBDDs is a subset of d-DNNFs (Darwiche and Marquis 2002). To compile SDDs, we use the PySDD package[4], which is implemented in Python and Cython. PySDD wraps the well-known SDD package[5] which offers canonical SDDs[6]. Employing canonical SDDs allows performing consistency and validity checking in a constant time (If the canonical SDD is inconsistent (resp. valid) then it is a single node labeled with $\bot$ (resp. $\top$) (Darwiche 2011)), so in practice it may improve the efficiency of explaining SDD classifiers. Besides, all presented algorithms are implemented in

[4]https://github.com/wannesm/PySDD

[5]http://reasoning.cs.ucla.edu/sdd/

[6]Since PySDD offers canonical SDDs, the **CD** transformation is not implemented in worst-case polynomial time (Van den Broeck and Darwiche 2015). However, in practice, this was never an issue in our experiments.

Python. [7] In addition, PySAT toolkit (Ignatiev, Morgado, and Marques-Silva 2018) is used to instrument incremental SAT oracle calls to enumerate AXp/CXp. As baseline comparison, we also include in this evaluation an heuristic explainer Anchor (Ribeiro, Singh, and Guestrin 2018) to assess the runtime performances of our approach. Lastly, we run the experiments on a MacBook Pro with a 6-Core Intel Core i7 2.6 GHz processor with 16 GByte RAM, running macOS Big Sur.

Results. Table 2 summarizes the obtained results of explaining d-DNNFs and SDDs. (Note that, for each dataset, the compiled d-DNNF and SDD represent the same decision function of the learned RODT. Hence, the computed explanations are the same as well. The size of the d-DNNF is on average twice as large as the corresponding SDD. Also note that compilation time is not included in the runtimes shown in the table, since these are not directly related with the computation of explanations.) Performance-wise, the maximum runtime to enumerate XPs is less than 1.4 sec for all the d-DNNFs, and less than 0.5 sec for all the SDDs. On average, total enumeration of XPs takes at most 0.4 sec for d-DNNFs; and for SDDs at most 0.1 sec. Thus the overall cost of the SAT oracle calls performed by Algorithm 3 is negligible. Given the results, one can conclude that the SAT calls do not constitute a bottleneck for enumerating the AXps/CXps of the classifiers represented as d-DNNFs or SDDs. However, in settings where the total number of explanations is much larger (i.e. exponentially large on the number of features), the cost of SAT calls could become dominant. Apart from the runtime, one observation is that the total number of AXps and CXps per instance is relatively small. Moreover, if compared with the number of features, the average length of an AXp/CXp is also relatively small. Despite that runtimes reported in Table 2 are small for AXp and CXp, one might not argue that the explanation problems studied in this paper are fairly easy. In fact, as can be observed Anchor's runtimes can exceed the running times of the d-DNNF non-heuristic explainer by several orders of magnitude.

To conclude, for the concrete case of classifiers that can be represented efficiently using d-DNNF and SDD, the experimental results confirm that, if a classifier can be represented with a propositional language that implements polynomial-time **CO** and **VA** queries as well as the **CD** transformation, then the computation and enumeration of explanations is not only practical, but substantially more efficient than alternative heuristic approaches. Regarding the limitations of proposed approach, these are the same as for all compilation-based methods: the off-line compilation phase may theoretically be very expensive in time and space. This limitation has not prevented compilation being used in large-scale industrial applications.

6 Related Work

Although recent years have witnessed a growing interest in finding explanations of machine learning (ML) models (Lipton 2018; Guidotti et al. 2019; Weld and Bansal 2019; Monroe 2021), explanations have been studied from different perspectives and in different branches of AI at least since the 80s (Shanahan 1989; Falappa, Kern-Isberner, and Simari 2002; Pérez and Uzcátegui 2003), including more recently in constraint programming (Amilhastre, Fargier, and Marquis 2002; Bogaerts et al. 2020; Gamba, Bogaerts, and Guns 2021). In the case of ML models, non-heuristic explanations have been studied in recent years (Shih, Choi, and Darwiche 2018; Ignatiev, Narodytska, and Marques-Silva 2019a; Shih, Choi, and Darwiche 2019; Narodytska et al. 2019; Ignatiev, Narodytska, and Marques-Silva 2019b,c; Darwiche and Hirth 2020; Ignatiev et al. 2020a; Ignatiev 2020; Audemard, Koriche, and Marquis 2020; Marques-Silva et al. 2020; Barceló et al. 2020; Ignatiev et al. 2020b; Izza, Ignatiev, and Marques-Silva 2020; Wäldchen et al. 2021; Izza and Marques-Silva 2021; Malfa et al. 2021; Ignatiev and Marques-Silva 2021; Cooper and Marques-Silva 2021; Huang et al. 2021; Audemard et al. 2021; Marques-Silva and Ignatiev 2022; Ignatiev et al. 2022; Shrotri et al. 2022). Some of these earlier works studied explanations for classifiers represented with propositional languages, namely those covered by the knowledge compilation map (Shih, Choi, and Darwiche 2018, 2019; Darwiche and Hirth 2020; Audemard, Koriche, and Marquis 2020; Barceló et al. 2020; Huang et al. 2021; Audemard et al. 2021). However, results on the efficient computation of explanations for classifiers represented with propositional languages are scarce. For example, (Shih, Choi, and Darwiche 2018, 2019; Darwiche and Hirth 2020) propose compilation algorithms (which are worst-case exponential) to generate the PI-explanations from OBDDs. Concretely, a classifier is compiled into an OBDD, which is then compiled into an OBDD representing the PI-explanations of the original classifier. Moreover, (Audemard, Koriche, and Marquis 2020) proves that, in the context of multi-class classification, if a propositional language satisfies **CD**, **FO**, and **IM**, then one PI-explanation can be computed in polynomial time. Our results in Section 4 consider multi-class classification with multiple classifiers. (Barceló et al. 2020) studies the computation complexity of computing smallest PI-explanations. Explanation enumeration based on the MARCO algorithm (Liffiton et al. 2016) was investigated in recent work (e.g. (Marques-Silva et al. 2021)). The main difference in Algorithm 3 is the explicit use of transformation and queries from the knowledge compilation map. Perhaps more importantly, the computation of AXp's and CXp's for a classifier represented as a d-DNNF circuit is fairly orthogonal to earlier work on the computation of explanations for propagators operating on d-DNNF circuits (Gange and Stuckey 2012). Indeed, in the case of propagators, the d-DNNF encodes valid assignments to a constraint, and explanations are always computed against a valuation of 1 of the d-DNNF, i.e. the allowed assignments to the constraint.

7 Conclusions

This paper proves that for any classifier that can be represented with a d-DNNF, both one AXp and one CXp can be computed in polynomial time on the size of the d-DNNF.

[7] All the materials for replicating the experiments are available at https://github.com/XuanxiangHuang/Xddnnf-experiments

Dataset	**(#F**	**#S)**	**Model**			**XPs**	**AXp**			**CXp**			**d-DNNF**		**SDD**		**Anchor**
			%A	**#ND**	**#NS**	**avg**	**M**	**avg**	**%L**	**M**	**avg**	**%L**	**M**	**avg**	**M**	**avg**	**avg**
adult	(12	1411)	80	189	111	7	6	3	34	9	5	14	0.065	0.021	0.010	0.002	3.61
chess	(36	320)	96	37	24	4	3	1	8	5	3	4	0.016	0.005	0.003	0.001	1.87
compas	(11	617)	68	351	262	11	15	4	42	17	6	18	0.210	0.059	0.015	0.004	2.71
corral	(6	16)	100	35	12	4	2	1	34	4	2	22	0.002	0.001	0.000	0.000	0.41
german	(21	100)	71	319	762	14	30	6	36	21	9	9	1.356	0.280	0.203	0.048	56.86
kr_vs_kp	(36	320)	96	62	44	6	12	2	11	8	4	5	0.054	0.010	0.006	0.002	2.46
lending	(9	1020)	83	177	101	6	8	2	34	8	3	23	0.065	0.013	0.009	0.002	1.51
Mammographic	(13	96)	82	131	51	11	10	5	36	8	6	16	0.040	0.020	0.004	0.002	1.06
mofn_3_7_10	(10	132)	97	107	47	11	19	4	33	27	6	23	0.065	0.013	0.002	0.001	1.10
monk1	(6	56)	85	98	75	5	4	2	54	7	3	22	0.007	0.003	0.001	0.001	1.38
monk2	(6	17)	70	177	96	6	5	2	63	6	5	24	0.015	0.008	0.003	0.002	3.25
monk3	(6	55)	97	14	11	3	2	1	31	2	2	20	0.001	0.000	0.000	0.000	0.52
mux6	(6	13)	92	58	23	5	4	2	54	5	3	22	0.005	0.002	0.000	0.000	0.56
parity5+5	(10	112)	86	427	101	10	8	3	67	15	8	15	0.175	0.056	0.003	0.001	18.64
postoperative	(8	9)	66	95	74	8	5	3	58	8	6	21	0.010	0.006	0.004	0.002	1.17
primary-tumor	(15	34)	80	109	67	11	13	5	28	9	5	14	0.046	0.018	0.005	0.003	1.19
promoters	(58	11)	81	32	17	6	4	2	7	5	4	3	0.036	0.018	0.003	0.002	19.25
recidivism	(15	634)	63	714	569	25	57	12	51	31	12	15	1.231	0.334	0.441	0.094	33.69
spect	(22	27)	77	114	61	10	14	5	18	12	5	10	0.080	0.025	0.004	0.002	1.41
threeOf9	(9	51)	100	87	36	7	12	3	36	9	4	18	0.022	0.007	0.001	0.000	1.10
tic_tac_toe	(9	96)	92	174	82	9	8	3	49	9	5	19	0.036	0.016	0.006	0.002	4.39
vote	(16	43)	95	47	29	4	5	2	17	5	3	12	0.012	0.004	0.001	0.001	0.76
xd6	(9	97)	99	88	32	7	18	4	34	26	3	20	0.031	0.009	0.003	0.001	1.51

Table 2: Listing all AXps /CXps for d-DNNFs and SDDs. Columns **#F** and **#S** report, resp. the number of features and the number of tested samples (instances) in the dataset. (The number of tested samples **#S** represents 10% of the data, selected randomly.) Sub-Column **%A** reports the (test) accuracy of the model and **#ND** (resp. **#NS**) shows the total number of nodes in the compiled d-DNNF (resp. SDD). Column **XPs** reports the average number of total explanations (AXp's and CXp's). Sub-columns **M** and **avg** of column **AXp** (resp. **CXp**) show, resp., the maximum and average number of explanations. The average length (in % of **#F**) of an AXp/CXp is given as **%L**. Sub-columns **M** and **avg** of column **d-DNNF** (resp. **SDD**) report, resp., maximum and average runtime (in seconds) to list all XPs of all tested instances. Finally, the average runtimes to compute *Anchor* explanations for the d-DNNFs is shown in **Anchor**.

Furthermore, the paper shows that enumeration of AXps and CXps can be implemented with one NP oracle call per explanation. The experimental evidence confirms that for small numbers of explanations, the cost of enumeration is negligible. In addition, the paper proposes conditions for generalized decision functions to be explained in polynomial time. Concretely, the paper develops conditions which allow generalized decision functions represented with DNNFs to be explainable in polynomial time. Finally, the paper proposes general conditions for a classifier to be explained in polynomial time. The experimental results validate the scability of the polynomial time algorithms and, more importantly, the scalability of oracle-based enumeration.

Acknowledgments

We are grateful to the anonymous referees for their constructive input. 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

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs Application to configuration. *Artif. Intell.*, 135(1-2): 199–234.

Anjomshoae, S.; Najjar, A.; Calvaresi, D.; and Främling, K. 2019. Explainable Agents and Robots: Results from a Systematic Literature Review. In *AAMAS*, 1078–1088.

Audemard, G.; Bellart, S.; Bounia, L.; Koriche, F.; Lagniez, J.; and Marquis, P. 2021. On the Computational Intelligibility of Boolean Classifiers. In *KR*, 74–86.

Audemard, G.; Koriche, F.; and Marquis, P. 2020. On Tractable XAI Queries based on Compiled Representations. In *KR*, 838–849.

Bakker, R. R.; Dikker, F.; Tempelman, F.; and Wognum, P. M. 1993. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. In *IJCAI*, 276–281.

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

Bogaerts, B.; Gamba, E.; Claes, J.; and Guns, T. 2020. Step-Wise Explanations of Constraint Satisfaction Problems. In *ECAI*, 640–647.

Bova, S. 2016. SDDs Are Exponentially More Succinct than OBDDs. In *AAAI*, 929–935.

Chinneck, J. W.; and Dravnieks, E. W. 1991. Locating Minimal Infeasible Constraint Sets in Linear Programs. *INFORMS J. Comput.*, 3(2): 157–168.

Cooper, M. C.; and Marques-Silva, J. 2021. On the Tractability of Explaining Decisions of Classifiers. In Michel, L. D., ed., *CP*, 21:1–21:18.

Darwiche, A. 2001. On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision. *J. Appl. Non Class. Logics*, 11(1-2): 11–34.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI*, 819–826.

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

Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17: 229–264.

Demšar, J.; Curk, T.; Erjavec, A.; Črt Gorup; Hočevar, T.; Milutinovič, M.; Možina, M.; Polajnar, M.; Toplak, M.; Starič, A.; Štajdohar, M.; Umek, L.; Žagar, L.; Žbontar, J.; Žitnik, M.; and Zupan, B. 2013. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 14: 2349–2353.

Falappa, M. A.; Kern-Isberner, G.; and Simari, G. R. 2002. Explanations, belief revision and defeasible reasoning. *Artif. Intell.*, 141(1/2): 1–28.

Fargier, H.; and Marquis, P. 2008. Extending the Knowledge Compilation Map: Krom, Horn, Affine and Beyond. In *AAAI*, 442–447.

Gamba, E.; Bogaerts, B.; and Guns, T. 2021. Efficiently Explaining CSPs with Unsatisfiable Subset Optimization. In *IJCAI*, 1381–1388.

Gange, G.; and Stuckey, P. J. 2012. Explaining Propagators for s-DNNF Circuits. In *CPAIOR*, 195–210.

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, J.; and Darwiche, A. 2007. The Language of Search. *J. Artif. Intell. Res.*, 29: 191–219.

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

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

Ignatiev, A.; Cooper, M. C.; Siala, M.; Hebrard, E.; and Marques-Silva, J. 2020a. Towards Formal Fairness in Machine Learning. In *CP*, 846–867.

Ignatiev, A.; Izza, Y.; Stuckey, P.; and Marques-Silva, J. 2022. Using MaxSAT for Efficient Explanations of Tree Ensembles. In *AAAI*.

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

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

Ignatiev, A.; Narodytska, N.; Asher, N.; and Marques-Silva, J. 2020b. From Contrastive to Abductive Explanations and Back Again. In *AIxIA*, 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.

Ignatiev, A.; Pereira, F.; Narodytska, N.; and Marques-Silva, J. 2018. A SAT-Based Approach to Learn Explainable Decision Sets. In *IJCAR*, 627–645.

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.

Koriche, F.; Lagniez, J.; Marquis, P.; and Thomas, S. 2013. Knowledge Compilation for Model Counting: Affine Decision Trees. In *IJCAI*, 947–953.

Lakkaraju, H.; Bach, S. H.; and Leskovec, J. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *KDD*, 1675–1684.

Liffiton, M. H.; Previti, A.; Malik, A.; and Marques-Silva, J. 2016. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2): 223–250.

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 *NeurIPS*, 4765–4774.

Malfa, E. L.; Zbrzezny, A.; Michelmore, R.; Paoletti, N.; and Kwiatkowska, M. 2021. On Guaranteed Optimal Robust Explanations for NLP Models. In *IJCAI*, 2658–2665.

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.

Marques-Silva, J.; Janota, M.; and Mencía, C. 2017. Minimal sets on propositional formulae. Problems and reductions. *Artif. Intell.*, 252: 22–50.

Miller, T. 2019a. "But why?" Understanding explainable artificial intelligence. *ACM Crossroads*, 25(3): 20–25.

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

Mittelstadt, B. D.; Russell, C.; and Wachter, S. 2019. Explaining Explanations in AI. In *FAT*, 279–288.

Monroe, D. 2021. Deceiving AI. *Commun. ACM*, 64.

Mueller, S. T.; Hoffman, R. R.; Clancey, W. J.; Emrey, A.; and Klein, G. 2019. Explanation in Human-AI Systems: A Literature Meta-Review, Synopsis of Key Ideas and Publications, and Bibliography for Explainable AI. *CoRR*, abs/1902.01876.

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.

Niveau, A.; Fargier, H.; and Pralet, C. 2011. Representing CSPs with Set-Labeled Diagrams: A Compilation Map. In *GKR*, 137–171.

Olson, R. S.; La Cava, W.; Orzechowski, P.; Urbanowicz, R. J.; and Moore, J. H. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1): 36.

Pérez, R. P.; and Uzcátegui, C. 2003. Preferences and explanations. *Artif. Intell.*, 149(1): 1–30.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1): 57–95.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *KDD*, 1135–1144.

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

Samek, W.; Montavon, G.; Vedaldi, A.; Hansen, L. K.; and Müller, K., eds. 2019. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer. ISBN 978-3-030-28953-9.

Samek, W.; and Müller, K. 2019. Towards Explainable Artificial Intelligence. In (Samek et al. 2019), 5–22.

Shanahan, M. 1989. Prediction is Deduction but Explanation is Abduction. In *IJCAI*, 1055–1060.

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

Shih, A.; Choi, A.; and Darwiche, A. 2019. Compiling Bayesian Network Classifiers into Decision Graphs. In *AAAI*, 7966–7974.

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*.

UCI. 2020. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml.

Valiant, L. G. 1984. A Theory of the Learnable. *Commun. ACM*, 27(11): 1134–1142.

Van den Broeck, G.; and Darwiche, A. 2015. On the Role of Canonicity in Knowledge Compilation. In *AAAI*, 1641–1648.

Wäldchen, S.; MacDonald, J.; Hauch, S.; and Kutyniok, G. 2021. The Computational Complexity of Understanding Binary Classifier Decisions. *J. Artif. Intell. Res.*, 70: 351–387.

Wegener, I. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM. ISBN 0-89871-458-3.

Weld, D. S.; and Bansal, G. 2019. The challenge of crafting intelligible intelligence. *Commun. ACM*, 62(6): 70–79.

Xu, F.; Uszkoreit, H.; Du, Y.; Fan, W.; Zhao, D.; and Zhu, J. 2019. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. In *NLPCC*, 563–574.