Expected Shapley-Like Scores of Boolean Functions: Complexity and Applications to Probabilistic Databases

PRATIK KARMAKAR, National University of Singapore, Singapore and CNRS@CREATE Ltd, Singapore
MIKAËL MONET, Inria, France and CRIStAL, Université de Lille, CNRS, France
PIERRE SENELLART*, DI ENS, ENS, PSL University, CNRS, France, Inria, France, and IUF, France
STÉPHANE BRESSAN*, National University of Singapore, Singapore

Shapley values, originating in game theory and increasingly prominent in explainable AI, have been proposed to assess the contribution of facts in query answering over databases, along with other similar power indices such as Banzhaf values. In this work we adapt these Shapley-like scores to probabilistic settings, the objective being to compute their expected value. We show that the computations of expected Shapley values and of the expected values of Boolean functions are interreducible in polynomial time, thus obtaining the same tractability landscape. We investigate the specific tractable case where Boolean functions are represented as deterministic decomposable circuits, designing a polynomial-time algorithm for this setting. We present applications to probabilistic databases through database provenance, and an effective implementation of this algorithm within the ProvSQL system, which experimentally validates its feasibility over a standard benchmark.

CCS Concepts: • **Mathematics of computing** → **Probabilistic representations**; • **Information systems** → *Relational database model*; • **Theory of computation** → *Algorithmic game theory and mechanism design.*

Additional Key Words and Phrases: Shapley value, Banzhaf value, probabilistic database, provenance, knowledge compilation, d-D circuit

ACM Reference Format:
Pratik Karmakar, Mikaël Monet, Pierre Senellart, and Stéphane Bressan. 2024. Expected Shapley-Like Scores of Boolean Functions: Complexity and Applications to Probabilistic Databases. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 92 (May 2024), 26 pages. https://doi.org/10.1145/3651593

1 INTRODUCTION

The *Shapley value* is a popular notion from cooperative game theory, introduced by Lloyd Shapley [37]. Its idea is to "fairly" distribute the rewards of a game among the players. The Banzhaf power index [8], another power distribution index with different weights, also plays an important role in voting theory. These are two instances of power indices for coalitions, which also include the Johnston [20, 21], Deegan–Packel [15], and Holler–Packel indices [18], see [27] for a survey. Shapley and Banzhaf values, in particular, have found recent applications in explainable machine learning [23, 40] and valuation of data inputs in data management [2, 17].

*Also with CNRS@CREATE Ltd & IPAL, CNRS.

Authors' addresses: Pratik Karmakar, NUS, School of Computing, Singapore, pratik.karmakar@u.nus.edu; Mikaël Monet, Inria, Villeneuve d'Ascq, France, mikael.monet@inria.fr; Pierre Senellart, DI ENS, ENS, PSL University, Paris, France, pierre@senellart.com; Stéphane Bressan, NUS, School of Computing, Singapore, steph@nus.edu.sg.

ACM 2836-6573/2024/5-ART92
https://doi.org/10.1145/3651593

In this work, we revisit the computation of such values (which we call *Shapley-like* values or scores) in a setting where data is uncertain. A long line of work [11–13, 25, 28, 32, 41] has studied various ways in which scoring mechanisms can be incorporated in a probabilistic setting. More specifically, the authors of [11] propose the notion of *expected Shapley value* and show that this is the only scoring mechanism satisfying a certain set of axioms in a probabilistic setting. Motivated by this, we extend their notion to other scores (e.g., Banzhaf), and study the complexity of the associated computational problems.

Our objective is then to investigate the tractability of expected Shapley-like value computations for Boolean functions, having in mind the potential application of computation of expected Shapley-like values of facts for a query over probabilistic databases. In particular, some results have been obtained in the literature that reduces the complexity of (non-probabilistic) Shapley-value computation to and from the computation of the model count of a Boolean function (or to the computation of the probability of a query in probabilistic databases) under some technical conditions [17, 22]; we aim at understanding this connection better by investigating whether *expected* Shapley(-like) value computation, which combines the computation of a power index and a probabilistic setting, is harder than each of these aspects taken in isolation.

We provide the following contributions. First (in Section 2), we formally introduce the notion of Shapley-like scores and of the expected value of such scores on Boolean functions whose variables are assigned independent probabilities. In Section 3, we investigate the connection between the computation of expected Shapley-like scores and the computation of the expected value of a Boolean function. In particular, we show a very general result (Corollary 3.7) that expected Shapley value computation is interreducible in polynomial time to the expected value computation problem over any class of Boolean functions for which it is possible to compute its value over the empty set in polynomial time; we also obtain a similar result (Corollary 3.12) for the computation of expected Banzhaf values. We then assume in Section 4 that we have a tractable representation of a Boolean function as a decomposable and deterministic circuit; in this case, we show a concrete polynomial-time algorithm for Shapley-like score computation (Algorithm 1) and some simplifications thereof for specific settings. We then apply in Section 5 these results to the case of probabilistic databases, showing (Corollary 5.3) that expected Shapley value computation is interreducible in polynomial time to probabilistic query evaluation. In Section 6 we show through an experimental evaluation that the algorithms proposed in this paper are indeed feasible in practical scenarios. Before concluding the paper, we discuss related work in Section 7.

For space reason, some proofs are relegated to the appendix.

2 PRELIMINARIES

For $n \in \mathbb{N}$ we write $[n] \overset{\text{def}}{=} \{0, \dots, n\}$. We denote by P the class of problems solvable in polynomial time. For a set V, we denote by 2^V its powerset.

Boolean functions. A *Boolean function over a finite set of variables* V is a mapping $\varphi : 2^V \to \{0, 1\}$. To talk about the complexity of problems over a class of Boolean functions, one must first specify how functions are specified as input. By a *class of Boolean functions*, we then mean a class of *representations* of Boolean functions; for instance, truth tables, decision trees, Boolean circuits, and so on, with any sensible encoding. In particular, we consider that the size of V is always provided in unary as part of the input. Let $\varphi : 2^V \to \{0, 1\}$ and $x \in V$. We denote by φ_{+x} (resp., φ_{-x}) the Boolean function on $V \setminus \{x\}$ that maps $Z \subseteq V \setminus \{x\}$ to $\varphi(Z \cup \{x\})$ (resp., to $\varphi(Z)$).

Expected value. For each $x \in V$, let $p_x \in [0, 1]$ be a probability value. For $V' \subseteq V$ and $Z \subseteq V'$, define $\Pi_{V'}(Z)$ to be the probability of Z being drawn from V' under the assumption that

Table 1. (Expected) Shapley values for $\varphi_{\mathrm{ex}} = (A \land a) \lor (C \land c)$

$x \in V$	p_x	$\mathrm{Score}_{c_{\mathrm{Shapley}}}(\varphi_{\mathrm{ex}}, V, x)$	$\mathrm{EScore}_{c_{\mathrm{Shapley}}}(\varphi_{\mathrm{ex}}, x)$
A	0.4	0.25	0.076
a	0.5	0.25	0.076
C	0.6	0.25	0.216
c	0.8	0.25	0.216

every $x \in V'$ is chosen independently with probability p_x. Formally: $\Pi_{V'}(Z) \stackrel{\mathrm{def}}{=} \left(\prod_{x \in Z} p_x\right) \times \left(\prod_{x \in V' \setminus Z}(1 - p_x)\right)$. The p-values do not appear in the notation $\Pi_{V'}(Z)$: this is to simplify notation. For $\varphi : 2^V \longrightarrow \{0, 1\}$, define then the *expected value of* φ as $\mathsf{EV}(\varphi) \stackrel{\mathrm{def}}{=} \sum_{Z \subseteq V} \Pi_V(Z)\varphi(Z)$. Note that this is simply the probability of φ being true. We then define the corresponding problem for a class of Boolean functions $\mathcal{F}$.

PROBLEM :	$\mathsf{EV}(\mathcal{F})$	(*Expected Value*)
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over variables V and probability values p_x for each $x \in V$	
OUTPUT :	The quantity $\mathsf{EV}(\varphi)$	

Here, we consider as usual that the probabilities values are rational numbers $\frac{p}{q}$ for $(p, q) \in \mathbb{N} \times (\mathbb{N} \setminus \{0\})$, provided as ordered pairs (p, q) where p and q themselves are encoded in binary.

Example 2.1. *Consider the Boolean function in disjunctive normal form:* $\varphi_{\mathrm{ex}} = (A \land a) \lor (C \land c)$ *on variables* $\{A, a, C, c\}$. *Assume the probabilities for these variables given in the second column of Table 1. Then its expected value can be computed (since* φ_{ex} *is read-once) as:* $\mathsf{EV}(\varphi_{\mathrm{ex}}) = 1 - (1 - 0.4 \times 0.5) \times (1 - 0.6 \times 0.8) = 1 - 0.8 \times 0.52 = 0.584$.

Shapley-like scores. Let $c : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{Q}$ be a function, that we call the *coefficient function*, and let $\varphi : 2^V \longrightarrow \{0, 1\}$ and $x \in V$. Define the *Shapley-like score with coefficients c of x in V with respect to* φ, or simply *score* when clear from context, by

$$\mathrm{Score}_c(\varphi, V, x) \stackrel{\mathrm{def}}{=} \sum_{E \subseteq V \setminus \{x\}} c(|V|, |E|) \times \left[\varphi(E \cup \{x\}) - \varphi(E)\right].$$

Example 2.2. *Let* $c_{\mathrm{Shapley}}(k, \ell) \stackrel{\mathrm{def}}{=} \frac{\ell!(k-l-1)!}{k!} = \binom{k-1}{l}^{-1} k^{-1}$ *and* $c_{\mathrm{Banzhaf}}(k, \ell) \stackrel{\mathrm{def}}{=} 1$. *Then, by definition,* $\mathrm{Score}_{c_{\mathrm{Shapley}}}(\varphi, V, x)$ *(resp.,* $\mathrm{Score}_{c_{\mathrm{Banzhaf}}}(\varphi, V, x)$*) is the usual Shapley (resp., Banzhaf) value, with set of players V and wealth function* φ. *The Penrose–Banzhaf power [24], a normalization of Banzhaf values, can also be defined by coefficients* $(k, \ell) \mapsto 2^{-k+1}$.

For each fixed coefficient function c and class of Boolean functions $\mathcal{F}$, we define the corresponding computational problem.

PROBLEM :	$\mathrm{Score}_c(\mathcal{F})$
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over variables V, a variable $x \in V$
OUTPUT :	The quantity $\mathrm{Score}_c(\varphi, V, x)$

Expected Shapley-like scores. We now introduce the probabilistic variant of Shapley-like scores, which is our main object of study.

Definition 2.3. *Let c be a coefficient function, $\varphi : 2^V \longrightarrow \{0,1\}$ a Boolean function over variables V, probability values p_y for $y \in V$, and $x \in V$. Define the* expected score of x for φ *as:*

$$\mathrm{EScore}_c(\varphi, x) \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq V \\ x \in Z}} \left(\Pi_V(Z) \times \mathrm{Score}_c(\varphi, Z, x)\right),$$

where in $\mathrm{Score}_c(\varphi, Z, x)$ *we see φ as a function from 2^Z to $\{0,1\}$.*

In words, this is the expected value of the corresponding score, when players are independently selected to be part of the cooperative game. Notice that subsets $Z \subseteq V$ not containing x are not summed over: this is because in this case x is not a player of the selected game and we thus declare its "contribution" to be null. We point out that the notion of expected Shapley value has been defined and studied in [11] in terms of the properties that it satisfies. The authors show that this is the only value satisfying some natural sets of axioms, making it a natural candidate for score attribution in a probabilistic setting. This definition is also strongly motivated by its applications to probabilistic databases (cf. Section 5). We then define the corresponding computational problem, for a fixed c and $\mathcal{F}$.

PROBLEM :	$\mathrm{EScore}_c(\mathcal{F})$	(*Expected Score*)
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over variables V, probability values p_y for each $y \in V$, a variable $x \in V$	
OUTPUT :	The quantity $\mathrm{EScore}_c(\varphi, x)$	

Example 2.4. *Continuing from Example 2.1, we can compute the Shapley value and the expected Shapley value of variables appearing in φ_{ex}. By symmetry, it is easy to see that all variables have the same Shapley value; on the other hand the expected Shapley values depend on the probability of each clause to be true. See Table 1 for all (expected) Shapley values. If other variables were in V, their Shapley and expected Shapley values w.r.t. φ_{ex} would be 0, regardless of their probabilities, since they do not play a role in the satisfaction of φ_{ex}.*

Reductions. For two computational problems A and B, we write $A \leqslant_{\mathrm{P}} B$ to assert the existence of a polynomial-time Turing reduction from A to B, i.e., a polynomial-time reduction that is allowed to use B as an oracle. We write $A \equiv_{\mathrm{P}} B$ when $A \leqslant_{\mathrm{P}} B$ and $B \leqslant_{\mathrm{P}} A$, meaning that the problems are equivalent under such reductions. Using this notation we can state a first trivial fact:

Fact 2.5. *We have* $\mathrm{Score}_c(\mathcal{F}) \leqslant_{\mathrm{P}} \mathrm{EScore}_c(\mathcal{F})$ *for any coefficient function c and class of Boolean functions $\mathcal{F}$.*

This is simply because $\mathrm{EScore}_c(\varphi, x) = \mathrm{Score}_c(\varphi, V, x)$ when $p_y = 1$ for all $y \in V$, so that our probabilistic variants of such scores are proper generalizations of the non-probabilistic ones.

3 EQUIVALENCE WITH EXPECTED VALUES

In this section we link the complexity of computing expected Shapley-like scores with that of computing expected values. The point is that $\mathrm{EV}(\mathcal{F})$ is a central problem that has already been studied in depth for most meaningful classes of Boolean functions, with classes for which that problem is in P while the general problem is #P-hard. In a sense then, if we can show for some problem A that $A \equiv_{\mathrm{P}} \mathrm{EV}(\mathcal{F})$, this settles the complexity of A. We start in Section 3.1 by the direction most interesting for efficient algorithms: from expected scores to expected values. We show that this is always possible, under the assumption that the coefficient function is computable in polynomial time. We then give results for the other direction in Section 3.2, where the picture is more complex.

3.1 From Expected Scores to Expected Values

Let us call a coefficient function c *tractable* if $c(k, \ell)$ can be computed in P when k and ℓ are given in unary as input. It is easy to see that c_{Banzhaf} and its normalized version are tractable. This is also the case of c_{Shapley}, using the fact that the $\binom{k}{\ell}$ binomial coefficient can be computed in time $O(k \times \ell)$ by dynamic programming (assuming arguments in unary). Under this assumption, we show that computing expected Shapley-like scores reduces in polynomial time to computing expected values.

Theorem 3.1. *We have* $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}(\mathcal{F})$ *for any tractable coefficient function c and any class $\mathcal{F}$ of Boolean functions.*

We obtain for instance that $\mathsf{EScore}_c(\mathcal{F})$ is in P for decision trees, ordered binary decision diagrams (OBDDs), deterministic and decomposable Boolean circuits, Boolean circuits of bounded treewidth [4], and so on, since $\mathsf{EV}(\mathcal{F})$ is in P for these classes. By Fact 2.5, this also recovers the results from [2, 17, 22] that (non-expected) Shapley and Banzhaf scores are in P for the tractable classes that they consider.

We prove Theorem 3.1 in the remaining of this section. To do so, we first define two intermediate problems.

PROBLEM :	$\mathsf{EV}_\star(\mathcal{F})$ (*Expected Value of Fixed Size*)
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over variables V, probabilities p_x for each $x \in V$, and $k \in [\lvert V\rvert]$
OUTPUT :	The quantity $\mathsf{EV}_k(\varphi) \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq V\\ \lvert Z\rvert = k}} \Pi_V(Z)\varphi(Z)$

PROBLEM :	$\mathsf{ENV}_{\star,\star}(\mathcal{F})$ (*Expected Nested Value of Fixed Sizes*)
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over V, probabilities p_x for each $x \in V$, and integers $k, \ell \in [\lvert V\rvert]$
OUTPUT :	The quantity $\mathsf{ENV}_{k,\ell}(\varphi) \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq V\\ \lvert Z\rvert = k}} \Pi_V(Z) \sum_{\substack{E \subseteq Z\\ \lvert E\rvert = \ell}} \varphi(E)$

Notice that $\mathsf{ENV}_{k,\ell}(\varphi) = 0$ when $k < \ell$. Also, observe that $\mathsf{EV}(\varphi) = \sum_{k=0}^{\lvert V\rvert} \mathsf{EV}_k(\varphi)$ and that $\mathsf{EV}_k(\varphi) = \mathsf{ENV}_{k,k}(\varphi)$, so that $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}_\star(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{ENV}_{\star,\star}(\mathcal{F})$ for any $\mathcal{F}$.

We now prove the chain of reductions $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{ENV}_{\star,\star}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}_\star(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}(\mathcal{F})$, in this order, which implies Theorem 3.1 indeed.

Lemma 3.2. *We have* $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{ENV}_{\star,\star}(\mathcal{F})$ *for any tractable coefficient function c and any class of Boolean functions $\mathcal{F}$.*

Proof. Let $\varphi : 2^V \to \{0, 1\}$ in $\mathcal{F}$, probability values p_y for $y \in V$, and $x \in V$. We wish to compute $\mathsf{EScore}_c(\varphi, x)$. Observe that $\mathsf{EScore}_c(\varphi, x) = A - B$, where

$$A = \sum_{\substack{Z \subseteq V\\ x \in Z}} \Pi_V(Z) \sum_{E \subseteq Z \setminus \{x\}} c(\lvert Z\rvert, \lvert E\rvert)\varphi(E \cup \{x\}) \qquad B = \sum_{\substack{Z \subseteq V\\ x \in Z}} \Pi_V(Z) \sum_{E \subseteq Z \setminus \{x\}} c(\lvert Z\rvert, \lvert E\rvert)\varphi(E).$$

Let us focus on A. Letting $V' \stackrel{\text{def}}{=} V \setminus \{x\}$, notice that these are the variables over which φ_{+x} is defined. Letting $n \stackrel{\text{def}}{=} |V'|$, we have

$$\begin{aligned} A &= \sum_{\substack{Z\subseteq V\\ x\in Z}} \Pi_V(Z) \sum_{E\subseteq Z\setminus\{x\}} c(|Z|,|E|)\varphi_{+x}(E) = p_x \sum_{Z\subseteq V'} \Pi_{V'}(Z) \sum_{E\subseteq Z} c(|Z|+1,|E|)\varphi_{+x}(E) \\ &= p_x \sum_{Z\subseteq V'} \sum_{E\subseteq Z} c(|Z|+1,|E|)\Pi_{V'}(Z)\varphi_{+x}(E) = p_x \sum_{k=0}^{n} \sum_{\substack{Z\subseteq V'\\ |Z|=k}} \sum_{\ell=0}^{k} \sum_{\substack{E\subseteq Z\\ |E|=\ell}} c(k+1,\ell)\Pi_{V'}(Z)\varphi_{+x}(E) \\ &= p_x \sum_{k=0}^{n}\sum_{\ell=0}^{k} c(k+1,\ell) \sum_{\substack{Z\subseteq V'\\ |Z|=k}} \sum_{\substack{E\subseteq Z\\ |E|=\ell}} \Pi_{V'}(Z)\varphi_{+x}(E) = p_x \sum_{k=0}^{n}\sum_{\ell=0}^{k} c(k+1,\ell)\mathsf{ENV}_{k,\ell}(\varphi_{+x}). \end{aligned}$$

We can do exactly the same for B (replacing φ_{+x} by φ_{-x}), after which we obtain:

$$\mathsf{EScore}_c(\varphi,x) = p_x \sum_{k=0}^{n}\sum_{\ell=0}^{k} c(k+1,\ell)\big(\mathsf{ENV}_{k,\ell}(\varphi_{+x}) - \mathsf{ENV}_{k,\ell}(\varphi_{-x})\big). \tag{1}$$

We can compute the coefficients $c(k+1,\ell)$ in P because c is tractable. Therefore, all that is left to show is that we can compute in P all the values $\mathsf{ENV}_{k,\ell}(\varphi_{+x})$ and $\mathsf{ENV}_{k,\ell}(\varphi_{-x})$ for $k,\ell \in [n]$. We point out that this is not obvious, because $\mathcal{F}$ might not be closed under conditioning, and unfortunately setting p_x to 0 or 1 is not enough to directly give us the values we want. In [22], this annoying subtlety is handled by using the closure under OR-substitutions of the class $\mathcal{F}$ (see the proof of their Lemma 3.2). In our case, we will overcome this problem by using the fact that we can freely choose the probabilities.

Let $z \in [0,1]$, and for $y \in V' = V \setminus \{x\}$ define $p^z_y \stackrel{\text{def}}{=} p_y$, and $p^z_x = z$. Define Π^z and $\mathsf{ENV}^z_{\star,\star}(\varphi)$ as expected. We claim that the following equation holds, for $i,j \in [n+1]$:

$$\mathsf{ENV}^z_{i,j}(\varphi) = z\big[\mathsf{ENV}_{i-1,j}(\varphi_{-x}) + \mathsf{ENV}_{i-1,j-1}(\varphi_{+x})\big] + (1-z)\mathsf{ENV}_{i,j}(\varphi_{-x}), \tag{2}$$

where we extended the definition of $\mathsf{ENV}_{\star,\star}(\varphi_{+x})$ and $\mathsf{ENV}_{\star,\star}(\varphi_{-x})$ to have value zero for out-of-bound (i,j)-indices. Before proving this claim, let us explain why this allows us to conclude. Indeed, we can then use the oracle to $\mathsf{ENV}_{\star,\star}(\mathcal{F})$ with $z=0$ to compute all the values $\mathsf{ENV}_{k,\ell}(\varphi_{-x})$. Once these are known, we use the oracle again but this time with $z=1$, and thanks to Equation (2) again we can recover all the values $\mathsf{ENV}_{k,\ell}(\varphi_{+x})$.

Therefore, all that is left to do is prove Equation (2). We have:

$$\mathsf{ENV}^z_{i,j}(\varphi) \stackrel{\text{def}}{=} \sum_{\substack{Z\subseteq V\\ |Z|=i}} \Pi_V(Z) \sum_{\substack{E\subseteq Z\\ |E|=j}} \varphi(E) = \underbrace{\sum_{\substack{Z\subseteq V\\ |Z|=i\\ x\notin Z}} \Pi_V(Z) \sum_{\substack{E\subseteq Z\\ |E|=j}} \varphi(E)}_{\alpha} + \underbrace{\sum_{\substack{Z\subseteq V\\ |Z|=i\\ x\in Z}} \Pi_V(Z) \sum_{\substack{E\subseteq Z\\ |E|=j}} \varphi(E)}_{\beta}.$$

With α and β the two terms defined above, we have:

$$\alpha = (1-z) \sum_{\substack{Z\subseteq V'\\ |Z|=i}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z\\ |E|=j}} \varphi(E) = (1-z) \sum_{\substack{Z\subseteq V'\\ |Z|=i}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z\\ |E|=j}} \varphi_{-x}(E) = (1-z)\mathsf{ENV}_{i,j}(\varphi_{-x}).$$

Let us now inspect β.

$$\beta = z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z\cup\{x\} \\ |E|=j}} \varphi(E) = \underbrace{z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z\cup\{x\} \\ |E|=j \\ x\notin E}} \varphi(E)}_{\alpha'} + \underbrace{z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z\cup\{x\} \\ |E|=j \\ x\in E}} \varphi(E)}_{\beta'}.$$

Again with α' and β' two terms above, we have:

$$\alpha' = z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z \\ |E|=j}} \varphi_{-x}(E) = z \times \mathsf{ENV}_{i-1,j}(\varphi_{-x}),$$

and

$$\beta' = z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z \\ |E|=j-1}} \varphi(E\cup\{x\}) = z \sum_{\substack{Z\subseteq V' \\ |Z|=i-1}} \Pi_{V'}(Z) \sum_{\substack{E\subseteq Z \\ |E|=j-1}} \varphi_{+x}(E) = z \times \mathsf{ENV}_{i-1,j-1}(\varphi_{+x}).$$

Putting it all together, we indeed obtain Equation (2), thus concluding the proof. □

The following lemma contains the most technical part of the proof of Theorem 3.1. It is proved using polynomial interpolation with carefully crafted probability values.

Lemma 3.3. *We have* $\mathsf{ENV}_{\star,\star}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}_{\star}(\mathcal{F})$ *for any* $\mathcal{F}$.

Proof. Let $\varphi \in \mathcal{F}$ over variables V, probability values p_x for each $x \in V$, and $k, \ell \in [|V|]$. Let $n \stackrel{\text{def}}{=} |V|$. Our goal is to compute $\mathsf{ENV}_{k,\ell}(\varphi)$. We will in fact use polynomial interpolation to compute *all* the values $\mathsf{ENV}_{j,\ell}(\varphi)$ for $j \in [n]$, and return $\mathsf{ENV}_{k,\ell}(\varphi)$.

Let $z_0, \ldots, z_n$ be $n+1$ distinct positive values in $\mathbb{Q}$. For $i \in [n]$ and $x \in V$, define $c_x^{z_i} \stackrel{\text{def}}{=} 2z_ip_x+1-p_x$ and $p_x^{z_i} \stackrel{\text{def}}{=} \frac{z_ip_x}{c_x^{z_i}}$, and define Π^{z_i} and $\mathsf{EV}^{z_i}(\varphi)$ as expected. Notice that these are all valid probability mappings, i.e., all values $p_x^{z_i}$ are well-defined and between 0 and 1, and observe that $1 - p_x^{z_i} = \frac{(z_ip_x)+(1-p_x)}{c_x^{z_i}}$. Define further $C_{z_i} \stackrel{\text{def}}{=} \prod_{x\in V} c_x^{z_i}$. Then:

$$\begin{aligned}\mathsf{EV}_\ell^{z_i}(\varphi) &= \sum_{\substack{E\subseteq V \\ |E|=\ell}} \Pi_V^{z_i}(E)\varphi(E) = \sum_{\substack{E\subseteq V \\ |E|=\ell}} \varphi(E) \prod_{x\in E} p_x^{z_i} \prod_{x\in V\setminus E} (1-p_x^{z_i}) \\ &= \frac{1}{C_{z_i}} \sum_{\substack{E\subseteq V \\ |E|=\ell}} \varphi(E) \prod_{x\in E} z_ip_x \prod_{x\in V\setminus E} [(z_ip_x) + (1-p_x)].\end{aligned}$$

Next we develop the innermost product as it is parenthesized and distribute the $\prod_{x\in E} z_ip_x$ term, obtaining:

$$\begin{aligned}\mathsf{EV}_\ell^{z_i}(\varphi) &= \frac{1}{C_{z_i}} \sum_{\substack{E\subseteq V \\ |E|=\ell}} \varphi(E) \sum_{E\subseteq Z\subseteq V} \prod_{x\in Z} z_ip_x \prod_{x\in V\setminus Z} (1-p_x) \\ &= \frac{1}{C_{z_i}} \sum_{\substack{E\subseteq V \\ |E|=\ell}} \varphi(E) \sum_{j=0}^{n} \sum_{\substack{E\subseteq Z\subseteq V \\ |Z|=j}} \prod_{x\in Z} z_ip_x \prod_{x\in V\setminus Z} (1-p_x) \\ &= \frac{1}{C_{z_i}} \sum_{\substack{E\subseteq V \\ |E|=\ell}} \varphi(E) \sum_{j=0}^{n} {z_i}^j \sum_{\substack{E\subseteq Z\subseteq V \\ |Z|=j}} \Pi_V(Z)\end{aligned}$$

$$= \frac{1}{C_{z_i}} \sum_{j=0}^{n} {z_i}^j \sum_{\substack{E\subseteq V\\|E|=\ell}} \varphi(E) \sum_{\substack{E\subseteq Z\subseteq V\\|Z|=j}} \Pi_V(Z).$$

$$= \frac{1}{C_{z_i}} \sum_{j=0}^{n} {z_i}^j \mathsf{ENV}_{j,\ell}(\varphi), \tag{3}$$

where in the last equality we have inverted the two innermost sums. Using the oracle to $\mathsf{EV}_\star(\mathcal{F})$, we compute $c_i \stackrel{\text{def}}{=} C_{z_i} \times \mathsf{EV}_\ell^{z_i}(\varphi)$ for $i \in [n]$ in polynomial time. By Equation (3), this gives us a system of linear equations $AX = C$, with $C_i \stackrel{\text{def}}{=} c_i$, $X_j \stackrel{\text{def}}{=} \mathsf{ENV}_{j,\ell}(\varphi)$ and $A_{ij} \stackrel{\text{def}}{=} {z_i}^j$. We see that A is a non-singular Vandermonde matrix, so we can in polynomial time recover all the values X_j and return $\mathsf{ENV}_{k,\ell}(\varphi)$, as promised. This concludes the proof. □

We can finally state the last step of the proof of Theorem 3.1, again proved using polynomial interpolation.

Lemma 3.4. *We have* $\mathsf{EV}_\star(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$.

Proof. Let $\varphi \in \mathcal{F}$ over variables V, probability values p_x for each $x \in V$, and $k \in [|V|]$. Let $n \stackrel{\text{def}}{=} |V|$. We wish to compute $\mathsf{EV}_k(\varphi)$. We use again polynomial interpolation to compute all the values $\mathsf{EV}_j(\varphi)$ for $j \in [n]$ and return $\mathsf{EV}_k(\varphi)$.

Let $z_0, \ldots, z_n$ be $n+1$ distinct positive values in $\mathbb{Q}$. For $i \in [n]$ and $x \in V$, define $c_x^{z_i} \stackrel{\text{def}}{=} 1-p_x+z_i p_x$, define $p_x^{z_i} \stackrel{\text{def}}{=} \frac{z_i p_x}{c_x^{z_i}}$, and define Π^{z_i} and $\mathsf{EV}^{z_i}(\varphi)$ as expected. Again, these are all valid probability mappings, and observe that this time $1 - p_x^{z_i} = \frac{1-p_x}{c_x^{z_i}}$. Defining as before $C_{z_i} \stackrel{\text{def}}{=} \prod_{x\in V} c_x^{z_i}$, it is this time much easier to derive the equality $\mathsf{EV}^{z_i}(\varphi) = \frac{1}{C_{z_i}} \sum_{j=0}^{n} {z_i}^j \mathsf{EV}_j(\varphi)$:

$$\mathsf{EV}^{z_i}(\varphi) \stackrel{\text{def}}{=} \sum_{Z\subseteq V} \Pi_V^{z_i}(Z)\varphi(Z) = \sum_{j=0}^{n} \sum_{\substack{Z\subseteq V\\|Z|=j}} \Pi_V^{z_i}(Z)\varphi(Z)$$

$$= \sum_{j=0}^{n} \sum_{\substack{Z\subseteq V\\|Z|=j}} \varphi(Z) \prod_{x\in Z} p_x^{z_i} \prod_{x\in V\setminus Z} (1-p_x^{z_i}) = \frac{1}{C_{z_i}} \sum_{j=0}^{n} \sum_{\substack{Z\subseteq V\\|Z|=j}} \varphi(Z) {z_i}^{|Z|} \prod_{x\in Z} p_x \prod_{x\in V\setminus Z} (1-p_x)$$

$$= \frac{1}{C_{z_i}} \sum_{j=0}^{n} {z_i}^j \sum_{\substack{Z\subseteq V\\|Z|=j}} \varphi(Z)\Pi_V(Z) = \frac{1}{C_{z_i}} \sum_{j=0}^{n} {z_i}^j \mathsf{EV}_j(\varphi).$$

We can then conclude just like in the proof of Lemma 3.3. □

3.2 From Expected Values to Expected Scores

In this section we show reductions in the other direction for c_{Shapley} and c_{Banzhaf}, under additional assumptions on the class $\mathcal{F}$.

Shapley score. Let us call a class of Boolean functions $\mathcal{F}$ *reasonable* if the following problem is in P: given as input (a representation of) $\varphi \in \mathcal{F}$, compute $\varphi(\emptyset)$. It is clear that all classes mentioned in this paper are reasonable in that sense. Then, under this assumption:

Proposition 3.5. *We have* $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EScore}_{c_{\mathrm{Shapley}}}(\mathcal{F})$ *for any reasonable class* $\mathcal{F}$ *of Boolean functions.*

Proof. For $\varphi : 2^Z \to \{0,1\}$, it is well known that the *efficiency property* holds: $\sum_{x \in Z} \mathrm{Score}_{c_{\mathrm{Shapley}}}(\varphi, Z, x) = \varphi(Z) - \varphi(\emptyset)$. Let then $\varphi \in \mathcal{F}$ over variables V and probability values p_x for each $x \in V$; our goal is to compute $\mathsf{EV}(\varphi)$. We have:

$$\begin{aligned}\sum_{x \in V} \mathrm{EScore}_{c_{\mathrm{Shapley}}}(\varphi, x) &= \sum_{x \in V} \sum_{\substack{Z \subseteq V \\ x \in Z}} \Pi_V(Z) \times \mathrm{Score}_{c_{\mathrm{Shapley}}}(\varphi, Z, x) \\ &= \sum_{Z \subseteq V} \sum_{x \in Z} \Pi_V(Z) \times \mathrm{Score}_{c_{\mathrm{Shapley}}}(\varphi, Z, x) \\ &= \sum_{Z \subseteq V} \Pi_V(Z) \big[\varphi(Z) - \varphi(\emptyset)\big] \\ &= \mathsf{EV}(\varphi) - \varphi(\emptyset). \end{aligned} \tag{4}$$

We note this is what [11, Axiom 9] calls *expected efficiency*, with the slight difference that they only consider cases in which $\varphi(\emptyset) = 0$. We can compute the left-hand size in P using oracle calls, and we can compute $\varphi(\emptyset)$ in P as well because $\mathcal{F}$ is reasonable, therefore we can compute $\mathsf{EV}(\varphi)$ in P indeed. This concludes the proof. ◻

Example 3.6. *The sum of all expected Shapley values in Table 1 is* 0.584*; as* $\varphi_{\mathrm{ex}}(\emptyset) = 0$*, this is exactly* $\mathsf{EV}(\varphi_{\mathrm{ex}})$ *computed in Example 2.1.*

This implies, for instance, that $\mathrm{EScore}_{c_{\mathrm{Shapley}}}(\mathcal{F})$ is intractable over arbitrary circuits, even monotone bipartite 2-DNF formulas [33]. Combining with Theorem 3.1, we obtain in particular:

Corollary 3.7. *We have* $\mathrm{EScore}_{c_{\mathrm{Shapley}}}(\mathcal{F}) \equiv_{\mathrm{P}} \mathsf{EV}(\mathcal{F})$ *for any reasonable class* $\mathcal{F}$ *of Boolean functions.*

Hence, at least with respect to polynomial-time computability, this settles the complexity of $\mathrm{EScore}_{c_{\mathrm{Shapley}}}(\mathcal{F})$ for such classes.

Banzhaf score. Next, we show a similar result for the Banzhaf value, under a different, though commonplace, assumption.

Definition 3.8. *A class* $\mathcal{F}$ *is said to be* closed under conditioning *if the following problem is in* P*: given* $\varphi \in \mathcal{F}$ *over variables* V *and* $x \in V$*, compute a representation in* $\mathcal{F}$ *of* φ_{+x}*. We say* $\mathcal{F}$ *is* closed under conjunctions (resp., disjunctions) with fresh variables *if the following is in* P*: given* $\varphi \in \mathcal{F}$ *over variables* V *and* $x \notin V$*, compute a representation in* $\mathcal{F}$ *of the Boolean function* $\varphi \wedge x$ *(resp.,* $\varphi \vee x$*).*

Proposition 3.9. *We have* $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathrm{EScore}_{c_{\mathrm{Banzhaf}}}(\mathcal{F})$ *for any class* $\mathcal{F}$ *that is closed under conditioning and that is also closed under either conjunctions or disjunctions with fresh variables.*

This implies that $\mathrm{EScore}_{c_{\mathrm{Banzhaf}}}$ is intractable, for instance, over monotone 2-CNFs or monotone 2-DNFs.

Proposition 3.9 requires more work than Proposition 3.5: we do it in two steps by introducing (yet) another intermediate problem.

PROBLEM :	$\mathsf{ENV}(\mathcal{F})$ (*Expected Nested Value*)
INPUT :	A Boolean function $\varphi \in \mathcal{F}$ over variables V and probability values p_x for each $x \in V$
OUTPUT :	The quantity $\mathsf{ENV}(\varphi) \stackrel{\text{def}}{=} \sum_{Z \subseteq V} \Pi_V(Z) \sum_{E \subseteq Z} \varphi(E)$

The next two lemmas then imply Proposition 3.9.

Lemma 3.10. *We have* $\mathsf{ENV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\mathcal{F})$ *for any* $\mathcal{F}$ *closed under conjunctions (resp., disjunctions) with fresh variables.*

Proof sketch. First, for $\varphi' : 2^{V'} \to \{0,1\}$ and $x \in V'$, we prove the equation

$$\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\varphi', x) = p_x[\mathsf{ENV}(\varphi'_{+x}) - \mathsf{ENV}(\varphi'_{-x})]. \tag{5}$$

Let then $\varphi : 2^V \to \{0,1\}$ be the Boolean function for which we want to compute $\mathsf{ENV}(\varphi)$, and let $x \notin V$ be a fresh variable. We use the closure property of $\mathcal{F}$ to compute a representation of $\varphi' \stackrel{\text{def}}{=} \varphi \odot x$, with $\odot$ being $\wedge$ or $\vee$ depending on the closure property. We then show using Equation (5) that $\mathsf{ENV}(\varphi)$ can be recovered from the single oracle call $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\varphi', x)$, with $V' \stackrel{\text{def}}{=} V \cup \{x\}$, with same probability values for $y \in V$ and $p_x = 1$. □

Lemma 3.11. *We have* $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{ENV}(\mathcal{F})$ *for any class* $\mathcal{F}$ *that is closed under conditioning.*

Proof sketch. This is again a rather technical proof by polynomial interpolation, in which we curiously seem to need the assumption of closure under conditioning. □

And thus, combining with Theorem 3.1, we obtain:

Corollary 3.12. *We have* $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\mathcal{F}) \equiv_{\mathrm{P}} \mathsf{EV}(\mathcal{F})$ *for any class* $\mathcal{F}$ *that is closed under conditioning and that is also closed under either conjunctions or disjunctions with fresh variables.*

We leave as future work a more systematic (tedious) study of when $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EScore}_c(\mathcal{F})$ holds for other coefficient functions.

4 DD CIRCUITS

We now present algorithms to compute expected Shapley-like scores in polynomial time over deterministic and decomposable Boolean circuits. Since computing expected values can be done in linear time over such circuits, the fact that computing expected Shapley-like scores over them is in P is already implied by our main result, Theorem 3.1. We nevertheless present more direct algorithms for these circuits as they are easier and more natural to implement than the convoluted chain of reductions with various oracle calls and matrix inversions from the previous section. We will moreover use these algorithms in our experimental evaluation in Section 6. We start by defining what are these circuits.

Boolean circuits. Let C be a Boolean circuit over variables V, featuring $\wedge$, $\vee$, $\neg$, constant 0- and 1-gates, and variable gates (i.e., gates labeled by a variable in V), with $\wedge$- and $\vee$-gates having arbitrary fan-in greater than 1. We write $\mathsf{Vars}(C) \subseteq V$ the set of variables that occur in the circuit. The size $|C|$ of C is its number of wires. For a gate g of C, we write C_g the subcircuit rooted at g, and write $\mathsf{Vars}(g)$ its set of variables. An $\wedge$-gate g of C is *decomposable* if for every two input gates $g_1 \neq g_2$ of g, $\mathsf{Vars}(g_1) \cap \mathsf{Vars}(g_2) = \emptyset$. We call C *decomposable* if all $\wedge$-gates in it are. An $\vee$-gate g of C is *deterministic* if the Boolean functions captured by each pair of distinct input gates of g are pairwise disjoint; i.e., there is no assignment that satisfies them both. We call C *deterministic* if all $\vee$-gates in it are. A *deterministic and decomposable (d-D) Boolean circuit* [31] is a Boolean circuit that is both deterministic and decomposable. An $\vee$-gate g is *smooth* if for any input g' of g we have $\mathsf{Vars}(g) = \mathsf{Vars}(g')$, and C is smooth is all its $\vee$-gates are. We say that C is *tight* if it satisfies the following three conditions: (1) $\mathsf{Vars}(C) = V$; (2) C is smooth; and (3) every $\wedge$ and every $\vee$ gate of C has exactly two children.

Example 4.1. *The formula* $\varphi_{\mathrm{ex}} = (A \wedge a) \vee (C \wedge c)$ *from our running example is not a d-D when interpreted as a circuit, since the* $\vee$ *gate is not deterministic. An equivalent d-D is for example* $\neg\,[\neg(A \wedge a) \wedge \neg(C \wedge c)]$*, interpreted as a circuit in the natural way.*

The following is folklore.

Lemma 4.2. *Given as input a d-D circuit C over variables V, we can compute in $O(|C| \times |V|)$ a d-D circuit C' over V that is equivalent to C and that is tight.*

General polynomial-time algorithm. It is thus enough to explain how to compute expected Shapley-like scores for tight d-Ds; let C be such a circuit on variables V. We start from Equation (1), restated here for convenience:

$$\mathsf{EScore}_c(C,x) = p_x \sum_{k=0}^{|V|-1} \sum_{\ell=0}^{k} c(k+1,\ell)\big(\mathsf{ENV}_{k,\ell}(C_1) - \mathsf{ENV}_{k,\ell}(C_0)\big).$$

Here, C_1 (resp., C_0) is the circuit C in which we have replaced every variable gate labeled by x by a constant 1-gate (resp., a constant 0-gate). It can easily be checked that C_0 and C_1 are tight d-Ds over $V \setminus \{x\}$. Therefore, it suffices to compute, for an arbitrary tight d-D circuit C', the $\mathsf{ENV}_{\star,\star}$ quantities. To do this, we crucially need the determinism and decomposability properties. The idea is to compute corresponding quantities for each gate of the circuit, in a bottom-up fashion. This is similar to what is done in [6, Theorem 2] and [17, Proposition 4.4], but the expressions we obtain are more involved because we have a quadratic number of parameters for each gate of the circuit, as opposed to a linear number of such parameters in these earlier works. The resulting algorithm for the whole procedure is shown in Algorithm 1. Intuitively, the values $\beta^g_{k,\ell}$ correspond to the $\mathsf{ENV}_{k,\ell}$-values for the subcircuit of C_1 rooted at gate g, the values $\gamma^g_{k,\ell}$ correspond to those for C_0, and δ values are intermediate quantities that we have to compute. Thus:

Theorem 4.3. *Let c be a tractable coefficient function. Given a d-D circuit C on variables V, probability values p_y for $y \in V$, and $x \in V$, Algorithm 1 correctly computes $\mathsf{EScore}_c(C,x)$ in polynomial time. Moreover, if we ignore the cost of arithmetic operations, it is in time $O\left(|C| \times |V|^5 + \mathrm{T}_c(|V|) \times |V|^2\right)$ where $\mathrm{T}_c(\alpha)$ is the cost of computing the coefficient function on inputs $\leqslant \alpha$.*

We can show that the number of bits of numerators and denominators of the β, γ and δ values is roughly $O(b \times |V|)$, for b the bound on the number of bits of numerators and denominators of all p_y values. Therefore to obtain the exact complexity, without ignoring the time to perform additions and multiplications over such numbers, one has to add an $O(b \times |V|)$ multiplicative factor.

In the case where all probabilities are identical, we can obtain a lower complexity by reusing techniques from [17]:

Proposition 4.4. *Let c be a tractable coefficient function. Given a d-D C on variables V, a unique probability value $p = p_y$ for all $y \in V$, and $x \in V$, $\mathsf{EScore}_c(C,x)$ can be computed in time $O\left(|V|^2 \times (|C||V| + |V|^2 + \mathrm{T}_c(|V|))\right)$ assuming unit-cost arithmetic.*

Quadratic-time algorithm for expected Banzhaf score. For the expected Shapley value, instantiating Algorithm 1 with $c = c_{\mathrm{Shapley}}$ seems to be the best that we can do. For the expected Banzhaf value however, we can design a more efficient algorithm. We start from Equation (5), restated here in terms of circuits:

$$\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(C,x) = p_x[\mathsf{ENV}(C_1) - \mathsf{ENV}(C_0)]. \tag{6}$$

We can show that ENV can be computed in linear time over tight d-D circuits, thus obtaining a $O(|C| \times |V|)$ complexity for $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}$ over arbitrary d-D circuits by Lemma 4.2:

Theorem 4.5. *Given a d-D C on variables V, probability values p_y for $y \in V$, and $x \in V$, we can compute in time $O(|C| \times |V|)$ (ignoring the cost of arithmetic operations) the quantity $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(C,x)$.*

Algorithm 1: Expected Shapley-like scores for deterministic and decomposable Boolean circuits

Input : A d-D C on variables V, probability values p_y for $y \in V$, and $x \in V$.
Output: The value $\mathrm{EScore}_c(C, x)$

Let $n' = |V| - 1$ and let g_{out} be the output gate of C;
Make C tight using Lemma 4.2, and call it C again;
Compute values δ_k^g for every gate g in C and $k \in [n']$ by bottom-up induction on C as follows:
  if *g is a constant gate or a variable gate with* $\mathrm{Vars}(g) = \{x\}$ **then**
    $\delta_0^g \leftarrow 1$ and $\delta_k^g \leftarrow 0$ for $k \geqslant 1$;
  else if *g is a variable gate with* $\mathrm{Vars}(g) = \{y\}$ *and $y \neq x$* **then**
    $\delta_0^g \leftarrow 1 - p_y$, $\delta_1^g \leftarrow p_y$, and $\delta_k^g \leftarrow 0$ for $k \geqslant 2$;
  else if *g is a $\neg$-gate with input gate g'* **then**
    $\delta_k^g \leftarrow \delta_k^{g'}$ for $k \in [n']$;
  else if *g is an $\vee$-gate with input gates g_1, g_2* **then**
    $\delta_k^g \leftarrow \delta_k^{g_1}$ for $k \in [n']$;
  else if *g is an $\wedge$-gate with input gates g_1, g_2* **then**
    $\delta_k^g \leftarrow \sum_{k_1=0}^{k} \delta_{k_1}^{g_1} \delta_{k-k_1}^{g_2}$ for $k \in [n']$;
end
Compute values $\beta_{k,\ell}^g$ and $\gamma_{k,\ell}^g$ for every gate g in C and $k, \ell \in [n']$ by bottom-up induction on C:
  if *g is a constant a-gate ($a \in \{0, 1\}$)* **then**
    $\beta_{0,0}^g, \gamma_{0,0}^g \leftarrow a$, and $\beta_{k,\ell}^g, \gamma_{k,\ell}^g \leftarrow 0$ for $(k, \ell) \neq (0, 0)$;
  else if *g is a variable gate with* $\mathrm{Vars}(g) = \{x\}$ **then**
    $\beta_{0,0}^g \leftarrow 1$, $\gamma_{0,0}^g \leftarrow 0$, and $\beta_{k,\ell}^g, \gamma_{k,\ell}^g \leftarrow 0$ for $(k, \ell) \neq (0, 0)$;
  else if *g is a variable gate with* $\mathrm{Vars}(g) = \{y\}$ *and $y \neq x$* **then**
    $\beta_{0,0}^g, \beta_{1,0}^g, \gamma_{0,0}^g, \gamma_{1,0}^g \leftarrow 0$, $\beta_{1,1}^g, \gamma_{1,1}^g \leftarrow p_x$, and $\beta_{k,\ell}^g, \gamma_{k,\ell}^g \leftarrow 0$ for all other values of k, ℓ;
  else if *g is a $\neg$-gate with input gate g'* **then**
    $\beta_{k,\ell}^g \leftarrow \binom{k}{\ell} \delta_k^g - \beta_{k,\ell}^{g'}$ for $k, \ell \in [n']$;
    $\gamma_{k,\ell}^g \leftarrow \binom{k}{\ell} \delta_k^g - \gamma_{k,\ell}^{g'}$ for $k, \ell \in [n']$;
  else if *g is an $\vee$-gate with input gates g_1, g_2* **then**
    $\beta_{k,\ell}^g \leftarrow \beta_{k,\ell}^{g_1} + \beta_{k,\ell}^{g_2}$ for $k, \ell \in [n']$;
    $\gamma_{k,\ell}^g \leftarrow \gamma_{k,\ell}^{g_1} + \gamma_{k,\ell}^{g_2}$ for $k, \ell \in [n']$;
  else if *g is an $\wedge$-gate with input gates g_1, g_2* **then**
    $\beta_{k,\ell}^g \leftarrow \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \beta_{k_1,\ell_1}^{g_1} \times \beta_{k-k_1,\ell-\ell_1}^{g_2}$ for $k, \ell \in [n']$;
    $\gamma_{k,\ell}^g \leftarrow \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \gamma_{k_1,\ell_1}^{g_1} \times \gamma_{k-k_1,\ell-\ell_1}^{g_2}$ for $k, \ell \in [n']$;
end
return $p_x \sum_{k=0}^{n'} \sum_{\ell=0}^{k} c(k+1, \ell)\left(\beta_{k,\ell}^{g_{\mathrm{out}}} - \gamma_{k,\ell}^{g_{\mathrm{out}}}\right)$;

Table 2. A TID with two relations Students and Grades

Students

ID	Name	Age	Prob.	Prov.
01	Alice	20	0.4	A
02	Bob	21	0.3	B
03	Charlie	22	0.6	C
04	Danny	25	0.8	D

Grades

ID	Grade	Prob.	Prov.
01	86	0.5	a
02	80	0.2	b
03	92	0.8	c
04	99	0.9	d

Comparing to and recovering the algorithms of [17] and [2]. We end this section by discussing how this relates to the algorithms proposed in [17] and [2], respectively for Shapley and Banzhaf value computation in a deterministic (non-probabilistic) setting.

First, we note that we can specialize Algorithm 1 to the computation of Shapley values by setting all p_y to 1, which means that, when computing $\mathsf{ENV}_{k,\ell}$-values, we only need to consider the case where $k = n$ as all $Z \subseteq V$ with $|Z| < n$ have $\Pi_V(Z) = 0$. This leads to the following simplifications: δ_k^g values need not be computed as they are all 0's except for $\delta^g_{|\mathsf{Vars}(g)|} = 1$; similarly, $\beta^g_{k,\ell}$ and $\gamma^g_{k,\ell}$ values need only be computed when $k = |\mathsf{Vars}(g)|$, all other being set to 0. This simplifies the computation to remove a factor of $|V|^2$, and we essentially recover the algorithm described in [17]. Note that the final complexity obtained is $O(|C| \times |V|^3)$, which is better than the complexity from Proposition 4.4.

Second, we observe that the algorithm underlying Theorem 4.5 has the same complexity as the exact algorithm in [2] for computing (non-expected) Banzhaf values. We note that [2] considers *decomposition trees* instead of d-D circuits, but any decomposition tree is in fact a d-D circuit in disguise, since a decomposable OR of the form $A \lor B$ can be rewritten as $\neg(\neg A \land \neg B)$, with the AND being decomposable. Their algorithm works in linear time on decomposition trees that are tight (see their Section 3.1), hence we obtain the same complexity while solving a seemingly more general problem: we study the *expected* Banzhaf values (which degenerates to the non-expected setting when all probabilities are 1), and d-D circuits are more general than decomposition trees as they allow sharing of subexpressions (i.e., the circuit is a DAG instead of a tree).

5 PROBABILISTIC DATABASES

(Probabilistic) databases and queries. Let $\Sigma = \{R_1, \dots, R_n\}$ be a *signature*, consisting of *relation names* each with their associated *arity* $\mathrm{ar}(R_i) \in \mathbb{N}$, and Const be a set of *constants*. A *fact* over (Σ, Const) is a term of the form $R(a_1, \dots, a_{\mathrm{ar}(R)})$, for $R \in \Sigma$ and $a_i \in \mathrm{Const}$. A (Σ, Const)*-database* D (or simply a *database* D) is a finite set of facts over (Σ, Const). We assume familiarity with the most common classes of query languages and refer the reader to [1, 7] for the basic definitions. A *Boolean query* is a query q that takes as input a database D and outputs $q(D) \in \{0, 1\}$. If $q(\bar{x})$ is a query with free variables $\bar{x}$ and $\bar{t}$ is a tuple of constants of appropriate length, we denote by $q[\bar{x}/\bar{t}]$ the Boolean query defined by $q[\bar{x}/\bar{t}](D) = 1$ if and only if $\bar{t}$ is in the output of $q(\bar{x})$ on D. A *tuple-independent probabilistic database*, or *TID* for short, consists of a database D together with probability values p_f for every fact $f \in D$. For a Boolean query q and TID $\mathbf{D} = (D, (p_f)_{f \in D})$, the *probability that* $\mathbf{D}$ *satisfies* q, written $\Pr(\mathbf{D} \models q)$, is defined as $\Pr(\mathbf{D} \models q) \stackrel{\text{def}}{=} \sum_{D' \subseteq D \text{ s.t. } q(D')=1} \Pr(D')$, where $\Pr(D')$ is $\prod_{f \in D'} p_f \times \prod_{f \in D \setminus D'} (1 - p_f)$. For a fixed Boolean query q, we denote by $\mathrm{PQE}(q)$ the computational problem that takes as input a TID $\mathbf{D}$ and outputs $\Pr(\mathbf{D} \models q)$.

(Expected) Shapley-like scores. Let $c : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ be a coefficient function, q a Boolean query, D a database and $f \in D$ a fact. Following the literature [17, 29, 30], we define the *Shapley-like score*

with coefficients c of f in D with respect to q, or simply *score* when clear, by[1] $\mathrm{Score}_c(q, D, f) \stackrel{\text{def}}{=} \sum_{E \subseteq D \setminus \{f\}} c(|D|, |E|) \times \left[q(E \cup \{f\}) - q(E)\right]$. We denote by $\mathrm{Score}_c(q)$ the corresponding computational problem. Let now $\mathbf{D} = (D, (p_f)_{f \in D})$ be a TID and $f \in D$, and define the *expected score of f for q in* $\mathbf{D}$ as:

$$\mathrm{EScore}_c(q, \mathbf{D}, f) \stackrel{\text{def}}{=} \sum_{Z \subseteq D, f \in Z} (\Pr(Z) \times \mathrm{Score}_c(q, Z, f)),$$

where in $\mathrm{Score}_c(q, Z, f)$ we see q as a function from 2^Z to $\{0, 1\}$. Define the problem $\mathrm{EScore}_c(q)$ as expected. Note that all of this matches our definitions of expected values, Shapley-like and expected Shapley-like scores for Boolean functions.

Example 5.1. *Consider the TID shown in Table 2. The probability of each tuple is shown in the next-to-last column of each table, and a tuple identifier used as a* provenance token *in the last one. Consider the Boolean query q_{ex} over these tables expressed in SQL as: "*`SELECT DISTINCT 1 FROM Students s JOIN Grades g ON s.ID = g.ID WHERE Age < 23 AND Grade >= 85`*". Its* Boolean provenance *[35] can be computed to be exactly the DNF formula $\varphi_{\text{ex}} = (A \wedge a) \vee (C \wedge c)$ from the running example. Its probability is thus 0.584, as computed in Example 2.1. The expected Shapley value of every tuple contributing to making q_{ex} true is given in Table 1.*

As usual, if the query $q(\bar{x})$ has free variables, for a tuple $\bar{t}$ of appropriate length we can define similarly, for $f \in D$, the expected score of f by using the Boolean query $q[\bar{x}/\bar{t}]$ in the above definition. This score then represents the contribution of f to $\bar{t}$ potentially being in the query result (one might in particular be interested in explaining why a tuple is *not* in the query result).

Then Theorem 3.1 directly translates into this setting of Shapley-like scores of facts over probabilistic databases:

Theorem 5.2. *We have* $\mathrm{EScore}_c(q) \leqslant_{\mathrm{P}} \mathrm{PQE}(q)$ *for any tractable coefficient function c and any Boolean query q.*

Proof. It suffices to instantiate Theorem 3.1 with the set of Boolean functions $\mathcal{F}_q \stackrel{\text{def}}{=} \{\varphi_{q,D} \mid D \text{ is a database}\}$, where $\varphi_{q,D} : 2^D \to \{0, 1\}$, the *Boolean provenance* [35] of query q over D, is defined by $\varphi_{q,D}(D') \stackrel{\text{def}}{=} q(D')$ for $D' \subseteq D$. Here, it is implicit that the Boolean function $\varphi_{q,D}$ is represented by D itself. □

In the case of the Shapley value, we can even get a full equivalence from Corollary 3.7:

Corollary 5.3. *We have* $\mathrm{EScore}_{c_{\text{Shapley}}}(q) \equiv_{\mathrm{P}} \mathrm{PQE}(q)$ *for any Boolean query q.*

Proof. One direction is Theorem 5.2. The other direction comes from Corollary 3.7, using the same $\mathcal{F}_q$ as in the proof of Theorem 5.2, and noticing that $\mathcal{F}_q$ is reasonable because the query is fixed. □

In particular, this gives a dichotomy on $\mathrm{EScore}_{c_{\text{Shapley}}}(q)$ between P and #P-hard for unions of conjunctive queries (UCQs), or more generally for queries that are *closed under homomorphisms* [3, 14]. This should be compared with the corresponding result for (non-expected) $\mathrm{Score}_{c_{\text{Shapley}}}(q)$, where a dichotomy is currently only known for *self-join–free conjunctive queries* [22, 29].

For Banzhaf values, even though $\mathrm{EScore}_{c_{\text{Banzhaf}}}(q) \leqslant_{\mathrm{P}} \mathrm{PQE}(q)$ is true for any Boolean query by Theorem 5.2, it is not clear how to obtain the other direction from Proposition 3.9: indeed, the class $\mathcal{F}_q$ from above has in general no reason to be closed under conditioning nor under taking

[1] We point out that the facts of D are traditionally partitioned between *endogenous* and *exogenous* facts, but we do not make this distinction in our work. This is to simplify the presentation, as usual definitions would extend in a straightforward manner.

Table 3. Provenance computation time, knowledge compilation time and method, and total Shapley/Banzhaf computation time for all output tuples and all facts, in the deterministic case and for expected values in the probabilistic case. The queries are the same TPC-H queries used in [17] (without the LIMIT operator used in [17]). All times reported are in seconds.

TPC-H query	# Output tuples	Provenance time	Compilation time	Compilation method	Avg d-D #gates	Shapley time Determ.	Shapley time Expect.	Banzhaf time Expect.
3	11620	2.125 ± 0.029	1.226 ± 0.008	33% dec., 67% tree dec.	22	0.762 ± 0.005	1.758 ± 0.011	0.468 ± 0.002
5	5	1.117 ± 0.022	0.044 ± 0.000	100% tree dec.	1115	0.766 ± 0.001	40.910 ± 0.447	0.191 ± 0.000
7	4	1.215 ± 0.053	0.017 ± 0.000	100% tree dec.	750	0.269 ± 0.001	9.381 ± 0.020	0.085 ± 0.000
10	1783	1.229 ± 0.023	0.018 ± 0.000	98% dec., 2% tree dec.	5	0.023 ± 0.000	0.037 ± 0.000	0.015 ± 0.000
11	61	0.174 ± 0.023	0.001 ± 0.000	87% dec., 13% tree dec.	7	0.001 ± 0.000	0.002 ± 0.000	0.001 ± 0.000
16	466	0.247 ± 0.027	0.084 ± 0.000	100% tree dec.	65	0.159 ± 0.001	0.455 ± 0.005	0.094 ± 0.001
18	91159	2.711 ± 0.298	0.749 ± 0.005	97% dec., 3% tree dec.	4	0.655 ± 0.002	1.008 ± 0.007	0.490 ± 0.003
19	56	1.223 ± 0.239	0.000 ± 0.000	100% dec.	3	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

conjunctions or disjunctions with fresh variables. Yet, we mention that [29, Proposition 5.6] shows a dichotomy for (non-expected) $\mathrm{Score}_{c_{\mathrm{Banzhaf}}}(q)$ for self-join–free CQs: the tractable queries are the *hierarchical* queries, while for non-hierarchical queries the problem is #P-hard. This dichotomy then directly extends to $\mathrm{EScore}_{c_{\mathrm{Banzhaf}}}(q)$: the tractable side follows from our Theorem 5.2 because $\mathrm{PQE}(q)$ is in P for hierarchical queries, while the hardness result is inherited by Fact 2.5 from the hardness of $\mathrm{Score}_{c_{\mathrm{Banzhaf}}}(q)$ shown in [29].

Provenance computation and compilation. Unfortunately, not all queries are tractable for probabilistic query evaluation. When faced with an intractable query, another approach is to use the *intensional method* [38], which is to compute and compile the Boolean provenance of the query on the database in a formalism from knowledge compilation that enjoys tractable computation of expected values, such as d-D circuits. When the provenance has been computed as a d-D circuit, we can use the results from Section 4 to compute the expected Shapley-like scores. This is the route that we take in the next section to compute these scores in practice.

6 IMPLEMENTATION AND EXPERIMENTS

In this section, we experimentally show that the computation of expected Shapley-like scores is feasible in practice on some realistic queries over probabilistic databases, despite the #P-hardness of the problem in general and the high $O(|C| \times |V|^5)$ upper bound (see Theorem 4.3) on the complexity of Algorithm 1 for d-Ds. The objective is not to provide a comprehensive experimental evaluation but to simply validate that algorithms presented in this work have reasonable complexity for practical applications.

Implementation. We rely on, and have extended, ProvSQL [36][2], an open-source PostgreSQL extension that computes (between other things) the Boolean provenance of a query over a database. We let ProvSQL compute the Boolean provenance of SQL queries over relational databases as a Boolean circuit, and have extended this system to add the following features: (1) We compile Boolean provenance into a d-D in the simple but common *decomposable* case where every $\land$- or $\lor$-gate g is decomposable, i.e., for every two inputs g_1 and g_2 to g, $\mathrm{Vars}(g_1) \cap \mathrm{Vars}(g_2) = \emptyset$. Note that, as we have already observed in Section 4, a decomposable $\lor$-gate of the form $A \lor B$ can be rewritten, using De Morgan's laws, into a decomposable $\land$-gate. (2) For cases where this is not possible, we attempt to compile Boolean provenance into a d-D by computing, if possible, a *tree decomposition* of the circuit of treewidth $\leqslant 10$, and by then following the construction detailed in

[2] https://github.com/PierreSenellart/provsql

[4, Section 5.1] to turn any Boolean circuit into a d-D in linear time when the treewidth is fixed. (3) Otherwise, we default to ProvSQL's default compilation of circuits into d-Ds, which amounts to coding the circuit as a CNF using the Tseitin transformation [39] and then calling an external knowledge compiler, d4 [26]. (4) We have implemented *directly within ProvSQL* Algorithm 1 to compute expected Shapley values on d-Ds, its simplification when all p_y are set to 1 detailed at the end of Section 4, as well as the algorithm to compute expected Banzhaf values in the proof of Theorem 4.5. They are all implemented with floating-point numbers.

In particular, this approach is not restricted to queries on the tractable side of the dichotomy of [14]. In addition, we benefit from the fact that, since late 2021, ProvSQL stores the provenance circuit in main memory, which speeds provenance computation up (earlier versions stored the provenance circuit within the database, on disk).

Experiment setup. Following [2, 17], we used the TPC-H 1 GB benchmark, with standard generated data and 8 standard TPC-H queries adapted to remove nesting and aggregation, as provided by the authors of [17][3]. We use the exact same queries as in [17], except that the `LIMIT` operator used for the experiments of that paper was removed, to obtain a larger and more realistic benchmark (we end up with 105 154 output tuples for these 8 queries). Probabilities were drawn uniformly at random for all facts.

Experiments were run on a desktop Linux PC with Xeon W3550 2.80GHz CPU, 64 GB RAM (8 GB of which were made available for PostgreSQL's shared buffers), running version 14.9 of PostgreSQL; our code has been incorporated in ProvSQL, and we ran the latest version of ProvSQL as of December 2023[4]. Data for PostgreSQL is stored on standard magnetic hard drives in RAID 1.

Results. We show in Table 3 results of these experiments. For each query, we report: the number of output tuples; the total time required by ProvSQL to evaluate the query and compute the provenance representation of every output tuple; the total time required by the compilation of the Boolean provenance circuits of all query results to d-Ds; the method used to produce these d-Ds[5]; the average number of gates of the resulting d-Ds; the total time needed to compute (expected) Shapley values of all query outputs for all relevant facts[6] in the deterministic case (where all probabilities are set to 1) and in the probabilistic case; the same for expected Banzhaf values in the probabilistic case[7]. All times are in seconds, repeated over 20 runs of each query, with the mean and standard deviation shown. To avoid caching provenance across multiple runs or multiple queries, the ProvSQL circuit was reset each time and PostgreSQL restarted.

We have the following observations regarding the experimental results (also compare with the results from Table 1 of [17]).

(1) ProvSQL is able to compute in a reasonable amount of time (at most a couple of seconds) the output of all queries, along with their Boolean provenance as a circuit; this contrasts with the results of [17] where provenance computation time could take up to 6 hours for query 3, even when limited to output only 100 tuples; we assume this is the result of recent ProvSQL optimizations and in-memory storage of the Boolean provenance circuit.

[3]https://github.com/navefr/ShapleyForDbFacts, archived as swh:1:rev:7a46a9fa381194097b81a7aae705d396872b26e3

[4]Archived on Software Heritage as swh:1:rev:bba98e1d96af4c5a25ac672a2f67ea44ed869f8f

[5]A different method might be used for each output tuple; we report the proportion of "dec." when the obtained circuit was already decomposable; and of "tree dec." when we used the tree decomposition approach; none of the circuits produced required using an external knowledge compiler.

[6]By *relevant facts*, we mean here the facts that appear in the provenance circuits. Indeed, the other facts have a score of zero.

[7]There is of course virtually no difference between computing deterministic and expected Banzhaf values, since the algorithm we use (that of Theorem 4.5), is the same.

(2) Compilation to a d-D takes a time that is comparable to provenance computation, and uses a combination of interpreting the circuit as a decomposable one and the tree decomposition algorithm of [4]; using an external knowledge compiler, which was what was done in [17], is never required. Note that compilation is much faster than reported in [17] (remember that the times in [17] need to be multiplied by the number of output tuples, whereas we report the sum of all compilation times); the provenance circuits for queries 5 and 7 could not even be compiled to a d-D in [17].

(3) The total time required for deterministic Shapley value computation is of similar magnitude as query evaluation as well and comparable to those reported in [17] (except in one case, for query 19, where numbers reported in [17] are abnormally high).

(4) Computing expected Shapley values in a probabilistic setting using Algorithm 1 incurs a higher cost, but remains more practical in practice than the high theoretical complexity of this algorithm may suggest – the maximum time required is for query 5, with 41 seconds to compute the expected Shapley values over five d-Ds whose average size is over a thousand of gates.

(5) The total time required for Banzhaf value computation is significantly lower than that of deterministic Shapley value computation, especially for circuits with large numbers of gates or variables, as consistent with the established complexities.

(6) Though query evaluation and provenance computation can be marred with significant performance differences from one run to the next (due to disk caching, interaction with PostgreSQL, etc.), there is little variation of the performance of knowledge compilation and expected Shapley-like value computation.

To summarize, the experiments validate the practicality of the algorithms presented in this paper for computation of expected Shapley-like scores over probabilistic databases.

7 RELATED WORK

Probabilistic scores from Game Theory. While Shapley value [37] is one of the most popular measures of player contribution in a game, it does not take into account the probabilistic aspect of some of these games. To address this issue, a long line of work [11–13, 25, 28, 32, 41] defines and studies various ways in which scoring mechanisms can be incorporated in the setting of probabilistic games. For instance, the expected Shapley value that we study in this paper has been defined in [11] (see Definition 9), where it is shown that, just like the non-probabilistic Shapley value, it is the only scoring mechanism that satisfies a natural set of axioms over probabilistic games (see Theorems 3 and 4 of [11]). In these works, authors mainly focus on analysing the characteristics of such scoring mechanisms, in particular in terms of axioms they verify. To the best of our knowledge, no complexity results or algorithms are proposed, whereas this is precisely what we study in this paper.

Shapley-like scores. The authors of [17, 29, 30, 34] study the complexity of the problems $\mathsf{Score}_{c_{\mathrm{Shapley}}}$ and $\mathsf{Score}_{c_{\mathrm{Banzhaf}}}$, over Boolean functions or, most often, instantiated in the setting of relational databases. See [9] for a survey for databases. In particular, [17] shows that, for any query, the problem $\mathsf{Score}_{c_{\mathrm{Shapley}}}$ can be reduced in polynomial time to probabilistic query evaluation for the same query. The authors of [22] show an analogous result for Boolean functions, also obtaining the other direction of the reduction (under some assumptions). Formally, define the *model counting problem* for class $\mathcal{F}$ of Boolean functions as follows: given as input $\varphi \in \mathcal{F}$ over variables V, compute $\#\varphi \stackrel{\text{def}}{=} \{Z \in 2^V \mid \varphi(Z) = 1\}$. They then show (we refer to their article for the definition of closure under OR-substitutions):

THEOREM 7.1 ([22, COROLLARY 7]). *We have* $\mathsf{Score}_{c_{\mathrm{Shapley}}}(\mathcal{F}) \equiv_{\mathrm{P}} \mathsf{MC}(\mathcal{F})$ *for any class* $\mathcal{F}$ *that is closed under OR-substitutions.*

Notice the resemblance between, on the one side, these last two results that we mention, and on the other side our Corollaries 3.7 and 5.3. The difference is that we study the *expected* Shapley values. There is a priori no reason for the tractable cases to be the same as the non-expected variant: indeed, the counting (or probabilistic) version of a problem is often much harder than the decision one — for instance probabilistic query evaluation is often intractable for queries for which regular evaluation is easy.[8] By our results, this phenomenon does not occur for $\mathrm{EScore}_{c_{\mathrm{Shapley}}}$. Since $\mathrm{EScore}_{c_{\mathrm{Shapley}}}$ strictly generalizes $\mathrm{Score}_{c_{\mathrm{Shapley}}}$ (by Fact 2.5), the reduction from $\mathrm{EScore}_{c_{\mathrm{Shapley}}}$ to EV is more challenging, and indeed one can check that our polynomial interpolation proofs are more involved than, say, [17, Proposition 3.1]. On the other hand, our life is made easier to prove the other direction of these equivalences. This explains why we do not need the assumption of closure under OR-substitutions in Corollary 3.7, and this is also what allows us to obtain a *complete* equivalence to probabilistic query evaluation in Corollary 5.3, *no matter the Boolean query*, whereas in the case of non-expected Shapley values this is only known for self-join–free CQs (see also [10] which makes progress on this problem, but does not solve it completely yet). As for algorithms for d-D circuits [2, 17], we refer to the end of Section 4.

SHAP-score. The authors of [5, 6, 16, 40] study the complexity of computing the SHAP-score. In particular [16, 40] show that it is equivalent to computing expected values, and [5, 6] propose polynomial-time algorithms for d-D circuits. Thus, the landscape is similar to what we obtain here. However, there is to the best of our knowledge no formal connection between the SHAP-score and the expected scores that we study here: in a nutshell, we compute the expected Shapley value where the game function is the Boolean function φ, whereas the SHAP score is computing the Shapley value where the game function is a conditional expectation of φ. Hence, the two sets of results seem to be independent.

8 CONCLUSION

We proposed the new notion of expected Shapley-like scores for Boolean functions, proved that computing these scores can always be reduced in polynomial-time to the well-studied problem of computing expected values, and that these two problems are often even equivalent (under commonplace assumptions). We designed algorithms for deterministic and decomposable Boolean circuits and implemented them in the setting of probabilistic databases, where our preliminary experimental results show that these scoring mechanisms could actually be used in practice. We leave as future work the study of approximation algorithms for this new notion. In particular, it is known that $\mathrm{Score}_{c_{\mathrm{Shapley}}}(q)$ has a *fully polynomial-time randomized scheme* [19] whenever q is a UCQ [9], and one could study whether this stays true for the probabilistic variant. Still we note that, since the reduction from Fact 2.5 is parsimonious, we inherit the few hardness results of the non-probabilistic setting, such as those of [34] for conjunctive queries with negations.

ACKNOWLEDGMENTS

This research is part of the program DesCartes (https://descartes.cnrsatcreate.cnrs.fr/) and is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) program.

[8]So one could say that expected Shapley scores are to Shapley scores what probabilistic query evaluation is to non-probabilistic query evaluation.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. 2023. Banzhaf Values for Facts in Query Answering. arXiv preprint arXiv:2308.05588.

[3] Antoine Amarilli. 2023. Uniform Reliability for Unbounded Homomorphism-Closed Graph Queries. In *ICDT (LIPIcs, Vol. 255)*. 14:1–14:17.

[4] Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. 2020. Connecting knowledge compilation classes and width parameters. *Theory of Computing Systems* 64 (2020), 861–914.

[5] Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. 2021. The Tractability of SHAP-Score-Based Explanations for Classification over Deterministic and Decomposable Boolean Circuits. In *AAAI*. 6670–6678.

[6] Marcelo Arenas, Pablo Barceló, Leopoldo E Bertossi, and Mikaël Monet. 2023. *J. Mach. Learn. Res.* 24, 63 (2023), 1–58.

[7] Marcelo Arenas, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris. 2022. Database Theory. Work in progress, latest version at https://github.com/pdm-book/community.

[8] John F Banzhaf III. 1964. Weighted voting doesn't work: A mathematical analysis. *Rutgers L. Rev.* 19 (1964), 317.

[9] Leopoldo Bertossi, Benny Kimelfeld, Ester Livshits, and Mikaël Monet. 2023. The Shapley value in database management. *SIGMOD Record* 52, 2 (2023), 6–17.

[10] Meghyn Bienvenu, Diego Figueira, and Pierre Lafourcade. 2024. When is Shapley Value Computation a Matter of Counting?. In *PODS*.

[11] Surajit Borkotokey, Sujata Gowala, and Rajnish Kumar. 2023. The Expected Shapley value on a class of probabilistic games. arXiv preprint arXiv:2308.03489.

[12] Francesc Carreras and Maria Albina Puente. 2015. Coalitional multinomial probabilistic values. *European Journal of Operational Research* 245, 1 (2015), 236–246.

[13] Francesc Carreras and María Albina Puente. 2015. Multinomial probabilistic values. *Group decision and negotiation* 24, 6 (2015), 981–991.

[14] Nilesh Dalvi and Dan Suciu. 2013. The dichotomy of probabilistic inference for unions of conjunctive queries. *Journal of the ACM (JACM)* 59, 6 (2013), 1–87.

[15] John Deegan and Edward W Packel. 1978. A new index of power for simple n-person games. *International Journal of Game Theory* 7 (1978), 113–123.

[16] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. 2021. On the Tractability of SHAP Explanations. In *AAAI*. 6505–6513.

[17] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. 2022. Computing the Shapley Value of Facts in Query Answering. In *SIGMOD Conference*. 1570–1583.

[18] Manfred J Holler and Edward W Packel. 1983. Power, luck and the right index. *Zeitschrift für Nationalökonomie* 43 (1983), 21–29.

[19] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* 43 (1986), 169–188.

[20] Ron J Johnston. 1977. National sovereignty and national power in European institutions. *Environment and Planning A* 9, 5 (1977), 569–577.

[21] Ronald John Johnston. 1978. On the measurement of power: Some reactions to Laver. *Environment and Planning A* 10, 8 (1978), 907–914.

[22] Ahmet Kara, Dan Olteanu, and Dan Suciu. 2024. From Shapley Value to Model Counting and Back. In *PODS*.

[23] Adam Karczmarz, Tomasz P. Michalak, Anish Mukherjee, Piotr Sankowski, and Piotr Wygocki. 2022. Improved feature importance computation for tree models based on the Banzhaf value. In *UAI*.

[24] Werner Kirsch and Jessica Langner. 2010. Power indices and minimal winning coalitions. *Social Choice and Welfare* 34, 1 (2010), 33–46.

[25] Maurice Koster, Sascha Kurz, Ines Lindner, and Stefan Napel. 2017. The prediction value. *Social Choice and Welfare* 48 (2017), 433–460.

[26] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *IJCAI*. 667–673.

[27] Annick Laruelle. 1999. *On the choice of a power index.* Technical Report. Instituto Valenciano de Investigaciones Económicas.

[28] Annick Laruelle and Federico Valenciano. 2008. Potential, value, and coalition formation. *Transactions in Operations Research* 16 (2008), 73–89.

[29] Ester Livshits, Leopoldo Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. The Shapley value of tuples in query answering. *Logical Methods in Computer Science* 17 (2021).

[30] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2020. The Shapley value of tuples in query answering. In *ICDT*, Vol. 155. 20:1–20:19.

[31] Mikaël Monet. 2020. Solving a Special Case of the Intensional vs Extensional Conjecture in Probabilistic Databases. In *PODS*. 149–163.

[32] Guillermo Owen. 1972. Multilinear extensions of games. *Management Science* 18, 5-part-2 (1972), 64–79.

[33] J Scott Provan and Michael O Ball. 1983. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.* 12, 4 (1983), 777–788.

[34] Alon Reshef, Benny Kimelfeld, and Ester Livshits. 2020. The impact of negation on the complexity of the Shapley value in conjunctive queries. In *PODS*. 285–297.

[35] Pierre Senellart. 2017. Provenance and Probabilities in Relational Databases: From Theory to Practice. *SIGMOD Record* 46, 4 (2017).

[36] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *Proc. VLDB Endow.* 11, 12 (2018), 2034–2037.

[37] Lloyd S. Shapley et al. 1953. A value for n-person games. (1953).

[38] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases.* Morgan & Claypool.

[39] G Tseitin. 1968. On the complexity of derivation in propositional calculus. *Studies in Constrained Mathematics and Mathematical Logic* (1968).

[40] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. 2022. On the tractability of SHAP explanations. *Journal of Artificial Intelligence Research* 74 (2022), 851–886.

[41] Robert J Weber. 1988. Probabilistic values for games. *The Shapley Value. Essays in Honor of Lloyd S. Shapley* (1988), 101–119.

Received December 2023; revised February 2024; accepted March 2024

A PROOFS FOR SECTION 3 (EQUIVALENCE WITH EXPECTED VALUES)

LEMMA 3.10. *We have* $\mathsf{ENV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\mathcal{F})$ *for any* $\mathcal{F}$ *closed under conjunctions (resp., disjunctions) with fresh variables.*

PROOF. Let us first show that for $\varphi' : 2^{V'} \to \{0,1\}$ and probability values p_y for $y \in V'$ and $x \in V'$ we have the equation claimed in the proof sketch, restated here:

$$\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\varphi', x) = p_x[\mathsf{ENV}(\varphi'_{+x}) - \mathsf{ENV}(\varphi'_{-x})].$$

The derivation is similar to that of Lemma 3.2, but simpler. Observe that $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(\varphi, x) = A - B$, where

$$A = \sum_{\substack{Z \subseteq V' \\ x \in Z}} \Pi_{V'}(Z) \sum_{E \subseteq Z \setminus \{x\}} \varphi'(E \cup \{x\}) \qquad B = \sum_{\substack{Z \subseteq V' \\ x \in Z}} \Pi_{V'}(Z) \sum_{E \subseteq Z \setminus \{x\}} \varphi'(E).$$

Let us focus on A. Letting $V'' \overset{\mathrm{def}}{=} V' \setminus \{x\}$, notice that these are the variables over which φ'_{+x} is defined. Letting $n \overset{\mathrm{def}}{=} |V''|$, we have

$$A = \sum_{\substack{Z \subseteq V' \\ x \in Z}} \Pi_{V'}(Z) \sum_{E \subseteq Z \setminus \{x\}} \varphi'_{+x}(E) = p_x \sum_{Z \subseteq V''} \Pi_{V''}(Z) \sum_{E \subseteq Z} \varphi'_{+x}(E) = p_x \mathsf{ENV}(\varphi'_{+x}).$$

We can do the same for B to obtain $B = p_x \mathsf{ENV}(\varphi'_{-x})$, hence the equation.

We now prove Lemma 3.10 in the case that $\mathcal{F}$ is closed under conjunctions with fresh variables. Let then $\varphi : 2^V \to \{0,1\}$, and probabilities p_y for $y \in V$. We want to compute $\mathsf{ENV}(\varphi)$. Since $\mathcal{F}$ is closed under conjunctions with fresh variables, let $x \notin V$ and compute a representation of $\varphi' \overset{\mathrm{def}}{=} \varphi \wedge x$ in $\mathcal{F}$. We call the oracle to $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}$ on φ' with same probabilities for $y \in V$ and with $p_x \overset{\mathrm{def}}{=} 1$. By the above equation (with $V' = V \cup \{x\}$) this immediately gives us $\mathsf{ENV}(\varphi)$ and concludes.

For the case when $\mathcal{F}$ is closed under disjunctions with fresh variables we do the same but with $\varphi' \overset{\mathrm{def}}{=} \varphi \vee x$: now by Equation (5) the oracle call returns $\left[\sum_{Z \subseteq V} \Pi_V(Z) \sum_{E \subseteq Z} 1\right] - \mathsf{ENV}(\varphi)$, which is equal to $\left[\sum_{Z \subseteq V} \Pi_V(Z) 2^{|Z|}\right] - \mathsf{ENV}(\varphi)$. We conclude the proof by showing that the first term is equal to $\prod_{y \in V}(1+p_y)$, which can be computed in polynomial time, hence we can indeed recover $\mathsf{ENV}(\varphi)$. Indeed, let $n \overset{\mathrm{def}}{=} |V|$, and order the variables of V arbitrarily as $y_1, \ldots, y_n$. For $i \in [n]$, define $V_i \overset{\mathrm{def}}{=} \{y_j \mid 1 \leqslant j \leqslant i \in [n]\}$ (note that $V_0 = \emptyset$), and $d_i \overset{\mathrm{def}}{=} \sum_{Z \subseteq V_i} \Pi_{V_i}(Z) 2^{|Z|}$. Observe that the quantity that we want is d_n. But it is clear that $d_0 = 1$ and that $d_i = \sum_{\substack{Z \subseteq V_i \\ y_i \notin Z}} \Pi_{V_i}(Z) 2^{|Z|} + \sum_{\substack{Z \subseteq V_i \\ y_i \in Z}} \Pi_{V_i}(Z) 2^{|Z|} = (1 - p_{y_i}) d_{n-1} + p_{y_i} \times d_{n-1} \times 2 = (1 + p_{y_i}) d_{n-1}$ for $1 \leqslant i \leqslant n$, which concludes. □

LEMMA 3.11. *We have* $\mathsf{EV}(\mathcal{F}) \leqslant_{\mathrm{P}} \mathsf{ENV}(\mathcal{F})$ *for any class* $\mathcal{F}$ *that is closed under conditioning.*

PROOF. Let $\varphi \in \mathcal{F}$ over variables V with $n \overset{\mathrm{def}}{=} |V|$ and probability values p_x for each $x \in V$; we want to compute $\mathsf{EV}(\varphi)$. We use polynomial interpolation to compute all the values $\mathsf{EV}_j(\varphi)$ for $j \in [n]$, after which we can simply return $\sum_{j=0}^{n} \mathsf{EV}_j(\varphi) = \mathsf{EV}(\varphi)$.

Without loss of generality, we can assume that $p_x < 1$ for all $x \in V$. Indeed, if there is x such that $p_x = 1$, we consider $V' = V \setminus \{x\}$ and $\varphi' = \varphi_{+x}$. Then $\mathsf{EV}_j(\varphi) = \mathsf{EV}_{j-1}(\varphi')$ for any $j \geqslant 1$ and $\mathsf{EV}_0(\varphi) = 0$. This is indeed without loss of generality because $\mathcal{F}$ is closed under conditioning, so that φ_{+x} is in $\mathcal{F}$.

Let $M \overset{\mathrm{def}}{=} \max_{x \in V} p_x < 1$. Let $z_0, \ldots, z_n$ be $n+1$ distinct rational values in $(0, 1-M)$. For $i \in [n]$ and $x \in V$, we define this time $p_x^{z_i} \overset{\mathrm{def}}{=} \frac{z_i p_x}{1-p_x}$, and define Π^{z_i} and $\mathsf{EV}^{z_i}(\varphi)$ as expected. Again, these are all valid probability mappings. Define $C \overset{\mathrm{def}}{=} \prod_{x \in V}(1 - p_x)$. We will show that we have $\mathsf{ENV}^{z_i}(\varphi) = \frac{1}{C} \sum_{j=0}^{n} z_i^j \mathsf{EV}_j(\varphi)$, which allows us to conclude as in the proof of Lemma 3.3. Indeed:

$$\mathsf{ENV}^{z_i}(\varphi) = \sum_{Z \subseteq V} \Pi_V^{z_i}(Z) \sum_{E \subseteq Z} \varphi(E)$$

$$= \sum_{E \subseteq V} \varphi(E) \sum_{E \subseteq Z \subseteq V} \Pi_V^{z_i}(Z) = \sum_{E \subseteq V} \varphi(E) \sum_{E \subseteq Z \subseteq V} \prod_{x \in Z} p_x^{z_i} \prod_{x \in V \setminus Z} (1 - p_x^{z_i})$$
$$= \frac{1}{C} \sum_{E \subseteq V} \varphi(E) \sum_{E \subseteq Z \subseteq V} \prod_{x \in Z} z_i p_x \prod_{x \in V \setminus Z} (1 - p_x - z_i p_x)$$
$$= \frac{1}{C} \sum_{E \subseteq V} \varphi(E) \prod_{x \in E} z_i p_x \prod_{x \in V \setminus E} [(z_i p_x) + (1 - p_x - z_i p_x)]$$
$$= \frac{1}{C} \sum_{E \subseteq V} \varphi(E) \prod_{x \in E} z_i p_x \prod_{x \in V \setminus E} (1 - p_x) = \frac{1}{C} \sum_{j=0}^{n} \sum_{\substack{E \subseteq V \\ |E| = j}} z_i^j \Pi_V(E) \varphi(E) = \frac{1}{C} \sum_{j=0}^{n} z_i^j \mathsf{EV}_j(\varphi). \qquad \square$$

B PROOFS FOR SECTION 4 (DD CIRCUITS)

B.1 Proof of Theorem 4.3

THEOREM 4.3. *Let c be a tractable coefficient function. Given a d-D circuit C on variables V, probability values p_y for $y \in V$, and $x \in V$, Algorithm 1 correctly computes $\mathrm{EScore}_c(C, x)$ in polynomial time. Moreover, if we ignore the cost of arithmetic operations, it is in time $O\left(|C| \times |V|^5 + \mathrm{T}_c(|V|) \times |V|^2\right)$ where $\mathrm{T}_c(\alpha)$ is the cost of computing the coefficient function on inputs $\leqslant \alpha$.*

Proving Theorem 4.3, as explained in Section 4, boils down to showing how we can compute, given a tight d-D circuit, the $\mathsf{ENV}_{\star,\star}$ quantities. We then show:

PROPOSITION B.1. *Given as input a tight d-D circuit C' on variables V' and probability values p_y for $y \in V'$ we can compute all the values $\mathsf{ENV}_{k,\ell}(C)$ for $k, \ell \in [|V'|]$ in $O(|C'| \times |V'|^4)$, ignoring the cost of arithmetic operations.*

Recall that this will be instantiated with $C' = C_0$ and $C' = C_1$ for the circuits C_0 and C_1 from Section 4 (which should not be confused with circuit C of that section). Also note that, even though by Equation (1) we only need to compute the values for $k \geqslant \ell$ (since they are zero when $k > \ell$), we still do as if we wanted to naively compute them all. This allows us to obtain cleaner expressions, in which the ranges for the sums are easier to read. Let us define $n' \stackrel{\text{def}}{=} |V'|$.

We first explain how to compute an intermediate quantity that will be needed later.

DEFINITION B.2. *For a gate $g \in C'$ and integer $k \in [n']$, define $\delta_k^g \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq \mathsf{Vars}(g) \\ |Z| = k}} \Pi_{\mathsf{Vars}(g)}(Z)$. (Note that $\delta_k^g = 0$ when $k > \mathsf{Vars}(g')$.)*

Notice that δ_k^g only depends on the "structure" of the circuit, but not on its semantics.

LEMMA B.3. *We can compute in $O(|C'| \times n'^2)$ all quantities δ_k^g, ignoring the cost of arithmetic operations.*

PROOF. We compute them by bottom-up induction on C'.

- **Constant gates.** Let g be a constant gate. Then $\mathsf{Vars}(g) = \emptyset$, so $\delta_k^g = 0$ for $k \geqslant 1$, and $\delta_0^g = 1$ (indeed $\Pi_\emptyset(\emptyset) = 1$ since this is the neutral element of multiplication).
- **Input gates.** Let g be an input gate, with variable y. Then $\mathsf{Vars}(g) = \{y\}$, so $\delta_k^g = 0$ for $k \geqslant 2$, while $\delta_0^g = \Pi_{\mathsf{Vars}(g)}(\emptyset) = 1 - p_y$ and $\delta_1^g = \Pi_{\mathsf{Vars}(g)}(\{y\}) = p_y$.
- **Negation gates.** Let g be a $\neg$-gate with input g'. Notice that $\mathsf{Vars}(g) = \mathsf{Vars}(g')$. So we have $\delta_k^g = \delta_k^{g'}$ for all $k \in [n']$ and we are done since the values $\delta_k^{g'}$ have already been computed inductively.
- **Deterministic smooth $\vee$-gates.** Let g be a smooth deterministic $\vee$-gate with inputs g_1, g_2. Since g is smooth we have $\mathsf{Vars}(g) = \mathsf{Vars}(g_1) = \mathsf{Vars}(g_2)$. In particular we have $\delta_k^g = \delta_k^{g_1}$ for all $k \in [n']$ and we are done.
- **Decomposable $\wedge$-gates.** Let g be a decomposable $\wedge$-gate with inputs g_1, g_2. Notice that $\mathsf{Vars}(g) = \mathsf{Vars}(g_1) \cup \mathsf{Vars}(g_2)$ with the union being disjoint. We can then decompose Z into a "left" part $Z_1 \subseteq$

$\mathsf{Vars}(g_1)$ of size $k_1 \in \{0, \dots, k\}$ and a "right" part $Z_2 \subseteq \mathsf{Vars}(g_2)$ of size $k - k_1$. We then have:

$$\delta_k^g = \sum_{\substack{Z \subseteq \mathsf{Vars}(g) \\ |Z|=k}} \Pi_{\mathsf{Vars}(g)}(Z) = \sum_{k_1=0}^{k} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1) \\ |Z_1|=k_1}} \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2) \\ |Z_2|=k-k_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1)\Pi_{\mathsf{Vars}(g_2)}(Z_2)$$

$$= \sum_{k_1=0}^{k} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1) \\ |Z_1|=k_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2) \\ |Z_2|=k-k_1}} \Pi_{\mathsf{Vars}(g_2)}(Z_2) = \sum_{k_1=0}^{k} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1) \\ |Z_1|=k_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1)\delta_{k-k_1}^{g_2}$$

$$= \sum_{k_1=0}^{k} \delta_{k-k_1}^{g_2} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1) \\ |Z_1|=k_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) = \sum_{k_1=0}^{k} \delta_{k_1}^{g_1}\delta_{k-k_1}^{g_2},$$

and we are done.

The complexity of every step is $O(n')$ except for $\land$-gates where the complexity is $O(n'^2)$; each step needs to be repeated for every gate of C', which gives the stated complexity. This concludes the proof of Lemma B.3. □

We next define $\mathsf{ENV}_{\star,\star}$-quantities for all gates of the circuit C'.

Definition B.4. *For a gate $g \in C'$ and $k, \ell \in [n']$, define*

$$\alpha_{k,\ell}^g \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq \mathsf{Vars}(g) \\ |Z|=k}} \sum_{\substack{E \subseteq Z \\ |E|=\ell}} \Pi_{\mathsf{Vars}(g)}(Z)C'_g(E).$$

If we can show that we can compute all quantities $\alpha_{k,\ell}^g$ in the required complexity then we are done: indeed, we can then take g to be the output gate of C', which gives us the quantities $\mathsf{ENV}_{k,\ell}(C')$ that we wanted. We show just that in the next lemma.

Lemma B.5. *We can compute in $O(|C'| \times n'^4)$ all the quantities $\alpha_{k,\ell}^g$.*

Proof. This is again done by bottom-up induction on C'.

Constant gates. Let g be a constant gate. Then $\mathsf{Vars}(g) = \emptyset$, so $\alpha_{k,\ell}^g = 0$ when $(k, \ell) \neq (0, 0)$, and $\alpha_{0,0}^g = 1$ if g is a constant 1-gate and $\alpha_{0,0}^g = 0$ if it is a constant 0-gate.

Input gates. Let g be an input gate, with variable y. Then $\mathsf{Vars}(g) = \{y\}$, so all values other than $\alpha_{0,0}^g$, $\alpha_{1,0}^g$ and $\alpha_{1,1}^g$ are null, and one can easily check that $\alpha_{0,0}^g = \alpha_{1,0}^g = 0$ and $\alpha_{1,1}^g = p_y$.

Negation gates. Let g be a $\neg$-gate with input g'. Notice that $\mathsf{Vars}(g) = \mathsf{Vars}(g')$ and that $C'_g(E) = 1 - C'_{g'}(E)$ for any $E \subseteq \mathsf{Vars}(g)$. We have

$$\alpha_{k,\ell}^g = \sum_{\substack{Z \subseteq \mathsf{Vars}(g) \\ |Z|=k}} \sum_{\substack{E \subseteq Z \\ |E|=\ell}} \Pi_{\mathsf{Vars}(g)}(Z)(1 - C'_{g'}(E)) = \left[\binom{k}{\ell} \sum_{\substack{Z \subseteq \mathsf{Vars}(g) \\ |Z|=k}} \Pi_{\mathsf{Vars}(g)}(Z)\right] - \alpha_{k,\ell}^{g'}$$

$$= \binom{k}{\ell}\delta_k^g - \alpha_{k,\ell}^{g'},$$

and we are done thanks to Lemma B.3.

Deterministic smooth $\lor$-gates. Let g be a smooth deterministic $\lor$-gate with inputs g_1, g_2. Since g is smooth we have $\mathsf{Vars}(g) = \mathsf{Vars}(g_1) = \mathsf{Vars}(g_2)$, and since it is deterministic we have $C'_g(E) = C'_{g_1}(E) + C'_{g_1}(E)$ for any $E \subseteq \mathsf{Vars}(g)$. Therefore we obtain $\alpha_{k,\ell}^g = \alpha_{k,\ell}^{g_1} + \alpha_{k,\ell}^{g_2}$ and we are done.

Decomposable $\land$-gates. Let g be a decomposable $\land$-gate with inputs g_1, g_2. Notice that $\mathsf{Vars}(g) = \mathsf{Vars}(g_1) \cup \mathsf{Vars}(g_2)$ with the union being disjoint, and that $C'_g(E) = C'_{g_1}(E \cap \mathsf{Vars}(g_1)) \times C'_{g_2}(E \cap \mathsf{Vars}(g_2))$ and $\Pi_{\mathsf{Vars}(g)}(Z) = \Pi_{\mathsf{Vars}(g_1)}(Z \cap \mathsf{Vars}(g_1)) \times \Pi_{\mathsf{Vars}(g_2)}(Z \cap \mathsf{Vars}(g_2))$ for any $Z, E \subseteq \mathsf{Vars}(g)$. We decompose the summations over Z and E as we did in the proof of Lemma B.3 for $\land$-gates. For readability we use colors to point out which parts of the expressions are modified or moved around.

$$
\begin{aligned}
\alpha^g_{k,\ell} &= \sum_{\substack{Z \subseteq \mathsf{Vars}(g)\\ |Z|=k}} \sum_{\substack{E \subseteq Z\\ |E|=\ell}} \Pi_{\mathsf{Vars}(g)}(Z) C'_g(E)\\
&= \sum_{k_1=0}^{k} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2)\\ |Z_2|=k-k_1}} \sum_{\substack{E \subseteq Z_1 \cup Z_2\\ |E|=\ell}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \Pi_{\mathsf{Vars}(g_2)}(Z_2) C'_g(E)\\
&= \sum_{k_1=0}^{k} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2)\\ |Z_2|=k-k_1}} \sum_{\ell_1=0}^{k_1} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \sum_{\substack{E_2 \subseteq Z_2\\ |E_2|=\ell-\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \Pi_{\mathsf{Vars}(g_2)}(Z_2) C'_{g_1}(E_1) C'_{g_2}(E_2)\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2)\\ |Z_2|=k-k_1}} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \sum_{\substack{E_2 \subseteq Z_2\\ |E_2|=\ell-\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \Pi_{\mathsf{Vars}(g_2)}(Z_2) C'_{g_1}(E_1) C'_{g_2}(E_2)\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2)\\ |Z_2|=k-k_1}} \sum_{\substack{E_2 \subseteq Z_2\\ |E_2|=\ell-\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \Pi_{\mathsf{Vars}(g_2)}(Z_2) C'_{g_1}(E_1) C'_{g_2}(E_2)\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) C'_{g_1}(E_1) \sum_{\substack{Z_2 \subseteq \mathsf{Vars}(g_2)\\ |Z_2|=k-k_1}} \sum_{\substack{E_2 \subseteq Z_2\\ |E_2|=\ell-\ell_1}} \Pi_{\mathsf{Vars}(g_2)}(Z_2) C'_{g_2}(E_2)\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) C'_{g_1}(E_1) \alpha^{g_2}_{k-k_1,\ell-\ell_1}\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \alpha^{g_2}_{k-k_1,\ell-\ell_1} \sum_{\substack{Z_1 \subseteq \mathsf{Vars}(g_1)\\ |Z_1|=k_1}} \sum_{\substack{E_1 \subseteq Z_1\\ |E_1|=\ell_1}} \Pi_{\mathsf{Vars}(g_1)}(Z_1) C'_{g_1}(E_1)\\
&= \sum_{k_1=0}^{k} \sum_{\ell_1=0}^{k_1} \alpha^{g_1}_{k_1,\ell_1} \times \alpha^{g_2}_{k-k_1,\ell-\ell_1}.
\end{aligned}
$$

and we are done.

The complexity is given by that of the step for $\neg$- and $\land$-gates, which are the most costly at $O(n'^4)$ (recall that computing $\binom{k}{\ell}$ is in $O(k \times \ell)$), which we need to multiply by the size of the circuit. This concludes the proof of Lemma B.5. ◻

In Algorithm 1, the β values are the α values for C_1 and the γ values are the α values for C_0, and they are computed in a single pass over the circuit C instead of first computing C_1 and C_0 and making passes over these two circuits. Therefore, Algorithm 1 is correct. To obtain the final complexity, we need to add the cost of line 32, which is in $O(n'^2 \times \mathrm{T}(n'))$ ignoring the cost of arithmetic operations, and remember that $|C_1|$ and $|C_0|$ are in $O(|C| \times |V|)$ by Lemma 4.2.

What remains to argue is that the number of bits (numerator and denominator) of all the α and δ values stays polynomial. But for $\alpha^g_{k,\ell}$ for instance we have

$$\alpha^g_{k,\ell} \stackrel{\text{def}}{=} \sum_{\substack{Z\subseteq \mathsf{Vars}(g)\\ |Z|=k}} \sum_{\substack{E\subseteq Z\\ |E|=\ell}} \Pi_{\mathsf{Vars}(g)}(Z) C'_g(E) \leqslant 2^{2|V|} \max_{Z\subseteq \mathsf{Vars}(g)} \Pi_{\mathsf{Vars}(g)}(Z).$$

If the number of bits of all numerators and denominators of all p_x is bounded by b, then the numerator of $\alpha^g_{k,\ell}$ is bounded by $2^{2|V|}2^{b|V|} = 2^{(b+2)|V|}$, so indeed have a polynomial number of bits for the numerators, and similar reasoning works for denominators and for the δ values.

B.2 Proof of Proposition 4.4

Proposition 4.4. *Let c be a tractable coefficient function. Given a d-D C on variables V, a unique probability value $p = p_y$ for all $y \in V$, and $x \in V$, $\mathsf{EScore}_c(C, x)$ can be computed in time $O\left(|V|^2 \times (|C||V| + |V|^2 + \mathrm{T}_c(|V|))\right)$ assuming unit-cost arithmetic.*

Proof. To prove Proposition 4.4 we consider a d-D circuit C over variables V with $n = |V|$. For a variable $x \in V$,

$$\begin{aligned}
\mathsf{EScore}_c(C, x) &= \sum_{\substack{Z\subseteq V\\ x\in Z}} \Pi_V(Z) \times \mathrm{Score}_c(C, Z, x)\\
&= \sum_{\substack{Z\subseteq V\\ x\in Z}} \Pi_V(Z) \sum_{E\subseteq Z\setminus\{x\}} c(|Z|, |E|) \times [C(E\cup\{x\}) - C(E)]\\
&= \sum_{E\subseteq V\setminus\{x\}} \left[\sum_{E\subseteq Z\subseteq V\setminus\{x\}} c(|Z|+1, |E|) \times p_x \times \Pi_V(Z)\right] [C(E\cup\{x\}) - C(E)]\\
&= \sum_{\ell=0}^{|V|-1} \sum_{\substack{E\subseteq V\setminus\{x\}\\ |E|=\ell}} \sum_{k=\ell}^{|V|-1} \left[\sum_{\substack{E\subseteq Z\subseteq V\setminus\{x\}\\ |Z|=k}} c(k+1, \ell) \times p^{k+1}(1-p)^{n-k-1}\right] [C(E\cup\{x\}) - C(E)]\\
&= \sum_{\ell=0}^{|V|-1} \sum_{k=\ell}^{|V|-1} \left[\binom{n-1-\ell}{k-\ell} \times c(k+1, \ell) \times p^{k+1}(1-p)^{n-k-1}\right] \sum_{\substack{E\subseteq V\setminus\{x\}\\ |E|=\ell}} [C(E\cup\{x\}) - C(E)]\\
&= \sum_{\ell=0}^{|V|-1} [\#\mathsf{SAT}_\ell(C_1) - \#\mathsf{SAT}_\ell(C_0)] \sum_{k=\ell}^{|V|-1} \left[\binom{n-1-\ell}{k-\ell} \times c(k+1, \ell) \times p^{k+1}(1-p)^{n-k-1}\right]
\end{aligned}$$

where we set C_1 and C_0 as usual and $\#\mathsf{SAT}_\ell(C')$ is the number of satisfying valuations of size ℓ of the circuit C'. Using the techniques of [17] (in particular, Lemma 4.5 of this paper), we can show that all the $\#\mathsf{SAT}\ell(C')$ values for a tight circuit C' over variables $|V'|$ can be computed in $O(|C'| \times |V'|^2)$. So, to compute all $\#SAT_\ell(C_0)$, we first need to make it tight (in $O(|C| \times |V|)$) and then we have a cost of $O(|C| \times |V|^3)$.

Now, to compute the rest of the sum, we need to compute for every ℓ and k a binomial coefficient in $O(n^2)$, a value of the coefficient function in $O(\mathrm{T}_c(n))$ and perform the other multiplications in $O(1)$ assuming unit cost arithmetic. We obtain thus an algorithm in $O\left(|C| \times |V|^3 + |V|^2 \times (|V|^2 + \mathrm{T}_c(|V|))\right)$. □

B.3 Proof of Theorem 4.5

Theorem 4.5. *Given a d-D C on variables V, probability values p_y for $y \in V$, and $x \in V$, we can compute in time $O(|C| \times |V|)$ (ignoring the cost of arithmetic operations) the quantity $\mathsf{EScore}_{c_{\mathrm{Banzhaf}}}(C, x)$.*

Proof. As argued in Section 4, we need only to prove, thanks to Equation (6), that ENV can be computed in linear time for tight d-D circuits. Let C' be a tight d-D over variables V' and p_x probability values for all $x \in V'$. We want to compute $\mathsf{ENV}(\varphi) \stackrel{\text{def}}{=} \sum_{Z\subseteq V'} \Pi_{V'}(Z) \sum_{E\subseteq Z} C'(E)$. (Recall that this will be instantiated with $C' = C_1$ and $C' = C_0$) from Equation (6).) We do this again by bottom-up induction on the circuit, computing the corresponding quantities for very gate. Formally, for a gate g of C', define: $\alpha^g \stackrel{\text{def}}{=} \sum_{Z\subseteq \mathsf{Vars}(g)} \Pi_{\mathsf{Vars}(g)}(Z) \sum_{E\subseteq Z} C'_g(E)$. Notice that we want α^g for g the output gate of C'. We show next how this can be done.

Constant gates. Let g be a constant gate. Then $\mathsf{Vars}(g) = \emptyset$, so α^g equals 1 if g is a constant 1-gate and 0 if it is a constant 0-gate.

Input gates. Let g be an input gate, with variable y. Then $\mathsf{Vars}(g) = \{y\}$, so $\alpha^g = p_y$.

Negation gates. Let g be a $\neg$-gate with input g'.
Then $C'_g(E) = 1 - C'_{g_1}(E)$, therefore $\alpha^g = \left[\sum_{Z\subseteq \mathsf{Vars}(g)} \Pi_{\mathsf{Vars}(g)}(Z) \sum_{E\subseteq Z} 1\right] - \alpha^{g'}$. We have already observed in the proof of Lemma 3.10 that the first term is equal to $\prod_{y\in\mathsf{Vars}(g)}(1+p_y)$, therefore we obtain $\alpha^g = \left[\prod_{y\in\mathsf{Vars}(g)}(1+p_y)\right] - \alpha^{g'}$.

Deterministic smooth $\vee$-gates. Let g be a smooth deterministic $\vee$-gate with inputs g_1, g_2. Since g is smooth we have $\mathsf{Vars}(g) = \mathsf{Vars}(g_1) = \mathsf{Vars}(g_2)$, and since it is deterministic we have $C'_g(E) = C'_{g_1}(E) + C'_{g_2}(E)$. Therefore $\alpha^g = \alpha^{g_1} + \alpha^{g_2}$.

Decomposable $\wedge$-gates. Let g be a decomposable $\wedge$-gate with inputs g_1, g_2. We decompose the sum similarly to what we did in the proof of Theorem 4.3:

$$\begin{aligned}
\alpha^g &= \sum_{Z\subseteq \mathsf{Vars}(g)} \Pi_{\mathsf{Vars}(g)}(Z) \sum_{E\subseteq Z} C'_g(E)\\
&= \sum_{Z_1\subseteq \mathsf{Vars}(g_1)} \sum_{Z_2\subseteq \mathsf{Vars}(g_2)} \Pi_{\mathsf{Vars}(g)}(Z_1\cup Z_2) \sum_{E_1\subseteq Z_1} \sum_{E_2\subseteq Z_2} C'_g(E_2\cup E_2)\\
&= \sum_{Z_1\subseteq \mathsf{Vars}(g_1)} \sum_{Z_2\subseteq \mathsf{Vars}(g_2)} \Pi_{\mathsf{Vars}(g_1)}(Z_1)\Pi_{\mathsf{Vars}(g_2)}(Z_2) \sum_{E_1\subseteq Z_1} \sum_{E_2\subseteq Z_2} C'_{g_1}(E_1)C'_{g_2}(E_2)\\
&= \sum_{Z_1\subseteq \mathsf{Vars}(g_1)} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \sum_{E_1\subseteq Z_1} C'_{g_1}(E_1) \sum_{Z_2\subseteq \mathsf{Vars}(g_2)} \Pi_{\mathsf{Vars}(g_2)}(Z_2) \sum_{E_2\subseteq Z_2} C'_{g_2}(E_2)\\
&= \sum_{Z_1\subseteq \mathsf{Vars}(g_1)} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \sum_{E_1\subseteq Z_1} C'_{g_1}(E_1)\alpha^{g_2}\\
&= \alpha^{g_2} \sum_{Z_1\subseteq \mathsf{Vars}(g_1)} \Pi_{\mathsf{Vars}(g_1)}(Z_1) \sum_{E_1\subseteq Z_1} C'_{g_1}(E_1)\\
&= \alpha^{g_2}\alpha^{g_1}.
\end{aligned}$$

This concludes the proof, as all of this can be done in $O(|C'|)$, ignoring the cost of arithmetic operations (and, in any case, the number of bits stays polynomial). For it to be true for negation gates, we need to compute $\prod_{y\in\mathsf{Vars}(g)}(1+p_y)$ for every gate, which can also be done during the bottom-up processing of the circuit. □

B.4 Complexity in the Case where All Probabilities are 1

As discussed at the end of Section 4, when all probabilities are set to 1, we recover the algorithm of [17] for non-probabilistic Shapley value computation. We briefly discuss its precise complexity.

In that setting, as discussed, we do not need to compute δ^g_k values (line 3–14) so we only need to discuss the cost of computing $\beta^g_{k,\ell}$ and $\gamma^g_{k,\ell}$ (lines 15–31) on the one hand, and of line 32 on the other hand. Further, recall that only the setting where $k = |\mathsf{Vars}(g)|$ is relevant. This means that the main loop to compute β and γ values is run $|\mathsf{Vars}(g)|$ times instead of $|\mathsf{Vars}(g)|^2$ times, and furthermore, that in the case where g is an $\wedge$-gate, its computation involves a single sum, as we can set k_1 to be $|\mathsf{Vars}(g_1)|$ (and thus k_2 to be $|\mathsf{Vars}(g_2)|$) on lines 29 and 30. Finally, on line 32, similarly, we only have one sum operator as $\beta^{g_{\mathrm{out}}}_{k,\ell}$ and $\gamma^{g_{\mathrm{out}}}_{k,\ell}$ are zero when $k \neq |\mathsf{Vars}(g_{\mathrm{out}})|$.

Remember that since the circuit needs to be made tight, its size is $O(|C| \times |V|)$. We therefore have for complexity: $O\left(|C| \times |V| \times |V|^2 + |V| \times \mathrm{T}_{c_{\mathrm{Shapley}}}(|V|)\right) = O\left(|C| \times |V|^3 + |V| \times |V|^2\right) = O\left(|C| \times |V|^3\right)$.