Banzhaf Values for Facts in Query Answering

OMER ABRAMOVICH, Tel Aviv University, Israel
DANIEL DEUTCH, Tel Aviv University, Israel
NAVE FROST, eBay, Israel
AHMET KARA, University of Zurich, Switzerland
DAN OLTEANU, University of Zurich, Switzerland

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.

CCS Concepts: • **Theory of computation → Data provenance**; • **Information systems → Database design and models**.

Additional Key Words and Phrases: Banzhaf Values, Lineage, Explanations

ACM Reference Format:
Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. 2024. Banzhaf Values for Facts in Query Answering. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 123 (June 2024), 26 pages. https://doi.org/10.1145/3654926

1 INTRODUCTION

Explaining the answer to a relational query is a fundamental problem in data management [9–12, 23, 24, 28, 36, 37]. Explanations typically rely on various forms of lineage [9, 12, 22] that records which database tuples participate in the computation of a given query answer. A detailed and compact explanation can be obtained by quantifying the contribution of each tuple in the lineage to the query answer. Several such quantification measures have been proposed in recent years (see Section 7). The *Banzhaf* value [7, 42] is one such measure and the focus of this work. It originated in Cooperative Game Theory where it assigns scores to players based on their marginal

Authors' addresses: Omer Abramovich, Tel Aviv University, Tel Aviv, Israel, omera1@mail.tau.ac.il; Daniel Deutch, Tel Aviv University, Tel Aviv, Israel, danielde@post.tau.ac.il; Nave Frost, eBay, Netanya, Israel, nafrost@ebay.com; Ahmet Kara, University of Zurich, Zurich, Switzerland, ahmet.kara@uzh.ch; Dan Olteanu, University of Zurich, Zurich, Switzerland, dan.olteanu@uzh.ch.

ACM 2836-6573/2024/6-ART123
https://doi.org/10.1145/3654926

contribution to every subset of the other players. It has found applications in many domains. Most prominently, it is used as a measure of voting power in the analysis of voting in the Council of the European Union [54]. It was shown to provide more robust data valuation across subsequent runs of stochastic gradient descent than alternative scores such as the Shapley value [55]. 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 [25]. In Banzhaf random forests [52], 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 [19].

Example 1.1. Figure 1 illustrates a DBLP-like database of research papers, their authors and topics, and a query that retrieves all topics of papers written by authors working in universities. Consider the output tuple "data science". One way to explain this output tuple is using its lineage, which is derived by the query using all database tuples. The analyst may ask: *how much* does each database tuple contribute to the output tuple? Which tuples contribute most? In case we only use the topmost two tuples (marked a_1–a_2) in the Author table and the topmost three tuples (marked w_1–w_3) in the Writes table, we would expect Alice to be more influential than Bob, since she wrote more papers on data science. Yet in case we consider all database tuples, is it less clear how to pick the most influential author in data science: Alice, Carol and David have each co-authored two papers, while Bob has one single-authored paper.

`Author` (endo)

	Name	Org
a_1	Alice	The Uni
a_2	Bob	The Uni
a_3	Carol	The Uni
a_4	Dave	The Uni

`Publication` (endo)

	Paper	Title	Conf
p_1	1	Bat in the hat	SIGMOD
p_2	2	Dog in the fog	SIGMOD
p_3	3	Hen in the den	VLDB

`Writes` (exo)

	Author	Paper
w_1	Alice	1
w_2	Alice	2
w_3	Bob	3
w_4	Carol	1
w_5	Carol	2
w_6	Dave	1
w_7	Dave	2

`Topics` (exo)

	Paper	Topic
t_1	1	data science
t_2	2	data science
t_3	3	data science

`Query Q`

$$Q(t) := \textsc{Author}(x, \text{“}The\ Uni\text{”}),\ \texttt{Writes}(x, y),\ \texttt{Publication}(y, \ell, c),\ \texttt{Topics}(y, t)$$

Fig. 1. A DBLP-like academic database and a query. Each fact is associated with a lineage variable written next to it. The facts in the relations `Author` and `Publication` are endogenous, all other facts are exogenous.

The Banzhaf value can be used to resolve this conundrum: Whereas in the first case Alice is more influential than Bob, in the second case Bob becomes more influential than Alice. The Banzhaf value quantifies the marginal contribution of each database tuple (paper and author) to each query output tuple (paper topic). More precisely, it identifies which input tuple f are *critical* to each output tuple t: f is critical for a subset D' of the database (or sub-database for short) if t is not an output tuple for D', but t becomes an output tuple for $D' \cup \{f\}$. The Banzhaf value of f is the number of such sub-databases to which it is critical. The Shapley value [49] can also be used for this quantification. The difference lies in the manner in which these quantities are aggregated: the Shapley value weighs the contribution based on the size of sub-database, whereas the Banzhaf value does not. Section 5 compares the two measures.

Example 1.2. Reconsider Figure 1 (with the Author and Writes tables restricted to the two and respectively three topmost tuples). We make the common distinction between endogenous and exogenous facts, where we only wish to compute the influence of the former and the latter are considered given. The tuple representing Alice (marked a_1) is critical for any sub-database that includes p_1 or p_2 but does not include p_3. Intuitively, this is because p_3 is essential for any derivation of the query result that does not involve Alice. By contrast, if the tuple representing Bob (a_2) were missing, alternative derivations could use either p_1 or p_2. Hence, there are less sub-databases for which a_2 is critical, compared to a_1.

Since Banzhaf values are defined by the contributions to all possible sub-databases, their definition does not give an efficient way of computing them. This paper starts a systematic investigation of both theoretical and practical facets of three computational problems for Banzhaf values in query answering: exact computation, approximation, and ranking. Our contributions are as follows.

1. Exact Banzhaf Computation (Section 3.1). 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 data structure called d-tree [21]. 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. This approach is justified by the observation that if we have the Banzhaf values for independent or mutually exclusive functions, we can then compute 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 [16], 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 (Section 3.2). 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 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 $\epsilon = 0.1$.

[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 [32] that is attained by AdaBan for $\epsilon = 0$.

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 (Section 4.1). We also introduce IchiBan, an algorithm that ranks facts and selects the top-k facts based on their Banzhaf values. Reconsidering our running example, we may ask which authors have the highest influence on a given topic. 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 answers, in practice it achieves near-perfect results. This is true even in cases where previous work [16], 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 (Section 4.2). 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 [32]. 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 [13, 18].

5. Comparison with Shapley Values (Section 5). The Shapley value [49] is a measure closely related to Banzhaf and previously used for explanation in query answering [16, 32]. We show that both

measures yield very similar rankings in practice. ExaBan can be easily adjusted to compute the exact Shapley values. This new algorithm, called ExaShap, runs slower than ExaBan since it needs to perform additional multiplications, yet it is faster than the state-of-the-art algorithm for Shapley values [16].

D-tree for fact attribution. To the best of our knowledge, there is no prior work on Banzhaf or Shapley value computation based on d-trees. D-trees, as well as other methods to compile Boolean functions into tractable forms, have been previously used to compute the probability of *output tuples* for queries over probabilistic databases [21]. Our use of d-tree is different: We compute the Banzhaf value for the contribution of each *input tuple* to each output tuple. Consequently, novel methods for the computation of exact values as well as bounds for approximation and ranking, are needed. The common denominator of both uses is the efficient counting of the satisfying assignments (or models) of a Boolean function. It is *weighted* model counting in probabilistic databases, where different satisfying assignments may have different probabilities or weights, and model counting for sub-functions in Banzhaf computation.

2 PRELIMINARIES

We denote by $\mathbb{N}$ the set of natural numbers including 0. For $n \in \mathbb{N}$, we denote $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. 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 $\varphi : \boldsymbol{X} \to \{0, 1\}$ 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 $\theta : \boldsymbol{X} \to \{0, 1\}$. 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.

Databases and Queries. We assume familiarity with the standard notions of relational databases (see [1]). Following prior work [32], we assume that the database is partitioned into a set D_n of *endogenous* and a set D_x of *exogenous* facts. A *conjunctive query* (CQ) Q has the form $Q(X) : -R_1(X_1), \ldots, R_n(X_n)$ where $R_1, \ldots, R_n$ are relation names; $X_1, \ldots, X_n$ are tuples including variables and constants whose arity match that of the corresponding relations; and X, called the head variables, is a subset of the variables occurring in $X_1, ..., X_n$. Each $R_i(X_i)$ is an atom of Q. If X is the empty set, we say that Q is *boolean*. Figure 1 includes an example of a CQ Q. A *union of conjunctive queries* (UCQ) consists of a set of CQs $\{Q_1, \ldots, Q_k\}$ whose head has the same arity.

We denote by $at(X)$ the set of atoms with the query variable X. 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.

Query Grounding. A grounding of a CQ Q w.r.t. a database D is an assignment G of constants to all variables of Q such that, after replacing every variable v by $G(v)$, every atom of Q corresponds to a fact in D. We use $facts(G)$ to denote this set of facts. A grounding for a UCQ is a grounding for one of its CQs. Each grounding G yields an output tuple, which is the restriction of G to the head variables of Q. Multiple groundings may yield the same tuple: we use $G(Q, D, t)$ to denote the set of groundings yielding t. We use $Q(D)$ to denote the set of output tuples for Q w.r.t. D.

Example 2.1. The following groundings of the query in Figure 1 yield the output (`data science`): $G_1 = \{x \mapsto \texttt{Alice}, y \mapsto 1, l \mapsto \texttt{Bat in the hat}, c \mapsto \texttt{SIGMOD}, t \mapsto \texttt{data science}\}$; $G_2 = \{x \mapsto \texttt{Bob}, y \mapsto 3, l \mapsto \texttt{Hen in the den}, c \mapsto \texttt{VLDB}, t \mapsto \texttt{data science}\}$.

Banzhaf Values of Database Facts. Consider a UCQ Q, a database $D = (D_n, D_x)$, an output tuple t, and an endogenous fact $f \in D_n$. We denote by $I(Q, D, t)$ the indicator function whose value is 1 if $t \in Q(D)$ and 0 otherwise. We then define the Banzhaf value [4] of a fact f as its marginal contribution to the existence of t in the query output, aggregated over all sub-databases:

$$Banzhaf(Q, D, f, t) = \Sigma_{D' \subseteq (D_n \setminus \{f\})} I(Q, D' \cup D_x \cup \{f\}, t) - I(Q, D' \cup D_x, t) \tag{1}$$

Note that since UCQs are monotone, $Banzhaf(Q, D, f, t)$ equals the number of sub-databases D' for which $t \in Q(D' \cup D_x \cup \{f\})$ while $t \notin Q(D' \cup D_x)$. By counting these sub-databases, the Banzhaf value intuitively reflects the extent to which the fact is replaceable in deriving a given output tuple of interest.

Example 2.2. Consider the query Q and database D in Figure 1. We assume that all facts in the relations `Author` and `Publication` are endogenous (contained in D_n) and all other facts are exogenous (contained in D_x). We identify the facts in the relations `Author` and `Publication` by the variables a_1-a_4 and respectively p_1-p_3.

We measure the impact of the existing authors on the topic data science using the Banzhaf value. A fact f is critical for a set $D' \subseteq D_n \setminus \{f\}$ if $(\texttt{data science}) \notin Q(D' \cup D_x)$ but $(\texttt{data science}) \in Q(D' \cup D_x \cup \{f\})$. Hence, to determine the Banzhaf value of f, it suffices to count all sets $D' \subseteq D_n \setminus \{f\}$ for which f is critical.

Let us first consider the relations `Author` and `Writes` restricted to their two and respectively three topmost facts. Alice (a_1) and Bob (a_2) have two and respectively one paper on data science. The Banzhaf value of a_1 is then indeed higher than the one of a_2, as detailed next. For each set D' of endogenous facts for which a_1 is critical, one of the following two cases holds: Either $a_2 \notin D'$ and $\{p_1, p_2\} \cap D' \neq \emptyset$; or $a_2 \in D'$, $\{p_1, p_2\} \cap D' \neq \emptyset$, and $p_3 \notin D'$. The overall number of such sets is nine. The sets for which a_2 is critical has one of the following properties: Either $a_1 \notin D'$ and $p_3 \in D'$; or $a_1 \in D'$, $\{p_1, p_2\} \cap D' = \emptyset$, and $p_3 \in D'$. There are five such sets. Hence, the Banzhaf value of Alice is higher than the one of Bob.

Now, consider the full database in Figure 1. In this case, Alice, Carol, and Dave have co-authored two papers (p_1 and p_2), whereas Bob has one paper (p_3), which is single-authored. Ranking the three academics by the number of their papers would put Bob below all the others. However, the Banzhaf value of the fact representing Bob is eleven, while the value for each of the other three academics is nine, which means that Bob has the most impact.

Query Lineage. Let a database $D = D_n \cup D_x$. We associate each endogenous fact f in D_n with a propositional variable denoted by $v(f)$. Given a UCQ Q, a database D and an output tuple t, the lineage of t with respect to Q over D, denoted by $lin(Q, D, t)$, is a positive Boolean function in Disjunctive Normal Form over the variables $v(f)$ of facts f in D_n, defined as follows:

$$lin(Q, D, t) \stackrel{\text{def}}{=} \bigvee_{G \in G(Q,D,t)} \bigwedge_{f \in facts(G) \cap D_n} v(f) \tag{2}$$

Each clause in the lineage is a conjunction of variables associated with facts participating in a grounding.

[4] Our results also apply to normalized versions of the Banzhaf value such as the *Penrose–Banzhaf power* or *Penrose–Banzhaf index* [27].

Example 2.3. Consider again the query Q and database D in Figure 1, where we restrict `Author` and `Writes` to their first two and respectively three facts. Then, $lin(Q, D, (\texttt{data science})) = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$. The variables of exogenous facts (in `Writes` and `Topics`) are excluded from the lineage.

Banzhaf values in query answering may be computed based on the lineage, rather than directly using the query and the database:

Definition 2.4 (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:

$$Banzhaf(\varphi, x) \stackrel{\text{def}}{=} \sum_{Y \subseteq \boldsymbol{X} \setminus \{x\}} \varphi[Y \cup \{x\}] - \varphi[Y] \tag{3}$$

Each choice of a subset Y of variables to be assigned *true* corresponds to a choice of sub-database including exactly the tuples whose variables are in Y. Each summand in Eq. (3) is either 0 or 1, depending on whether adding x to the subset Y of variables chosen to be assigned *true*, flips the truth value of φ from 0 to 1.

When $\varphi = lin(Q, D, t)$ is positive, it can be shown:

$$Banzhaf(Q, D, f, t) = Banzhaf(lin(Q, D, t), v(f)) \tag{4}$$

Example 2.5. Consider the lineage from Example 2.3: $lin(Q, D, (\texttt{data science})) = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$. Eq. (4) gives $Banzhaf(Q, D, a_1, (\texttt{data science})) = Banzhaf(lin(Q, D, (\texttt{data science})), a_1)$. This relies on the fact that any satisfying assignment for this lineage corresponds to a database over which the query outputs the tuple (`data science`). For instance, any assignment that sets a_1 and p_1 to true satisfies the lineage. Indeed, Q outputs (`data science`) when evaluated over any database that contains all exogenous facts and the facts (`Alice`, `The Uni`) and (1, `Bat in the hat`, `SIGMOD`) identified by a_1 and p_1, respectively.

Computing Banzhaf values based on the lineage is in practice much more efficient than computing them based on their direct definition, since the latter involves costly re-evaluation of the query on sub-databases. Because of this, and since the lineage itself is efficiently computable [22], we will focus in the remainder of the paper on the problem of Banzhaf computation over lineage expressions.

Finally, we give a useful characterization of Banzhaf values via counting satisfying assignments. Let $\#\varphi$ be the number of satisfying assignments for a positive formula φ. Based on [32], we get:

$$Banzhaf(\varphi, x) = \#\varphi[x := 1] - \#\varphi[x := 0] \tag{5}$$

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 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:

$$\#\varphi = \#\varphi_1 \cdot \#\varphi_2 \tag{6}$$

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

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

$$\#\varphi = \#\varphi_1 \cdot 2^{n_2} + 2^{n_1} \cdot \#\varphi_2 - \#\varphi_1 \cdot \#\varphi_2 \tag{8}$$

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) \cdot (2^{n_2} - \#\varphi_2), \tag{9}$$

  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:

$$\#\varphi = \#\varphi_1 + \#\varphi_2 \tag{10}$$

$$Banzhaf(\varphi, x) = Banzhaf(\varphi_1, x) + Banzhaf(\varphi_2, x) \tag{11}$$

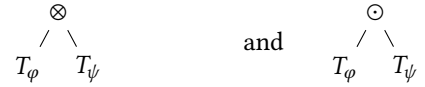
The derivations of these formulas are given in the extended version of this paper [2].

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 [39]. For such functions, Eq. (6) to (9) 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 [51] and exact Shapley computation [16], and borrowed from knowledge compilation [14], 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.

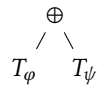
In this paper, we compile the query lineage into a *decomposition tree* [21]. Such trees have inner nodes that are the logical operators enhanced with information about independence and mutual exclusiveness of their children: $\otimes$ stands for independent-or, $\odot$ for independent-and, and $\oplus$ for mutual exclusion.

Definition 3.1. [21] 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 ψ (resp.), the following are d-trees for $\varphi \vee \psi$ and $\varphi \wedge \psi$ (resp.).



- If T_φ and T_ψ are d-trees for mutually exclusive functions φ and resp. ψ, then the following is a d-tree for $\varphi \vee \psi$.



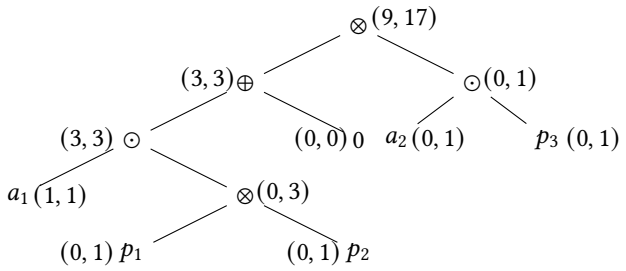


Fig. 2. Complete d-tree of the lineage $lin(Q, D, \texttt{data science}) = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$ for the query Q and database D from Figure 1, where the relations `Author` and `Writes` are restricted to the first two and respectively three facts. Each node is associated with the pair of the Banzhaf value of a_1, representing Alice, and the model count computed for the subtree rooted at that node.

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 [21]. 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. Figure 2 shows a complete d-tree for the lineage $\varphi = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$ from example Ex. 2.3 (ignore the pair of numbers associated with each node for now). We first observe that the function $\varphi_1 = (a_1 \wedge p_1) \vee (a_1 \wedge p_2)$ consisting of the first two conjunctive clauses of φ is independent of the last conjunctive clause $\varphi_2 = (a_2 \wedge p_3)$. This means that each pair of satisfying assignments for φ_1 and φ_2 can be composed into a single satisfying assignment for φ. We decompose φ into the independent-or of φ_1 and φ_2. To decompose φ_1, we apply Shannon expansion on a_1 and obtain the two mutually exclusive functions $\varphi_{10} = \neg a_1 \wedge \varphi_1[a_1 := 0] = 0$ and $\varphi_{11} = a_1 \wedge \varphi_1[a_1 := 1] = a_1 \wedge (p_1 \vee p_2)$. The functions φ_{11} and φ_2 can be further decomposed into independent functions until we obtain a complete d-tree.

Alternatively, we can factor out a_1 in $\varphi_1 = (a_1 \wedge p_1) \vee (a_1 \wedge p_2)$ to obtain the function $a_1 \wedge (p_1 \vee p_2)$, and compile φ_1 into the d-tree $a_1 \odot (p_1 \otimes p_2)$. Our algorithm computing d-trees does this simplification whenever a variable occurs in all clauses.

Fig. 3 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. (6) to (11) 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}.

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

$B := 0$; $\# := 0$; //initialization
switch T_φ
  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_{φ_1} op T_{φ_2}:
    $(B_i, \#_i) :=$ ExaBan(T_{φ_i}, x) for $i \in [2]$
    $n_i :=$ number of variables in T_{φ_i} for $i \in [2]$
    switch op
      case $\odot$: //wlog if x is in φ, then it is in φ_1
      $B := B_1 \cdot \#_2$; $\# := \#_1 \cdot \#_2$
      case $\otimes$: //wlog if x is in φ, then it is in φ_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 φ_1 and φ_2 have same variables
        $B := B_1 + B_2$; $\# := \#_1 + \#_2$
return $(B, \#)$

Fig. 3. Computing the exact Banzhaf value for a variable x and the model count over a complete d-tree.

Proposition 3.3. *For any positive DNF function φ, complete d-tree T_φ for φ, and variable x in φ, ExaBan$(T_\varphi, x) = (Banzhaf(\varphi, x), \#\varphi)$.*

Example 3.4. The value pairs associated with the nodes of the complete d-tree in Figure 2 show the trace of the computation of ExaBan for the d-tree and the variable a_1, which stands for Alice in the database from Figure 1. Each node is associated with the pair of the Banzhaf value and the model count for the subtree rooted at that node. Hence, the first component 9 of the pair associated with the root is the Banzhaf value of Alice within the decomposed lineage. We detail the computation of the values $(3, 3)$ at the $\odot$-node in the left child tree of the root. This node is an independent-and. The variable a_1 is in the left subtree. ExaBan computes the Banzhaf value 3 of a_1 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 $\psi = a_1 \wedge (p_1 \vee p_2)$. Indeed, every model of the function must satisfy a_1 and at least one of p_1 and p_2, which implies $\#\psi = 3$. Using Eq. (5), we get $Banzhaf(\psi, a_1) = \#\psi[a_1 := 1] - \#\psi[a_1 := 0] = 3 - 0 = 3$.

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

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 bound 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.

We next explain how to efficiently compute upper and lower bounds for positive DNF functions, albeit without any error guarantee. We then introduce AdaBan, which uses the bounds to compute and incrementally refine approximation intervals for d-trees.

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 [21], which are positive DNF functions where every variable occurs once. Whereas the first three aforementioned properties are already known to hold for iDNF functions [21], 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. (6) and (8) 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 Prop. 3.5:

Proposition 3.5. *For any positive DNF φ and variable x in φ:*

$$\begin{aligned} \#L(\varphi) &\leq \#\varphi \leq \#U(\varphi) \\ \#L(\varphi[x := 1]) - \#U(\varphi[x := 0]) &\leq Banzhaf(\varphi, x) \\ &\leq \#U(\varphi[x := 1]) - \#L(\varphi[x := 0]) \end{aligned}$$

Example 3.6. Consider the lineage $\varphi = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$ from Example 2.3. We compute lower and upper bounds for the model count $\#\varphi$ and the Banzhaf value of a_1, representing Alice in the database from Figure 1. The function φ is a disjunction of two independent functions $\varphi_1 = (a_1 \wedge p_1) \vee (a_1 \wedge p_2)$ and $\varphi_2 = (a_2 \wedge p_3)$. From Ex. 3.4, $Banzhaf(\varphi, a_1) = 9$ and $\#\varphi = 17$. The functions $\varphi[x := 0] = (0 \wedge p_1) \vee (0 \wedge p_2) \vee (a_2 \wedge p_3) \equiv a_2 \wedge p_3$ and $\varphi[x := 1] = (1 \wedge p_1) \vee (1 \wedge p_2) \vee (a_2 \wedge p_3) \equiv p_1 \vee p_2 \vee (a_2 \wedge p_3)$ are in iDNF, so $L(\varphi[a_1 := 0]) = U(\varphi[a_1 := 0]) = \varphi[a_1 := 0]$ and $L(\varphi[a_1 := 1]) = U(\varphi[a_1 := 1]) = \varphi[a_1 := 1]$. Note that $\varphi[a_1 := 0] = a_2 \wedge p_3$, yet it is defined over four variables, which is important for computing its correct model count.

We may also obtain the following iDNF functions: $L(\varphi) = (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$ by skipping the clause $(a_1 \wedge p_1)$ in φ; and $U(\varphi) = (a_1 \wedge p_1) \vee p_2 \vee (a_2 \wedge p_3)$ by removing a_1 from the second clause

BOUNDS(d-tree T_φ for function φ, 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_φ
  case literal or constant ℓ:
    $(L_b, L_\#) := (U_b, U_\#) :=$ EXABAN(ℓ, x)
  case non-trivial leaf ψ: `//no literal nor constant`
    `//Compute bounds by Prop. 3.5`
    $L_b := \#L(\psi[x := 1]) - \#U(\psi[x := 0])$
    $U_b := \#U(\psi[x := 1]) - \#L(\psi[x := 0])$
    $L_\# := \#L(\psi); \quad U_\# := \#U(\psi)$
  case T_{φ_1} op T_{φ_2}:
    $(L_b^{(i)}, L_\#^{(i)}, U_b^{(i)}, U_\#^{(i)}) :=$ BOUNDS(T_{φ_i}, x), for $i \in [2]$
    $n_i :=$ number of variables in φ_i, for $i \in [2]$
    switch op
      case $\odot$: `//wlog if` x `is in` φ`, then it is in` φ_1
        $L_b := L_b^{(1)} \cdot L_\#^{(2)}; \quad U_b := U_b^{(1)} \cdot U_\#^{(2)}$
        $L_\# := L_\#^{(1)} \cdot L_\#^{(2)}; \quad U_\# := U_\#^{(1)} \cdot U_\#^{(2)}$
      case $\otimes$: `//wlog if` x `is in` φ`, then it is in` φ_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` φ_1 `and` φ_2 `have same variables`
        $L_b := L_b^{(1)} + L_b^{(2)}; \quad U_b := U_b^{(1)} + U_b^{(2)}$
        $L_\# := L_\#^{(1)} + L_\#^{(2)}; \quad U_\# := U_\#^{(1)} + U_\#^{(2)}$
return $(L_b, L_\#, U_b, U_\#)$

Fig. 4. 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.

of φ. Using Eq. (6) and (8):

$$\#L(\varphi[a_1 := 0]) = \#U(\varphi[a_1 := 0]) = 4,$$
$$\#L(\varphi[a_1 := 1]) = \#U(\varphi[a_1 := 1]) = 13,$$
$$\#L(\varphi) = 7, \text{ and } \#U(\varphi) = 23.$$

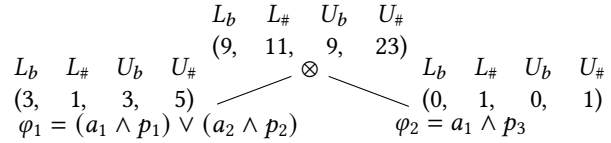
Hence, it indeed holds that $\#L(\varphi) = 7 \leq \#\varphi = 17 \leq \#U_\varphi = 23$ and $\#L(\varphi[a_1 := 1]) - \#U(\varphi[a_1 := 0]) = 9 \leq Banzhaf(\varphi, a_1) = 9 \leq \#U(\varphi[a_1 := 1]) - \#L(\varphi[a_1 := 0]) = 9$.

3.2.2 Efficient Computation of Lower and Upper Bounds for D-trees. The procedure BOUNDS in Fig. 4 computes lower and upper bounds on the Banzhaf value and model count for any d-tree, whose leaves are positive DNF functions, (possibly negated) literals, or constants. It does so in linear time in one bottom-up pass over the d-tree.

The procedure's input is 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 Prop. 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. (9), 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. We use the upper bound since the term occurs negatively.

Example 3.7. Consider the following partial d-tree representing the lineage $\varphi = (a_1 \wedge p_1) \vee (a_1 \wedge p_2) \vee (a_2 \wedge p_3)$ from Example 2.3. Each node is assigned a quadruple of bounds for the Banzhaf value of a_1 and the model count for the d-tree rooted at that node. Following the notation in the procedure BOUNDS in Fig. 4, the first and the third entries in a quadruple are the lower and respectively upper bound for the Banzhaf value; the second and the fourth entries are the lower and respectively upper bound for the model count.



The function φ is a disjunction of the two independent functions $\varphi_1 = (a_1 \wedge p_1) \vee (a_1 \wedge p_2)$ and $\varphi_2 = a_2 \wedge p_3$. Since φ_2 is already in iDNF, the quadruple labelling φ_2 consists of the exact model count 1 for φ_2 and the exact Banzhaf value 0 for a_1 in φ.

We explain the bounds in the quadruple labelling φ_1. Since $\varphi_1[a_1 := 0] = (0 \wedge p_1) \vee (0 \wedge p_2) \equiv 0$ and $\varphi_1[a_1 := 1] = p_1 \vee p_2$ are in iDNF, L_b and U_b give the exact Banzhaf value $Banzhaf(\varphi_1, a_1) = \varphi_1[a_1 := 1] - \varphi_1[a_1 := 0] = 3$. The lower and upper bound for the model count of φ are $L_\# = \#L(\varphi_1) = \#(a_1 \wedge p_1) = 1$ and respectively $U_\# = \#U(\varphi_1) = \#\big(p_1 \vee (a_1 \wedge p_2)\big) = 5$.

Eq. (6) to (11) and Prop. 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. Fig. 5 introduces our approximation algorithm ADABAN. Its input is a partial d-tree T_φ, a variable x, a relative error ϵ, and initial trivial bounds $[0, 2^{n-1}]$ on $Banzhaf(\varphi, x)$, where n is the number of variables in φ. It computes an interval of ϵ-approximations for $Banzhaf(\varphi, x)$. First, it calls the procedure BOUNDS from Fig. 4 to obtain lower and upper bounds L_b, U_b for $Banzhaf(\varphi, x)$ based on the current partial d-tree T_φ. It then updates the best lower bound L and upper bound U seen so far. If $(1 - \epsilon) \cdot U - (1 + \epsilon) \cdot L \leq 0$, then it returns the interval $[(1 - \epsilon) \cdot U, (1 + \epsilon) \cdot L]$. For any value B in this non-empty interval, we have $B \geq (1 - \epsilon) \cdot U \geq (1 - \epsilon) \cdot Banzhaf(\varphi, x)$ and $B \leq (1 + \epsilon) \cdot L \leq (1 + \epsilon) \cdot Banzhaf(\varphi, x)$, i.e., B is a relative ϵ-approximation for $Banzhaf(\varphi, x)$. If the condition does not hold, it picks a non-trivial (no literal/constant) leaf ψ, decomposes it, and checks again whether the new bounds are satisfactory. Such a leaf ψ always exists unless T_φ is complete, in which case $U = L$. The decomposition of ψ replaces ψ 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.

AdaBan(d-tree T_φ, variable x, error ϵ, bounds $[L, U]$)
outputs bounds for $Banzhaf(\varphi, x)$ satisfying relative error ϵ

$(L_b\ ,\ \cdot\ ,\ U_b\ ,\ \cdot\ ) :=$ bounds(T_φ, x) //get bounds on T_φ
$\ell := u := 0$ //initialize the bounds to return
$L := \max\{L, L_b\}; U := \min\{U, U_b\}$ //update bounds
if $(1-\epsilon)\cdot U - (1+\epsilon)\cdot L \leq 0$ //error satisfied
  $\ell := (1-\epsilon)\cdot U; u := (1+\epsilon)\cdot L$
else
  pick a non-trivial leaf ψ of T_φ //no literal/constant
  switch ψ
    case $\psi_1 \wedge \psi_2$ for independent ψ_1 and ψ_2:
      replace ψ by $\psi_1 \odot \psi_2$ in T_φ
    case $\psi_1 \vee \psi_2$ for independent ψ_1 and ψ_2:
      replace ψ by $\psi_1 \otimes \psi_2$ in T_φ
    default:
      pick a variable y in ψ
      replace ψ by $(y \odot \psi[y := 1]) \oplus (\neg y \odot \psi[y := 0])$ in T_φ
  $[\ell, u] :=$ AdaBan$(T_\varphi, x, \epsilon, [L, U])$
return $[\ell, u]$

Fig. 5. Approximating Banzhaf values with relative error ϵ using incremental decomposition and bound refinement.

Proposition 3.9. *For any positive DNF function φ, d-tree T_φ for φ, variable x in φ, error ϵ, and bounds $L \leq Banzhaf(\varphi, x) \leq U$, it holds* AdaBan$(T_\varphi, x, \epsilon, [L, U]) = [\ell, u]$ *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 Figs. 4 and 5 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:

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

4 BANZHAF-BASED RANKING AND TOP-k

When exact computation of Banzhaf values is out of reach, it is still highly valuable 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 IchiBan 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 then show a dichotomy result. #BIS is the problem of counting independent sets in a bipartite graph; it is conjectured that there is no FPTAS for #BIS [13, 18].

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 [32]. 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, and is deferred to the extended version of this paper [2].

5 ON BANZHAF VS. SHAPLEY VALUES

We have focused in this paper on Banzhaf values as a measure of facts contribution in query answering. The Shapley value is a related notion that is commonly used in previous work on query evaluation as well as in the literature on explaining predictions of Machine Learning models. We next recall this notion, compare it to Banzhaf values, and discuss which of our algorithms can be adapted to Shapley values. Section 6 further compares them.

Definition 5.1. As in Definition 2.4, let $D = (D_n, D_x)$ be a database where we distinguish between endogenous (D_n) and exogenous (D_x), let Q be a boolean query and let f be a fact in D. The Shapley value of f with respect to Q and D is defined by:

$$Shap(Q, D_n, D_x, f) = \sum_{D' \subseteq D_n \setminus \{f\}} \frac{|D'|!(|D_n| - |D'| - 1)!}{|D_n|!} \cdot \big(Q(D' \cup D_x \cup \{f\}) - Q(D' \cup D_x)\big) \quad (12)$$

The definition of Shapley value resembles that of Banzhaf value (Definition 2.4), except for the coefficients multiplying the contribution of subsets. These coefficients stem from a probabilistic interpretation, illustrated in [32]: Consider a probabilistic process in which players join the game one by one, in a random order, and where the reward of each player is based on their marginal contribution to the subset of players that have joined the game before them. The Shapley value of a player is their expected reward in this probabilistic process.

Example 5.2. Revisiting Example 2.2, recall that in the computation of Banzhaf value of a fact f we have counted the number of sets for which f is critical. For Shapley values, we need to distinguish between the sets based on their sizes, so that we can weigh their contribution accordingly. For instance, a_1 is critical for the set $\{a_2, p_1\}$. This set is of size 2, so it contributes $\frac{2!(|D_n|-2-1)!}{D_n!}$ to the Shapley value of a_1. For the variant of the example including the topmost two tuples in the Author relation, we have $|D_n| = 5$ (the total number of endogenous facts), and so the set $\{a_2, p_1\}$ contributes $\frac{2!\cdot 2!}{5!} = 0.0333$ to the Shapley value of a_1. We repeat for all critical sets and sum up the obtained values, yielding a total Shapley value of 0.3666 for a_1. Similarly, for a_2 we obtain a Shapley value of 0.2.

There is no definitive answer to whether the Shapley or the Banzhaf value are the better fit for the task of quantifying contribution. In domains outside query answering, both have been extensively studied. The main property satisfied by Shapley and not by Banzhaf is that the Shapley values of all players add up to the game value, while the Banzhaf values do not. This property is crucial when the measure of contribution is used to distribute a reward corresponding to the game value amongst members of the coalition, which is the original motivation behind Shapley values. It is not necessarily a desideratum for importance attribution [44]; if needed, normalization methods can be applied to Banzhaf [25]. The main advantage of Banzhaf over Shapley values is that the lack of the combinatorial coefficients make it simpler.

We next revisit the computational problems we have studied for Banzhaf and briefly explain which of the algorithms may be adapted to Shapley.

Exact Computation. For exact computation, our ExaBan algorithm can be easily adapted to an exact algorithm for Shapley values computation, which we refer to as ExaShap. Given a complete d-tree for φ, ExaShap differs from ExaBan in that it computes, for each d-tree node, the number of critical sets *of each size* (rather than just their total number), by adapting Equations (6) - (11) . Namely, instead of computing the overall number of satisfying assignments $\#\varphi$ we compute the number of satisfying assignments $\#_k\varphi$ for every size k. Then, for instance, Equation (6) is replaced

by

$$\#_k\varphi = \sum_{i=0}^{i=k} \#_i\varphi_1 \cdot \#_{k-i}\varphi_2 \tag{13}$$

We follow similarly for the other equations (see [2]). The computed quantities are then propagated up the d-tree, multiplied by the corresponding combinatorial coefficients and summed up.

Approximate Computation. Adapting AdaBan to Shapley values is more challenging, as explained next. Recall that AdaBan calculates lower and upper bounds for multiple (usually many) incomplete d-trees. Bound calculations over incomplete d-trees tend to be significantly more expensive for Shapley than for Banzhaf because of the relative difficulty to calculate lower and upper bounds for functions at leaves. Furthermore, we need to compute and propagate these bounds for each critical set size. This overhead leads to ineffectiveness of this approach for Shapley values, calling for further study of approximation of Shapley values.

Ranking and top-k. Our ranking algorithm relies on our approximation method AdaBan, and so the difficulties in adapting it to Shapley values carry over the ranking. Due to their similarity, it is natural to ask whether quantifying the contribution based on Shapley or Banzhaf values makes a difference in terms of ranking. We show a somewhat disparity between theory and practice in this respect: theoretically, the ranking may be different, and this holds even for simple queries such as $Q : -R(x), S(x), T(x)$ and for a particular choice of database [2]. However, in our experiments in Section 6.6, the two rankings are often the same.

6 EXPERIMENTS

This section details our experimental setup and results.

6.1 Experimental Setup and Benchmarks

We 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.

Algorithms. We benchmarked our algorithms ExaBan, ExaShap, AdaBan, and IchiBan against the following three competitors: Sig22, for exact computation using an off-the-shelf knowledge compilation package [16]; MC, a Monte Carlo-based randomized approximation [30]; and CNF-Proxy, an heuristic for ranking facts based on their contribution [16]. These competitors were originally developed for Shapley value. We adapted them to compute Banzhaf values (see Sec. 7). This adaptation improved the execution time of the competitors (since the Banzhaf formula is simpler to compute than Shapley), but the improvement was marginal compared to their overall execution time (up to 0.1%). We only show for the baselines the (faster) execution times observed for their Banzhaf variant. 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$\mathbf{X}$ to denote the execution of an algorithm Algo with parameter value $\mathbf{X}$.

Datasets. We tested the algorithms using 301 queries evaluated over three datasets: Academic, IMDB and TPC-H (SF1). The workload (see Table 1) is based on previous work on Shapley values for query answering [4, 16]: as in [16], 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 [4] (Academic was not used in [16]). The lineage for all output tuples of these queries was constructed using ProvSQL [48]. All algorithms take as input the same lineage. The computation time for the lineage is therefore the same for all algorithms and as reported in [16]. We only report next the execution times of the algorithms for Banzhaf-related computation given

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

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

the lineage. 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.

Measurements. We measure 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 report failure in case an algorithm did not terminate an instance within one hour. We also report 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.

6.2 Summary of Experimental Findings

Our experimental findings lead to the following main conclusions:

(1) *ExaBan consistently outperforms Sig22 for exact computation.* Sec. 6.3 shows that ExaBan significantly outperforms Sig22 and also succeeds in many cases where Sig22 times out (41.7%-99.2% of these cases for the different datasets). ExaShap generally outperforms Sig22 as well, but to a lesser extent than ExaBan.

(2) *AdaBan outperforms ExaBan already for small relative errors.* Sec. 6.4 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. 6.4 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. 6.5 shows that IchiBan quickly and accurately separates the approximation intervals of the first k Banzhaf values (demonstrated for k up to 10) from the remaining values, and it is significantly more accurate than previous approaches based on MC or CNF Proxy.

6.3 Exact Banzhaf computation

We first compare ExaBan, ExaShap and Sig22.

Success Rate. Table 2 gives the success rate of the exact algorithms for each dataset. ExaBan succeeded for far more queries and lineages than Sig22. For Academic and IMDB, both algorithms succeeded for the majority of instances; a breakdown analysis shows that whenever Sig22 failed for a query, it failed for all lineages (output tuples) of this query. ExaBan succeeds for 15% and 17% more queries for Academic and IMDB respectively. For TPC-H, the query success rate is significantly lower for both algorithms. Still, ExaBan failed for only 9% of the queries (Sig22 failed for 14%). ExaShap succeeds in less cases than ExaBan, due to the additional operations needed for Shapley value computation, yet it still outperforms Sig22, indicating that our approach is also competitive for exact Shapley value computation.

Runtime Performance. We first analyze the instances for which both algorithms succeed (there are no instances for which Sig22 succeeds and ExaBan fails). Table 3 shows that ExaBan clearly

Dataset	Algorithm	Query Success Rate	Lineage Success Rate
Academic	ExaBan	98.91%	99.99%
	ExaShap	97.82%	98.84%
	Sig22	83.91%	98.40%
	AdaBan0.1	98.91%	99.99%
	MC50#vars	96.74%	98.83%
IMDB	ExaBan	82.23%	99.63%
	ExaShap	65.48%	99.53%
	Sig22	65.48%	98.35%
	AdaBan0.1	88.32%	99.81%
	MC50#vars	83.76%	99.74%
TPC-H	ExaBan	58.33%	91.52%
	ExaShap	50.00%	85.46%
	Sig22	50.00%	85.46%
	AdaBan0.1	75.00%	92.73%
	MC50#vars	50.00%	85.46%

Table 2. Query success rate: Percentage of queries for which the algorithms finish 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 finish within one hour.

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
	ExaShap	0.023	0.001	0.003	0.029	0.219	0.346	0.756
	Sig22	0.290	0.124	0.134	0.303	0.537	2.433	81.54
IMDB	ExaBan	0.149	0.002	0.003	0.014	0.046	0.631	1793
	ExaShap	0.727	0.003	0.013	0.090	0.470	6.414	>3600
	Sig22	2.840	0.146	0.365	1.710	5.909	54.63	2271
TPC-H	ExaBan	0.003	0.003	0.003	0.003	0.004	0.011	0.063
	ExaShap	0.007	0.003	0.003	0.005	0.009	0.015	0.558
	Sig22	1.217	0.080	0.140	0.200	0.260	1.450	157.3

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

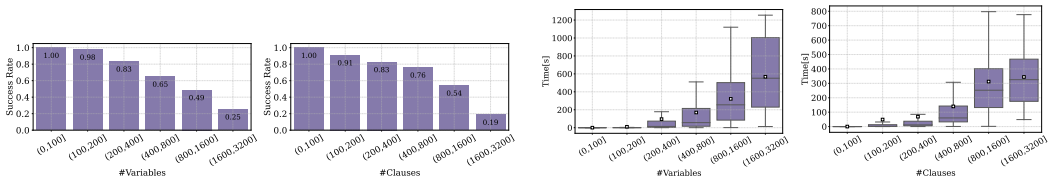
outperforms Sig22: For instances that are hard for Sig22, ExaBan achieves a speedup of up to 166x (229x) for TPC-H (Academic). For IMDB, ExaBan's speedup over Sig22 is already visible for simple instances, with a speedup of 128x for the 95-th percentiles. ExaShap generally performs somewhat worse than ExaBan yet generally still better than Sig22. ExaBan and ExaShap have a few performance outliers for IMDB.

Runtime Performance of ExaBan when Sig22 fails. Sig22 fails 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 achieves near-perfect success and finishes in less than ten minutes for all these instances. For IMDB, ExaBan succeeds in 77.4% of these instances. For 95% of these success cases, ExaBan finishes in under ten minutes. For TPC-H, ExaBan succeeds in 41.7% of these instances; whenever it succeeds, its computation time is just over one minute. To summarize, ExaBan is generally faster and more robust than Sig22. One reason is that, in contrast to ExaBan, Sig22 requires to turn the lineage into a CNF representation, which may increase its size and complexity. This advantage is also shared by ExaShap, which also succeeds in many cases when Sig22 fails, notably for IMDB (albeit to a lesser extent than ExaBan).

The effect of lineage size and structure. Figure 6 shows the performance of ExaBan grouped by the number of variables or clauses. ExaBan achieves near-perfect success rates and terminates in

Dataset	algorithm	Success rate	Execution times [sec]						
			Mean	p50	p75	p90	p95	p99	Max
Academic	ExaBan	99.2%	128.9	168.4	172.0	174.4	175.0	189.0	563.5
	ExaShap	27.7%	93.64	3.375	15.14	64.51	711.6	1187	1397
IMDB	ExaBan	77.4%	111.9	24.10	95.95	348.8	597.1	1055	1381
	ExaShap	71.3%	225.1	91.94	272.5	640.5	931.4	1392	3574
TPC-H	ExaBan	41.7%	53.77	56.44	60.24	63.27	66.23	68.59	69.18
	ExaShap	0%	-	-	-	-	-	-	-

Table 4. ExaBan and ExaShap's runtime performance for instances on which Sig22 fails.



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

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

Fig. 6. Success rate and execution time of ExaBan across all databases/queries, grouped by the number of variables (clauses) in the lineage. An interval $[i, j]$ represents the set of lineages with #vars (# clauses) between i and j.

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 5. Approximate versus exact Banzhaf computation for instances on which ExaBan succeeds.

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

6.4 Approximate Banzhaf Computation

We next examine the performance of AdaBan0.1 (i.e., AdaBan 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 succeeds at least for all instances for which the latter also succeeds. 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 observe 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).

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

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

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 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.

Runtime Performance. Table 5 focuses on instances on which ExaBan (and also AdaBan0.1) succeeds. AdaBan0.1 consistently outperforms both ExaBan and MC50#vars. The average runtime gains over ExaBan range from a factor of 3 for Academic to 20 for TPC-H. MC50#vars is slower than ExaBan for over 99% of the instances, and even fails for some instances for which ExaBan succeeds. Running MC with a larger number of samples to improve its accuracy (see below) will naturally take even more time.

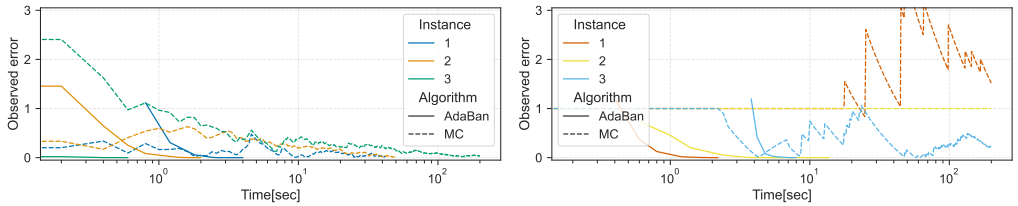
Table 6 shows that, when only considering the instances on which ExaBan fails, AdaBan0.1 succeeds in nearly 50% (15%) of these instances for IMDB (TPC-H). Both ExaBan and AdaBan0.1 fail for just one instance in Academic (not shown).

Approximation Quality. AdaBan0.1 guarantees a deterministic relative error of 0.1. MC50#vars only guarantees a probabilistic absolute error, where the number of required samples depends quadratically on the inverse of the error. Table 7 compares the observed approximation quality of 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. The results are shown for all instances for which ExaBan succeeds, and separately for the "hard" instances for which ExaBan took at least five seconds. AdaBan0.1's approximation is consistently closer to the ground truth than MC50#vars's approximation by several orders of magnitude.

Approximation Error as a Function of Time. Figure 7 presents, for several instances, the evolution of the observed error for AdaBan and MC over time. These instances appear in [3] and were selected, for illustration, from the set of "hard" lineages for which ExaBan needs at least 200 seconds to compute the Banzhaf values of all variables (then, variables appearing in these lineages were selected at random). The error of AdaBan decreases 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 for some instances it may not even converge within 200 seconds.

6.5 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



(a) 3 instances for which MC converged to the Banzhaf value.

(b) 3 instances for which MC did not converge to the Banzhaf value.

Fig. 7. Convergence rate of approximate Banzhaf value $\hat{v}$ to the exact Banzhaf value v as a function of time, for representative instances. The observed error (y-axis) is calculated as $\frac{|v-\hat{v}|}{v}$. AdaBan is stopped as soon as it reaches the exact Banzhaf value.

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

Table 8. Observed precision@10 / precision@5 for instances for which ExaBan succeeds.

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; in many of the considered 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 top-3 and top-5 computation took longer than ExaBan. We attribute this to a large number of ties in the Banzhaf values of facts for the TPC-H workload, whose lineages are more symmetric in the variables. IchiBan0.1 is a good alternative for such instances.

6.6 Banzhaf vs Shapley based ranking

As explained in Section 5, there are queries and databases for which Banzhaf-based and Shapley-based rankings differ. We next study their similarity in practice. Table 9 shows an empirical comparison of the rankings for the Academic dataset (results for other datasets show similar trends and are deferred to [2]), for cases where we have the exact values as ground truth (i.e. cases where Sig22 succeeds). The measures used for comparison are precision@k for $k = 3, 5, 10$, Kendall's τ_b [47] and Spearman's correlation [46]. We report statistics for these measures aggregated over all instances. For 90% of all instances, Banzhaf-based and Shapley-based rankings completely agree,

and for 95-99% of the instances they are very similar. A few cases exhibit significantly different rankings.

Measure	Mean	p75	p90	p95	p99	Worst
Kendall τ	0.995	1	1	0.989	0.881	0.2
Spearman ρ	0.997	1	1	0.999	0.949	0.239
Precision@3	0.999	1	1	1	1	0.666
Precision@5	0.997	1	1	1	0.8	0.6
Precision@10	0.998	1	1	1	0.9	0.6

Table 9. Comparison of Banzhaf and Shapley rankings. Percentiles are from most similar to least similar rankings. Values closer to 1 indicate higher similarity of the two rankings.

7 RELATED WORK

We compare our work to multiple lines of related work.

Explaining Query Results. Many approaches for explanations in databases provide a detailed record of the computation that took place using various models of lineage or provenance (e.g., [9, 12, 22]). Another approach (e.g. [17, 32, 36, 45]) is to quantify the contribution of individual tuples and/or subsets thereof. These approaches are complementary: the lineage may be detailed but is often too verbose and convoluted to be directly presented as explanation. Our work falls into the second category, together with multiple lines of previous work that are based, for example, on: Shapley values [32], causality, responsibility, and counterfactuals [34–36, 45], or methods for credit distribution [17]. While there is no silver bullet for choosing a function that quantifies contribution, the Shapley and Banzhaf values are of interest as they are grounded in Game Theory, where they are extensively investigated (e.g., [20, 29, 50, 53]). Further approaches for explanations include those based on provenance summaries [28] and explanations of outliers [38].

Shapley value. Recent works [8, 15, 16, 26, 30–32, 43] investigated the use of the Shapley value [20, 29, 49, 50, 53] for explanations in query answering, with particular focus on algorithms and the complexity of computing exact and approximate Shapley values for facts. Our novel algorithm for *exact computation* performs better than the state-of-the-art, in both the Banzhaf and Shapley variants. Our novel *anytime approximation and ranking algorithms* for Banzhaf values have no counterparts with *provable guarantees* of anytime approximation and/or ranking with provable relative approximation for Shapley/Banzhaf values. In contrast, previous work [32] has proposed a polynomial time randomized absolute approximation scheme for Banzhaf (and Shapley) values based on Monte Carlo sampling. *Sec. 6 shows experimentally that AdaBan significantly outperforms this randomized approach. Randomized approximations based on Monte Carlo sampling have three important limitations*: (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; (3) Monte Carlo views the input functions as black boxes and does not exploit their structure. *These limitations are not shared by our deterministic approximation AdaBan.* Our algorithm outperforms the ranking method proposed in previous work, namely the CNF Proxy heuristic [16]; the latter has two further disadvantages: (a) no theoretical guarantees; (b) the proxy values only reflect the relative order of contribution and not an approximation of its magnitude. Furthermore, our dichotomy result for ranking has no counterparts for neither Shapley nor Banzhaf, and our comparative study of Banzhaf and Shapley values for query answering is also novel.

Hardness of exact Banzhaf computation. Prior work shows, via reduction from query evaluation in probabilistic databases, that for non-hierarchical self-join free CQs, computing *exact* Banzhaf values of facts is $\text{FP}^{\#\text{P}}$-hard [32]. *We are the first to show hardness of Banzhaf-based* ranking. *For*

that, we needed a completely different technique, relying on the conjecture that there is no PTIME approximation for counting independent sets in a bipartite graph [13, 18].

The SHAP score. SHAP (SHapley Additive exPlanations) is used for feature importance in machine learning (ML) [33]. Unlike Shapley/Banzhaf values, it models missing "players" (feature values in ML) according to their expected values. Prior work [5, 6] showed that under common complexity assumptions, there is no PTIME algorithm for ranking based on SHAP scores. This result uses a different technique from our work, and due to the differences between SHAP and Banzhaf values, their results could not be used here.

Probabilistic Databases. In probabilistic databases, the goal is to compute/approximate/rank the probabilities of query answers. The work of [21, 40, 41] has developed approximation and ranking algorithms for probabilistic databases that are based on incremental compilation of the query lineage into partial d-trees. *Our work differs from this prior work as it is tailored at Banzhaf value computation and Banzhaf-based ranking as opposed to probability computation.* In particular, in probabilistic databases one needs to compute one probability value for a given lineage. For Banzhaf values, we need to compute such value (or range) *for every variable* in the lineage. Furthermore, the computation of Banzhaf values is of course different from the computation of probabilities, and we consequently had to derive novel formulas and optimizations to compute/bound Banzhaf values. We also showed an adaptation of our solution for exact computation of Shapley values, yet achieving a practically effective anytime approximation algorithm for Shapley values remains a challenge for future work.

8 CONCLUSION

We introduced algorithms for the exact and anytime deterministic approximate computation of the Banzhaf values that quantify the contribution of database facts to the answers of select-project-join-union queries. We also showed the use of these algorithms for Banzhaf-based ranking and gave a dichotomy in the complexity of ranking. Our exact algorithm may further be adapted to compute Shapley values. Our algorithms outperform prior work in both runtime and accuracy for a wide range of problem instances.

There are several exciting directions for future work. First, we plan to extend our algorithmic framework to more expressive queries that also have aggregates and negation. We would also like to generalize our algorithms to further fact attribution measures, such as the SHAP score and causality-based measures.

ACKNOWLEDGMENTS

Ahmet Kara and Dan Olteanu would like to acknowledge the UZH Global Strategy and Partnerships Funding Scheme. The research of Daniel Deutch and Omer Abramovich has been partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 804302).

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Vol. 8. Addison-Wesley Reading. http://webdam.inria.fr/Alice/

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

[3] Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. 2023. GitHub Repository. https://github.com/Omer-Abramovich/AdaBan.

[4] Dana Arad, Daniel Deutch, and Nave Frost. 2022. LearnShapley: Learning to Predict Rankings of Facts Contribution Based on Query Logs. In *Proc. of CIKM*. 4788–4792.

[5] Marcelo Arenas, Pablo Barceló, Leopoldo Bertossi, and Mikaël Monet. 2021. The Tractability of SHAP-Score-Based Explanations over Deterministic and Decomposable Boolean Circuits. In *Proc. of AAAI*. https://arxiv.org/abs/2007.14045

[6] Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. 2023. On the Complexity of SHAP-Score-Based Explanations: Tractability via Knowledge Compilation and Non-Approximability Results. *J. Mach. Learn. Res.* 24 (2023), 63:1–63:58. http://jmlr.org/papers/v24/21-0389.html

[7] J.F. Banzhaf. 1965. Weighted Voting Doesn't Work: A Mathematical Analysis. *Rutgers Law Review* 19, 2 (1965), 317–343.

[8] Leopoldo Bertossi, Benny Kimelfeld, Ester Livshits, and Mikaël Monet. 2023. The Shapley Value in Database Management. *SIGMOD Rec.* 52, 2 (2023).

[9] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and Where: A Characterization of Data Provenance. In *Proc. of ICDT*. Springer, 316–330.

[10] Adriane Chapman and HV Jagadish. 2009. Why Not?. In *Proc. of SIGMOD*. 523–534.

[11] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474. https://doi.org/10.1561/1900000006

[12] Yingwei Cui, Jennifer Widom, and Janet L Wiener. 2000. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Trans. Datab. Syst.* 25, 2 (2000), 179–227. http://ilpubs.stanford.edu:8090/252/1/1997-3.pdf

[13] Radu Curticapean, Holger Dell, Fedor V. Fomin, Leslie Ann Goldberg, and John Lapinskas. 2019. A Fixed-Parameter Perspective on #BIS. *Algorithmica* 81, 10 (2019), 3844–3864.

[14] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *JAIR* 17 (2002), 229–264. https://arxiv.org/abs/1106.1819

[15] Susan B. Davidson, Daniel Deutch, Nave Frost, Benny Kimelfeld, Omer Koren, and Mikaël Monet. 2022. ShapGraph: An Holistic View of Explanations through Provenance Graphs and Shapley Values. In *Proc. of SIGMOD*. 2373–2376. https://doi.org/10.1145/3514221.3520172

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

[17] Dennis Dosso, Susan B. Davidson, and Gianmaria Silvello. 2022. Credit Distribution in Relational Scientific Databases. *Inf. Syst.* 109 (2022), Article 102060.

[18] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. 2004. The Relative Complexity of Approximate Counting Problems. *Algorithmica* 38, 3 (2004), 471–500.

[19] Algaba E, Prieto A, and Saavedra-Nieves A. 2023. Risk Analysis Sampling Methods in Terrorist Networks based on the Banzhaf Value. *Risk Analysis* PMID: 37210375 (May 2023). https://doi.org/10.1111/risa.14156

[20] Vincent Feltkamp. 1995. Alternative Axiomatic Characterizations of the Shapley and Banzhaf Values. *Int. J. Game Theory* 24 (1995), 179–186.

[21] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime Approximation in Probabilistic Databases. *VLDB J.* 22, 6 (2013), 823–848. https://doi.org/10.1007/s00778-013-0310-5

[22] Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *Proc. of PODS*. 31–40. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1022&context=db_research

[23] Todd J Green and Val Tannen. 2017. The Semiring Framework for Database Provenance. In *Proc. of PODS*. 93–99.

[24] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A Survey on Provenance: What for? What form? What from? *VLDB J.* 26, 6 (2017), 881–906. https://doi.org/10.1007/s00778-017-0486-1

[25] 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 *Proc. of UAI (PMLR, Vol. 180)*. PMLR, 969–979. https://proceedings.mlr.press/v180/karczmarz22a.html

[26] Majd Khalil and Benny Kimelfeld. 2023. The Complexity of the Shapley Value for Regular Path Queries. In *Proc. of ICDT (LIPIcs, Vol. 255)*. 11:1–11:19. https://doi.org/10.4230/LIPIcs.ICDT.2023.11

[27] Werner Kirsch and Jessica Langner. 2010. Power Indices and Minimal Winning Coalitions. *Soc. Choice Welf.* 34, 1 (2010), 33–46. https://doi.org/10.1007/s00355-009-0387-3

[28] Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2020. Approximate Summaries for Why and Why-not Provenance. *Proc. of VLDB Endow. (PVLDB)* 13, 6 (2020), 912–924. https://doi.org/10.14778/3380750.3380760

[29] Ehud Lehrer. 1988. An Axiomatization of the Banzhaf Value. *Int. J. Game Theory* 17 (1988), 89–99.
[30] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2020. The Shapley Value of Tuples in Query Answering. In *Proc. of ICDT (LIPIcs, Vol. 155)*. 20:1–20:19. https://arxiv.org/abs/1904.08679
[31] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. Query Games in Databases. *SIGMOD Rec.* 50, 1 (2021), 78–85. https://doi.org/10.1145/3471485.3471504
[32] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. The Shapley Value of Tuples in Query Answering. *Log. Methods Comput. Sci.* 17, 3 (2021). https://doi.org/10.46298/lmcs-17(3:22)2021
[33] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Proc. of NeurIPS*. 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf
[34] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *Proc. of VLDB Endow. (PVLDB)* 4, 1 (2010), 34–45.
[35] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. 2011. Bringing Provenance to Its Full Potential Using Causal Reasoning. In *TaPP*, Peter Buneman and Juliana Freire (Eds.). USENIX Association.
[36] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and Explanations in Databases. *Proc. of VLDB Endow. (PVLDB)* 7, 13 (2014), 1715–1716. http://www.vldb.org/pvldb/vol7/p1715-meliou.pdf
[37] Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2019. Going Beyond Provenance: Explaining Query Answers with Pattern-based Counterbalances. In *Proc. of SIGMOD*. ACM, 485–502. https://doi.org/10.1145/3299869.3300066
[38] Zhengjie Miao, Qitian Zeng, Chenjie Li, Boris Glavic, Oliver Kennedy, and Sudeepa Roy. 2019. CAPE: Explaining Outliers by Counterbalancing. *Proc. VLDB Endow.* 12, 12 (2019), 1806–1809. https://doi.org/10.14778/3352063.3352071
[39] Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In *Proc. of SUM*. Springer, 326–340.
[40] Dan Olteanu, Jiewen Huang, and Christoph Koch. 2010. Approximate Confidence Computation in Probabilistic Databases. In *Proc. of ICDE*. 145–156.
[41] Dan Olteanu and Hongkai Wen. 2012. Ranking Query Answers in Probabilistic Databases: Complexity and Efficient Algorithms. In *Proc. of ICDE*. 282–293. https://doi.org/10.1109/ICDE.2012.61
[42] L. S. Penrose. 1946. The Elementary Statistics of Majority Voting. *Journal of the Royal Statistical Society* 109, 1 (1946), 53–57. http://www.jstor.org/stable/2981392
[43] Alon Reshef, Benny Kimelfeld, and Ester Livshits. 2020. The Impact of Negation on the Complexity of the Shapley Value in Conjunctive Queries. In *Proc. of PODS*. 285–297. https://doi.org/10.1145/3375395.3387664
[44] Mustapha Ridaoui, Michel Grabisch, and Christophe Labreuche. 2018. An Axiomatisation of the Banzhaf Value and Interaction Index for Multichoice Games. In *MDAI*. https://shs.hal.science/halshs-02381119
[45] Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. 2016. Quantifying Causal Effects on Query Answering in Databases. In *Proc. of TaPP*. USENIX Association. http://web.cs.ucla.edu/~guyvdb/papers/SalimiTaPP16.pdf
[46] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & analgesia* 126, 5 (2018), 1763–1768.
[47] Pranab Kumar Sen. 1968. Estimates of the Regression Coefficient based on Kendall's tau. *Journal of the American statistical association* 63, 324 (1968), 1379–1389.
[48] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. Provsql: Provenance and Probability Management in PostgreSQL. *Proc. of VLDB Endow. (PVLDB)* 11, 12 (2018), 2034–2037. https://hal.inria.fr/hal-01851538/file/p976-senellart.pdf
[49] Lloyd S Shapley. 1953. A Value for n-Person Games. *Contributions to the Theory of Games* 2, 28 (1953), 307–317. http://www.library.fa.ru/files/Roth2.pdf#page=39
[50] Philip D Strafiin Jr. 1988. The Shapley—Shubik and Banzhaf Power Indices as Probabilities. *The Shapley value: essays in honor of Lloyd S. Shapley* (1988), 71.
[51] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool. https://www.morganclaypool.com/doi/abs/10.2200/S00362ED1V01Y201105DTM016
[52] Jianyuan Sun, Guoqiang Zhong, Kaizhu Huang, and Junyu Dong. 2018. Banzhaf Random Forests: Cooperative Game Theory Based Random Forests with Consistency. *Neural Networks* 106 (2018), 20–29.
[53] Rene Van den Brink and Gerard Van der Laan. 1998. Axiomatizations of the Normalized Banzhaf Value and the Shapley Value. *Social Choice and Welfare* 15 (1998), 567–582.
[54] Diego Varela and Javier Prado-Dominguez. 2012. Negotiating the Lisbon Treaty: Redistribution, Efficiency and Power Indices. *Czech Economic Review* 6, 2 (July 2012), 107–124.
[55] Jiachen T. Wang and Ruoxi Jia. 2023. Data Banzhaf: A Robust Data Valuation Framework for Machine Learning. In *Proc. of AISTATS*. 6388–6421.

Received October 2023; revised January 2024; accepted February 2024