

The Tractability of SHAP-Score-Based Explanations over Deterministic and Decomposable Boolean Circuits

Marcelo Arenas^{1,2,3}, Pablo Barceló^{2,3}, Leopoldo Bertossi^{4,3}, Mikaël Monet⁵

¹ Department of Computer Science, Universidad Católica de Chile

² Institute for Mathematical and Computational Engineering, Universidad Católica de Chile

³ IMFD Chile

⁴ Universidad Adolfo Ibáñez, FIC, Chile

⁵ Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRISTAL, F-59000 Lille, France
marenas@ing.puc.cl, pbarcelo@uc.cl, leopoldo.bertossi@uai.cl, mikaël.monet@inria.fr

Abstract

Scores based on Shapley values are widely used for providing explanations to classification results over machine learning models. A prime example of this is the influential SHAP-score, a version of the Shapley value that can help explain the result of a learned model on a specific entity by assigning a score to every feature. While in general computing Shapley values is a computationally intractable problem, it has recently been claimed that the SHAP-score can be computed in polynomial time over the class of decision trees. In this paper, we provide a proof of a stronger result over Boolean models: the SHAP-score can be computed in polynomial time over *deterministic and decomposable Boolean circuits*. Such circuits, also known as *tractable Boolean circuits*, generalize a wide range of Boolean circuits and binary decision diagrams classes, including binary decision trees, Ordered Binary Decision Diagrams (OBDDs) and Free Binary Decision Diagrams (FBDDs). We also establish the computational limits of the notion of SHAP-score by observing that, under a mild condition, computing it over a class of Boolean models is always polynomially as hard as the model counting problem for that class. This implies that both determinism and decomposability are essential properties for the circuits that we consider, as removing one or the other renders the problem of computing the SHAP-score intractable (namely, #P-hard).

1 Introduction

Explainable artificial intelligence has become an active area of research. Central to it is the observation that artificial intelligence (AI) and machine learning (ML) models cannot always be blindly applied without being able to interpret and explain their results. For example, when someone applies for a loan and sees their application rejected by an algorithmic decision-making system, the system should be able to provide an explanation for that decision. Explanations can be *global* – focusing on the general input/output relation of the model –, or *local* – focusing on how features affect the decision of the model for a specific input. Recent literature has strengthened the importance of the latter by showing their ability to provide explanations that are often overlooked by global explanations (Molnar 2020).

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

One natural way of providing local explanations for classification models consists in assigning *numerical scores* to the feature values of an entity that has gone through the classification process. Intuitively, the higher the score of a feature value, the more relevant it should be considered. It is in this context that the SHAP-score has been introduced (Lundberg and Lee 2017; Lundberg et al. 2020). This recent notion has rapidly gained attention and is becoming influential. There are two properties of the SHAP-score that support its rapid adoption. First, its definition is quite general and can be applied to any kind of classification model. Second, the definition of the SHAP-score is grounded on the well-known *Shapley value* (Shapley 1953; Roth 1988), that has already been used successfully in several domains of computer science; see, e.g., (Hunter and Konieczny 2010; Livshits et al. 2020; Michalak et al. 2013; Cesari et al. 2018). Thus, SHAP-scores have a clear, intuitive, combinatorial meaning, and inherit all the desirable properties of the Shapley value.

For a given classifier M , entity e and feature x , the SHAP-score $\text{SHAP}(M, e, x)$ intuitively represents the importance of the feature value $e(x)$ to the classification result $M(e)$. In its general formulation, $\text{SHAP}(M, e, x)$ is a weighted average of differences of expected values of the outcomes (c.f. Section 2 for its formal definition). Unfortunately, computing quantities that are based on the notion of Shapley value is in general intractable. Indeed, in many scenarios the computation turns out to be #P-hard (Faigle and Kern 1992; Deng and Papadimitriou 1994; Livshits et al. 2020; Bertossi et al. 2020), which makes the notion difficult to use – if not impossible – for practical purposes (Arora and Barak 2009). Therefore, a natural question is: For what kinds of classification models the computation of the SHAP-score can be done efficiently? This is the subject of this paper.

In this work, we focus on classifiers working with *binary feature values* (i.e., propositional features that can take the values 0 or 1), and that return 1 (accept) or 0 (reject) for each entity. We will call these *Boolean classifiers*. The second assumption that we make is that the underlying probability distribution on the population of entities is what we call a *product distribution*, where each binary feature x has a probability $p(x)$ of being equal to 1, independently of the other features. We note here that the restriction to binary in-

puts can be relevant in many practical scenarios where the features are of a propositional nature.

More specifically, we investigate Boolean classifiers defined as *deterministic and decomposable Boolean circuits*, a widely studied model in *knowledge compilation* (Darwiche 2001; Darwiche and Marquis 2002). Such circuits encompass a wide range of Boolean models and binary decision diagrams classes that are considered in knowledge compilation, and in AI more generally. For instance, they generalize *binary decision trees*, *ordered binary decision diagrams* (OBDDs), *free binary decision diagrams* (FBDDs), and *deterministic and decomposable negation normal forms* (d-DNNFs) (Darwiche 2001; Amarilli et al. 2020; Darwiche and Hirth 2020). These circuits are also known under the name of *tractable Boolean circuits*, that is used in recent literature (Shih, Darwiche, and Choi 2019; Shi et al. 2020; Shih, Choi, and Darwiche 2018b,a; Shih et al. 2019; Peharz et al. 2020). We provide an example of a deterministic and decomposable Boolean circuit next (and give the formal definition in Section 2).

Example 1.1. We want to classify papers submitted to a conference as rejected (Boolean value 0) or accepted (Boolean value 1). Papers are described by features **fg**, **dtr**, **nf** and **na**, which stand for “follows guidelines”, “deep theoretical result”, “new framework” and “nice applications”, respectively. The Boolean classifier for the papers is given by the Boolean circuit in Figure 1. The input of this circuit are the features **fg**, **dtr**, **nf** and **na**, each of which can take value either 0 or 1, depending on whether the feature is present (1) or absent (0). The nodes with labels \neg , \vee or \wedge are logic gates, and the associated Boolean value of each one of them depends on the logical connective represented by its label and the Boolean values of its inputs. The output value of the circuit is given by the top node in the figure.

The Boolean circuit in Figure 1 is said to be *decomposable*, because for each \wedge -gate, the sets of features of its inputs are pairwise disjoint. For instance, in the case of the top node in Figure 1, the left-hand side input has $\{\mathbf{fg}\}$ as its set of features, while its right-hand side input has $\{\mathbf{dtr}, \mathbf{nf}, \mathbf{na}\}$ as its set of features, which are disjoint. Also, this circuit is said to be *deterministic*, which means that for every \vee -gate, two (or more) of its inputs cannot be given value 1 by the same Boolean assignment for the features. For instance, in the case of the only \vee -gate in Figure 1, if a Boolean assignment for the features gives value 1 to its left-hand side input, then feature **dtr** has to be given value 1 and, thus, such an assignment gives value 0 to the right-hand side input of the \vee -gate. In the same way, it can be shown that if a Boolean assignment for the features gives value 1 to the right-hand side input of this \vee -gate, then it gives value 0 to its left-hand side input. \square

Readers who are not familiar with knowledge compilation can simply think about deterministic and decomposable circuits as a tool for establishing in a uniform manner the tractability of computing SHAP-scores on several Boolean classifier classes. Our main contributions are the following:

1. We provide a polynomial time algorithm that computes the SHAP-score for deterministic and decompos-

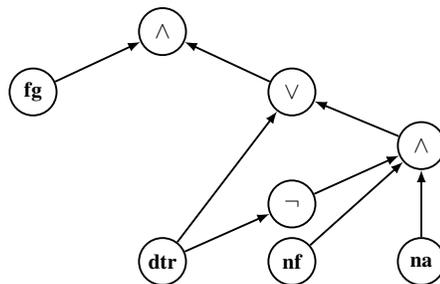


Figure 1: A deterministic and decomposable Boolean Circuit as a classifier.

able Boolean circuits, in the special case of *uniform probability distributions* (that is, when each $p(x)$ is $\frac{1}{2}$). In particular, this provides a precise proof of the claim made in (Lundberg et al. 2020) that the SHAP-score for Boolean classifiers given as decision trees can be computed in polynomial time. Moreover, we also obtain as a corollary that the SHAP-score for Boolean classifiers given as OBDDs, FBDDs and d-DNNFs can be computed in polynomial time.

2. We observe that computing the SHAP-score on Boolean circuits in a class is always polynomially as hard as the *model counting* problem for that class (under a mild condition). By using this observation, we obtain that each one of the *determinism* assumption and the *decomposability* assumption is necessary for tractability.
3. Last, we show that the results above (and most interestingly, the polynomial-time algorithm) can be extended to the SHAP-score defined on product distributions for the entity population.

Our contributions should be compared to the results obtained in the contemporaneous paper (Van den Broeck et al. 2020). There, the authors establish the following theorem: for every class \mathcal{C} of classifiers and under product distributions, the problem of computing the SHAP-score for \mathcal{C} is polynomial-time equivalent to the problem of computing the expected value for the models in \mathcal{C} . Since computing expectations is in polynomial time for tractable Boolean circuits, this in particular implies that computing the SHAP-score is in polynomial time for the circuits that we consider; in other words, their results capture ours. However, there is a fundamental difference in the approach taken to show tractability: their reduction uses multiple oracle calls to the problem of computing expectations, whereas we provide a more direct algorithm to compute the SHAP-score on these circuits.

Our algorithm for computing the SHAP-score could be used in practical scenarios. Indeed, recently, some classes of classifiers have been compiled into tractable Boolean circuits. This is the case, for instance, of Bayesian Classifiers (Shih, Choi, and Darwiche 2018a), Binary Neural Networks (Shi et al. 2020), and Random Forests (Choi et al. 2020). The idea is to start with a Boolean classifier M given in a formalism that is hard to interpret – for instance a Binary neural network – and to compute a tractable Boolean

circuit M' that is equivalent to M (this computation can be expensive). One can then use M' and the nice properties of tractable Boolean circuits to interpret the decisions of the model. Hence, this makes it possible to apply the results in this paper on the SHAP-score to those classes of classifiers.

Paper structure. We give preliminaries in Section 2. In Section 3, we prove that the SHAP-score can be computed in polynomial time for deterministic and decomposable Boolean circuits for uniform probability distributions. In Section 4 we establish the limits of the tractable computation of the SHAP-score. Next we show in Section 5 that our results extend to the setting where we consider product distributions. We conclude and discuss future work in Section 6.

2 Preliminaries

2.1 Entities, Distributions and Classifiers

Let X be a finite set of *features*, also called *variables*. An *entity* over X is a function $\mathbf{e} : X \rightarrow \{0, 1\}$. We denote by $\text{ent}(X)$ the set of all entities over X . On this set, we consider the *uniform probability distribution*, i.e., for an event $E \subseteq \text{ent}(X)$, we have that $P(E) := \frac{|E|}{2^{|X|}}$. We will come back to this assumption in Section 5, where we will consider the more general product distributions (we start with the uniform distribution to ease the presentation).

A *Boolean classifier* M over X is a function $M : \text{ent}(X) \rightarrow \{0, 1\}$ that maps every entity over X to 0 or 1. We say that M *accepts* an entity \mathbf{e} when $M(\mathbf{e}) = 1$, and that it *rejects* it if $M(\mathbf{e}) = 0$. Since we consider $\text{ent}(X)$ to be a probability space, M can be regarded as a random variable.

2.2 The SHAP-Score over Boolean Classifiers

Let $M : \text{ent}(X) \rightarrow \{0, 1\}$ be a Boolean classifier over the set X of features. Given an entity \mathbf{e} over X and a subset $S \subseteq X$ of features, the set $\text{cw}(\mathbf{e}, S) := \{\mathbf{e}' \in \text{ent}(X) \mid \mathbf{e}'(x) = \mathbf{e}(x) \text{ for each } x \in S\}$ contains those entities that coincide with \mathbf{e} over each feature in S . In other words, $\text{cw}(\mathbf{e}, S)$ is the set of entities that are *consistent with* \mathbf{e} on S . Then, given an entity $\mathbf{e} \in \text{ent}(X)$ and $S \subseteq X$, we define the *expected value of M over $X \setminus S$ with respect to \mathbf{e}* as

$$\phi(M, \mathbf{e}, S) := \mathbb{E}[M(\mathbf{e}') \mid \mathbf{e}' \in \text{cw}(\mathbf{e}, S)].$$

Since we consider the uniform distribution over $\text{ent}(X)$, we have that

$$\phi(M, \mathbf{e}, S) = \sum_{\mathbf{e}' \in \text{cw}(\mathbf{e}, S)} \frac{1}{2^{|X \setminus S|}} M(\mathbf{e}').$$

Intuitively, $\phi(M, \mathbf{e}, S)$ is the probability that $M(\mathbf{e}') = 1$, conditioned on the inputs $\mathbf{e}' \in \text{ent}(X)$ to coincide with \mathbf{e} over each feature in S . This function is then used in the general formula of the Shapley value (Shapley 1953; Roth 1988) to obtain the SHAP-score for feature values in \mathbf{e} .

Definition 2.1. *Given a Boolean classifier M over a set of features X , an entity \mathbf{e} over X , and a feature $x \in X$, the SHAP score of feature x on \mathbf{e} with respect to M is defined as*

$$\text{SHAP}(M, \mathbf{e}, x) := \sum_{S \subseteq X \setminus \{x\}} \frac{|S|!(|X| - |S| - 1)!}{|X|!} \left(\phi(M, \mathbf{e}, S \cup \{x\}) - \phi(M, \mathbf{e}, S) \right). \quad (1)$$

Thus, $\text{SHAP}(M, \mathbf{e}, x)$ is a weighted average of the contribution of feature x on \mathbf{e} to the classification result, i.e., of the differences between having it and not, under all possible permutations of the other feature values. Observe that, from this definition, a high positive value of $\text{SHAP}(M, \mathbf{e}, x)$ intuitively means that setting x to $\mathbf{e}(x)$ strongly leans the classifier towards acceptance, while a high negative value of $\text{SHAP}(M, \mathbf{e}, x)$ means that setting x to $\mathbf{e}(x)$ strongly leans the classifier towards rejection.

2.3 Deterministic and Decomposable Boolean Circuits

A Boolean circuit over a set of variables X is a directed acyclic graph C such that

- (i) Every node without incoming edges is either a *variable gate* or a *constant gate*. A variable gate is labeled with a variable from X , and a constant gate is labeled with either 0 or 1;
- (ii) Every node with incoming edges is a *logic gate*, and is labeled with a symbol \wedge , \vee or \neg . If it is labeled with the symbol \neg , then it has exactly one incoming edge;¹
- (iii) Exactly one node does not have any outgoing edges, and this node is called the *output gate of C* .

Such a Boolean circuit C represents a Boolean classifier in the expected way – we assume the reader to be familiar with Boolean logic \neg , and we write $C(\mathbf{e})$ for the value in $\{0, 1\}$ of the output gate of C when we evaluate C over the entity \mathbf{e} .

Several restrictions of Boolean circuits with good computational properties have been studied. Let C be a Boolean circuit over a set of variables X and g a gate of C . The Boolean circuit C_g over X is defined by considering the sub-graph of C induced by the set of gates g' in C for which there exists a path from g' to g in C . Notice that g is the output gate of C_g . The set $\text{var}(g)$ is defined as the set of variables $x \in X$ such that there exists a variable gate with label x in C_g . Then, an \vee -gate g of C is said to be *deterministic* if for every pair g_1, g_2 of distinct input gates of g , the Boolean circuits C_{g_1} and C_{g_2} are disjoint in the sense that there is no entity \mathbf{e} that is accepted by both C_{g_1} and C_{g_2} (that is, there is no entity $\mathbf{e} \in \text{ent}(X)$ such that $C_{g_1}(\mathbf{e}) = C_{g_2}(\mathbf{e}) = 1$). The circuit C is called *deterministic* if every \vee -gate of C is deterministic. An \wedge -gate g of C is said to be *decomposable* if for every pair g_1, g_2 of distinct input gates of g , we have that $\text{var}(g_1) \cap \text{var}(g_2) = \emptyset$. Then, C is called *decomposable* if every \wedge -gate of C is decomposable.

Example 2.2. In Example 1.1, we explained at an intuitive level why the Boolean circuit in Figure 1 is deterministic and decomposable. By using the terminology defined in

¹Recall that the fan-in of a gate is the number of its input gates. In our definition of Boolean circuits, we allow unbounded fan-in \wedge - and \vee -gates.

the previous paragraph, it can be formally checked that this Boolean circuit indeed satisfies these conditions. \square

As mentioned before, deterministic and decomposable Boolean circuits generalize many decision diagrams and Boolean circuits classes. We refer to (Darwiche 2001; Amarilli et al. 2020) for detailed studies of knowledge compilation classes and of their precise relationships. For the reader’s convenience, we explain in the supplementary material in <https://arxiv.org/pdf/2007.14045.pdf> how FBDDs and binary decision trees can be encoded in linear time as deterministic and decomposable Boolean circuits.

3 Tractable Computation of the SHAP-Score

In this section, we prove our first tractability result, namely, that computing the SHAP-score for Boolean classifiers given as deterministic and decomposable Boolean circuits can be done in polynomial time, for uniform probability distributions. Formally:

Theorem 3.1. *The following problem can be solved in polynomial time. Given as input a deterministic and decomposable Boolean circuit C over a set of features X , an entity $e : X \rightarrow \{0, 1\}$, and a feature $x \in X$, compute the value $\text{SHAP}(C, e, x)$.*

In particular, since binary decision trees, OBDDs, FBDDs and d-DNNFs are all restricted kinds of deterministic and decomposable circuits, we obtain as a consequence of Theorem 3.1 that this problem is also in polynomial time for these classes. For instance, for binary decision trees we obtain:

Corollary 3.2. *The following problem can be solved in polynomial time. Given as input a binary decision tree T over a set of features X , an entity $e : X \rightarrow \{0, 1\}$, and a feature $x \in X$, compute the value $\text{SHAP}(T, e, x)$.*

The authors of (Lundberg et al. 2020) give a proof of this result, but, unfortunately, with few details to fully understand it. Moreover, it is important to notice that Theorem 3.1 is a nontrivial extension of the result for decision trees, as it is known that deterministic and decomposable circuits can be exponentially more succinct than binary decision trees (in fact, than FBDDs) at representing Boolean classifiers (Darwiche 2001; Amarilli et al. 2020).

In order to prove Theorem 3.1, we need to introduce some notation. Let M be a Boolean classifier over a set of features X . We write $\text{SAT}(M) \subseteq \text{ent}(X)$ for the set of entities that are accepted by M , and $\#\text{SAT}(M)$ for the cardinality of this set. Let $e, e' \in \text{ent}(X)$ be a pair of entities over X . We define $\text{sim}(e, e') := \{x \in X \mid e(x) = e'(x)\}$ to be the set of features on which e and e' coincide. Given a Boolean classifier M over X , an entity $e \in \text{ent}(X)$ and a natural number $k \leq |X|$, we define the set $\text{SAT}(M, e, k) := \text{SAT}(M) \cap \{e' \in \text{ent}(X) \mid |\text{sim}(e, e')| = k\}$, in other words, the set of entities e' that are accepted by M and which coincide with e in exactly k features. Naturally, we write $\#\text{SAT}(M, e, k)$ for the size of $\text{SAT}(M, e, k)$.

Example 3.3. Let M be the Boolean classifier represented by the circuit in Example 1.1. Then $\text{SAT}(M)$ is the set

containing all papers that are accepted according to M , so that $\#\text{SAT}(M) = 5$. Now, consider the entity e such that $e(\mathbf{fg}) = 1$, $e(\mathbf{dtr}) = 1$, $e(\mathbf{nf}) = 0$ and $e(\mathbf{na}) = 1$. Then one can check that $\#\text{SAT}(M, e, 0) = 0$, $\#\text{SAT}(M, e, 1) = 0$, $\#\text{SAT}(M, e, 2) = 2$, $\#\text{SAT}(M, e, 3) = 2$ and $\#\text{SAT}(M, e, 4) = 1$. \square

Our proof of Theorem 3.1 is technical and is divided into two modular parts. The first part, which is developed in Section 3.1, consists in showing that the problem of computing $\text{SHAP}(\cdot, \cdot, \cdot)$ can be reduced in polynomial time to that of computing $\#\text{SAT}(\cdot, \cdot, \cdot)$. This part of the proof is a sequence of formula manipulations, and it only uses the fact that deterministic and decomposable circuits can be efficiently *conditioned* on a variable value (to be defined in Section 3.1). In the second part of the proof, which is developed in Section 3.2, we show that computing $\#\text{SAT}(\cdot, \cdot, \cdot)$ can be done in polynomial time for deterministic and decomposable Boolean circuits. It is in this part that the properties of deterministic and decomposable circuits are really used.

3.1 Reducing $\text{SHAP}(\cdot, \cdot, \cdot)$ to $\#\text{SAT}(\cdot, \cdot, \cdot)$

In this section, we show that for deterministic and decomposable Boolean circuits, the computation of the SHAP-score can be reduced in polynomial time to the computation of $\#\text{SAT}(\cdot, \cdot, \cdot)$. To achieve this, we will need two more definitions. Let M be a Boolean classifier over a set of features X and $x \in X$, and let Boolean classifiers $M_{+x} : \text{ent}(X \setminus \{x\}) \rightarrow \{0, 1\}$ and $M_{-x} : \text{ent}(X \setminus \{x\}) \rightarrow \{0, 1\}$ be defined as follows. For $e \in \text{ent}(X \setminus \{x\})$, we write e_{+x} and e_{-x} the entities over X such that $e_{+x}(x) = 1$, $e_{-x}(x) = 0$ and $e_{+x}(y) = e_{-x}(y) = e(y)$ for every $y \in X \setminus \{x\}$. Then define $M_{+x}(e) := M(e_{+x})$ and $M_{-x}(e) := M(e_{-x})$. In the literature, M_{+x} (resp., M_{-x}) is called the *conditioning by x* (resp., *by $\neg x$*) of M . Conditioning can be done in linear time for a Boolean circuit C by replacing every gate with label x by a constant gate with label 1 (resp., 0). We write C_{+x} (resp., C_{-x}) for the Boolean circuit obtained via this transformation. One can easily check that, if C is deterministic and decomposable, then C_{+x} and C_{-x} are deterministic and decomposable as well.

We now introduce the second definition needed for the proof. For a Boolean classifier M over a set of variables X , an entity $e \in \text{ent}(X)$ and an integer $k \leq |X|$, we define

$$H(M, e, k) := \sum_{\substack{S \subseteq X \\ |S|=k}} \sum_{e' \in \text{cw}(e, S)} M(e'). \quad (2)$$

We first explain how computing $\text{SHAP}(\cdot, \cdot, \cdot)$ can be reduced in polynomial time to the problem of computing $H(\cdot, \cdot, \cdot)$, and then how computing $H(\cdot, \cdot, \cdot)$ can be reduced in polynomial time to computing $\#\text{SAT}(\cdot, \cdot, \cdot)$.

Reducing from $\text{SHAP}(\cdot, \cdot, \cdot)$ to $H(\cdot, \cdot, \cdot)$. We need to compute $\text{SHAP}(C, e, x)$, for a given deterministic and decomposable circuit C over a set of variables X , entity $e \in \text{ent}(X)$, and feature $x \in X$. Let $n = |X|$, and define

$$\text{Diff}_k(C, e, x) := \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} (\phi(C, e, S \cup \{x\}) - \phi(C, e, S)).$$

Then by the definition of the SHAP-score in (1), we have:

$$\text{SHAP}(C, \mathbf{e}, x) = \sum_{k=0}^{n-1} \frac{k!(n-k-1)!}{n!} \text{Diff}_k(C, \mathbf{e}, x).$$

Observe that all arithmetical terms (such as $k!$ or $n!$) can be computed in polynomial time: this is simply because n is given in unary, as it is bounded by the size of the circuit. Therefore, it is enough to show how to compute in polynomial time the quantities $\text{Diff}_k(C, \mathbf{e}, x)$ for each $k \in \{0, \dots, n-1\}$, as $n = |X|$ is bounded by the size of the input (C, \mathbf{e}, x) . By definition of $\phi(\cdot, \cdot, \cdot)$, we have that $\text{Diff}_k(C, \mathbf{e}, x) = \alpha - \beta$, where:

$$\begin{aligned} \alpha &= \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \frac{1}{2^{n-(k+1)}} \sum_{\mathbf{e}' \in \text{cw}(\mathbf{e}, S \cup \{x\})} C(\mathbf{e}') \\ \beta &= \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \frac{1}{2^{n-k}} \sum_{\mathbf{e}' \in \text{cw}(\mathbf{e}, S)} C(\mathbf{e}'). \end{aligned}$$

Next we show how the computation of α and β can be reduced in polynomial-time to the computation of $H(\cdot, \cdot, \cdot)$. For an entity $\mathbf{e} \in \text{ent}(X)$ and $S \subseteq X$, let $\mathbf{e}|_S$ be the entity over S that is obtained by restricting \mathbf{e} to the domain S (that is, formally $\mathbf{e}|_S \in \text{ent}(S)$ and $\mathbf{e}|_S(y) := \mathbf{e}(y)$ for every $y \in S$). Then, starting with β , we have that:

$$\begin{aligned} \beta &= \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \frac{1}{2^{n-k}} \sum_{\mathbf{e}' \in \text{cw}(\mathbf{e}, S)} C(\mathbf{e}') \\ &= \left[\sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \frac{1}{2^{n-k}} \sum_{\substack{\mathbf{e}' \in \text{cw}(\mathbf{e}, S) \\ \mathbf{e}'(x)=1}} C(\mathbf{e}') \right] \\ &\quad + \left[\sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \frac{1}{2^{n-k}} \sum_{\substack{\mathbf{e}' \in \text{cw}(\mathbf{e}, S) \\ \mathbf{e}'(x)=0}} C(\mathbf{e}') \right] \\ &= \left[\frac{1}{2^{n-k}} \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \sum_{\mathbf{e}'' \in \text{cw}(\mathbf{e}|_{X \setminus \{x\}}, S)} C_{+x}(\mathbf{e}'') \right] \\ &\quad + \left[\frac{1}{2^{n-k}} \sum_{\substack{S \subseteq X \setminus \{x\} \\ |S|=k}} \sum_{\mathbf{e}'' \in \text{cw}(\mathbf{e}|_{X \setminus \{x\}}, S)} C_{-x}(\mathbf{e}'') \right] \\ &= \frac{1}{2^{n-k}} \left(H(C_{+x}, \mathbf{e}|_{X \setminus \{x\}}, k) + H(C_{-x}, \mathbf{e}|_{X \setminus \{x\}}, k) \right). \end{aligned}$$

The last equality is obtained by using the definition of $H(\cdot, \cdot, \cdot)$. A similar analysis allows us to conclude that:

$$\alpha = \begin{cases} \frac{1}{2^{n-(k+1)}} H(C_{+x}, \mathbf{e}|_{X \setminus \{x\}}, k), & \text{if } C(\mathbf{e}) = 1 \\ \frac{1}{2^{n-(k+1)}} H(C_{-x}, \mathbf{e}|_{X \setminus \{x\}}, k), & \text{if } C(\mathbf{e}) = 0 \end{cases}.$$

Hence, if we can compute in polynomial time $H(\cdot, \cdot, \cdot)$ for deterministic and decomposable Boolean circuits, then

we can compute α and β in polynomial time (because C_{+x} and C_{-x} can be computed in linear time from C , and they are deterministic and decomposable as well). Thus, we can compute $\text{Diff}_k(C, \mathbf{e}, x)$ in polynomial time for each $k \in \{0, \dots, n-1\}$ and, hence, $\text{SHAP}(C, \mathbf{e}, x)$ as well. In conclusion, $\text{SHAP}(C, \mathbf{e}, x)$ can be computed in polynomial time if there is a polynomial-time algorithm to compute $H(\cdot, \cdot, \cdot)$ for deterministic and decomposable Boolean circuits.

Reducing from $H(\cdot, \cdot, \cdot)$ to $\#\text{SAT}(\cdot, \cdot, \cdot)$. We now show that computing $H(\cdot, \cdot, \cdot)$ can be reduced in polynomial time to computing $\#\text{SAT}(\cdot, \cdot, \cdot)$. Given as input a deterministic and decomposable circuit C over a set of variables X , an entity $\mathbf{e} \in \text{ent}(X)$, and an integer $k \leq |X|$, recall the definition of $H(C, \mathbf{e}, x)$ in (2). Then consider an entity $\mathbf{e}'' \in \text{ent}(X)$ and reason about how many times \mathbf{e}'' will occur as a summand in the expression (2). First of all, it is clear that if $|\text{sim}(\mathbf{e}, \mathbf{e}'')| < k$, then \mathbf{e}'' will not appear in the sum; this is because if $\mathbf{e}' \in \text{cw}(\mathbf{e}, S)$ for some $S \subseteq X$ such that $|S| = k$, then $S \subseteq \text{sim}(\mathbf{e}, \mathbf{e}')$ and, thus, $k \leq |\text{sim}(\mathbf{e}, \mathbf{e}')|$. Now, how many times does an entity $\mathbf{e}'' \in \text{ent}(X)$ such that $|\text{sim}(\mathbf{e}, \mathbf{e}'')| \geq k$ occur as a summand in the expression? The answer is simple: once per $S \subseteq \text{sim}(\mathbf{e}, \mathbf{e}'')$ of size k . Since there are $\binom{|\text{sim}(\mathbf{e}, \mathbf{e}'')|}{k}$ such sets S , we obtain that $H(C, \mathbf{e}, k)$ is equal to

$$\begin{aligned} &\sum_{\substack{\mathbf{e}'' \in \text{ent}(X) \\ |\text{sim}(\mathbf{e}, \mathbf{e}'')| \geq k}} \binom{|\text{sim}(\mathbf{e}, \mathbf{e}'')|}{k} \cdot C(\mathbf{e}'') \\ &= \sum_{\substack{\mathbf{e}'' \in \text{SAT}(C) \\ |\text{sim}(\mathbf{e}, \mathbf{e}'')| \geq k}} \binom{|\text{sim}(\mathbf{e}, \mathbf{e}'')|}{k} \\ &= \sum_{\ell=k}^n \sum_{\substack{\mathbf{e}'' \in \text{SAT}(C) \\ |\text{sim}(\mathbf{e}, \mathbf{e}'')| = \ell}} \binom{|\text{sim}(\mathbf{e}, \mathbf{e}'')|}{k} \\ &= \sum_{\ell=k}^n \binom{\ell}{k} \sum_{\substack{\mathbf{e}'' \in \text{SAT}(C) \\ |\text{sim}(\mathbf{e}, \mathbf{e}'')| = \ell}} 1 = \sum_{\ell=k}^n \binom{\ell}{k} \cdot \#\text{SAT}(C, \mathbf{e}, \ell), \end{aligned}$$

with the last equality being obtained by using the definition of $\#\text{SAT}(\cdot, \cdot, \cdot)$. This concludes the reduction of this section and, hence, the first part of the proof.

3.2 Computing $\#\text{SAT}(\cdot, \cdot, \cdot)$ in Polynomial Time

We now take care of the second part of the proof of Theorem 3.1, i.e., proving that computing $\#\text{SAT}(\cdot, \cdot, \cdot)$ for deterministic and decomposable Boolean circuits can be done in polynomial time. To do this, given a deterministic and decomposable Boolean circuit C , we first perform two preprocessing steps on C , which will simplify the proof.

- **Rewriting to fan-in at most 2.** First, we modify the circuit C so that the fan-in of every \vee - and \wedge -gate is at most 2. This can simply be done in linear time by rewriting every \wedge -gate (resp., and \vee -gate) of fan-in $m > 2$ with a chain of $m-1$ \wedge -gates (resp., \vee -gates) of fan-in 2. It is clear that the resulting Boolean circuit is deterministic

and decomposable. Hence, from now on we assume that the fan-in of every \vee - and \wedge -gate of C is at most 2.

- **Smoothing the circuit.** A deterministic and decomposable circuit C is *smooth* (Darwiche 2001; Shih et al. 2019) if for every \vee -gate g and input gates g_1, g_2 of g , we have that $\text{var}(g_1) = \text{var}(g_2)$, and we call such a \vee -gate smooth. A standard construction allows to transform in polynomial time a deterministic and decomposable Boolean circuit C into an equivalent smooth deterministic and decomposable Boolean circuit, and where each gate has fan-in at most 2. Thus, from now on we also assume that C is smooth. We illustrate how the construction works in Example 3.4. Full details can be found in the supplementary material in <https://arxiv.org/pdf/2007.14045.pdf>.

We have all the ingredients to prove that $\#\text{SAT}(\cdot, \cdot, \cdot)$ can be computed in polynomial time. Let C be a deterministic and decomposable Boolean circuit over a set of variables X , $\mathbf{e} \in \text{ent}(X)$, ℓ a natural number such that $\ell \leq |X|$ and $n = |X|$. For a gate g of C , let R_g be the Boolean circuit over $\text{var}(g)$ that is defined by considering the subgraph of C induced by the set of gates g' in C for which there exists a path from g' to g in C . Notice that R_g is a deterministic and decomposable Boolean circuit with output gate g . Moreover, for a gate g and natural number $k \leq |\text{var}(g)|$, define $\alpha_g^k := \#\text{SAT}(R_g, \mathbf{e}_{|\text{var}(g)}, k)$, which we recall is the number of entities $\mathbf{e}' \in \text{ent}(\text{var}(g))$ such that \mathbf{e}' satisfies R_g and $|\text{sim}(\mathbf{e}_{|\text{var}(g)}, \mathbf{e}')| = k$. We will show how to compute all the values α_g^k for every gate g of C and $k \in \{0, \dots, |\text{var}(g)|\}$ in polynomial time. This will conclude the proof since, for the output gate g_{out} of C , we have that $\alpha_{g_{\text{out}}}^\ell = \#\text{SAT}(C, \mathbf{e}, \ell)$. Next we explain how to compute these values in a bottom-up manner.

Variable gate. g is a variable gate with label $y \in X$, so that $\text{var}(g) = \{y\}$. Then $\alpha_g^0 = 1 - \mathbf{e}(y)$ and $\alpha_g^1 = \mathbf{e}(y)$.

Constant gate. g is a constant gate with label $a \in \{0, 1\}$. Then $\text{var}(g) = \emptyset$ and $\alpha_g^0 = a$.²

\neg -gate. g is a \neg -gate with input gate g' . Then $\text{var}(g) = \text{var}(g')$, and the values $\alpha_{g'}^k$ for $k \in \{0, \dots, |\text{var}(g)|\}$ have already been computed. Fix $k \in \{0, \dots, |\text{var}(g)|\}$. Since $\binom{|\text{var}(g)|}{k}$ is equal to the number of entities $\mathbf{e}' \in \text{ent}(\text{var}(g))$ such that $|\text{sim}(\mathbf{e}_{|\text{var}(g)}, \mathbf{e}')| = k$, we have that

$$\alpha_g^k = \binom{|\text{var}(g)|}{k} - \alpha_{g'}^k.$$

Therefore, given that $\binom{|\text{var}(g)|}{k}$ can be computed in polynomial time since $k \leq |\text{var}(g)| \leq n = |X|$, we have an efficient way to compute α_g^k .

\vee -gate. g is an \vee -gate. By assumption, g is deterministic, smooth and has fan-in at most 2. If g has only one input g' , then clearly $\text{var}(g) = \text{var}(g')$ and $\alpha_g^k = \alpha_{g'}^k$ for every $k \in \{0, \dots, |\text{var}(g)|\}$.

Thus, assume that g has exactly two input gates g_1 and g_2 , and recall that $\text{var}(g_1) = \text{var}(g_2) = \text{var}(g)$, because g is smooth. Also, recall that $\alpha_{g_1}^k$ and $\alpha_{g_2}^k$, for each $k \in \{0, \dots, |\text{var}(g)|\}$, have already been computed. Fix $k \in \{0, \dots, |\text{var}(g)|\}$. Given that g is deterministic and smooth, we have that $\text{SAT}(R_g) = \text{SAT}(R_{g_1}) \cup \text{SAT}(R_{g_2})$, where $\text{SAT}(R_{g_1}) \cap \text{SAT}(R_{g_2}) = \emptyset$. By intersecting these three sets with the set $\{\mathbf{e}' \in \text{var}(g) \mid |\text{sim}(\mathbf{e}_{|\text{var}(g)}, \mathbf{e}')| = k\}$, we obtain that $\text{SAT}(R_g, \mathbf{e}_{|\text{var}(g)}, k) = \text{SAT}(R_{g_1}, \mathbf{e}_{|\text{var}(g)}, k) \cup \text{SAT}(R_{g_2}, \mathbf{e}_{|\text{var}(g)}, k)$, where $\text{SAT}(R_{g_1}, \mathbf{e}_{|\text{var}(g)}, k) \cap \text{SAT}(R_{g_2}, \mathbf{e}_{|\text{var}(g)}, k) = \emptyset$. Hence:

$$\begin{aligned} \#\text{SAT}(R_g, \mathbf{e}_{|\text{var}(g)}, k) &= \\ \#\text{SAT}(R_{g_1}, \mathbf{e}_{|\text{var}(g)}, k) &+ \#\text{SAT}(R_{g_2}, \mathbf{e}_{|\text{var}(g)}, k), \end{aligned}$$

or, in other words, we have that $\alpha_g^k = \alpha_{g_1}^k + \alpha_{g_2}^k$. Hence, we have an efficient way to compute α_g^k .

\wedge -gate. g is an \wedge -gate. By assumption, recall that g is decomposable and has fan-in at most 2. If g has only one input g' , then clearly $\text{var}(g) = \text{var}(g')$ and $\alpha_g^k = \alpha_{g'}^k$ for every $k \in \{0, \dots, |\text{var}(g)|\}$. Thus, assume that g has exactly two input gates g_1 and g_2 . Recall then that the values $\alpha_{g_1}^i$ and $\alpha_{g_2}^j$, for each $i \in \{0, \dots, |\text{var}(g_1)|\}$ and $j \in \{0, \dots, |\text{var}(g_2)|\}$, have already been computed. Fix $k \in \{0, \dots, |\text{var}(g)|\}$. Given that g is a decomposable \wedge -gate, in this case it is possible to prove that:

$$\alpha_g^k = \sum_{\substack{i \in \{0, \dots, |\text{var}(g_1)|\} \\ j \in \{0, \dots, |\text{var}(g_2)|\} \\ i+j=k}} \alpha_{g_1}^i \cdot \alpha_{g_2}^j. \quad (3)$$

The complete proof of this property can be found in the supplementary material in <https://arxiv.org/pdf/2007.14045.pdf>. Therefore, as in the previous cases, we conclude that there is an efficient way to compute α_g^k .

This concludes the proof that $\#\text{SAT}(\cdot, \cdot, \cdot)$ can be computed in polynomial time for deterministic and decomposable Boolean circuits and, hence, the proof of Theorem 3.1.

Example 3.4. We illustrate how the algorithm for computing the SHAP-score operates on the Boolean circuit C given in Example 1.1. Recall that C is defined over $X = \{\mathbf{fg}, \mathbf{dtr}, \mathbf{nf}, \mathbf{na}\}$, and assume we want to compute $\text{SHAP}(C, \mathbf{e}, \mathbf{nf})$ for the entity \mathbf{e} with $\mathbf{e}(x) = 1$ for each $x \in X$. By the polynomial time reductions shown in Section 3.1, to compute $\text{SHAP}(C, \mathbf{e}, \mathbf{nf})$ it suffices to compute $\text{H}(C_{-\mathbf{nf}}, \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}, \ell)$ and $\text{H}(C_{+\mathbf{nf}}, \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}, \ell)$ for each $\ell \in \{0, \dots, 3\}$, which in turn reduces to the computation of $\#\text{SAT}(C_{-\mathbf{nf}}, \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}, \ell)$ and $\#\text{SAT}(C_{+\mathbf{nf}}, \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}, \ell)$ for each $\ell \in \{0, \dots, 3\}$. In what follows, we show how to compute the values $\#\text{SAT}(C_{+\mathbf{nf}}, \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}, \ell)$.

For the sake of presentation, let $D := C_{+\mathbf{nf}}$ and $\mathbf{e}^* = \mathbf{e}_{|X \setminus \{\mathbf{nf}\}}$, so that we need to compute $\#\text{SAT}(D, \mathbf{e}^*, \ell)$ for

²We recall the mathematical convention that there is a unique function with the empty domain and, hence, a unique entity over \emptyset .

That is, the product distribution that is determined by pre-specified marginal distributions, and that makes the features take values independently from each other. Observe the effect of the probability distribution on the SHAP-score: intuitively, the higher the probability of an entity, the more impact this entity will have on the computation. This can be used, for instance, to avoid bias in the explanations (Lundberg and Lee 2017; Bertossi et al. 2020).

Notice that the uniform space is a special case of product space, with Π_p invoking $p(x) := 1/2$ for every $x \in X$. Thus, our hardness results from Theorem 4.1 also hold in the case where the probabilities $p(x)$ are given as input. What is more interesting is the fact that our tractability result from Theorem 3.1 extends to product distributions. Formally:

Theorem 5.1. *The following problem can be solved in polynomial time. Given as input a deterministic and decomposable circuit C over a set of features X , rational probability values $p(x)$ for every feature $x \in X$, an entity $e : X \rightarrow \{0, 1\}$, and a feature $x \in X$, compute the value $\text{SHAP}(C, e, x)$ under the probability distribution Π_p .*

The proof of Theorem 5.1 is more involved than that of Theorem 3.1, and is provided in the supplementary material in <https://arxiv.org/pdf/2007.14045.pdf>. In particular, the main difficulty is that $\phi(M, e, S)$ is no longer equal to $\sum_{e' \in \text{cw}(e, S)} \frac{1}{2^{|\bar{X} \setminus S|}} M(e')$ (as it was the case for the uniform space), because the entities do not all have the same probability. This prevents us from being able to reduce to the computation of $\#\text{SAT}(\cdot, \cdot, \cdot)$. Instead, we use a different definition of $H(\cdot, \cdot, \cdot)$, and prove that it can directly be computed in a bottom-up fashion on the circuits. We show in Algorithm 1 our algorithm to compute the SHAP score for deterministic and decomposable Boolean circuits under product distributions, which can be extracted from the proof in the supplementary material. Notice that by using the techniques presented in Section 3, the first step of the algorithm transforms the input circuit C into an equivalent smooth circuit D where each \vee -gate and \wedge -gate has fan-in 2.

6 Extensions and Future Work

We leave open many interesting directions for future work. For instance, we intend to extend our algorithm for efficiently computing the SHAP-score to work with non-Boolean classifiers, and to consider more general probability distributions that could better capture possible correlations and dependencies between features. We also aim to provide an experimental comparison of our algorithm, but specialized for decision trees, with the one provided in (Lundberg et al. 2020, Alg. 2). Last, we intend to test our algorithm on real-world scenarios.

Acknowledgments

Pablo Barceló was funded by Fondecyt grant 1200967.

References

Amarilli, A.; Capelli, F.; Monet, M.; and Senellart, P. 2020. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.* 64(5): 861–914.

Algorithm 1: SHAP-scores for deterministic and decomposable Boolean circuits

Input : Deterministic and decomposable Boolean circuit C over features X with output gate g_{out} , rational probability values $p(x)$ for all $x \in X$, entity $e \in \text{ent}(X)$, and feature $x \in X$.

Output: The value $\text{SHAP}(C, e, x)$ under the probability distribution Π_p .

```

1 Transform  $C$  into an equivalent smooth circuit  $D$ 
  where each  $\vee$ -gate and  $\wedge$ -gate has fan-in 2;
2 Compute the set  $\text{var}(g)$  for every gate  $g$  in  $D$ ;
3 Compute values  $\gamma_g^\ell$  and  $\delta_g^\ell$  for every gate  $g$  in  $D$ 
  and  $\ell \in \{0, \dots, |\text{var}(g) \setminus \{x\}|\}$  by bottom-up
  induction on  $D$  as follows:
4   if  $g$  is a constant gate with label  $a \in \{0, 1\}$  then
5      $\gamma_g^0, \delta_g^0 \leftarrow a$ ;
6   else if  $g$  is a variable gate with  $\text{var}(g) = \{x\}$  then
7      $\gamma_g^0 \leftarrow 1$ ;
8      $\delta_g^0 \leftarrow 0$ ;
9   else if  $g$  is a variable gate with  $\text{var}(g) = \{y\}$  and
   $y \neq x$  then
10     $\gamma_g^0, \delta_g^0 \leftarrow p(y)$ ;
11     $\gamma_g^1, \delta_g^1 \leftarrow e(y)$ ;
12  else if  $g$  is a  $\neg$ -gate with input gate  $g'$  then
13    for  $\ell \in \{0, \dots, |\text{var}(g) \setminus \{x\}|\}$  do
14       $\gamma_g^\ell \leftarrow \binom{|\text{var}(g) \setminus \{x\}|}{\ell} - \gamma_{g'}^\ell$ ;
15       $\delta_g^\ell \leftarrow \binom{|\text{var}(g) \setminus \{x\}|}{\ell} - \delta_{g'}^\ell$ ;
16    end
17  else if  $g$  is an  $\vee$ -gate with input gates  $g_1, g_2$  then
18    for  $\ell \in \{0, \dots, |\text{var}(g) \setminus \{x\}|\}$  do
19       $\gamma_g^\ell \leftarrow \gamma_{g_1}^\ell + \gamma_{g_2}^\ell$ ;
20       $\delta_g^\ell \leftarrow \delta_{g_1}^\ell + \delta_{g_2}^\ell$ ;
21    end
22  else if  $g$  is an  $\wedge$ -gate with input gates  $g_1, g_2$  then
23    for  $\ell \in \{0, \dots, |\text{var}(g) \setminus \{x\}|\}$  do
24       $\gamma_g^\ell \leftarrow \sum_{\substack{\ell_1 \in \{0, \dots, |\text{var}(g_1) \setminus \{x\}|\} \\ \ell_2 \in \{0, \dots, |\text{var}(g_2) \setminus \{x\}|\} \\ \ell_1 + \ell_2 = \ell}} \gamma_{g_1}^{\ell_1} \cdot \gamma_{g_2}^{\ell_2}$ ;
25       $\delta_g^\ell \leftarrow \sum_{\substack{\ell_1 \in \{0, \dots, |\text{var}(g_1) \setminus \{x\}|\} \\ \ell_2 \in \{0, \dots, |\text{var}(g_2) \setminus \{x\}|\} \\ \ell_1 + \ell_2 = \ell}} \delta_{g_1}^{\ell_1} \cdot \delta_{g_2}^{\ell_2}$ ;
26    end
27 end
28 return
    
$$\sum_{k=0}^{|\bar{X}|-1} \frac{k! (|\bar{X}| - k - 1)!}{|\bar{X}|!} \cdot [(\mathbf{e}(x) - p(x))(\gamma_{g_{\text{out}}}^k - \delta_{g_{\text{out}}}^k)];$$


```

- Arora, S.; and Barak, B. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.
- Bertossi, L.; Li, J.; Schleich, M.; Suci, D.; and Vagena, Z. 2020. Causality-based explanation of classification outcomes. In *Proceedings of the Fourth Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2020*, 6:1–6:10.
- Cesari, G.; Algaba, E.; Moretti, S.; and Nepomuceno, J. A. 2018. An application of the Shapley value to the analysis of co-expression networks. *Applied network science* 3(1): 35.
- Choi, A.; Shih, A.; Goyanka, A.; and Darwiche, A. 2020. On Symbolically Encoding the Behavior of Random Forests. *CoRR* abs/2007.01493.
- Darwiche, A. 2001. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Applied Non-Classical Logics* 11(1-2).
- Darwiche, A.; and Hirth, A. 2020. On the reasons behind decisions. *arXiv preprint arXiv:2002.09284* .
- Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.* 17: 229–264.
- Deng, X.; and Papadimitriou, C. H. 1994. On the complexity of cooperative solution concepts. *Mathematics of operations research* 19(2): 257–266.
- Faigle, U.; and Kern, W. 1992. The Shapley value for cooperative games under precedence constraints. *International Journal of Game Theory* 21(3): 249–266.
- Hunter, A.; and Konieczny, S. 2010. On the measure of conflicts: Shapley inconsistency values. *Artificial Intelligence* 174(14): 1007–1026.
- Livshits, E.; Bertossi, L. E.; Kimelfeld, B.; and Sebag, M. 2020. The Shapley value of tuples in query answering. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155, 20:1–20:19.
- Lundberg, S. M.; Erion, G.; Chen, H.; DeGrave, A.; Prutkin, J. M.; Nair, B.; Katz, R.; Himmelfarb, J.; Bansal, N.; and Lee, S.-I. 2020. From local explanations to global understanding with explainable AI for trees. *Nature machine intelligence* 2(1): 2522–5839.
- Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, 4765–4774.
- Michalak, T. P.; Aadithya, K. V.; Szczepanski, P. L.; Ravindran, B.; and Jennings, N. R. 2013. Efficient computation of the Shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research* 46: 607–650.
- Molnar, C. 2020. *Interpretable machine learning: A guide for making black box models explainable*. <https://christophm.github.io/interpretable-ml-book>.
- Peharz, R.; Lang, S.; Vergari, A.; Stelzner, K.; Molina, A.; Trapp, M.; Van den Broeck, G.; Kersting, K.; and Ghahramani, Z. 2020. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. *arXiv preprint arXiv:2004.06231* .
- Provan, J. S.; and Ball, M. O. 1983. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing* 12(4): 777–788.
- Roth, A. E. 1988. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press.
- Shapley, L. S. 1953. A value for n-person games. *Contributions to the Theory of Games* 2(28): 307–317.
- Shi, W.; Shih, A.; Darwiche, A.; and Choi, A. 2020. On tractable representations of binary neural networks. *arXiv preprint arXiv:2004.02082* .
- Shih, A.; Choi, A.; and Darwiche, A. 2018a. A symbolic approach to explaining Bayesian network classifiers. In *Proceedings IJCAI*, 5103–5111.
- Shih, A.; Choi, A.; and Darwiche, A. 2018b. Formal verification of Bayesian network classifiers. In *International Conference on Probabilistic Graphical Models*, 427–438.
- Shih, A.; Darwiche, A.; and Choi, A. 2019. Verifying binarized neural networks by Angluin-style learning. In *International Conference on Theory and Applications of Satisfiability Testing*, 354–370. Springer.
- Shih, A.; Van den Broeck, G.; Beame, P.; and Amarilli, A. 2019. Smoothing structured decomposable circuits. In *Advances in Neural Information Processing Systems*, 11416–11426.
- Van den Broeck, G.; Lykov, A.; Schleich, M.; and Suci, D. 2020. On the Tractability of SHAP Explanations. *arXiv preprint arXiv:2009.08634* .