srtp analysis

shapley value

idea:

if we have a coalition C that collaborates to produce a value V

how much did each individual member contribute to the final value

1. compare the coalition formed w/ and w/o the member

V_{1234}-V_{234} = marginal contribution of member to C_{234}

enumerate all pairs of coalition and compute the mean

A unified approach to interpreting model predictions

- how individual features contribute to a model's outputs

shapley additive explanations

- local accuracy

let x' be simplified local inputs

g(x') be a explanatory model

if x is roughly equal to x'

then f(x) must be roughly equal to g(x)

missingness

x' = 0 → phi must be 0

consistency

- if feature contribution changes, the feature effect cannot change in the opposite direction

Data provenance is **the documentation of where a piece of data comes from and the processes and methodology by which it was produced**.

shapley kernel → universal used

exogenous → given facts

endogenous → attribute contributions

Relational databases and queries. Let $\Sigma = \{R_1, \ldots, R_n\}$ be a *signature*, consisting of *relation names* R_i each with its associated *arity* $\mathrm{ar}(R_i) \in \mathbb{N}$, and Const be a set of *constants*. A *fact* over (Σ, Const) is simply a term of the form $R(a_1, \ldots, a_{\mathrm{ar}(R)})$, for $R \in \Sigma$ and $a_i \in \mathrm{Const}$. A (Σ, Const)-*database* D, or simply a *database* D, is a finite set of facts over (Σ, Const). We assume familiarity with the most common classes of query languages and refer the reader to [1] for the basic definitions. In particular, we recall the equivalence between relational algebra and relational calculus [1], and the fact that Select-Project-Join-Union (SPJU) queries are equivalent to unions of conjunctive queries (UCQs). Depending of the context and for consistency with relevant past publications, we will use terminology of either relational calculus or relational algebra. What we call a *Boolean query* is a query q that takes as input a database D and outputs $q(D) \in \{0, 1\}$. If $q(\bar{x})$ is a query with free variables $\bar{x}$ and $\bar{t}$ is a tuple of constants of same length as $\bar{x}$, we denote by $q[\bar{x}/\bar{t}]$ the Boolean query defined by: $q[\bar{x}/\bar{t}](D) = 1$ if and only if $\bar{t}$ is in the output of $q(\bar{x})$ on D.

Shapley values of facts. Following [20, 27], we use the notion of Shapley values [32] to attribute a contribution to facts of an input database. In this context, the database D is traditionally partitioned into two sets of facts: a set D_{x} of so-called *exogenous* facts, and a set D_{n} of *endogenous* facts. The idea is that exogenous facts are considered as given, while endogenous facts are those to which we would like to attribute contributions. Let q be a Boolean query

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

Notice that here, $|E|!(|D_n| - |E| - 1)!$ is the number of permutations of D_n with all endogenous facts in E appearing first, then f, and finally, all the other endogenous facts. Intuitively then, the value $\text{Shapley}(q, D_n, D_x, f)$ represents the contribution of f to the query's output: the higher this value is, the more f helps in satisfying q.

For non-Boolean queries $q(\bar{x})$, we are interested in the Shapley value of the fact f for every individual tuple $\bar{t}$ in the output [20]. The extension to non-Boolean $q(\bar{x})$ is then straightforward: the Shapley value of the fact f *for the answer* $\bar{t}$ *to* $q(\bar{x})$ is the value $\text{Shapley}(q[\bar{x}/\bar{t}], D_n, D_x, f)$. Therefore, the computational challenge reduces to that of the Boolean query $q[\bar{x}/\bar{t}]$. Hence, in the theoretical analysis we focus on Boolean queries, and we go back to considering non-Boolean queries when we study the implementation aspects (starting in Section 4.2).

non boolean queries can be reduced to boolean query problem

Livshits et al. [20, 27] initiated the study of the computational complexity of calculating Shapley values in query answering. They showed mainly lower bounds on the complexity of the problem, with the exception of the sub-class of self-join free SPJ queries called *hierarchical*, where they gave a polynomial-time algorithm. The results are more positive if imprecision is allowed, as they showed that the problem admits a tractable approximation scheme (FPRAS, to be precise) via Monte Carlo sampling. The state of affairs is that the class of known tractable cases (namely the hierarchical conjunctive queries) is highly restricted, and the approximation algorithms with theoretical guarantees are impractical in the sense that they require a large number of executions of the query over database subsets (the samples). Hence, the theoretical analysis of Livshits et al. [20, 27] does not provide sufficient evidence of practical feasibility for adopting the Shapley value as a measure of responsibility

- showed lower bounds
- the results are more positive if imprecision allowed

in query answering. Moreover, the results of Livshits et al. [20] imply that, for self-join free SPJ queries the class of tractable queries for computing Shapley values coincides with the class of tractable queries in probabilistic tuple-independent databases [7]. Yet, no direct connection has been made between these two problems and, theoretically speaking, it has been left unknown whether algorithms for probabilistic databases can be used for Shapley computation.

Contribution of sigmod 22

One of our contributions is to show that the techniques developed by [2 , 3, 36] can be adapted to the context of Shapley values for databases. For instance, by adapting to our context the proof of Van den Broeck et al. [36] that computing the SHAP-score reduces to computing the expected value of the model, we resolve the aforementioned open question a rmatively: we prove that Shapley computation can be e ciently (polynomial-time) reduced to probabilistic query answering. Importantly, this applies not only to the restricted class of SPJ queries without self-joins, but to every database query.

Sol1

Compute the Boolean provenance of a given output tuple in the sense of Imielinski and Lipski and then to compile it into a circuit form (deterministic and decomposable circuit)

While the aforementioned properties of the circuit are not guaranteed in general (beyond the class of hierarchical queries), we empirically show the applicability and usefulness of the approach even for non-hierarchical queries.

Their method involves analyzing the dependencies between the input and output attributes of a query or update transaction, and using this information to construct a set of Boolean expressions that represent the possible combinations of input values that could have produced the output.

The Boolean provenance of an output tuple refers to the set of input tuples that contributed to the computation of the output tuple. In other words, it is the set of all possible combinations of input values that could have led to the given output tuple.

C = (A AND NOT B) OR (NOT A AND B)

Suppose the output tuple we are interested in is (C=true). To compute its Boolean provenance, we need to identify all the possible combinations of input values that could have produced C=true. In this case, there are two such combinations:

- A=true, B=false
- A=false, B=true

We can represent these combinations as Boolean expressions:

- (A=true) AND (B=false)
- (A=false) AND (B=true)

Therefore, the Boolean provenance of the output tuple (C=true) is the set of these two Boolean expressions:

{ (A=true) AND (B=false), (A=false) AND (B=true) }

- results indicate that exact computation is fast in most cases, but costly in others
- For the latter cases, we propose a heuristic approach to retrieve the relative order of the facts by their Shapley values, without actually computing these values. Indeed, determining the most in□uential facts is in many cases already highly useful, even if their precise

Solution 2

contribution remains unknown. The solution that we propose to this end is termed *CNF Proxy*; it is based on a transformation of the provenance to Conjunctive Normal Form (CNF) and using it to compute proxy values intuitively based on (1) the number of clauses in which a variable occur and (2) its alternatives in each clause. These are two aspects that are correlated with Shapley values. The proxy values may be very different from the real Shapley values, and yet, when we order facts according to their proxy values we may intuitively get an ordering that is similar to the order via Shapley. Our experiments validate that this intuition indeed holds for examined benchmarks.

We have experimented with multiple queries from the two stan-

HAP score

Results

```
In most cases (98.67% of the IMDB output tuples and 83.83%
for TPC-H), our exact computation algorithm terminates in 2.5 sec-
onds or less, given the provenance expression. In the vast majority
of remaining cases the execution is very costly, typically running
out of memory already in the Knowledge Compilation step.
```

```
By
contrast, our inexact solution CNF Proxy is extremely fast even for
these hard cases - it typically terminates in a few milliseconds with
the worst observed case (an outlier) being 4 seconds
```

Simple hybrid approach

execute the exact
algorithm until it either terminates or a timeout elapses. If we have
reached the timeout, resort to executing CNF Proxy and rank the
facts based on the obtained values

Hence, our contributions are both of a theoretical and practical nature and can be summarized as follows. *(1)* By adapting the proof technique of [36], we establish a fundamental result about the complexity of computing Shapley values over relational queries: Shapley values can be computed in polynomial time–in data complexity–whenever the query can be evaluated in polynomial time over tuple-independent probabilistic databases (Proposition 3.1). This holds for *every* query. *(2)* By adapting the proof technique of [2, 3], we devise a novel algorithm for computing Shapley values for query evaluation via compilation to a deterministic and decomposable circuit (Proposition 4.4). We show that this algorithm is practical and has the theoretical guarantee of running in polynomial time in the size of the circuit. *(3)* We present a novel heuristic, CNF Proxy, that is fast yet inexact, and is practically effective if we are interested in ranking input facts by their contribution rather than computing exact Shapley values (Section 5). *and (4)* We describe a thorough experimental study of our algorithms over realistic data and show their efficiency (Section 6).