Banzhaf Values for Facts in Query Answering

Omer Abramovich
Tel Aviv University
Tel Aviv, Israel

Daniel Deutch
Tel Aviv University
Tel Aviv, Israel

Nave Frost
eBay Research
Netanya, Israel

Ahmet Kara
University of Zurich
Zurich, Switzerland

Dan Olteanu
University of Zurich
Zurich, Switzerland

ABSTRACT

Quantifying the contribution of database facts to query answers has been studied as means of explanation. The Banzhaf value, originally developed in Game Theory, is a natural measure of fact contribution, yet its efficient computation for select-project-join-union queries is challenging. In this paper, we introduce three algorithms to compute the Banzhaf value of database facts: an exact algorithm, an anytime deterministic approximation algorithm with relative error guarantees, and an algorithm for ranking and top-k. They have three key building blocks: compilation of query lineage into an equivalent function that allows efficient Banzhaf value computation; dynamic programming computation of the Banzhaf values of variables in a Boolean function using the Banzhaf values for constituent functions; and a mechanism to compute efficiently lower and upper bounds on Banzhaf values for any positive DNF function.

We complement the algorithms with a dichotomy for the Banzhaf-based ranking problem: given two facts, deciding whether the Banzhaf value of one is greater than of the other is tractable for hierarchical queries and intractable for non-hierarchical queries.

We show experimentally that our algorithms significantly outperform exact and approximate algorithms from prior work, most times up to two orders of magnitude. Our algorithms can also cover challenging problem instances that are beyond reach for prior work.

PVLDB Reference Format:
Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. Banzhaf Values for Facts in Query Answering. PVLDB, XX(X): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Omer-Abramovich/AdaBan.

1 INTRODUCTION

Explaining the answer to a relational query is a fundamental problem in data management [10–13, 25, 27, 31, 38, 39]. One main approach to explanation is based on attribution, where each tuple from the input database is assigned a score reflecting its contribution to the query answer. A measure that quantifies the contribution of a fact to the query answer is the *Banzhaf* value [7, 44]. It has found applications in various domains. Most prominently, it is used as a measure of voting power in the analysis of voting in the Council of the European Union [53]. It was shown to provide more robust data valuation across subsequent runs of stochastic gradient descent than alternative scores such as the Shapley value [54]. It is used for understanding feature importance in training tree ensemble models, where it is preferable over the Shapley value as it can be computed faster and it can be numerically more robust [28]. In Banzhaf random forests [51], it is used to evaluate the importance of each feature across several possible feature sets used for training random forests. It is also used as a measure of risk analysis in terrorist networks [22].

This paper starts a systematic investigation of both theoretical and practical facets of three computational problems for Banzhaf-based fact attribution in query answering: exact computation, approximation, and ranking. Our contribution is fourfold.

1. Exact Banzhaf Computation. We introduce ExaBan, an algorithm that computes the exact Banzhaf scores for the contributions of facts in the answers to positive relational queries (Select-Project-Join-Union in SQL). Its input is the query lineage, which is a Boolean positive function whose variables are the database facts. Its output is the Banzhaf value of each variable. It relies on the compilation of the lineage into a d-tree, a data structure previously used for efficient computation in probabilistic databases [24]. The compilation recursively decomposes the function into a disjunction or conjunction of (independent) functions over disjoint sets of variables, or into a disjunction of (mutually exclusive) functions with disjoint sets of satisfying variable assignments. Our use of d-tree is justified by the observation that if we have the Banzhaf values for independent or mutually exclusive functions, we can then compute in linear time the Banzhaf values for the conjunction or disjunction of these functions. In our experiments with over 300 queries and three widely-known datasets (TPC-H, IMDB, Academic), ExaBan consistently outperforms the state-of-the-art solution [19], which we adapted to compute Banzhaf instead of Shapley values. The performance gap is up to two orders of magnitude on those workloads for which the prior work finishes within one hour, while ExaBan also succeeds to terminate within one hour for 41.7%-99.2% (for the different datasets) of the cases for which prior work failed.

2. Anytime Deterministic Banzhaf Approximation. We also introduce AdaBan, an algorithm that computes approximate Banzhaf values of facts. AdaBan is an *approximation algorithm* in the sense that it computes an interval $[\ell, u]$ that contains the exact Banzhaf value of a given fact. It is *deterministic* in the sense that the exact

Proceedings of the VLDB Endowment, Vol. XX, No. X ISSN 2150-8097.
doi:XX.XX/XXX.XX

value is guaranteed to be contained in the approximation interval[1]. It is *anytime* in the sense that it can be stopped at any time and provides a correct approximation interval for the exact Banzhaf value. Each decomposition step cannot enlarge the approximation interval. Given any error $\epsilon \in [0, 1]$ and an approximation interval $[\ell, u]$ computed by AdaBan, if $(1 - \epsilon)u \leq (1 + \epsilon)\ell$, then any value in the interval $[(1 - \epsilon)u, (1 + \epsilon)\ell]$ is a (relative) ϵ-approximation of the exact Banzhaf value. AdaBan provably reaches the desired approximation error[2] after a number of steps. *In the worst case*, any deterministic approximation algorithm needs exponentially many steps in the number of facts[3]. Yet in practical settings including our experiments, AdaBan's behavior is much better than the theoretical worst case. For instance, AdaBan takes up to one order of magnitude less time than ExaBan to reach $\epsilon = 0.1$.

AdaBan has two main ingredients: (1) the incremental decomposition of the query lineage into a d-tree, and (2) a mechanism to compute lower and upper bounds on the Banzhaf value for a variable in any positive DNF function.

The first ingredient builds on ExaBan. Unlike ExaBan, AdaBan does not exhaustively compile the lineage into a d-tree before computing the Banzhaf values. Instead, it intertwines the incremental compilation of the lineage with the computation of approximation intervals for the Banzhaf value. If an interval reaches the desired approximation error, then AdaBan stops the computation; otherwise, it further expands the d-tree. Thus, it may finish after much fewer decomposition steps than ExaBan. This is the main reason behind AdaBan's speedup over ExaBan, as reported in our experiments.

The second ingredient is the computation of approximation intervals. AdaBan can derive lower and upper bounds on the Banzhaf value for any variable in positive DNF functions at the leaves of a d-tree. While the bounds may be arbitrarily loose, they can be computed in time linear in the function size. Given approximation intervals at the leaves of a d-tree, AdaBan computes an approximation interval for the entire d-tree, and thus for the query lineage.

3. Banzhaf-based Ranking and Top-k Facts. We also introduce IchiBan, an algorithm that ranks facts and selects the top-k facts based on their Banzhaf values. IchiBan is a natural generalization of AdaBan: It incrementally refines the approximation intervals for the Banzhaf values of all facts until the intervals are separated or become the same Banzhaf value. Two intervals are separated when the lower bound of one becomes larger than the upper bound of the other. IchiBan also supports approximate ranking, where the approximation intervals are ordered by their middle points.

The top-k problem is to find k facts whose Banzhaf values are the largest across all facts in the database. To obtain such top-k facts, we proceed similarly to ranking. We start by incrementally tightening the approximation intervals for the Banzhaf values of all facts. Once the approximation interval for a fact is below the lower bound of at least k other facts, we discard that fact from our computation. Alternatively, we can stop the execution when the overlapping approximation intervals reach a given error, at the cost of allowing approximate top-k.

Our experiments show that when IchiBan is prompted to produce approximate ranking or top-k results, in practice it achieves near-perfect results. This is true even in cases where previous work [19], which gives no top-k correctness guarantees, produces inaccurate results. Furthermore, IchiBan is by up to an order of magnitude faster than computing the exact Banzhaf values.

4. Dichotomy for Banzhaf-based Ranking. Our fourth contribution is a dichotomy for the complexity of the ranking problem in case of self-join-free Boolean conjunctive queries: Given two facts, deciding whether the Banzhaf value of one fact is greater than the Banzhaf value of the other fact is tractable (i.e., in polynomial time) for hierarchical queries and intractable (i.e., not in polynomial time) for non-hierarchical queries. This dichotomy coincides with the dichotomy for the exact computation of Banzhaf values [35]. This is surprising, since ranking facts does not require in principle their exact Banzhaf values but just an approximation sufficient to rank them (as done in IchiBan). The tractability for ranking is implied by the tractability for exact computation (since we can first compute the exact Banzhaf values of all facts in polynomial time and then sort the facts by their Banzhaf values), yet the intractability for ranking is *not* implied by the intractability for exact computation. Our intractability result relies on the conjecture that an efficient (i.e., polynomial in the inverse of the error and in the graph size) approximation for counting the independent sets in a bipartite graph is not possible [14, 21].

The paper is organized as follows. Sec. 2 introduces the notions of Banzhaf value, Boolean functions, relational databases and queries, and query lineage. Sec. 3 introduces the algorithms for exact and approximate computation of Banzhaf values. Sec. 4 introduces our algorithm for Banzhaf-based top-k and ranking and our dichotomy for ranking. Sec. 5 details our experimental findings. Sec. 6 contrasts our contributions to prior work on approximate computation and attribution by Shapley values. Sec. 7 concludes. Full proofs of formal statements are deferred to the extended version of this paper [2].

2 PRELIMINARIES

We denote by $\mathbb{N}$ the set of natural numbers including 0. For $n \in \mathbb{N}$, we denote $[n] \overset{\text{def}}{=} \{1, 2, \ldots, n\}$. In case $n = 0$, we have $[n] = \emptyset$.

Boolean Functions. Given a set $\boldsymbol{X}$ of Boolean variables, a *Boolean function* over $\boldsymbol{X}$ is a function $\varphi : \boldsymbol{X} \rightarrow \{0, 1\}$ defined recursively as: a variable in $\boldsymbol{X}$; a conjunction $\varphi_1 \wedge \varphi_2$ or a disjunction $\varphi_1 \vee \varphi_2$ of two Boolean functions φ_1 and φ_2; or a negation $\neg(\varphi_1)$ of a Boolean function φ_1. A *literal* is a variable or its negation. The size of φ, denoted by $|\varphi|$, is the number of symbols in φ. For a variable $x \in \boldsymbol{X}$ and a constant $b \in \{0, 1\}$, $\varphi[x := b]$ denotes the function that results from replacing x by b in φ. An *assignment* for φ is a function $\theta : \boldsymbol{X} \rightarrow \{0, 1\}$. We also denote an assignment θ by the set $\{x \mid \theta(x) = 1\}$ of its variables mapped to 1. The Boolean value of φ under the assignment θ is denoted by $\varphi[\theta]$. If $\varphi[\theta] = 1$, then θ is a *satisfying assignment* or *model* of φ. We denote the number of models of φ by $\#\varphi$. A function is *positive* if its literals are positive.

Definition 2.1 (Banzhaf Value of Boolean Variable). Given a Boolean function φ over $\boldsymbol{X}$, the *Banzhaf value* of a variable $x \in \boldsymbol{X}$ in φ is:

[1]This is in stark contrast to randomized approximation schemes, where the exact value is contained in the approximation interval with a probability $\delta \in (0, 1)$.
[2]In contrast, the randomized approximation schemes cannot guarantee that by executing one more iteration step the approximation interval does not enlarge.
[3]Otherwise, it would contradict the hardness of exact Banzhaf value computation [35] that is attained by AdaBan for $\epsilon = 0$.

$$Banzhaf(\varphi, x) \overset{\text{def}}{=} \sum_{Y \subseteq X \setminus \{x\}} \varphi[Y \cup \{x\}] - \varphi[Y] \quad (1)$$

Normalized versions of the Banzhaf value $Banzhaf(\varphi, x)$ can be obtained by dividing it by (1) the number $2^{|X|-1}$ of all possible assignments of the variables in X except x (*Penrose–Banzhaf power*), or by (2) the sum $\sum_{y \in X} Banzhaf(\varphi, y)$ of the Banzhaf values of all variables (*Penrose–Banzhaf index*) [30]. In this paper, we use the definition in Eq. (1), but our results immediately apply to the normalized versions as well.

Example 2.2. Consider the Boolean function $\varphi = x_1 \lor (x_2 \land \neg x_3)$. The following table shows all possible assignments Y for φ and the Boolean value of φ under Y. For simplicity, we identify variables by their indices, e.g., x_1 is identified by 1.

θ	$\emptyset$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
$\varphi[\theta]$	0	1	1	0	1	1	0	1

Recall the set notation for an assignment; e.g., $Y = \{2, 3\}$ means that $x_2 = x_3 = 1$ and $x_1 = 0$. To compute the Banzhaf value of x_1, we sum up the differences $\varphi[Y \cup \{x_1\}] - \varphi[Y]$ for all $Y \subseteq \{x_2, x_3\}$:

$$\begin{aligned} Banzhaf(\varphi, x_1) = {} & \big(\varphi[\{1\}] - \varphi[\emptyset]\big) + \big(\varphi[\{1,2\}] - \varphi[\{2\}]\big) + \\ & \big(\varphi[\{1,3\}] - \varphi[\{3\}]\big) + \varphi[\{1,2,3\}] - \varphi[\{2,3\}] \\ = {} & 1 + 0 + 1 + 1 = 3 \end{aligned}$$

Similarly, $Banzhaf(\varphi, x_2) = 1$ and $Banzhaf(\varphi, x_3) = -1$. The latter is negative, because x_3 appears negated in φ.

An alternative characterization of the Banzhaf value, adapted from prior work [35], is the difference between the numbers of the models of the function where x is set to 1 and respectively to 0.

Proposition 2.3. *The following holds for any Boolean function φ over X and variable $x \in X$:*

$$Banzhaf(\varphi, x) = \#\varphi[x := 1] - \#\varphi[x := 0] \quad (2)$$

Example 2.4. Consider again the function $\varphi = x_1 \lor (x_2 \land \neg x_3)$ from Example 2.2. We compute the Banzhaf value of the variable x_1 using Eq. (2). The function $\varphi[x_1 := 1] = 1 \lor (x_2 \land \neg x_3)$ evaluates to 1 under any assignment for the variables x_2 and x_3, hence $\#\varphi[x_1 := 1] = 4$. The only model of the function $\varphi[x_1 := 0] = 0 \lor (x_2 \land \neg x_3)$ is $\{x_2\}$, hence $\#\varphi[x_1 := 0] = 1$. We obtain $Banzhaf(\varphi, x_1) = 4 - 1 = 3$, which is the same as the value computed in Example 2.2.

Databases. Let a countably infinite set Dom of constants. A *database schema* S is a finite set of *relation symbols*, with each relation symbol R having a fixed *arity*. A database D over S associates with each relation symbol R of arity k a finite k-ary relation $R^D \subseteq \text{Dom}^k$. We identify a database D with its finite set of *facts* $R(c_1, \ldots, c_k)$, stating that the k-ary relation R^D contains the tuple $(c_1, \ldots, c_k)$. Following prior work, we assume that the database is partitioned into a set D_n of *endogenous* and a set D_x of *exogenous* facts [35].

Queries. A *conjunctive query* (CQ) over database schema S has the form: $Q = \exists Y \bigwedge_{j \in [m]} R_j(Y_j)$, where R_j is a relation symbol from S, each Y_j is a tuple of variables and constants, and Y is a set of variables included in $\bigcup_{j \in [m]} Y_j$. To distinguish variables in queries from variables in Boolean functions, we denote the query variables by uppercase letters and the function variables by lowercase letters. All variables in Y are *bound*, whereas the variables included in $\bigcup_{j \in [m]} Y_j$ but not in Y are *free*. Each $R_j(Y_j)$ is an *atom* of Q. We denote by $at(X)$ the set of atoms with the query variable X. A *Boolean* query is a query without free variables.

A CQ is *hierarchical* if for any two variables X and Y, one of the following conditions holds: $at(X) \subset at(Y)$, $at(X) \supseteq at(Y)$, or $at(X) \cap at(Y) = \emptyset$. A CQ is *self-join free* if there are no two atoms with the same relation symbol.

Example 2.5. The query $Q = \exists X, Y, Z, V, U\, R(X, Y, Z) \land S(X, Y, V) \land T(X, U)$ is hierarchical: $at(V) \subset at(Y) \subset at(X)$, $at(U) \subset at(X)$, and $at(U) \cap at(Y) = \emptyset$. The query $Q = \exists X, Y\, R(X) \land S(X, Y) \land T(Y)$ is non-hierarchical: the sets $at(X) = \{R(X), S(X, Y)\}$ and $at(Y) = \{T(Y), S(X, Y)\}$ are neither disjoint nor one is included in the other.

A *union of conjunctive queries* (UCQ) has the form $Q = Q_1 \lor \cdots \lor Q_n$ where $Q_1, \ldots, Q_n$ are CQs. The query Q is Boolean if $Q_1, \ldots, Q_n$ are Boolean. Given a non-Boolean query Q with free variables $X_1, \ldots, X_n$, a *residual* query of Q is a Boolean query, where each free variable X_i is replaced by a constant a_i for $i \in [n]$. We denote this residual query by $Q[a_1/X_1, \ldots, a_n/X_n]$.

Selection conditions of the form $X\ \theta$ `const`, where X is a query variable, `const` is a constant, and the comparison θ is any of $<, \leq, =, \neq, \geq, >, \geq$, are also supported for practical reasons. UCQs with selections correspond to select-project-join-union queries in SQL.

Query Lineage. Let a database $D = D_n \cup D_x$. Each endogenous fact f in D_n is associated with a propositional variable denoted by $v(f)$. Given a Boolean UCQ Q and a database D, the lineage of Q over D, denoted by $\varphi_{Q,D}$, is a positive Boolean function in DNF over the variables $v(f)$ of facts f in D_n. Each clause is a conjunction of m variables, where m is the number of atoms in Q. We define lineage recursively on the structure of Q (we skip D from the subscript):

$$\varphi_{Q_1 \land Q_2} \overset{\text{def}}{=} \varphi_{Q_1} \land \varphi_{Q_2} \qquad \varphi_{Q_1 \lor Q_2} \overset{\text{def}}{=} \varphi_{Q_1} \lor \varphi_{Q_2}$$

$$\varphi_{\exists X Q} \overset{\text{def}}{=} \bigvee_{a \in \text{Dom}} \varphi_{Q[a/X]} \qquad \varphi_{R(t)} \overset{\text{def}}{=} \begin{cases} v(R(t)) & \text{if } R(t) \in D_n \\ 1 & \text{if } R(t) \in D_x \\ 0 & \text{otherwise} \end{cases}$$

where $Q[a/X]$ is Q where the variable X is set to the constant a. If Q is the conjunction (disjunction) of subqueries, the lineage of Q is the conjunction (disjunction) of the lineages of the subqueries. In case of an existential quantifier $\exists X$, the lineage is the disjunction of the lineages of the residual queries obtained by replacing X with each value in the domain. If Q is an atom $R(t)$ where all variables are already replaced by constants, we check whether $R(t)$ is a fact in the database. If it is not, then the constant 0 is added to the lineage. Otherwise, we have two cases. If $R(t)$ is an endogenous fact, then the variable $v(R(t))$ associated with $R(t)$ is added to the lineage. If $R(t)$ is an exogenous fact, then the constant 1 is added instead to the lineage. This means that exogenous facts are not in the lineage, even though they are used to create the lineage.

The lineage for any non-Boolean query Q is defined using the case of Boolean queries. Each tuple in the result of Q defines a residual query of Q, which is Boolean and for which we can compute the lineage as defined above. In other words, the lineage of Q is given by the set of lineages of the tuples in the result of Q.

Example 2.6. Reconsider the first query Q from Example 2.5 and the database $D = \{R(1,2,3), S(1,2,4), S(1,2,5), T(1,6)\}$, where all facts are endogenous. There are two groundings of the query in the database, obtained by replacing X, Y, Z, V, U with $1, 2, 3, 4, 6$ respectively or $1, 2, 3, 5, 6$ respectively. Each grounding is intuitively an alternative reason for the query satisfaction and yields a clause in the lineage. Thus, the lineage is $\varphi_{Q,D} = [v(R(1,2,3)) \wedge v(S(1,2,4)) \wedge v(T(1,6))] \vee [v(R(1,2,3)) \wedge v(S(1,2,5)) \wedge v(T(1,6))]$.

Banzhaf Values of Database Facts. We use the Banzhaf value of an endogenous database fact f as a measure of contribution of f to the result of a given query. An equivalent formulation is via the query lineage: We want the Banzhaf value of the variable $v(f)$ associated with f in the lineage of the query.

Consider a Boolean query Q, a database $D = (D_n, D_x)$, and an endogenous fact $f \in D_n$. Let $v(f)$ be the variable associated to f. We define:

$$Banzhaf(Q, D, f) \overset{\text{def}}{=} Banzhaf(\varphi_{Q,D}, v(f)) \tag{3}$$

Since the function $\varphi_{Q,D}$ is positive, it follows from Eq. (1) that $Banzhaf(Q, D, f)$ is the number of subsets $D' \subseteq (D_n \setminus \{f\})$ such that $Q(D' \cup D_x) = 0$ and $Q(D' \cup D_x \cup \{f\}) = 1$.

For a non-Boolean query Q with free variables $\boldsymbol{Z}$, the Banzhaf value of f is defined with respect to a tuple $\boldsymbol{t}$ in the result of Q:

$$Banzhaf(Q, D, f, \boldsymbol{t}) \overset{\text{def}}{=} Banzhaf(Q[\boldsymbol{t}/\boldsymbol{Z}], D, f)$$

where $Q[\boldsymbol{t}/\boldsymbol{Z}]$ is the Boolean residual query of Q, where the tuple of free variables $\boldsymbol{Z}$ is replaced by the tuple $\boldsymbol{t}$ of constant values.

Example 2.7. Consider again the lineage $\varphi_{Q,D}$ from Example 2.6. We have $\varphi_{Q,D}[v(R(1,2,3)) := 1] - \varphi_{Q,D}[v(R(1,2,3)) := 0] = 2 - 0 = 2$ and $\varphi_{Q,D}[v(S(1,2,4)) := 1] - \varphi_{Q,D}[v(S(1,2,4)) := 0] = 2 - 1 = 1$. Hence, $Banzhaf(\varphi_{Q,D}, v(R(1,2,3))) = Banzhaf(Q, D, R(1,2,3)) = 2$ and $Banzhaf(\varphi_{Q,D}, v(S(1,2,4))) = Banzhaf(Q, D, S(1,2,4)) = 1$.

3 BANZHAF COMPUTATION

This section introduces our algorithmic framework for computing the exact or approximate Banzhaf value for a fact (variable) in a query lineage (Boolean positive DNF function). Sec. 3.1 gives our exact algorithm, which allows us to introduce the building blocks of decomposition trees and formulas for Banzhaf value computation that exploit the independence and mutual exclusion of functions. Then, Sec. 3.2 extends the exact algorithm to an anytime deterministic approximation algorithm, which incrementally refines approximation intervals for the Banzhaf values until the desired error is reached.

3.1 Exact Computation

The main idea of our exact algorithm is as follows. Assume we have the Banzhaf value for a variable x in a function φ_1. Then, we can compute efficiently the Banzhaf value for x in a function $\varphi = \varphi_1$ op φ_2, where op is one of the logical connectors OR ($\vee$) or AND ($\wedge$) and in case the functions φ_1 and φ_2 are independent, i.e., they have no variable in common, or mutually exclusive, i.e., they have no satisfying assignment in common. The following formulas make this argument precise, where we keep track of both the Banzhaf value for x in φ and also of the model count $\#\varphi$ for φ:

- If $\varphi = \varphi_1 \wedge \varphi_2$ and φ_1 and φ_2 are independent, then:

$$\#\varphi = \#\varphi_1 \cdot \#\varphi_2 \tag{4}$$

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) \cdot \#\varphi_2 \tag{5}$$

- If $\varphi = \varphi_1 \vee \varphi_2$ and φ_1 and φ_2 are independent, then:

$$\#\varphi = \#\varphi_1 \cdot 2^{n_2} + 2^{n_1} \cdot \#\varphi_2 - \#\varphi_1 \cdot \#\varphi_2 \tag{6}$$

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) \cdot (2^{n_2} - \#\varphi_2), \tag{7}$$

  where n_i is the number of variables in φ_i for $i \in [2]$.
- If $\varphi = \varphi_1 \vee \varphi_2$, and φ_1 and φ_2 are mutually exclusive and over the same variables, then:

$$\#\varphi = \#\varphi_1 + \#\varphi_2 \tag{8}$$

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) + Banzhaf(\varphi_2, x) \tag{9}$$

The derivations of these formulas are given in the extended version of this paper [2].

For functions representing the lineage of hierarchical queries, it is known that they can be decomposed efficiently into independent functions down to trivial functions of one variable [40]. For such functions, Eq. (4) to (7) are then sufficient to compute efficiently the Banzhaf values. For non-hierarchical queries, however, this is not the case. A common general approach, which is widely used in probabilistic databases [50] and exact Shapley computation [19], and borrowed from knowledge compilation [17], is to decompose, or *compile*, the query lineage into an equivalent Boolean function, where all logical connectors are between functions that are either independent or mutually exclusive. While in the worst case this necessarily leads to a blow-up in the number of decomposition steps (unless P=NP), it turns out that in many practical cases (including our own experiments), this number remains reasonably small.

In this paper, we compile the query lineage into a *decomposition tree* [24]. Such trees have inner nodes that are the logical operators enhanced with information about independence and mutual exclusiveness of their children: $\otimes$ stands for independent-or, $\odot$ for independent-and, and $\oplus$ for mutual exclusion.

Definition 3.1. [24] A *decomposition tree*, or d-tree for short, is defined recursively as follows:

- Every function φ is a d-tree for φ.
- If T_φ and T_ψ are d-trees for independent functions φ and respectively ψ, then

$$\begin{array}{ccccc} & \otimes & & & & \odot & \\ & / \ \backslash & & \text{and} & & / \ \backslash & \\ T_\varphi & & T_\psi & & T_\varphi & & T_\psi \end{array}$$

  are d-trees for $\varphi \vee \psi$ and respectively $\varphi \wedge \psi$.
- If T_φ and T_ψ are d-trees for mutually exclusive functions φ and respectively ψ, then

$$\begin{array}{ccc} & \oplus & \\ & / \ \backslash & \\ T_\varphi & & T_\psi \end{array}$$

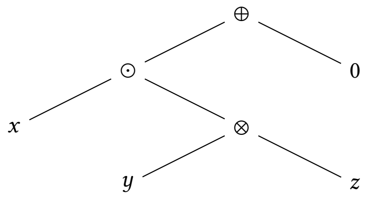
  is a d-tree for $\varphi \vee \psi$.

A d-tree, whose leaves are Boolean constants or literals, is *complete*.

Any Boolean function can be compiled into a complete d-tree by decomposing it into conjunctions or disjunctions of independent functions or into disjunctions of mutually exclusive functions. The latter is always possible via Shannon expansion: Given a function φ

and a variable x, φ can be equivalently expressed as the disjunction of two mutually exclusive functions defined over the same variables as φ: $\varphi = (x \wedge \varphi[x := 1]) \vee (\neg x \wedge \varphi[x := 0])$. This expression yields the d-tree: $(x \odot \varphi[x := 1]) \oplus (\neg x \odot \varphi[x := 0])$. The details of d-tree construction are given in prior work [24]. In a nutshell, it first attempts to partition the function into independent functions using a standard algorithm for finding connected components in a graph representation of the function. If this fails, then it applies Shannon expansion on a variable that appears most often in the function (other heuristics are possible, e.g., pick variables whose conditioning allow for independence partitioning). The functions $\varphi[x := 1]$ and $\varphi[x := 0]$ are subject to standard simplifications for conjunctions and disjunctions with the constants 0 and 1. In the worst case, d-tree compilation may (unavoidably) require a number of Shannon expansion steps exponential in the number of variables.

Example 3.2. We construct a d-tree for the Boolean function $\varphi = (x \wedge y) \vee (x \wedge z)$. We first observe that the two conjunctive clauses are not independent, so we apply Shannon expansion on x and decompose the function into the two mutually exclusive functions $\varphi_1 = x \wedge \varphi[x := 1] = x \wedge (y \vee z)$ and $\varphi_0 = \neg x \wedge \varphi[x := 0] = 0$. The left branch representing φ_1 can be further decomposed into independent functions until we obtain a complete d-tree:



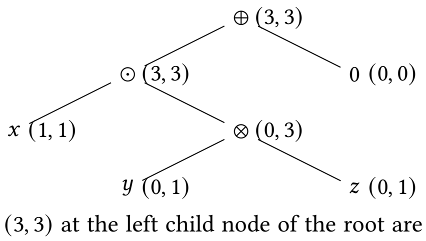
Alternatively, we can factor out x to obtain the function $x \wedge (y \vee z)$, and compile it into the d-tree $x \odot (y \otimes z)$. Our algorithm computing d-trees does this whenever a variable occurs in all clauses.

Figure 1 gives our algorithm ExaBan that computes the exact Banzhaf value for any variable x in an input function φ. It takes as input a complete d-tree for φ and uses Eq. (4) to (9) to express the Banzhaf value of a variable x in a function φ represented by a d-tree T_φ using the Banzhaf values of x in sub-trees T_{φ_1} and T_{φ_2}.

Proposition 3.3. *For any positive DNF function φ, complete d-tree T_φ for φ, and variable x in φ, it holds*

$$\textsc{ExaBan}(T_\varphi, x) = (Banzhaf(\varphi, x), \#\varphi).$$

Example 3.4. The next figure shows the trace of the computation of ExaBan for the input d-tree from Example 3.2 and the variable x. Each node of the d-tree is labelled by the pair of the Banzhaf value and the model count computed for the subtree rooted at that node:



The values (3, 3) at the left child node of the root are computed as follows. This node is an independent-and ($\odot$). The variable x is in the left subtree. ExaBan computes the Banzhaf value 3 of x by multiplying the Banzhaf value 1 at the left child node with the model count 3 at the right child node. The model count of 3 is obtained by multiplying the model counts at the child nodes. The function represented by the tree rooted at this $\odot$-node is $\varphi_1 = x \wedge (y \vee z)$. Indeed, every model of the function must satisfy x and at least one of y and z, which implies $\#\varphi_1 = 3$. Using Eq. (2), we have $Banzhaf(\varphi_1, x) = \varphi_1[x := 1] - \varphi_1[x := 0] = 3 - 0 = 3$.

```
ExaBan(d-tree $T_\varphi$ for function $\varphi$, variable $x$)
outputs $(Banzhaf(\varphi, x), \#\varphi)$

$B := 0$;   $\# := 0$;   //initialization
switch $T_\varphi$
  case $x$:  $B := 1$; $\# := 1$
  case $\neg x$: $B := -1$; $\# := 1$
  case 1 or a literal not $x$ nor $\neg x$: $B := 0$; $\# := 1$
  case 0: $B := 0$; $\# := 0$
  case $T_{\varphi_1}$ op $T_{\varphi_2}$:
    $(B_i, \#_i) :=$ ExaBan$(T_{\varphi_i}, x)$ for $i \in [2]$
    $n_i :=$ number of variables in $T_{\varphi_i}$ for $i \in [2]$
    switch op
      case $\odot$:  //wlog if $x$ is in $\varphi$,then it is in $\varphi_1$
      $B := B_1 \cdot \#_2$;   $\# := \#_1 \cdot \#_2$
      case $\otimes$:  //wlog if $x$ is in $\varphi$,then it is in $\varphi_1$
        $B := B_1 \cdot (2^{n_2} - \#_2)$;   $\# := \#_1 \cdot 2^{n_2} + 2^{n_1} \cdot \#_2 - \#_1 \cdot \#_2$
      case $\oplus$:  //wlog $\varphi_1$ and $\varphi_2$ have same variables
        $B := B_1 + B_2$;   $\# := \#_1 + \#_2$
return $(B, \#)$
```

Figure 1: Computing the exact Banzhaf value for a variable x and the model count over a complete d-tree.

ExaBan can be immediately generalized to compute the Banzhaf values for any number of variables $x_1, \ldots, x_n$. For all variables, it uses the same d-tree and shares the computation of the counts $\#_i$.

3.2 Anytime Deterministic Approximation

As explained in Sec. 3.1, to obtain exact Banzhaf values for the variables in a function, we first compile the function into a complete d-tree and then compute in a bottom-up traversal of the d-tree the exact Banzhaf values and model counts at each node of the d-tree. Approximate computation does not require in general a complete d-tree for the function. In this section, we introduce an anytime deterministic approximation algorithm, called AdaBan, that *gradually* expands the d-tree and computes after each expansion step upper and lower bounds on the Banzhaf values and model counts for the new leaves. It then uses the bounds to compute an approximation interval for the partial d-tree. If the approximation interval meets the desired error, it stops. Otherwise, it continues with the function compilation and bounds computation at another leaf in the d-tree. Eventually, the approximation interval becomes tight enough to meet the allowed error. Unlike ExaBan, AdaBan merges the construction of the d-tree with the computation of the bounds so it can intertwine them at each expansion step.

Sec. 3.2.1 explains how to efficiently compute upper and lower bounds for positive DNF functions, albeit without any error guarantee. Sec. 3.2.3 introduces AdaBan, which uses such bounds to compute approximation intervals and incrementally refine them.

3.2.1 Efficient Computation of Lower and Upper Bounds for Positive DNF Functions. We introduce two procedures L (for lower bound) and U (for upper bound) that map any positive DNF function φ to positive DNF functions that enjoy the following four desirable properties: (1) $L(\varphi)$ and $U(\varphi)$ admit linear-time computation of model counting; (2) $L(\varphi)$ and $U(\varphi)$ can be synthesized from φ in time linear in the size of φ; (3) the number of models of $L(\varphi)$ is less than or equal to the number of models of φ, which in turn is less than or equal to the number of models of $U(\varphi)$; and (4) lower and upper bounds on the Banzhaf value of x in φ can be obtained by applying L and U to the functions $\varphi[x := 0]$ and $\varphi[x := 1]$.

The co-domain of L and U is the class of iDNF functions [24], which are positive DNF functions where every variable occurs once. Whereas the first three aforementioned properties are already known to hold for iDNF functions [24], the fourth one is new and key to our approximation approach.

For the first property, we note that since each variable in an iDNF function only occurs once, we can decompose the function in linear time into a complete d-tree with $\odot$ or $\otimes$ as inner nodes and literals or constants at leaves. Then, we can traverse the d-tree bottom up and use Eq. (4) and (6) to compute at each node the model count for the function represented by the subtree rooted at that node. Overall, model counting for iDNF functions takes linear time.

For the second property, we explain the procedures L and U for a given DNF function φ. The iDNF function $L(\varphi)$ is any subset of the clauses such that no two selected clauses share variables. The iDNF function $U(\varphi)$ is a transformation of φ, where we keep one occurrence of each variable and eliminate all other occurrences.

The third and fourth properties follow by Proposition 3.5:

Proposition 3.5. *For any positive DNF function φ and variable x in φ, it holds:*

$$\#L(\varphi) \leq \#\varphi \leq \#U(\varphi)$$
$$\#L(\varphi[x:=1]) - \#U(\varphi[x:=0]) \leq Banzhaf(\varphi, x) \leq \#U(\varphi[x:=1]) - \#L(\varphi[x:=0])$$

Example 3.6. Consider the positive DNF function $\varphi = (x \wedge y) \vee (x \wedge z) \vee u$. The function is a disjunction of two independent functions $\varphi_1 = (x \wedge y) \vee (x \wedge z)$ and $\varphi_2 = u$. Since φ_1 is the function analyzed in Example 3.4, we know that $Banzhaf(\varphi_1, x) = \#\varphi_1 = 3$. It is easy to see that $Banzhaf(\varphi_2, x) = 0$ and $\#\varphi_2 = 1$. Using Eq. (6) and 7, we obtain

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) \cdot (2^1 - 1) = 3 \cdot 1 = 3$$
$$\#\varphi = \#\varphi_1 \cdot \#\varphi_2 + \#\varphi_1 \cdot (2^1 - 1) + (2^3 - \#\varphi_1) \cdot \#\varphi_2 = 3 + 3 + 5 = 11.$$

The conditioned functions $\varphi[x := 0] = (0 \wedge y) \vee (0 \wedge z) \vee u$ and $\varphi[x := 1] = (1 \wedge y) \vee (1 \wedge z) \vee u = y \vee z \vee u$ are already in iDNF, so $L(\varphi[x := 0]) = U(\varphi[x := 0]) = \varphi[x := 0]$ and $L(\varphi[x := 1]) = U(\varphi[x := 1]) = \varphi[x := 1]$. Note that $\varphi[x := 0] = u$, yet it is defined over three variables, which is important for computing its correct model count.

```
bounds(d-tree $T_\varphi$ for function $\varphi$, variable $x$)
outputs lower and upper bounds for $Banzhaf(\varphi, x)$ and $\#\varphi$

$(L_b, L_\#, U_b, U_\#) := (0,0,0,0)$ // Initialize the bounds
switch $T_\varphi$
  case literal or constant $\ell$:
    $(L_b, L_\#) := (U_b, U_\#) :=$ ExaBan$(\ell, x)$
  case non-trivial leaf $\psi$:   //no literal nor constant
    //Compute bounds by Proposition 3.5
    $L_b := \#L(\psi[x:=1]) - \#U(\psi[x:=0])$
    $U_b := \#U(\psi[x:=1]) - \#L(\psi[x:=0])$
    $L_\# := \#L(\psi)$;   $U_\# := \#U(\psi)$
  case $T_{\varphi_1}$ op $T_{\varphi_2}$:
    $(L_b^{(i)}, L_\#^{(i)}, U_b^{(i)}, U_\#^{(i)}) :=$ bounds$(T_{\varphi_i}, x)$, for $i \in [2]$
    $n_i :=$ number of variables in $\varphi_i$, for $i \in [2]$
    switch op
      case $\odot$: //wlog if $x$ is in $\varphi$, then it is in $\varphi_1$
        $L_b := L_b^{(1)} \cdot L_\#^{(2)}$;   $U_b := U_b^{(1)} \cdot U_\#^{(2)}$
        $L_\# := L_\#^{(1)} \cdot L_\#^{(2)}$;   $U_\# := U_\#^{(1)} \cdot U_\#^{(2)}$
      case $\otimes$: //wlog if $x$ is in $\varphi$, then it is in $\varphi_1$
        $L_b := L_b^{(1)} \cdot (2^{n_2} - U_\#^{(2)})$; $U_b := U_b^{(1)} \cdot (2^{n_2} - L_\#^{(2)})$
        $L_\# := L_\#^{(1)} \cdot 2^{n_2} + L_\#^{(2)} \cdot 2^{n_1} - L_\#^{(1)} \cdot L_\#^{(2)}$
        $U_\# := U_\#^{(1)} \cdot 2^{n_2} + U_\#^{(2)} \cdot 2^{n_1} - U_\#^{(1)} \cdot U_\#^{(2)}$
      case $\oplus$: //wlog $\varphi_1$ and $\varphi_2$ have same variables
        $L_b := L_b^{(1)} + L_b^{(2)}$;   $U_b := U_b^{(1)} + U_b^{(2)}$
        $L_\# := L_\#^{(1)} + L_\#^{(2)}$;   $U_\# := U_\#^{(1)} + U_\#^{(2)}$
return $(L_b, L_\#, U_b, U_\#)$
```

Figure 2: Computation of bounds for the Banzhaf value $Banzhaf(\varphi, x)$ and model count $\#\varphi$, given a (possibly partial) d-tree T_φ for the function φ and a variable x.

We may also obtain the following iDNF functions: $L(\varphi) = (x \wedge y) \vee u$ by skipping the clause $(x \wedge z)$ in φ; and $U(\varphi) = (x \wedge y) \vee z \vee u$ by removing x from the second clause of φ. Using Eq. (4) and (6):

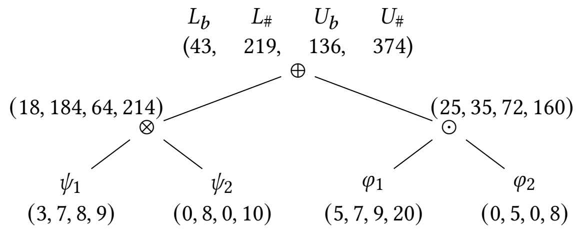
$$\#L(\varphi[x:=0]) = \#U(\varphi[x:=0]) = 4,$$
$$\#L(\varphi[x:=1]) = \#U(\varphi[x:=1]) = 7,$$
$$\#L(\varphi) = 5, \text{ and } \#U(\varphi) = 13.$$

Hence, it indeed holds that $\#L(\varphi) = 5 \leq \#\varphi = 11 \leq \#U_\varphi = 13$ and $\#L(\varphi[x := 1]) - \#U(\varphi[x := 0]) = 3 \leq Banzhaf(\varphi, x) = 3 \leq \#U(\varphi[x := 1]) - \#L(\varphi[x := 0]) = 3$.

3.2.2 Efficient Computation of Lower and Upper Bounds for D-trees. Figure 2 gives the procedure bounds that computes lower and upper bounds on the Banzhaf value and model count for any d-tree, whose leaves may be positive DNF functions, literals (possibly negated variables), or constants. The computation is done in linear time in one bottom-up pass over the d-tree.

The procedure takes as input a d-tree T_φ for a function φ and a variable x for which we want to compute the Banzhaf value. At a leaf ℓ of T_φ that is a literal or a constant, it calls ExaBan(ℓ, x) to compute the exact Banzhaf value and model count for ℓ. At a leaf ψ that is not a literal nor a constant, the algorithm first computes the iDNF functions $L(\psi)$, $U(\psi)$, $L(\psi[x := b])$, and $U(\psi[x := b])$ for $b \in \{0,1\}$ By Proposition 3.5, these functions can be used to derive lower and upper bounds on $Banzhaf(\psi, x)$ and $\#\psi$. If T_φ has children, then it recursively computes bounds on them and then combines them into bounds for itself. We next discuss the lower bound for the Banzhaf value of x in case φ is a disjunction of independent functions φ_1 and φ_2. The other cases are handled analogously. By Eq. (7), the formula for the exact Banzhaf value is $Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) \cdot (2^{n_2} - \#\varphi_2)$. To obtain a lower bound on $Banzhaf(\varphi, x)$, we replace the term $Banzhaf(\varphi_1, x)$ by its lower bound and the term $\#\varphi_2$ by its upper bound. The reason for using the upper bound is that the term occurs negatively.

Example 3.7. Consider the following partial d-tree representing a function φ. Each node is assigned a quadruple of bounds for the Banzhaf value of some variable x and the model count for the d-tree rooted at that node. Following the notation in the procedure BOUNDS in Figure 2, the first and the third entry in a quadruple are the lower and respectively upper bound for the Banzhaf value; the second and the fourth entry are the lower and respectively upper bound for the model count. For the computation of the bounds at the node $\otimes$ assume that each of the functions ψ_i has four variables.



Assume we have already computed the bounds for the leaves of the d-tree. We explain how the procedure BOUNDS uses these bounds to derive bounds for the Banzhaf values at the nodes $\odot$ and $\oplus$. Assume that the variable x appears in φ_1 but not in φ_2. At the node $\odot$, the lower bound for the Banzhaf value is $5 \cdot 5 = 25$ and its upper bound is $9 \cdot 8 = 72$. Similarly, at the node $\oplus$, the lower and upper bounds for the Banzhaf value are $L_b = 18 + 25 = 43$ and respectively $U_b = 64 + 72 = 136$.

We cannot use the bounds L_b and U_b to derive a 0.5-approximation for the Banzhaf value, since $(1 - 0.5) \cdot U_b = 68$ is larger than $(1 + 0.5) \cdot L_b = 64.5$. However, every value within the interval from $(1-0.6) \cdot U_b = 14.4$ to $(1+0.6) \cdot L_b = 68.8$ is a 0.6-approximation. For instance, it holds that $20 \geq (1-0.6) \cdot U_b \geq (1-0.6) \cdot Banzhaf(\varphi, x)$ and $20 \leq (1 + 0.6) \cdot L_b \leq (1 + 0.6) \cdot Banzhaf(\varphi, x)$.

Eq. (4) to (9) and Proposition 3.5 imply:

PROPOSITION 3.8. *For any positive DNF function φ, d-tree T_φ for φ, and variable x in φ, it holds* BOUNDS$(T_\varphi, x) = (L_b, L_\#, U_b, U_\#)$ *such that $L_b \leq Banzhaf(\varphi, x) \leq U_b$ and $L_\# \leq \#\varphi \leq U_\#$.*

3.2.3 *Refining Bounds for D-Trees.*

Given a partial d-tree T_φ, a variable x, and a relative error ϵ, the algorithm ADABAN in Figure 3 computes an interval that consists of ϵ-approximations for $Banzhaf(\varphi, x)$. The algorithm uses recursion to gradually improve the interval. First, it calls the function BOUNDS from Figure 2 to obtain a lower bound L_b and an upper bound U_b for $Banzhaf(\varphi, x)$ based on the current structure of T_φ. If $(1-\epsilon) \cdot U_b - (1+\epsilon) \cdot L_b \leq 0$, it returns the interval $[(1-\epsilon) \cdot U_b, (1+\epsilon) \cdot L_b]$. Observe that, in this case, for any value $B \in [(1-\epsilon) \cdot U_b, (1+\epsilon) \cdot L_b]$, it holds $B \geq (1-\epsilon) \cdot U_b \geq (1-\epsilon) \cdot Banzhaf(\varphi, x)$ and $B \leq (1-\epsilon) \cdot L_b \leq (1-\epsilon) \cdot Banzhaf(\varphi, x)$, which means that B is a relative ϵ-approximation for $Banzhaf(\varphi, x)$.

```
ADABAN(d-tree $T_\varphi$, variable $x$, error $\epsilon$) outputs approximation
interval for $Banzhaf(\varphi, x)$ satisfying relative error $\epsilon$

$(L_b\ ,\ \cdot\ ,\ U_b\ ,\ \cdot\ )$ := BOUNDS($T_\varphi, x$)    //get bounds on $T_\varphi$
$\ell := 0; u := 0$    //initialize approximation interval
if $(1-\epsilon) \cdot U_b - (1+\epsilon) \cdot L_b \leq 0$    //error satisfied
  $\ell := (1-\epsilon) \cdot U_b; u := (1+\epsilon) \cdot L_b$
else
  pick a non-trivial leaf $\psi$ of $T_\varphi$ //no literal nor constant
  switch $\psi$
    case $\psi_1 \wedge \psi_2$ for independent $\psi_1$ and $\psi_2$:
      replace $\psi$ by $\psi_1 \odot \psi_2$ in $T_\varphi$
    case $\psi_1 \vee \psi_2$ for independent $\psi_1$ and $\psi_2$:
      replace $\psi$ by $\psi_1 \otimes \psi_2$ in $T_\varphi$
    default:    pick a variable $y$ in $\psi$
      replace $t$ by $(y \odot \psi[y := 1]) \oplus (\neg y \odot \psi[y := 0])$ in $T_\varphi$
  $[\ell, u]$ := ADABAN($T_\varphi, x, \epsilon$)
return $[\ell, u]$
```

Figure 3: Computing an approximate Banzhaf value with relative error ϵ using incremental decomposition and bound refinement.

If $(1-\epsilon) \cdot U_b - (1+\epsilon) \cdot L_b \leq 0$ does not hold, the algorithm picks a non-trivial leaf ψ, which is neither a literal nor a constant, decomposes it, and checks again whether the interval based on the new bounds is satisfactory. Such a leaf ψ always exists unless T_φ is complete, in which case $U_b = L_b$. The decomposition of ψ means its replacement by ψ_1 op ψ_2 where op represents independent-and ($\odot$), independent-or ($\otimes$), or mutual exclusion ($\oplus$). The decomposition of ψ into mutually exclusive functions ψ_1 and ψ_2 is always possible using Shannon expansion.

PROPOSITION 3.9. *For any positive DNF function φ, d-tree T_φ for φ, variable x in φ, and error ϵ, it holds ADABAN$(T_\varphi, x, \epsilon) = [\ell, u]$ such that every value in $[\ell, u]$ is an ϵ-approximation of $Banzhaf(\varphi, x)$.*

3.2.4 *Optimizations.*

The algorithms ADABAN and BOUNDS presented in Figures 2 and 3 are subject to four key optimizations implemented in our prototype.

(1) Instead of *eagerly* recomputing the bounds for a partial d-tree after each decomposition step, we follow a *lazy* approach that does not recompute the bounds after independence partitioning steps and instead only recomputes them after Shannon expansion steps.

(2) To avoid recomputation of bounds for subtrees whose leaves have not changed, we cache the bounds for each subtree. Hence, whenever a new bound is calculated for some leaf, it suffices to propagate the bound along the path to the root of the d-tree.

(3) To approximate the Banzhaf values for several variables, we do not compute bounds for each variable after each expansion step. Instead, we compute the approximation for one variable at a time.

After having achieved a satisfying approximation for one variable, we reuse the partial d-tree constructed so far to obtain a desired approximation for the next variable. This reduces the number of bounds calls and improves the overall runtime of AdaBan.

(4) Instead of computing bounds for $\#\varphi[x := 1]$ and $\#\varphi[x := 0]$, as done in bounds, it suffices to compute bounds for $\#\varphi$ and $\#\varphi[x := 0]$ for each variable x. This is justified by the following insight:

$$\begin{aligned} Banzhaf(\varphi, x) &= \#\varphi[x := 1] - \#\varphi[x := 0] \\ &= \#\varphi[x := 1] + \#\varphi[x := 0] - 2 \cdot \#\varphi[x := 0] = \#\varphi - 2 \cdot \#\varphi[x := 0], \end{aligned}$$

where the first equality is by the characterization of the Banzhaf value in Eq. (2) and the last equality states that the set of models of φ is the disjoint union of the set of models where x is 0 and the set of models where x is set to 1. In many practical scenarios, the lower bound for $Banzhaf(\varphi, x)$ computed using bounds for $\#\varphi$ and $\#\varphi[x := 0]$ is tighter than the lower bound computed by AdaBan.

4 BANZHAF-BASED RANKING AND TOP-k

Common uses of fact attribution in query answering and explanations are to identify the k most influential facts and to rank the facts by their influence to the query result. Our anytime approximation of Banzhaf values lends itself naturally to fast ranking and computation of top-k facts, as follows.

4.1 The Algorithm

We introduce a new algorithm called IchiBan, that uses AdaBan to find the variables in a given function with the top-k Banzhaf values. It starts by running AdaBan for all variables at the same time. Whenever AdaBan computes the bounds for the Banzhaf values of the variables, IchiBan identifies those variables whose upper bounds are smaller than the lower bounds of at least k other variables. These former variables are not in top-k and are discarded. It then resumes AdaBan for the remaining variables and repeats the selection process using the refined bounds. Eventually, it obtains the variables with the top-k Banzhaf values. For ranking, IchiBan runs until the approximation intervals for the variables do not overlap or collapse to the same Banzhaf value.

IchiBan may also be executed with a parameter $\epsilon \in [0, 1]$. In this case, it may finish as soon as each approximation interval reaches a relative error ϵ. IchiBan then ranks the facts based on the order of the mid-points of their respective intervals.

4.2 A Dichotomy Result

The time complexity of IchiBan is unavoidably exponential in the worst case. We next analyze in further depth the complexity of the ranking problem and show a dichotomy in the complexity of Banzhaf-based ranking of database facts. We first formalize the following ranking problem, parameterized by a Boolean CQ Q:

Problem:	RankBan$_Q$
Description:	*Banzhaf-based ranking of database facts*
Parameter:	Boolean CQ Q
Input:	Database $D = (D_n, D_x)$ and facts $f_1, f_2 \in D_n$
Question:	Is $Banzhaf(Q, D, f_1) \leq Banzhaf(Q, D, f_2)$?

We now state the dichotomy and then explain it.

Theorem 4.1. *For any Boolean CQ Q without self-joins, it holds:*

- *If Q is hierarchical, then RankBan$_Q$ can be solved in polynomial time.*
- *If Q is not hierarchical, then RankBan$_Q$ cannot be solved in polynomial time, unless there is an FPTAS for #BIS.*

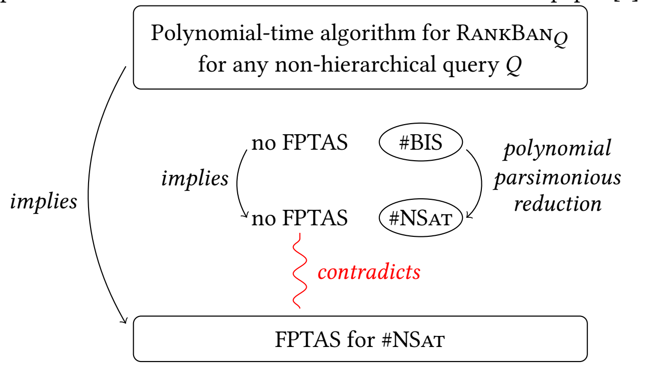
The tractability part of our dichotomy follows from prior work: In case of hierarchical queries, *exact* Banzhaf values of database facts can be computed in polynomial time [35]. Hence, we can first compute the exact Banzhaf values and then rank the facts. Showing the intractability part of our dichotomy is more involved and requires novel development. It is based on the widely accepted conjecture that there is no polynomial-time approximation scheme (FPTAS) for counting independent sets in bipartite graphs (#BIS) [14, 21]. In the following, we make these notions more precise.

A bipartite graph is an undirected graph $G = (V, E)$ where the set V of nodes is partitioned into two disjoint sets U and W and the edges $E \subseteq U \times W$ connect nodes from U with nodes from W. An independent set V' of G is a subset of V such that no two nodes in V' are connected by an edge. The problem #BIS is defined as:

Problem:	#BIS
Description:	*Counting independent sets in bipartite graphs*
Input:	Bipartite graph G
Compute:	Number of independent sets of G

An algorithm A for a numeric function g is a *fully polynomial-time approximation scheme* (FPTAS) for g if for any error $0 < \epsilon < 1$ and input x, A computes, in time polynomial in the size of x and in ϵ^{-1}, a value $A(x)$ such that $(1 - \epsilon)g(x) \leq A(x) \leq (1 + \epsilon)g(x)$.

The hardness result in Theorem 4.1 assumes the widely accepted conjecture that there is no FPTAS for #BIS [14, 21]. We next outline our proof strategy, which is visualized by the following diagram; the proof details are deferred to the extended version of this paper [2].


Polynomial-time algorithm for RankBan$_Q$ for any non-hierarchical query Q
implies
no FPTAS
#BIS
implies
polynomial parsimonious reduction
no FPTAS
#NSat
contradicts
FPTAS for #NSat

We use the intermediate problem #NSat: Given a positive bipartite DNF function, compute the number of its non-satisfying assignments. We first give a parsimonious polynomial-time reduction from #BIS to #NSat, i.e., a polynomial-time reduction that also preserves the output; this means that the number of non-satisfying assignments equals the number of independent sets. Assuming that there is no FPTAS for #BIS, this reduction implies that there is no FPTAS for #NSat. Yet, given a polynomial-time algorithm A for RankBan$_Q$ for any non-hierarchical query Q, we can design an FPTAS for #NSat. This contradicts the assumption that there is no FPTAS for #NSat. Consequently, there cannot be any polynomial-time algorithm for RankBan$_Q$ for non-hierarchical queries Q.

Table 1: Statistics of the datasets used in the experiments.

Dataset	# Queries	# Lineages	# Vars (avg/max)	# Clauses (avg/max)
Academic	92	7,865	79 / 6,027	74 / 6,025
IMDB	197	986,030	25 / 27,993	15 / 13,800
TPC-H	12	165	1,918 / 139,095	863 / 75,983

5 EXPERIMENTS

This section details our experimental setup and results.

5.1 Experimental Setup and Benchmarks

We have implemented all algorithms in Python 3.9 and performed experiments on a Linux Debian 14.04 machine with 1TB of RAM and an Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz processor. We set a timeout for each run of an algorithm to one hour.

Datasets. We tested the algorithms using 301 queries evaluated over three datasets: Academic, IMDB and TPC-H (SF1). The workload is based on previous work on Shapley values for query answering [3, 19]: as in [19], for TPC-H we used all queries without nested subqueries and with aggregates removed, so expressible as SPJU queries. For IMDB and Academic, we used all queries from [3] (Academic was not used in [19]). We constructed lineage for all output tuples of these queries using ProvSQL [47]. The resulting set of nearly 1M lineage expressions is the most extensive collection for which attribution in query answering has been assessed in academic papers. Table 1 includes statistics on the datasets.

Algorithms. We benchmarked our algorithms ExaBan, AdaBan, and IchiBan against the following three competitors: Sig22, for exact computation using an off-the-shelf knowledge compilation package [19]; MC, a Monte Carlo-based randomized approximation [33]; and CNFProxy, an heuristic for ranking facts based on their contribution [19]. These competitors were originally developed for Shapley value. We adapted them to compute Banzhaf values (see Sec. 6). AdaBan, MC, and IchiBan expect as input: the error bound, the number of samples, and respectively the number of top results to retrieve. We use the notation Algo**X** to denote the execution of an algorithm Algo with parameter value **X**.

Measurements. We measured the execution time of all algorithms and the accuracy of AdaBan and MC. We define an instance as the (exact, approximate or top-k) computation of the Banzhaf values for all variables in a lineage of an output tuple of a query over one dataset. We reported failure in case an algorithm did not terminate an instance within one hour. We also reported the success rate of each algorithm and statistics of its execution times across all instances (average, median, maximal execution time, and percentiles). The pX columns in the following tables show the execution times for the X-th percentile of the considered instances.

5.2 Exact Banzhaf computation

We first compare the two exact algorithms: ExaBan and Sig22.

Success Rate. Table 2 gives the success rate of ExaBan and Sig22 for each dataset, where success means that an algorithm finished each instance in one hour. ExaBan succeeded in strictly far more instances than Sig22. For Academic and IMDB, both algorithms succeeded for the vast majority of instances; a breakdown based on queries shows that the failure cases of Sig22 lead to a failure to compute Banzhaf values with respect to all output tuples of a given query for a significant portion of the workload (success rate of 83.91% and 65.48% for the two datasets). For these instances, ExaBan achieves success rates of 98.91% and 82.23% respectively. For TPC-H, the success rate is significantly lower for both algorithms. Here ExaBan improves the success rate from 85.45% to 91.52% of all instances, and the success rate from 50% to 58.33% of all instances for a query.

Table 2: Query success rate: Percentage of queries for which the algorithms finished for all instances of a query within one hour. Lineage success rate: Percentage of instances (over all queries in each dataset) for which the algorithms finished within one hour.

Dataset	Algorithm	Query Success Rate	Lineage Success Rate
Academic	ExaBan	98.91%	99.99%
	Sig22	83.91%	98.40%
	AdaBan0.1	98.91%	99.99%
	MC50#vars	96.74%	98.83%
IMDB	ExaBan	82.23%	99.63%
	Sig22	65.48%	98.35%
	AdaBan0.1	88.32%	99.81%
	MC50#vars	83.76%	99.74%
TPC-H	ExaBan	58.33%	91.52%
	Sig22	50.00%	85.46%
	AdaBan0.1	75.00%	92.73%
	MC50#vars	50.00%	85.46%

Runtime Performance. We first analyze the instances for which both algorithms succeeded. There are instances for which Sig22 failed and ExaBan succeeded, and there are no instances for which Sig22 succeeded and ExaBan failed. Table 3 shows that ExaBan clearly outperforms Sig22: Whenever both succeed for Academic and TPC-H, they are very fast, bar a few outliers for Sig22. ExaBan needs less than 0.4 and respectively 0.95 seconds for all instances. For hard instances for Sig22, ExaBan even achieves a speedup of up to 166x (229x) for TPC-H (Academic). For IMDB, ExaBan's speedup over Sig22 is already visible for simple instances, with a speedup of 25x for the 95-th percentiles. ExaBan also has a few performance outliers for IMDB.

Runtime Performance of ExaBan when Sig22 fails. Sig22 failed for 126 instances in Academic, 16239 instances in IMDB, and 24 instances in TPC-H. Table 4 summarizes the success rate and runtime performance of ExaBan for these instances. For Academic, ExaBan achieved near-perfect success and finished in less than ten minutes for all these instances. For IMDB, ExaBan succeeded in 77.4% of these instances. For 95% of these success cases, ExaBan finished in under ten minutes. For TPC-H, ExaBan succeeded in 41.7% of these instances; whenever it succeeded, its computation time was just over one minute. To summarize, ExaBan is generally faster and more robust than Sig22. We primarily attribute this to the fact that, in contrast to ExaBan, Sig22 requires to turn the lineage into a CNF representation, which may increase its size and complexity.

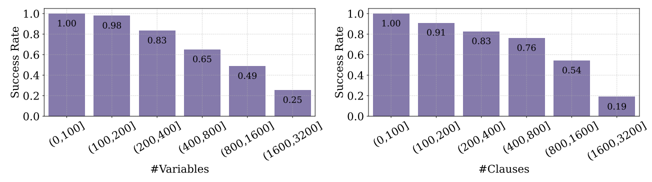
The effect of lineage size and structure. Figure 4 gives a breakdown analysis of the performance of ExaBan, grouped by the number of variables or clauses. ExaBan achieves near-perfect success rates and terminates in under a few seconds for instances with less than

Table 3: Runtime performance for exact Banzhaf computation in instances for which Sig22 succeeded.

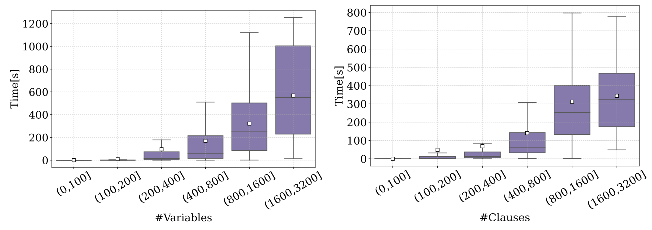
Dataset	Algorithm	Execution times [sec]						
		Mean	p50	p75	p90	p95	p99	Max
Academic	ExaBan	0.004	0.001	0.002	0.003	0.004	0.080	0.356
	Sig22	0.290	0.124	0.134	0.303	0.537	2.433	81.54
IMDB	ExaBan	0.323	0.002	0.008	0.066	0.231	2.174	1793
	Sig22	2.840	0.146	0.365	1.710	5.909	54.63	2271
TPC-H	ExaBan	0.713	0.892	0.905	0.935	0.935	0.941	0.941
	Sig22	1.217	0.080	0.140	0.200	0.260	1.450	157.3

Table 4: ExaBan runtime performance for instances on which Sig22 failed.

Dataset	Success rate	Execution times [sec]						
		Mean	p50	p75	p90	p95	p99	Max
Academic	99.2%	128.9	168.4	172.0	174.4	175.0	189.0	563.5
IMDB	77.4%	111.9	24.10	95.95	348.8	597.1	1055	1381
TPC-H	41.7%	53.77	56.44	60.24	63.27	66.23	68.59	69.18



(a) Success rate (average over all instances in each group)



(b) Execution time (ranges over all instances in each group)

Figure 4: Success rate and execution time of ExaBan across all database and queries, grouped by the number of variables (clauses) in the lineage. $[i, j]$ on the x-axis stands for the set of lineages with #vars (# clauses) between i and j.

200 variables or less than 100 clauses. ExaBan is successful in 25% (18%) of the instances with 1600-3200 variables (clauses).

5.3 Approximate Banzhaf Computation

We next examine the performance of AdaBan0.1 (with relative error 0.1) compared to ExaBan and MC.

Success Rate. Table 2 shows that AdaBan0.1's success rate is higher than that of ExaBan. Indeed, the former succeeded at least for all instances for which the latter also succeeded. For Academic, where the success rate of ExaBan is already near perfect, there is no further improvement brought by AdaBan0.1. For IMDB and TPC-H, however, AdaBan0.1 succeeds for 88.32% and respectively 75% of queries, a significant increase relative to ExaBan, which only succeeds for 82.23 % and respectively 58.33 % of queries. In particular, we have observed that AdaBan0.1 achieves a success rate of 74% (68 %) even for lineages with 1600-3200 variables (clauses), a significant improvement compared to the success rate of ExaBan for these cases. MC50#vars's success rate is comparable to that of ExaBan (but see the discussion below on execution time).

Table 5: Approximate versus exact Banzhaf computation for instances on which ExaBan succeeded.

Dataset	Algorithm	Execution times [sec]						
		Mean	p50	p75	p90	p95	p99	Max
Academic	AdaBan0.1	0.761	0.001	0.002	0.007	0.048	60.05	173.7
	ExaBan	2.065	0.001	0.002	0.012	0.197	164.5	563.5
	MC50#vars	>42.77	0.003	0.013	0.072	0.239	>3600	>3600
IMDB	AdaBan0.1	0.624	0.001	0.003	0.014	0.044	4.740	984.9
	ExaBan	1.579	0.002	0.003	0.009	0.077	10.374	1793
	MC50#vars	>13.99	0.012	0.039	0.386	2.613	257.1	>3600
TPC-H	AdaBan0.1	0.198	0.003	0.005	0.013	2.590	3.421	3.460
	ExaBan	4.227	0.895	0.931	0.938	51.05	61.98	69.18
	MC50#vars	>260.7	0.003	0.009	0.066	>3600	>3600	>3600

Table 6: AdaBan0.1 runtime performance and success rate for instances on which ExaBan failed.

Dataset	Success rate	Execution times [sec]						
		Mean	p50	p75	p90	p95	p99	Max
IMDB	49.53%	644.1	575.3	847.0	1105	1247	1584	1802
TPC-H	15.39%	166.3	166.3	166.4	166.4	166.4	166.4	166.4

Runtime Performance. Table 5 focuses on the instances on which ExaBan (and also AdaBan0.1) succeeds. AdaBan0.1 consistently outperforms both ExaBan and MC50#vars. The gains in the average runtime over ExaBan range from a factor of 3 for Academic to 20 for TPC-H. We further observe that MC50#vars is slower than ExaBan for over 99% of the examined instances, and even fails to terminate in 1 hour for some of the instances for which ExaBan succeeds. Thus, running MC with a larger number of samples to improve its accuracy (see below) is not useful.

Runtime Performance and Success Rate of AdaBan0.1 when other algorithms fail. Table 6 shows that, when only considering the instances on which ExaBan fails, AdaBan0.1 succeeds in nearly 50% (15%) of these instances for IMDB (TPC-H). Both ExaBan and AdaBan0.1 fail for just one instance in Academic.

Approximation Quality. AdaBan0.1 guarantees a (deterministic) relative error of 0.1. MC50#vars, however, only guarantees a (probabilistic) absolute error, where the number of samples required for a given error depends quadratically on the inverse of the error. Table 7 compares the observed approximation quality obtained for AdaBan0.1 and MC50#vars. These are measured as the ℓ_1 distance between the vectors of estimated Banzhaf values computed by each algorithm, compared to the ground truth exact Banzhaf values as computed by ExaBan. We show the results over all instances on which ExaBan succeeded, and separately over the ("Hard") instances for which ExaBan took at least five seconds. For all these instances, AdaBan0.1's approximation is consistently closer to the ground truth than MC50#vars's approximation by several orders of magnitude.

Table 7: Observed error ratio as ℓ_1 distance between the vectors of algorithm's output and of the exact normalized Banzhaf values for instances on which ExaBan succeeded.

Dataset	Algorithm	Mean	p50	p75	p90	p95	p99	Max
Academic	AdaBan0.1	5.24E-05	0	0	0	0	1.18E-03	2.09E-02
	MC50#vars	0.60	0.56	0.78	1.00	1.30	1.34	1.67
IMDB	AdaBan0.1	1.35E-04	0	0	0	7.77E-04	3.34E-03	1.92E-02
	MC50#vars	0.56	0.51	0.67	0.87	1.00	1.20	1.71
TPC-H	AdaBan0.1	9.04E-18	0	0	0	1.24E-24	3.23E-23	1.37E-15
	MC50#vars	0.50	0.44	0.67	1.00	1.34	1.34	1.34
Hard	AdaBan0.1	3.96E-04	2.40E-05	3.61E-04	1.19E-03	2.06E-03	4.21E-03	1.65E-02
	MC50#vars	0.312	0.303	0.383	0.4.65	0.516	0.64	1.13

Table 8: Observed precision@10 and precision@5 for instances on which ExaBan succeeded.

Dataset	Algorithm	Mean	p50	p75	p90	p95	p99	Min
Academic	IchiBan0.1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	0.9 / 1
	MC50#vars	0.87 / 0.90	0.9 / 1	0.8 / 0.8	0.7 / 0.6	0.5 / 0.6	0.3 / 0.4	0.2 / 0.2
	CNF Proxy	0.87 / 0.95	0.9 / 1	0.8 / 1	0.7 / 0.8	0.6 / 0.8	0.5 / 0.6	0.3 / 0.4
IMDB	IchiBan0.1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	0.6 / 0.4
	MC50#vars	0.90 / 0.87	0.9 / 1	0.8 / 0.8	0.7 / 0.6	0.6 / 0.6	0.5 / 0.4	0 / 0
	CNF Proxy	0.93 / 0.98	1 / 1	0.9 / 1	0.8 / 1	0.7 / 0.8	0.6 / 0.6	0.2 / 0.2
TPC-H	IchiBan0.1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	MC50#vars	0.34 / 0.84	0.1 / 1	0.1 / 1	0.1 / 0.2	0.1 / 0.2	0.1 / 0.11	0.1 / 0
	CNF Proxy	0.88 / 0.97	0.9 / 1	0.8 / 1	0.7 / 0.8	0.7 / 0.8	0.7 / 0.6	0.7 / 0.6

Approximation Error as Function of Time. Figure 5 presents, for several instances (appearing in [1]), the evolution of the observed error for AdaBan and MC over time. The error of AdaBan decreases exponentially and consistently over time, reaching a very small error within a few seconds. This is consistent with our observation that a small error ($\epsilon = 0.1$) is typically reached very quickly. In contrast, the behavior of MC is erratic and may not even converge within two hundred seconds.

5.4 Top-k Computation

We evaluate the accuracy of IchiBan0.1, which allows a relative error of up to 0.1, MC50#vars, and CNF Proxy using the standard measure of precision@k, which is the fraction of reported top-k tuples that are in the ground truth top-k set. Table 8 gives the distribution of precision@k values observed for different instances and $k \in \{5, 10\}$. With the exception of some outliers for IMDB, IchiBan0.1 achieves near perfect precision@k, while MC50#vars is much less stable and consistently inferior. CNF Proxy is more accurate than MC50#vars, but is also consistently outperformed by IchiBan0.1. The results for $k = 1, 3$ are omitted for lack of space: for $k = 1$, all algorithms achieve high success rates; for $k = 3$ the observed trends are similar to those in the table. The execution time of IchiBan0.1 is essentially the same as reported for AdaBan0.1, i.e. typically an order of magnitude better than ExaBan.

We further run the variant of IchiBan that decides the top-k results with certainty (deferred to the extended report [2]): for top-1, it is extremely fast; we found out that in many instances, there is a clear top-1 fact, whose Banzhaf value is much greater than of the others. For top-3 and top-5, it achieved better performance over IMDB than both ExaBan and AdaBan0.1. This was however not the case for TPC-H, where separating the top-3 or top-5 facts from the rest took longer than ExaBan. We attribute this to a large number of ties in the Banzhaf values of facts for the TPC-H workload. IchiBan0.1 is a good alternative for such instances.

5.5 Summary of Experimental Findings

Our experimental findings lead to the following main conclusions:

(1) *ExaBan consistently outperforms Sig22 for exact computation.* Sec. 5.2 shows that ExaBan not only outperforms Sig22 on the workloads previously used for Sig22, but it also succeeds in many cases where Sig22 times out (41.7%-99.2% of these cases for the different datasets).

(2) *AdaBan outperforms ExaBan already for small relative errors.* Sec. 5.3 shows that AdaBan is up to an order of magnitude, and on average three times faster than ExaBan for relative error 0.1.

(3) *The accuracy of MC can be orders of magnitude worse than that of AdaBan.* Sec. 5.3 shows that if we only run MC for a sufficiently small number of steps so that its runtime remains competitive to AdaBan, then its accuracy can be up to four orders of magnitude worse than that of AdaBan. On the other hand, if we were to run MC sufficiently many steps to achieve a comparable accuracy, then its runtime becomes infeasible.

(4) *IchiBan can quickly identify the top-k facts.* Sec. 5.4 shows that IchiBan quickly and accurately separates the approximation intervals of the first k Banzhaf values (demonstrated for $k = 1, 3, 5, 10$) from the rest, and is significantly more accurate than previous approaches based on MC or CNF Proxy.

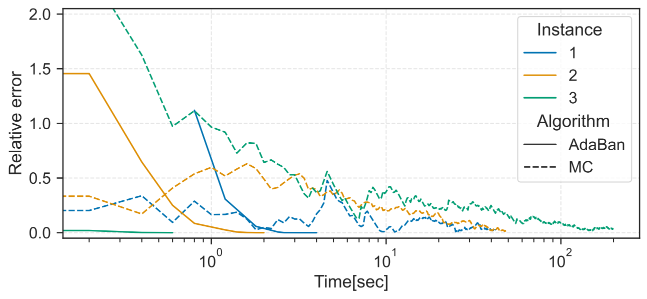
6 RELATED WORK

We compare our work to multiple lines of related work.

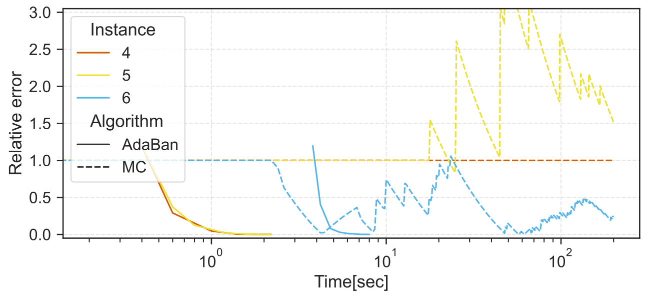
Shapley Value. Recent work [9, 18, 19, 29, 33–35, 45] investigated the use of the Shapley value [48] to define attribution scores in query answering, with particular focus on algorithms and the complexity of computing exact and approximate Shapley values for facts. The Banzhaf value [7, 44] is very closely related to the Shapley value, and both have been extensively investigated in Game Theory [23, 32, 49, 52]. They have the same formula up to combinatorial coefficients that are present in the Shapley value formula and missing in the Banzhaf value formula; different coefficients need to be computed for each size of variable set, and are multiplied by the number of sets of this size. Computationally, we have empirically shown advantages of the approach presented here over prior work. Furthermore, our algorithmic and theoretical contributions do not have a parallel in the literature on Shapley or Banzhaf values for query answering (to our knowledge). Specifically, ours are the first deterministic approximation and ranking algorithms with provable guarantees, whereas approximation in previous works is based on Monte Carlo [19, 35] and ranking is only heuristic [19].

Banzhaf-based ranking and Shapley-based ranking can differ already for the simple query $Q(X) = R(X) \wedge S(X,Y) \wedge T(X,Z)$ (details in [2]). Our dichotomy result, that has no known parallel for Shapley-based ranking, establishes that Banzhaf-based ranking is tractable precisely for the same class of hierarchical queries for which also exact computation of the Shapley (and Banzhaf) value [35] is tractable. The hierarchical property has led to several other dichotomies, e.g., for probabilistic query evaluation [15], incremental view maintenance [8], one-step streaming evaluation in the finite cursor model [26], and readability of query lineage [43].

Hardness of Exact Banzhaf Computation. Prior work shows that for non-hierarchical self-join free CQs, computing exact Banzhaf



(a) Three instances for which MC converged to the Banzhaf value.



(b) Three instances for which MC did not converge to the Banzhaf value.

Figure 5: Relative approximation error as function of AdaBan and MC runtime for representative instances.

values of database facts is $FP^{\#P}$-hard [35]. The proof is by a reduction from the $FP^{\#P}$-hard problem of evaluating non-hierarchical queries over probabilistic databases [16]. Our argument for the hardness of Banzhaf-based ranking is different. It relies on the conjecture that there is no polynomial-time approximation for counting the independent sets in a bipartite graph [14, 21].

Further attribution measures in Query Answering. In addition to game theory-based methods for attribution in query answering, previous works have proposed several other approaches. Causality-based methods focus on uncovering the causal responsibility of database facts for a query outcome [37, 38, 46]. The causal responsibility of a fact f is a score proportional to the largest fact set such that including f to the set turns the result of the query from false to true. Furthermore, recent work has empirically evaluated various attribution methods for the problem of credit distribution [20]. Their study compares game theory-based methods with approaches based on causal responsibility and simpler methods like fact frequency counting in the provenance. They highlight both the similarities and differences among these attribution approaches.

Attribution in machine learning. SHAP (SHapley Additive exPlanations) values is a popular method for attributing feature importance in machine learning models [36]. It builds upon the concept of Shapley values, but crucially differs in that it models missing "players" (feature values in the context of machine learning) according to their expectation. Recent line of work studies the complexity of SHAP score computation [4–6]. In [6] the authors show that under commonly accepted complexity assumptions, there is no polynomial-time algorithm for ranking based on SHAP scores, even for monotone DNF. Our proof uses different techniques, and it is not trivial to establish a direct reduction between the two problems.

Approximation algorithms. Our algorithm AdaBan relies on the anytime deterministic approximation framework originally introduced for (ranked) query evaluation in probabilistic databases [24, 41, 42]. In particular, AdaBan uses the incremental and shared compilation of query lineage into partial d-trees for approximate computation, ranking, and top-k. Besides the general approximation framework, AdaBan differs significantly from this prior work as it is tailored at Banzhaf value computation and Banzhaf-based ranking as opposed to probability computation. In particular, AdaBan uses lower and upper bounds for the Banzhaf values in functions represented (1) in independent DNF and (2) by disjunctions and conjunctions of mutually exclusive or independent functions. These bounds need also be computed for each variable in the function rather than for the entire function.

Prior work [35] gives a polynomial time randomized approximation scheme for Shapley (and Banzhaf) values based on Monte Carlo sampling. Sec. 5 shows experimentally that our AdaBan significantly outperforms this randomized approach. As also witnessed for ranking in probabilistic databases [42], randomized approximations based on Monte Carlo sampling have three important limitations, which are not shared by our determinstic approximation AdaBan: (1) the achieved ranking is only a probabilistic approximation of the correct one; (2) running one more Monte Carlo step does not necessarily lead to a refinement of the approximation interval, and hence the approximation is not truly incremental; (3) The sampling approach sees the functions as black boxes and does not exploit their structure. Sec. 5 also reports on experiments with the CNF Proxy heuristic [19], which efficiently rank facts based on a proxy value; though the algorithms has no theoretical guarantees, [19] has shown that the produced ranking is often similar to ranking based on the real Shapley values (though the proxy values are typically *not* similar to the Shapley values). Here we show that our algorithm also outperforms CNF Proxy in terms of the produced ranking.

7 CONCLUSION

We have studied in this paper the problem of approximating the Banzhaf values, quantifying fact contribution in query answering, and the related problem of ranking facts based on their underlying Banzhaf values. We have shown a dichotomy for the complexity of ranking for self-join-free CQs: ranking is achievable in PTIME precisely for the class of heirarchical queries (under plausible complexity assumptions). This is in contrast to the existence of a fully polynomial approximation scheme for the entire class of UCQs. We have then introduced an exact (but possibly EXPTIME) computation algorithm and an anytime approximation algorithm, and shown that both are significantly superior to the state-of-the-art for the respective problems they solve. Remaining open problems include the extension of our results to other attribution measures, including those mentioned in Sec. 6. We also plan to study the problems for more expressive query classes, including aggregation and negation, and to explore the development of further dedicated optimizations.

REFERENCES

[1] Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. Banzhaf values for facts in query answering. https://github.com/Omer-Abramovich/AdaBan, 2023.

[2] Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. Banzhaf Values for Facts in Query Answering (Extended version). https://github.com/Omer-Abramovich/AdaBan/blob/main/Ranking_Banzhaf_Values.pdf, 2023.

[3] Dana Arad, Daniel Deutch, and Nave Frost. Learnshapley: Learning to predict rankings of facts contribution based on query logs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4788–4792, 2022.

[4] Marcelo Arenas, Pablo Barceló, Leopoldo Bertossi, and Mikaël Monet. On the complexity of shap-score-based explanations: Tractability via knowledge compilation and non-approximability results. *arXiv preprint*, 2021.

[5] Marcelo Arenas, Pablo Barceló, Leopoldo Bertossi, and Mikaël Monet. The tractability of SHAP-score-based explanations over deterministic and decomposable boolean circuits. In *Proceedings of AAAI*, 2021.

[6] Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. On the complexity of shap-score-based explanations: Tractability via knowledge compilation and non-approximability results. *J. Mach. Learn. Res.*, 24:63:1–63:58, 2023.

[7] J.F. Banzhaf. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19(2):317–343, 1965.

[8] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318, 2017.

[9] Leopoldo Bertossi, Benny Kimelfeld, Ester Livshits, and Mikaël Monet. The shapley value in database management.

[10] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330. Springer, 2001.

[11] Adriane Chapman and HV Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534, 2009.

[12] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[13] Yingwei Cui, Jennifer Widom, and Janet L Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179–227, 2000.

[14] Radu Curticapean, Holger Dell, Fedor V. Fomin, Leslie Ann Goldberg, and John Lapinskas. A fixed-parameter perspective on #bis. *Algorithmica*, 81(10):3844–3864, 2019.

[15] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.

[16] Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30:1–30:87, 2012.

[17] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[18] Susan B. Davidson, Daniel Deutch, Nave Frost, Benny Kimelfeld, Omer Koren, and Mikaël Monet. Shapgraph: An holistic view of explanations through provenance graphs and shapley values. In *SIGMOD*, pages 2373–2376, 2022.

[19] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. Computing the shapley value of facts in query answering. In *SIGMOD*, pages 1570–1583, 2022.

[20] Dennis Dosso, Susan B. Davidson, and Gianmaria Silvello. Credit distribution in relational scientific databases. *Inf. Syst.*, 109:102060, 2022.

[21] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

[22] Algaba E, Prieto A, and Saavedra-Nieves A. Risk analysis sampling methods in terrorist networks based on the Banzhaf value. *Risk Anal.*, PMID: 37210375, May 2023.

[23] Vincent Feltkamp. Alternative axiomatic characterizations of the shapley and banzhaf values. *International Journal of Game Theory*, 24:179–186, 1995.

[24] Robert Fink, Jiewen Huang, and Dan Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6):823–848, 2013.

[25] Todd J Green and Val Tannen. The semiring framework for database provenance. In *Proceedings of PODS*, pages 93–99, 2017.

[26] Martin Grohe, Yuri Gurevich, Dirk Leinders, Nicole Schweikardt, Jerzy Tyszkiewicz, and Jan Van den Bussche. Database query processing using finite cursor machines. *Theory Comput. Syst.*, 44(4):533–560, 2009.

[27] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. A survey on provenance: What for? what form? what from? *VLDB J.*, 26(6):881–906, 2017.

[28] Adam Karczmarz, Tomasz P. Michalak, Anish Mukherjee, Piotr Sankowski, and Piotr Wygocki. Improved feature importance computation for tree models based on the banzhaf value. In *UAI*, volume 180 of *Proceedings of Machine Learning Research*, pages 969–979. PMLR, 2022.

[29] Majd Khalil and Benny Kimelfeld. The complexity of the shapley value for regular path queries. In Floris Geerts and Brecht Vandevoort, editors, *26th International Conference on Database Theory, ICDT 2023, March 28-31, 2023, Ioannina, Greece*, volume 255 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[30] Werner Kirsch and Jessica Langner. Power indices and minimal winning coalitions. *Soc. Choice Welf.*, 34(1):33–46, 2010.

[31] Seokki Lee, Bertram Ludäscher, and Boris Glavic. Approximate summaries for why and why-not provenance. *Proc. VLDB Endow.*, 13(6):912–924, 2020.

[32] Ehud Lehrer. An axiomatization of the banzhaf value. *International Journal of Game Theory*, 17:89–99, 1988.

[33] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The shapley value of tuples in query answering. In *ICDT*, volume 155, pages 20:1–20:19. Schloss Dagstuhl, 2020.

[34] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. Query games in databases. *SIGMOD Rec.*, 50(1):78–85, 2021.

[35] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The shapley value of tuples in query answering. *Log. Methods Comput. Sci.*, 17(3), 2021.

[36] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[37] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proc. VLDB Endow.*, 4(1):34–45, 2010.

[38] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. Causality and explanations in databases. *Proceedings of the VLDB Endowment (PVLDB)*, 7(13):1715–1716, 2014.

[39] Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. Going beyond provenance: Explaining query answers with pattern-based counterbalances. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 485–502. ACM, 2019.

[40] Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340. Springer, 2008.

[41] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

[42] Dan Olteanu and Hongkai Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*, pages 282–293, 2012.

[43] Dan Olteanu and Jakub Zavodny. Factorised representations of query results: size bounds and readability. In *ICDT*, pages 285–298, 2012.

[44] L. S. Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109(1):53–57, 1946.

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

[46] Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. Quantifying causal effects on query answering in databases. In *TaPP*. USENIX Association, 2016.

[47] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. Provsql: Provenance and probability management in postgresql. *Proceedings of the VLDB Endowment (PVLDB)*, 11(12):2034–2037, 2018.

[48] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[49] Philip D Strafiin Jr. The shapley—shubik and banzhaf power indices as probabilities. *The Shapley value: essays in honor of Lloyd S. Shapley*, page 71, 1988.

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

[51] Jianyuan Sun, Guoqiang Zhong, Kaizhu Huang, and Junyu Dong. Banzhaf random forests: Cooperative game theory based random forests with consistency. *Neural Networks*, 106:20–29, 2018.

[52] Rene Van den Brink and Gerard Van der Laan. Axiomatizations of the normalized banzhaf value and the shapley value. *Social Choice and Welfare*, 15:567–582, 1998.

[53] Diego Varela and Javier Prado-Dominguez. Negotiating the Lisbon Treaty: Redistribution, Efficiency and Power Indices. *Czech Economic Review*, 6(2):107–124, July 2012.

[54] Jiachen T. Wang and Ruoxi Jia. Data banzhaf: A robust data valuation framework for machine learning. In *AISTATS*, volume 206 of *Proceedings of Machine Learning Research*, pages 6388–6421. PMLR, 2023.