Computing the Shapley Value of Facts in Query Answering
Literature Review

COMPUTER SCIENCE

CSP 520 – Data Integration, Warehousing and Provenance

Group - 15

Rohith Reddy Bairi - A20526972

BrijeshKumarYadav Bandi - A20528371

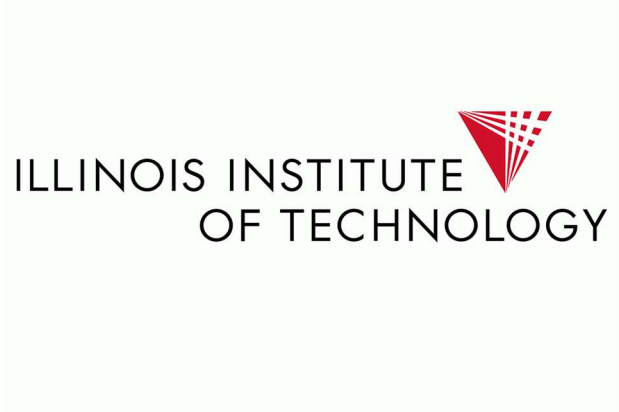
Shirisha Vaddegoni - A20553499

Deekshita Tummala – A20551936

Under The Guidance of

Prof. Boris Glavic

Date: 11/30/2023



Chicago, Illinois, United States of America

Abstract

In exploring solutions for fair value distribution in data-intensive computation, the Shapley value emerges as a pivotal concept. Despite its theoretical prominence, practical application in query evaluation has remained a challenge (Author et al., Year). This gap is addressed by Author et al., who propose novel computational strategies to derive Shapley values efficiently. They introduce a tight theoretical evaluation for probabilistic databases, culminating in a polynomial-time algorithm suitable for a subset of queries. Their approach utilizes Boolean expressions to represent database query answers, which are then transformed via Knowledge Compilation into a format conducive for Shapley value computation. Furthermore, they present a heuristic method based on Conjunctive Normal Form, broadening the computational toolkit available for such complex evaluations. Empirical tests on TPC-H and IMDB datasets validate their methodologies, marking a significant advancement in practical database query optimization. This research not only contributes a new perspective to the ongoing discourse on value distribution in data analytics but also offers tangible solutions with proven efficacy, thus representing a substantial stride in operational database management.

Contents

1. Introduction

In the realm of database systems and machine learning, the elucidation of query answers is not merely a technical concern but a fundamental aspect of interpretability and accountability. With the proliferation of data-intensive applications, the impetus to demystify the inner workings of query results has never been more pronounced. The initiative to provide explanations has burgeoned, with scholarly endeavors converging on the derivation of such explanations from the factual bedrock utilized in procuring the query answers, commonly denoted as provenance or lineage. This focus on provenance is more than an academic exercise; it is a practical necessity for transparency in decision-making processes that affect everything from logistical optimizations to personalized recommendations.

To illustrate, let us consider a seemingly straightforward query: is there a route from the USA to France requiring no more than one connection, derived from a comprehensive database of airports and flights? While the query's response may be binary—affirmative or negative—the explanatory groundwork is far from binary. It encompasses all pertinent routes factoring into the answer's derivation, a dataset potentially as vast as the databases themselves. This deluge of information presents its own challenges, not the least of which is the varying significance of individual facts. Some flights are indispensable in bridging the transatlantic divide, while others are redundant, replaceable links in the aerial network.

In addressing these complexities, the academic community has put forth several principled methodologies to quantify the input facts' contribution to query answers. Among these, the approach of applying Shapley values—a game-theoretic construct for equitably apportioning the fruits of collective endeavors among contributors—stands out. The Shapley value, with its robust theoretical underpinnings, has permeated diverse fields such as economics, environmental science, and network analysis, and has lately made inroads into data-centric paradigms including knowledge representation and machine learning. In the database context, Shapley values intuitively represent the contribution of a fact to the presence of a result tuple, given a query and a database.

Yet, the journey from theoretical formulation to computational feasibility is fraught with hurdles. The seminal work of Livshits et al. embarked on demystifying the computational complexity associated with Shapley values in query answering. They illuminated primarily the lower bounds of this complexity, offering a tractable algorithm only for a narrow subclass of queries. While approximation through Monte Carlo sampling provided a glimpse of tractability, this approach's impracticality is evident in its requirement for numerous query executions over subsets of the

database, rendering the theoretical findings of Livshits et al. insufficient for practical adoption of Shapley values as a tool for responsibility measurement in query answering.

The landscape of computational complexity for Shapley values is further nuanced when considering the SHAP-score, employed in machine learning to elucidate model predictions. Though both SHAP-score and Shapley values for databases are rooted in the same game-theoretic principles, they diverge significantly in their application—SHAP-score concerns model features and conditional expectations, while database Shapley values concern tuples and query values. Nevertheless, the techniques developed for the SHAP-score have opened doors to adapting these principles to database contexts, a task we undertake with vigor in this report.

Our investigation commences with an exploration of probabilistic databases as a gateway to Shapley computation. We posit that algorithms designed for probabilistic query answering can be extended to compute Shapley values effectively, a postulate not limited to simple query structures but applicable to a broad spectrum of database queries. This theoretical bridge is not merely academic; it has tangible implications for utilizing probabilistic query engines to calculate Shapley values.

We then pivot to the practical domain, advocating for Knowledge Compilation as an expedient approach for probabilistic databases. Knowledge Compilation entails transforming the Boolean provenance of an output tuple into a circuit form amenable to probability calculation. We refine this approach, bypassing the probabilistic intermediary, to directly compute Shapley values from these circuits—ushering in a pragmatic algorithm with polynomial-time execution guarantees.

Yet, not all scenarios lend themselves to exact computations. In instances where precision can be sacrificed for speed, we introduce a novel heuristic—CNF Proxy. This heuristic ranks input facts by their contribution without the computational overhead of exact Shapley value calculations. Our empirical investigations across standard benchmarks like TPC-H and IMDB corroborate the effectiveness of this heuristic.

The contributions of this report are thus twofold: we establish a foundational result in the computational complexity of Shapley values over relational queries, and we furnish a suite of algorithms—from exact computations to practical heuristics—that promise to elevate the practicality of Shapley value computations in the database arena.

The Shapley Value of Facts in Relational Databases

The Shapley value, a concept borrowed from cooperative game theory, finds a novel application in the realm of relational databases, serving as a measure to attribute the significance or contribution of individual facts to the outcome of a query. The foundation of this application is built on the structure and operations of relational databases, which are comprised of facts and the queries that operate on them.

A relational database is a collection of facts, which are essentially true statements about relationships among data items. Each fact is associated with a relation (also known as a table) and is expressed in the form of a tuple. For instance, if we consider Σ to be the signature of the database, which includes all relation names like R1, R2, ..., Rn with their respective arities (the number of elements in the tuples), and Const as a set of constants, a fact over (Σ, Const) is an instantiation of a relation with constants as elements, such as R(a1, ..., an).

In the context of query languages, Boolean queries are of particular interest. A Boolean query is a type of query that returns a binary outcome, true or false, in response to a question about the database. This binary nature simplifies the output to either 0 (false) or 1 (true), denoting the absence or presence of a tuple in the query result, respectively.

When it comes to attributing contributions of facts in a database, we partition the database into exogenous and endogenous facts. Exogenous facts are considered fixed and given; they are not the subject of contribution analysis. In contrast, endogenous facts are the variables under investigation, the elements to which we want to attribute value.

The Shapley value of a fact is a measure of its contribution to the result of a query. Mathematically, it is defined for an endogenous fact f in a database D for a Boolean query q as:

$$\text{Shapley}(q, D_n, D_x, f) \stackrel{\text{def}}{=} \sum_{E \subseteq D_n \setminus \{f\}} \frac{|E|!(|D_n| - |E| - 1)!}{|D_n|!} \big(q(D_x \cup E \cup \{f\}) - q(D_x \cup E)\big). \quad (1)$$

where:

D_n is the set of endogenous facts,

D_x is the set of exogenous facts,

E is a subset of D_n excluding the fact f,

|E| denotes the cardinality of the set E,

|D_n| is the total number of endogenous facts,

! denotes the factorial of a number,

value (q, D_x ∪ E ∪ {f}) is the value of the query when the fact f is included,

value (q, D_x ∪ E) is the value of the query when the fact f is not included.

The Shapley value thus calculated represents the average marginal contribution of the fact f across all possible combinations of facts. In essence, it quantifies the change in the query's outcome caused by the inclusion of the fact f.

To illustrate with an example, consider a database with a relation Flights detailing direct flights between cities. Suppose we have a Boolean query q asking whether there is a direct flight from New York (NY) to Paris (Paris). The facts in our database are flight connections, and we want to determine the Shapley value of the fact Flights (NY, Paris). If we have additional facts such as Flights (NY, London) and Flights (London, Paris), we can compute the Shapley value of Flights (NY, Paris) to understand its importance in connecting NY to Paris directly, as opposed to connecting through London.

By using the Shapley value in this context, we can discern the relative importance of different flights in the network, which is crucial for decision-making in route optimization, pricing strategies, and network expansion.

Incorporating Shapley values into database analysis allows us to transcend traditional provenance tracking, which merely outlines the lineage of a query's outcome, by quantifying the impact of individual data elements, thereby offering a nuanced view of data significance within the complex interplay of relational queries.

Reduction to Probabilistic Databases and Complexity of Computing Shapley Values

The computation of Shapley values in database systems intersects intriguingly with the complexity theory, specifically in terms of its reduction to the framework of probabilistic databases. This section delves into the intricate relationship between the complexity of computing Shapley values and probabilistic query evaluation (PQE), establishing a vital link that extends the tractability of Shapley values to a broader class of Boolean queries.

At the outset, it's important to define what we mean by Shapley values in this context. For a given Boolean query q over a database D consisting of endogenous (internal) facts Dn and exogenous (external) facts Dx, the Shapley value of an endogenous fact f is essentially a numerical representation of f's contribution to the query's outcome. The challenge is to compute this value effectively; a task whose complexity depends on the structure of the query itself.

Complexity Analysis and Data Complexity

The complexity of the problem Shapley(q) is approached through the lens of data complexity, which assumes that the query q is fixed and much smaller compared to the database size. This is a reasonable assumption in practice, where databases are voluminous compared to the queries executed on them. Past research, particularly in [20, 27], has revealed a dichotomy in complexity for self-join-free Boolean conjunctive queries: such queries are either tractable, specifically hierarchical, or intractable.

The section's main thrust is to demonstrate that the computation of Shapley values (Shapley(q)) is no harder than probabilistic query evaluation (PQE(q)), for any Boolean query. This relationship is encapsulated in Proposition 3.1, which asserts that Shapley(q) can be polynomial-time Turing reduced to PQE(q). This proposition is a significant bridge, connecting the well-studied domain of PQE with the relatively new territory of Shapley values in databases.

Probabilistic Query Evaluation (PQE)

Probabilistic databases deal with uncertainty in data. A tuple-independent database assigns a probability to each fact, indicating the likelihood of its truth. The problem of PQE involves computing the probability that a given query q holds true in a probabilistic database setting. The tractability of PQE for a query also implies the tractability of computing the Shapley value for the same query, leading to Corollary 3.2, which states that Shapley values can be computed in polynomial time for all 'safe' queries.

Computational Reductions and Their Implications

The proof of Proposition 3.1 hinges on the ability to compute the number of 'slices' (subsets of a certain size that satisfy the query) using an oracle for PQE. The intricacies of the proof involve complex arithmetic and polynomial-time computations, leveraging the structure of a Vandermonde matrix — a square matrix with a distinct structure that allows for easy computation of its determinant and inversion.

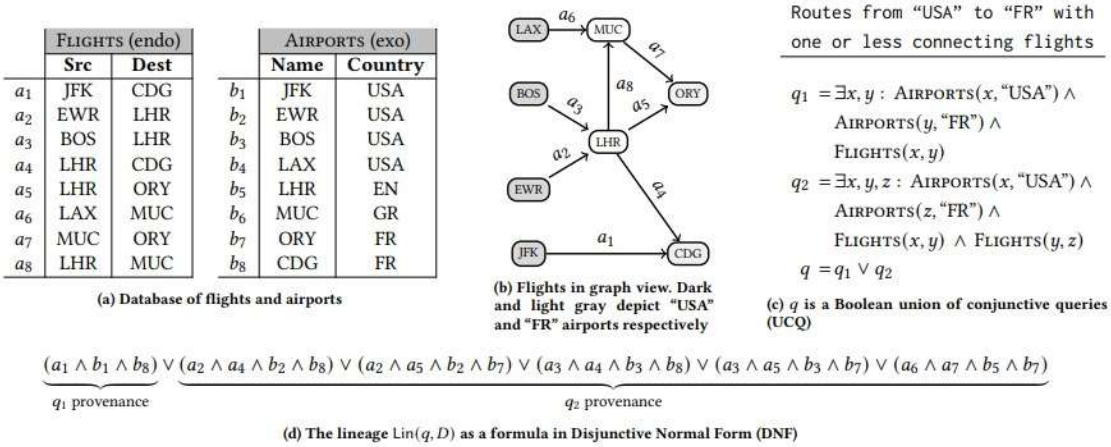
Open Problems and Further Directions

Despite these advancements, open questions remain, notably whether the converse of Proposition 3.1 is true. The report discusses the intriguing possibility of a reduction from PQE to Shapley values for every Boolean query, an area that remains ripe for exploration.

Illustrative Example

To bring these concepts to life, the section includes an example using a database of flights and airports. The database is divided into exogenous facts (airports, with their associated countries) and endogenous facts (flights, with their source and destination). The query in question asks whether there is a route from the USA to France with at most one connecting flight. The Shapley value computation for this Boolean query is illustrated by considering the provenance of the query answer as a Disjunctive Normal Form (DNF) formula. This tangible example elucidates how the abstract concepts of Shapley values play out in real-world scenarios.

In conclusion, the reduction of Shapley value computation to probabilistic query evaluation signifies a pivotal advancement in the application of game-theoretic concepts to database theory. It not only broadens the horizon of tractable cases for Shapley value computation but also strengthens the theoretical underpinnings of database management with regards to complexity analysis.



	Flights (endo)	
	Src	**Dest**
a_1	JFK	CDG
a_2	EWR	LHR
a_3	BOS	LHR
a_4	LHR	CDG
a_5	LHR	ORY
a_6	LAX	MUC
a_7	MUC	ORY
a_8	LHR	MUC

	Airports (exo)	
	Name	**Country**
b_1	JFK	USA
b_2	EWR	USA
b_3	BOS	USA
b_4	LAX	USA
b_5	LHR	EN
b_6	MUC	GR
b_7	ORY	FR
b_8	CDG	FR

(a) Database of flights and airports

(b) Flights in graph view. Dark and light gray depict "USA" and "FR" airports respectively

Routes from "USA" to "FR" with one or less connecting flights

$q_1 = \exists x, y : \text{Airports}(x, \text{"USA"}) \wedge \text{Airports}(y, \text{"FR"}) \wedge \text{Flights}(x, y)$

$q_2 = \exists x, y, z : \text{Airports}(x, \text{"USA"}) \wedge \text{Airports}(z, \text{"FR"}) \wedge \text{Flights}(x, y) \wedge \text{Flights}(y, z)$

$q = q_1 \vee q_2$

(c) q is a Boolean union of conjunctive queries (UCQ)

$$\underbrace{(a_1 \wedge b_1 \wedge b_8)}_{q_1 \text{ provenance}} \vee \underbrace{(a_2 \wedge a_4 \wedge b_2 \wedge b_8) \vee (a_2 \wedge a_5 \wedge b_2 \wedge b_7) \vee (a_3 \wedge a_4 \wedge b_3 \wedge b_8) \vee (a_3 \wedge a_5 \wedge b_3 \wedge b_7) \vee (a_6 \wedge a_7 \wedge b_5 \wedge b_7)}_{q_2 \text{ provenance}}$$

(d) The lineage $\text{Lin}(q, D)$ as a formula in Disjunctive Normal Form (DNF)

Figure 1: The database, query q and its lineage used for our running example

Here's a detailed breakdown of above figures:

Subfigure (a) - Database of flights and airports:

This subfigure lists two tables representing a relational database. The first table is labeled 'FLIGHTS (endo)' and contains endogenous facts, each represented by a tuple with a source airport (Src) and a destination airport (Dest), alongside an identifier (a1 to a8). These facts represent the available flights between different airports.

The second table is labeled 'AIRPORTS (exo)' and contains exogenous facts. This table lists airports (Name) and their corresponding countries (Country), with identifiers from b1 to b8. These facts are considered given and are not the subject of the Shapley value analysis.

Subfigure (b) - Flights in graph view:

This is a graphical representation of the flights listed in the FLIGHTS table from subfigure (a). Airports are shown as nodes, and flights are depicted as edges between these nodes. The graph distinguishes between 'USA' airports (in dark gray) and 'FR' (France) airports (in light gray), showing the connections between them.

Subfigure (c) - Routes from “USA” to “FR” with one or less connecting flights:

Here, two queries (q1 and q2) are presented. They are both Boolean union of conjunctive queries (UCQ) which are used to determine if there are direct flights (q1) or one-stop flights (q2) from the USA to France.

q1 checks for the existence of a direct flight from a 'USA' airport to a 'FR' airport.

q2 extends the check to flights with exactly one connection: it looks for a route from the 'USA' to 'FR' that involves an intermediate connection at any other airport.

q is the combination of q1 and q2, capturing all routes from 'USA' to 'FR' with zero or one stop.

Subfigure (d) - The lineage Lin (q, D) as a formula in Disjunctive Normal Form (DNF):

This subfigure shows the lineage (or provenance) of the query q when applied to the database D, which is represented as a formula in Disjunctive Normal Form. This formula is constructed using logical ORs (∨) of logical ANDs (∧) of facts from the FLIGHTS and AIRPORTS tables, corresponding to the possible routes from 'USA' to 'FR'.

The provenance for q1 includes a single term since there's only one direct flight in the database from 'USA' to 'FR' (JFK to CDG).

The provenance for q2 is more complex, as it includes multiple terms representing different one-stop connections.

Shapley Value Calculation Process:

The Shapley value in this context is used to evaluate the contribution of each flight (endogenous fact) to the Boolean query q. To compute these values, one would consider the presence and absence of each fact in the provenance of q and assess the impact of each fact on the outcome of q. The Shapley value formula (not shown in this image) would then be applied, considering all subsets of endogenous facts, to determine the average marginal contribution of each fact to the truth of the Boolean query.

This image is a comprehensive illustration of how databases are structured, how queries are represented and evaluated, and the beginning steps of how the Shapley value can be computed in the context of database queries. The entire process encapsulates database management, query evaluation, and an application of game theory to databases.5. Experiment

Thorough assessments were conducted of various algorithms using real-world data, demonstrating that our method surpasses baseline techniques in both quality and efficiency. The labeled benchmark data employed for this evaluation is publicly accessible on GitHub, facilitating future research.

Exact Computation through Knowledge Compilation

The section expands on the idea that knowledge compilation, a method traditionally used in computing probabilities in probabilistic databases, can also be applied to compute Shapley values. This technique is grounded in transforming the lineage (or provenance) of a Boolean query into a certain circuit form amenable to efficient probability computation. This concept has been adapted from methodologies used to calculate SHAP-scores in machine learning.

Understanding Lineage and Circuits

To lay the groundwork, we begin with the notion of lineage in the context of databases. The lineage of a Boolean query is a Boolean function representing the contribution of database facts to the query's outcome. In a more formal sense, given a set of variables (facts from a database), an assignment of these variables is a subset indicating which facts are considered present. The set of all assignments is denoted as 2^X. A Boolean function, then, maps these assignments to 0 or 1, indicating whether the combination of facts satisfies the query.

The satisfiability (SAT) of a Boolean function includes all assignments that result in the function evaluating to true. When considering the number of such assignments, we denote this quantity as #SAT. More specifically, SAT_k(φ) refers to the set of satisfying assignments with a specific number of facts present, known as the Hamming weight.

In the context of databases, the lineage of a query is a Boolean function where the variables are facts from the database, mapping each sub-database to the output of the query. For queries with free variables, the lineage is defined similarly but is specific to the tuple of constants that replace the free variables.

From DNF to Knowledge Compilation

Traditionally, the lineage can be expressed as a Disjunctive Normal Form (DNF), a Boolean expression comprising ORs of ANDs. However, for computational purposes, we refine this expression to account for exogenous (given) and endogenous (subject to analysis) facts. The endogenous lineage, ELin(q, Dx, Dn), is the Boolean function considering only endogenous facts, assuming exogenous facts are always present. This lineage is crucial for computing Shapley values as it encapsulates the influence of facts we can control or alter.

Decomposable Deterministic Circuits

The paper then introduces the concept of decomposable deterministic (d-D) circuits, a specific class of Boolean circuits used in knowledge compilation. These circuits consist of AND (∧), OR (∨), NOT (¬), and variable gates with the standard Boolean logic semantics.

A circuit is decomposable if the inputs of every AND gate do not share common variables, ensuring independent sub-functions.

A circuit is deterministic if the inputs to every OR gate represent disjoint functions, meaning no single assignment of variables can make both inputs true simultaneously.

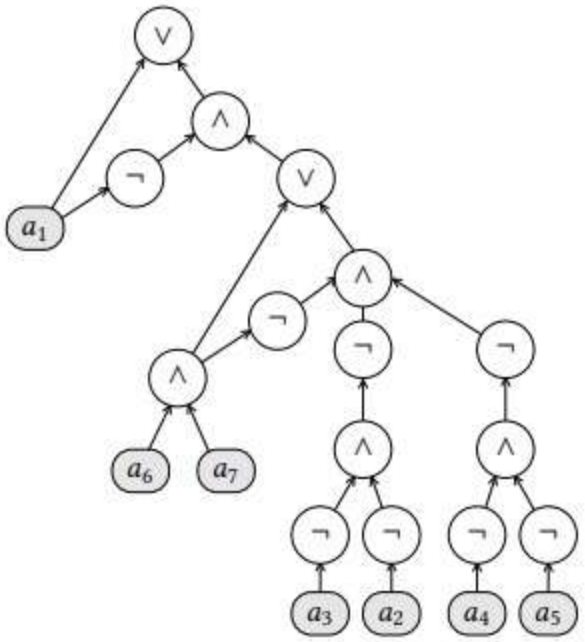


Figure 2: Deterministic and decomposable circuit for ELin(q, D_x, D_n) from the running example.

Figure 2 Analysis

The provided image depicts a d-D circuit for ELin(q, Dx, Dn), representing the endogenous lineage of the query from the running example. The circuit illustrates the structured, tree-like composition of the Boolean function for ELin(q, Dx, Dn), where:

The root node (∨) represents the logical OR operation, indicating that satisfying any of the subfunctions suffices to satisfy the query.

The AND (∧) nodes indicate combinations of facts that must all be present to satisfy the query.

The NOT (¬) nodes invert the truth of the subsequent fact, reflecting the absence of a fact in a given sub-database.

The figure highlights the gates where these operations occur, with the determinism and decomposability properties ensuring efficient computation. For instance, the topmost ∨ gate is deterministic because it separates the possibility of fact a1 being present or absent. The ∧ gates below it are decomposable as they combine independent sets of variables.

In essence, the paper argues that by representing the endogenous lineage of a query as a d-D circuit, one can efficiently compute Shapley values. This is because the d-D structure allows for polynomial-time computation of the relevant probabilities. The methodology presents a significant advancement in database theory, linking the computation of Shapley values to established techniques in knowledge compilation and probabilistic database evaluation.

Algorithmic Computation of Shapley Values via Deterministic and Decomposable Circuits

Proposition 4.4 is the cornerstone of this section, presenting an algorithmic breakthrough for calculating Shapley values in polynomial time. Specifically, given a deterministic and decomposable circuit C representing the endogenous lineage Elin (q,Dn,Dx) for a database

D = Dx U Dn and a boolean query q, along with endogenous fact f belongs to Dn , the Shapley value of f can be computed efficiently relative to the size of the circuit |C|.

Technical Background and Definitions:

Deterministic and Decomposable Circuit (d-D Circuit): A type of Boolean circuit that guarantees efficient computation. A circuit is deterministic if for every OR gate, the subfunctions it represents are mutually exclusive. It's decomposable if for every AND gate, the subfunctions do not share any common variables. This structure is crucial for the polynomial-time computation of Shapley values.

Endogenous Lineage Elin (q,Dn,Dx): A Boolean function representing the influence of variable facts within a database on the outcome of a query, considering that exogenous facts are always present.

Proof Outline of Proposition 4.4:

The proof begins by ensuring that the circuit C includes all variables from Dn. If any variables are missing (i.e., facts that don't appear in the circuit), they are incorporated through a conjunction

that does not alter the circuit's semantics, maintaining its deterministic and decomposable properties.

The proof continues by constructing two circuits, C1 and C2, from C, where C1 assumes the fact f is always present (replaced by a constant 1-gate), and C2 assumes f is absent (replaced by a constant 0-gate). The circuits C1 and C2 thus represent all combinations of endogenous facts with and without the fact f, respectively.

The Shapley value is then calculated using a formula that involves summing over all possible subsets of endogenous facts, adjusting the contribution of each fact by its presence or absence. This formula takes the form:

$$Shapley(q, D_n, D_x, f) = \frac{1}{|D_n|!} \sum_{k=0}^{|D_n|-1} \frac{k!(|D_n| - k - 1)!}{|D_n|} (valuewithf - valuewithoutf)$$

where the "value with f" and "value without f" are computed using C1 and C2, respectively.

Algorithmic Implementation:

By leveraging the properties of C1 and C2, we can compute the quantities required for the Shapley value efficiently. The specific architecture of this computation is outlined in Section 4.2.

Significance and Application:

The ability to compute Shapley values quickly has profound implications for systems that rely on query explanations, such as decision-making models and interpretability in machine learning. This algorithm furnishes a practical approach to what was previously an intractable problem for large databases, enabling precise attribution of database fact contributions to query results.

Graphical Representation and Examples:

While the text does not provide a visual example, the implementation of this algorithm would likely involve constructing and manipulating the circuits C1 and C2. A table or graph depicting the computation steps for the Shapley value, considering various subsets of Dn, could be illustrative in demonstrating the algorithm's efficiency and the polynomial relationship between the circuit size and computation time.

Implementation Architecture

The architecture primarily leverages two tools, ProvSQL and c2d, to process a given database $D = Dx \cup Dn$, a query q(x), a tuple t of same arity as x, and an endogenous fact f belongs to Dn.

Key Components and Tools:

ProvSQL: This is a PostgreSQL extension for computing query provenance, which can be understood as the lineage of query answers. It computes the lineage of a query in various semirings, effectively tracking how input data contributes to query results.
c2d: A knowledge compiler that transforms a Boolean function in Conjunctive Normal Form (CNF) into another formalism called "d-DNNF" (deterministic, decomposable negation normal form).
Process Overview:

Lineage Computation: ProvSQL takes the database D, the query q(x) and the tuple t computing the lineage Lin(q[x/t],D) as a Boolean circuit, denoted as C.

Circuit Conversion: The circuit C is then manipulated to set all exogenous facts to 1 resulting in a new circuit C` representing Elin(q[x/t], Dx, Dn).

CNF Transformation: Since knowledge compilers typically accept CNF input, the circuit C' undergoes the Tseytin transformation to convert it into an equivalent CNF formula ϕ, with additional variables.

Knowledge Compilation: The CNF ϕ is fed into c2d, producing a d-DNNF C'' , equivalent to ϕ.

Variable Elimination: Using Lemma 4.6, the d-DNNF C'' is processed to produce a d-DNNF C''' that is equivalent to C' but without the additional variables introduced by the Tseytin transformation.
Algorithm 1 (Shapley Values from Deterministic and Decomposable Boolean Circuits):

The algorithm, given a d-D circuit with an output gate g Elin(q, Dx, Dn) computes the Shapley value for an endogenous fact f. It involves the following steps:

Completion: Ensuring that all variables in Dn appear in the circuit.

Partial Evaluation: Computing partial evaluations C1 and C2 by setting the fact f to 1 and 0, respectively.

Computation of Satisfying Assignments: Determining the number of satisfying assignments for each k in C1 and C2.

Shapley Value Calculation: Using the computed values to calculate the Shapley value according to the formula:

$$\text{Shapley}(q, D_n, D_x, f) = \frac{1}{|D_n|!} \sum_{k=0}^{|D_n|-1} \frac{k!(|D_n| - k - 1)!}{|D_n|} (\Gamma[k] - \Delta[k])$$

where Γ and Δ are arrays of #SAT values from C_1 and C_2, respectively.

Significance and Challenges:

The architecture provides a novel approach to compute Shapley values, previously impractical for large databases, by harnessing knowledge compilation techniques. However, the paper notes that the efficiency of transforming a CNF into a d-D circuit is not guaranteed, as it is FP#P-hard in general.

Implementation and Experimental Analysis:

Section 6 of the paper contains an experimental analysis that evaluates the practical tractability of the approach. This implementation architecture aims to bridge the gap between theoretical complexity and practical application, enabling the calculation of Shapley values for real-world databases.

The paper presents a sophisticated framework that integrates existing database tools and knowledge compilation methods to compute Shapley values, a task of increasing importance in data-centric applications such as explainable AI and database management systems. The architecture is designed to optimize the computation process, making efficient use of algorithms and transformations to handle the complexities involved in such calculations.

Inexact Computation via CNF Proxy

The CNF Proxy is a heuristic that provides an efficient estimation of Shapley values without necessarily performing exact computations. The heuristic is based on the intuition that facts contributing significantly to the Shapley value tend to have high occurrence rates in the provenance data and possess fewer alternatives, i.e., there are fewer facts that can replace their function.

Key Concepts and Definitions:

Shapley Value: A measure used in cooperative game theory to distribute a total payoff to players according to their individual contributions.
CNF (Conjunctive Normal Form): A way of structuring a Boolean expression as an AND of ORs.

Provenance (Lineage): The history of the data, showing how the data was derived and through which transformations or calculations.
Tseytin Transformation: A method for converting Boolean circuits into CNF while preserving satisfiability.

CNF Proxy Method:

Provenance Occurrences: The heuristic begins by assessing the frequency of each fact's appearance in the provenance. This is directly observable from the CNF form of the provenance circuit, where a higher occurrence suggests a higher importance or contribution.
Alternatives in Provenance: The CNF form also provides insights into the number of alternatives for each fact, which is inferred from the number of alternatives present in each clause of the CNF.
Formal Definition:

The CNF Proxy defines the Shapley value of a function $h : 2^X \to \mathbb{R}$ for a variable $x \in X$ as follows:

$$Shapley(h, x) := \sum_{S \subseteq X \setminus \{x\}} \frac{|S|!(|X| - |S| - 1)!}{|X|!}$$

This formula takes into account the contributions of the fact x across all subsets S of the entire set X, weighted by the size of S and the factorial of the number of facts not in S.

Experimental Validation:

The paper mentions that experimental results, which will be detailed in a later section, confirm that rankings produced by CNF Proxy align closely with those obtained from exact Shapley value calculations. This implies that while CNF Proxy may not provide the precise Shapley value, it can reliably rank the importance of facts in terms of their contributions to a given outcome.

Practical Use:

In practice, the CNF Proxy provides a valuable approximation tool for situations where computing exact Shapley values is infeasible due to computational constraints. By focusing on the occurrence and replaceability of facts in the provenance, this method allows for a quick and efficient estimation of their significance.

Implementation Architecture:

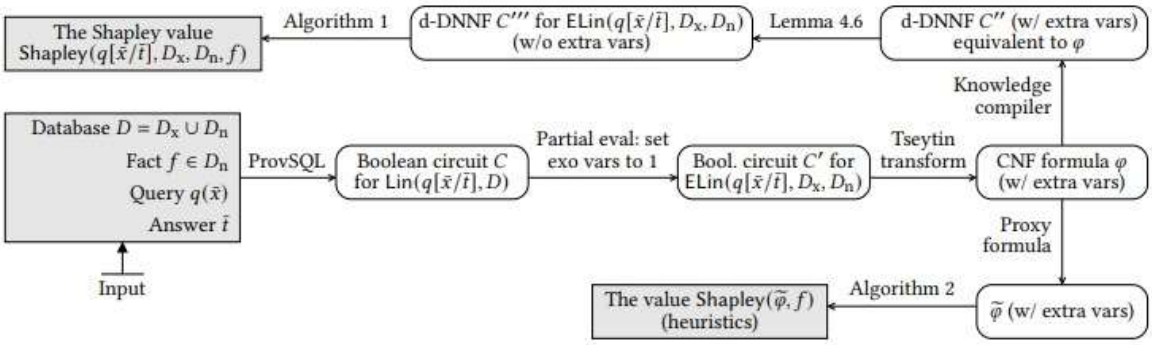


Figure 3: Our implementation architecture.

The figure illustrates the overall architecture for implementing the CNF Proxy method, showing the flow from database input through the various transformations and computations to the output of heuristic Shapley values. This visual representation would be helpful in understanding the step-by-step process of the CNF Proxy methodology.

The CNF Proxy method serves as an efficient heuristic for estimating the Shapley value of facts in a database, particularly useful in large-scale or complex scenarios where exact computation is impractical. This approach enhances the scalability of Shapley value analysis, making it more accessible for a wider range of applications.

The experimental section of the paper reports on the performance of an exact computation algorithm for Shapley values across a set of complex queries. This performance is evaluated on a system implemented in Python 3.6, using PostgreSQL for provenance computation and the c2d compiler for knowledge compilation.

Experiments and Results

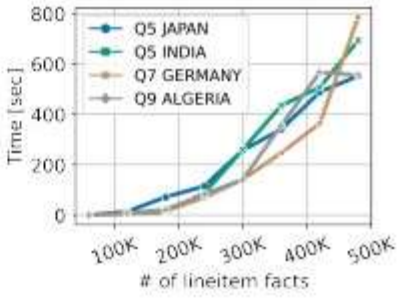
Experimental Setup:

Hardware: The experiments were conducted on a Linux Debian machine with significant computational resources (1TB RAM, Intel Xeon Gold CPU).
Software: The provenance was captured using ProvSQL, and knowledge compilation was performed with the c2d compiler. The source code for the implementation is publicly available.
Benchmark: A new benchmark consisting of 40 complex queries was created over TPC-H and IMDB databases, from which 95,803 output tuples and their provenance expressions were derived.
Key Terms:

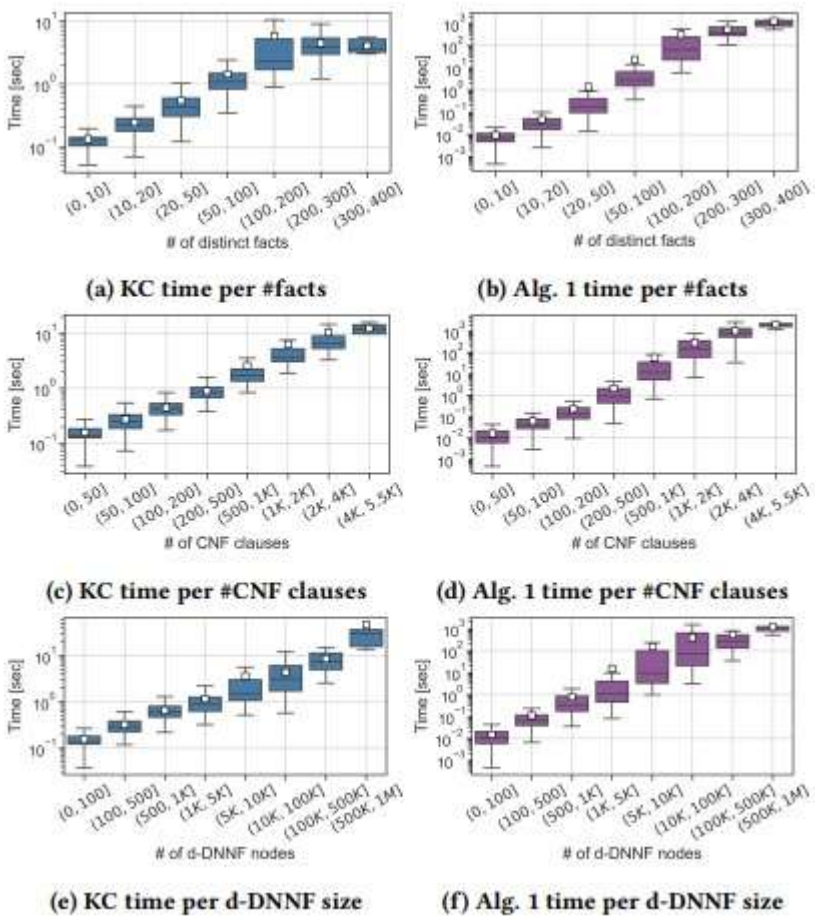
Provenance: The history or origin of data, which in database systems refers to the derivation path of query results.
Knowledge Compilation (KC): A process that transforms complex logical expressions into a form that allows for more efficient query evaluation.
d-DNNF: Deterministic Decomposable Negation Normal Form, a structured representation of Boolean functions that enables efficient logical operations.
Shapley Values: Numerical values that quantify the contribution of each participant (or fact in this context) within a cooperative setting.
Percentiles: Statistical measures indicating the value below which a given percentage of observations fall.
Performance Indicators:

The success rate of the algorithm was high, with the IMDB queries having a 99.96% success rate and the TPC-H queries having an 84.43% success rate.
Execution times for knowledge compilation and Algorithm 1, which computes Shapley values, varied across different queries and are presented in percentiles to capture the distribution of times.
Observations from Results:

(a) Representative outputs



(b) "Difficult" outputs



Execution Time: The execution time of the system includes provenance generation using ProvSQL and the computation time for both the knowledge compilation and Shapley evaluation steps.
Scalability: Figures 4 and 5 depict the running times related to various features such as the number of facts, CNF clauses, and d-DNNF sizes, indicating the system's scalability.
Outliers: Certain queries like q11d had a longer average execution time, demonstrating that while the system is typically efficient, some queries can be outliers.
Tables and Figures:

Table 1: Statistics on the exact computation of Shapley values for 16 representative queries

Dataset	Query	#Joined tables	#Filter conditions	Execution time [sec]	#Output tuples	Success rate	KC execution times [sec]					Alg. 1 execution times [sec]				
							Mean	p25	p50	p75	p99	Mean	p25	p50	p75	p99
TPC-H	3	3	5	20980.71	100	100%	0.06	0.04	0.07	0.08	0.13	0.00	0.00	0.00	0.00	0.01
	5	6	9	48.67	5	0%	-	-	-	-	-	-	-	-	-	-
	7	6	8	30.57	4	0%	-	-	-	-	-	-	-	-	-	-
	10	4	6	4.72	10	100%	0.14	0.13	0.13	0.14	0.19	0.01	0.01	0.01	0.01	0.01
	11	6	7	0.13	10	100%	0.09	0.08	0.08	0.08	0.13	0.01	0.00	0.00	0.01	0.03
	16	3	5	1.25	10	100%	0.26	0.13	0.18	0.33	0.57	0.18	0.01	0.03	0.29	0.88
	18	4	3	37.31	10	100%	0.13	0.08	0.08	0.18	0.23	0.01	0.00	0.01	0.01	0.03
	19	2	21	2.04	1	100%	1.20	1.20	1.20	1.20	1.20	156.06	156.06	156.06	156.06	156.06
IMDB	1a	5	10	0.25	35	100%	0.17	0.08	0.08	0.13	2.10	5.92	0.01	0.01	0.01	206.72
	6b	5	8	2.61	1	100%	0.44	0.44	0.44	0.44	0.44	0.08	0.08	0.08	0.08	0.08
	7c	8	21	77.33	2415	99%	0.82	0.18	0.28	0.63	9.44	24.28	0.02	0.05	0.39	787.12
	8d	7	10	145.10	44517	99.9%	0.26	0.13	0.18	0.29	1.09	0.36	0.01	0.02	0.05	2.28
	11a	8	18	3.20	10	100%	0.75	0.18	0.48	1.14	2.15	23.34	0.01	0.13	1.27	151.99
	11d	8	16	56.99	210	98.1%	0.99	0.19	0.32	0.78	5.97	96.19	0.02	0.06	0.48	1650.72
	13c	9	19	2.44	14	100%	0.22	0.13	0.18	0.28	0.53	0.02	0.01	0.01	0.01	0.06
	15d	9	18	24.25	207	97.6%	1.89	0.24	0.48	1.28	10.01	70.05	0.06	0.30	3.52	1821.13
	16a	8	15	5.56	173	100%	0.18	0.13	0.14	0.20	0.53	0.02	0.01	0.01	0.02	0.12

Table 1: Presents detailed statistics on the execution of Shapley values for 16 representative queries, showing the success rate and execution times across different percentiles.
Figures 4a-f: Show the knowledge compilation time and Algorithm 1 time against various metrics, such as the number of distinct facts, CNF clauses, and d-DNNF nodes.
Figures 5a-b: Display the running time for various TPC-H query outputs as a function of table size, distinguishing between representative and "difficult" outputs.

The experimental results showcase the system's capability to efficiently compute exact Shapley values for a majority of cases within a practical timeframe. The variation in execution times across queries provides insights into the algorithm's performance under different data complexities, confirming its applicability to real-world databases with a high degree of success. These findings highlight the potential of the system for applications that require an understanding of individual contributions to complex database queries.

The experimental analysis in the paper evaluates inexact computation methods for estimating Shapley values, comparing the performance of CNF Proxy against two sampling-based methods: Monte Carlo and Kernel SHAP. These methods offer a means to estimate Shapley values when exact computation is infeasible due to resource constraints or complexity.

Sampling Methods:

Monte Carlo: A probabilistic method that approximates Shapley values by randomly sampling permutations of input facts and observing their impact on the outcome.
Kernel SHAP: A method developed for machine learning explainability that uses a weighted linear regression model to estimate the SHAP values, accounting for feature contributions to a model’s output.
Performance Metrics:

The performance of these algorithms is evaluated using several metrics:

Execution Time: How long it takes for each method to compute the Shapley values.

nDCG (Normalized Discounted Cumulative Gain): A measure of ranking quality, assessing how well the inexact methods order the facts compared to the ground truth.
Precision@k: The accuracy of the inexact methods in identifying the top-k most influential facts.
L1 and L2 Norms: Measures of the absolute and squared differences, respectively, between the estimated and actual Shapley values.
Findings:
Execution Speed: CNF Proxy, which does not rely on sampling, consistently exhibits faster execution times compared to Monte Carlo and Kernel SHAP.
Quality of Estimation: In terms of ranking quality (nDCG), CNF Proxy generally outperforms the sampling methods. It maintains high nDCG scores regardless of the provenance size, suggesting its robustness in ranking fact contributions.
Precision: CNF Proxy also demonstrates superior precision in identifying the most influential facts, especially notable when dealing with a larger number of facts in the provenance.
Dependency on Provenance Size:
The analysis observes that the performance of Monte Carlo and Kernel SHAP tends to deteriorate as the number of facts in the provenance increases, whereas CNF Proxy remains relatively stable.
Figures and Tables:

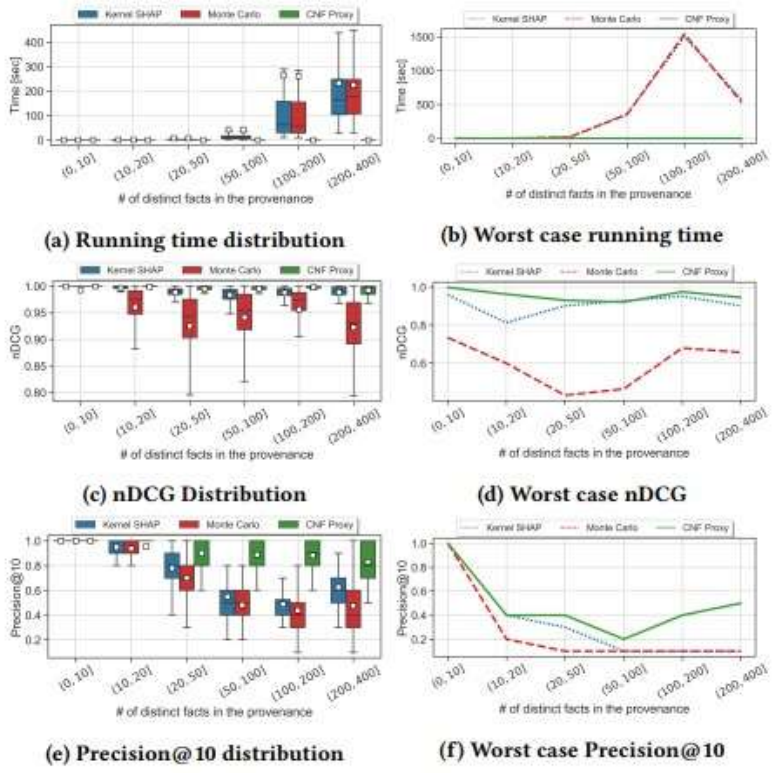
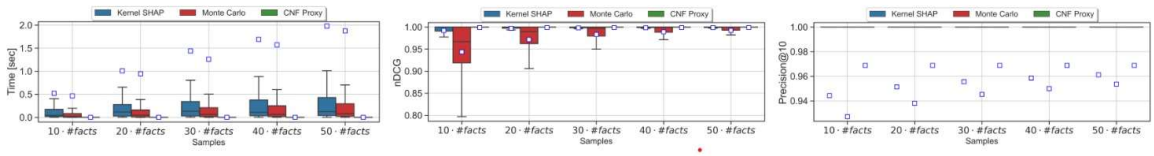


Figure 7: Comparing inexact methods as a function the number of distinct facts in the provenance (n). Sampling methods (Kernel SHAP and Monte Carlo) presented with sampling budget of $m = 20n$; different budgets led to similar trends.



(a) Execution time (b) nDCG (c) Precision@10

Figure 6: Comparing various metrics for inexact methods as a function of the sampling budget; CNF Proxy does not rely on sampling, thus remains constant for all budgets.

Figures 6 and 7: These figures illustrate the performance of the three methods across different metrics and as a function of the number of distinct facts in the provenance. They highlight the robustness of CNF Proxy in various scenarios, including worst-case analyses.
Table 2: This table provides a detailed comparison of the median and mean performance of the three methods, with a focus on execution time and accuracy metrics.

The experiments suggest that while sampling methods like Monte Carlo and Kernel SHAP can provide accurate estimates of Shapley values, they may become less practical as the complexity of the provenance increases. The CNF Proxy method emerges as a fast and reliable alternative, particularly well-suited for large-scale or complex datasets. This method can efficiently approximate fact rankings, which is crucial in applications where understanding the influence of individual data points is necessary.

Implications for Practice:

The findings underscore the usefulness of CNF Proxy in settings where quick estimations of fact contributions are needed. It provides a balance between computational efficiency and accuracy in ranking, making it a valuable tool for practitioners working with large databases and complex queries.
Hybrid Approach Overview:

The hybrid method begins with an attempt at exact computation through knowledge compilation and Algorithm 1. If this exact process completes within a configurable timeout

t seconds, its result is used. If it doesn't complete within the timeout, the process switches to using the CNF Proxy method, which offers an efficient ranking of input tuples based on estimated Shapley contributions.

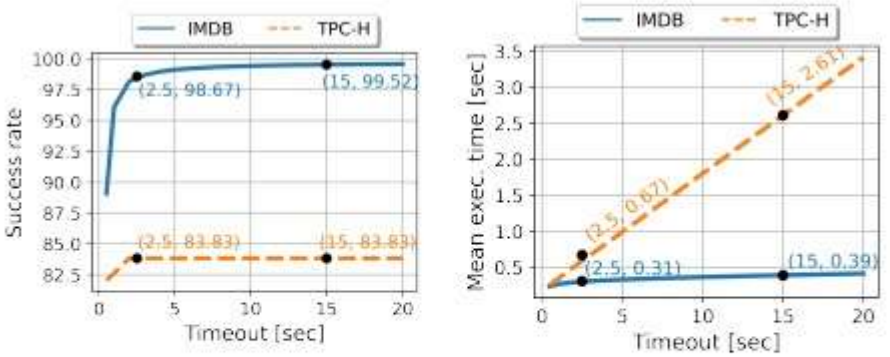
Success Rate and Execution Time:

The exact computation has a high success rate, with 99.96% for IMDB and 84.43% for TPC-H when given ample time (up to an hour).
Under a strict timeout of 2.5 seconds, the success rates for exact computations are 98.67% for IMDB and 83.83% for TPC-H.

Increasing the timeout to 15 seconds only marginally improves the success rate for IMDB and does not affect the rate for TPC-H.
The mean execution time for the hybrid method is modest, with 0.31 seconds for IMDB and 0.67 seconds for TPC-H under a 2.5-second timeout.
Key Conclusions:

For the majority of cases, exact computation of Shapley values is feasible within a few seconds.
The CNF Proxy offers a much faster alternative for cases where exact computation is not possible within the timeout, providing results in milliseconds to a few seconds.
Rankings derived from the CNF Proxy are shown to be accurate in comparison to the ground truth where exact computation is successful.
Technical Terms:

Knowledge Compilation: The process of transforming logical expressions into a form that is easier to work with computationally.
Algorithm 1: A specific procedure outlined in the paper for computing exact Shapley values.
CNF Proxy: An inexact method that estimates the ranking of Shapley values based on a Boolean function's CNF representation.
nDCG (Normalized Discounted Cumulative Gain): A metric for evaluating the quality of a ranking system, with respect to a set of ground truth values.
Precision@k: A measure of how many of the top-k ranked items by an algorithm are in the top-k of the ground truth ranking.
Figures and Tables:



(a) Exact computation success rate as function of the computation timeout

(b) Mean execution time of the hybrid approach as a function of the computation timeout

Figure 8: Hybrid approach performance

Figure 8a: Illustrates the exact computation success rate as a function of the computation timeout, revealing the threshold beyond which additional time yields minimal benefit.
Figure 8b: Demonstrates the mean execution time of the hybrid approach in relation to the computation timeout, showing how the mean time increases with the timeout.
Implications for Practice:

The hybrid approach provides a practical solution for computing Shapley values, particularly in settings where computational resources are limited or the dataset is too large for exact computations. It allows for accurate and efficient estimations of the importance of facts in a database, balancing the need for speed with the desire for precision.

Applications:

Such a hybrid method is valuable in fields where decision-making relies on understanding contributions of individual factors, such as in data science, economics, and cooperative game theory, particularly when interpretability and explainability of models are critical.

In the conclusion of the paper, the authors summarize their contributions to the field of database management and analysis through the development of a practical framework for computing Shapley values, which quantify the contributions of individual database facts to the results of a query.

Key Contributions:

Practical Framework: A novel and practical method for assessing the contributions of database facts to query answers using Shapley values.
Exact and Inexact Algorithms: The paper describes an exact algorithm for computing these contributions and a faster, inexact algorithm that ranks the contributions effectively.
Theoretical Foundations: A significant theoretical contribution is the establishment of a connection between computing Shapley values and probabilistic query evaluation, suggesting that the computation of Shapley values can be polynomially reduced to the latter.
Technical Terms:

Shapley Values: A concept from cooperative game theory applied to determine the fair distribution of a total payoff to participants according to their contributions. In databases, it represents the contribution of individual facts to the outcome of a query.
SPJU Queries: Select-Project-Join-Union queries, a class of SQL queries that involve basic operations like selecting, projecting, joining, and unioning tables.
Probabilistic Query Evaluation (PQE): A method for evaluating queries when the data involved has an associated probability, reflecting uncertainty or incomplete information.
Open Questions and Future Directions:

Open Problem 1: The authors mention an unresolved question about whether the reduction between Shapley values and PQE also exists in the opposite direction.
Further Constructs: Suggestions for future work include studying the application of Shapley value computations to other SQL constructs like aggregates and negation.

Bag Semantics: The current framework treats tuples individually, but there's an opportunity to adapt the framework to account for multiple identical tuples within a database, considering fact multiplicities more thoroughly.
Implications for Practice:

The framework is designed for the class of queries supported by ProvSQL, which makes it directly applicable to a wide range of practical database management systems. The exact algorithm provides detailed contribution calculations, while the faster algorithm allows for quick ranking of contributions, which can be particularly useful for large datasets where computational resources or time are constraints.

Applications:

The methodology proposed in this paper is essential for situations where it is necessary to understand the influence of individual data entries on the outcome of database queries. This is particularly relevant in fields like business intelligence, data analytics, and the emerging domain of explainable AI.

Conclusion:

The paper concludes with a robust contribution to the database analysis field, providing both practical tools and theoretical insights. It sets the stage for further exploration in effectively computing and interpreting the impact of database facts on query results, paving the way for more transparent and fair data management practices.